Servidor.pl

```
#!/usr/local/bin/perl

use Socket;

($port) = @ARGV;

$port = 2345 unless $port;
```

Empleamos el módulo Socket, equivalente a las definiciones que encontramos en «/usr/include/sys/socket.h». Además incluye valores habituales que habría que incluir a mano, y que dependen de la plataforma, como:

$AF_INET = 2;

$SOCK_STREAM = 1;

Toma el valor de los argumentos del programa. Si se recibe alguno, debe ser el número de puerto del socket. Se sitúa en «port»

Comprobamos si «port» tiene un valor asignado, y si no es así, se le da el valor «2345»

Servidor.pl (2)

```
($name, $aliases, $protocol) = getprotobyname('tcp');
```

Carga en nuestro programa, información importante sobre el protocolo orientado a conexión «tcp». Buscar más en «/etc/protocols»

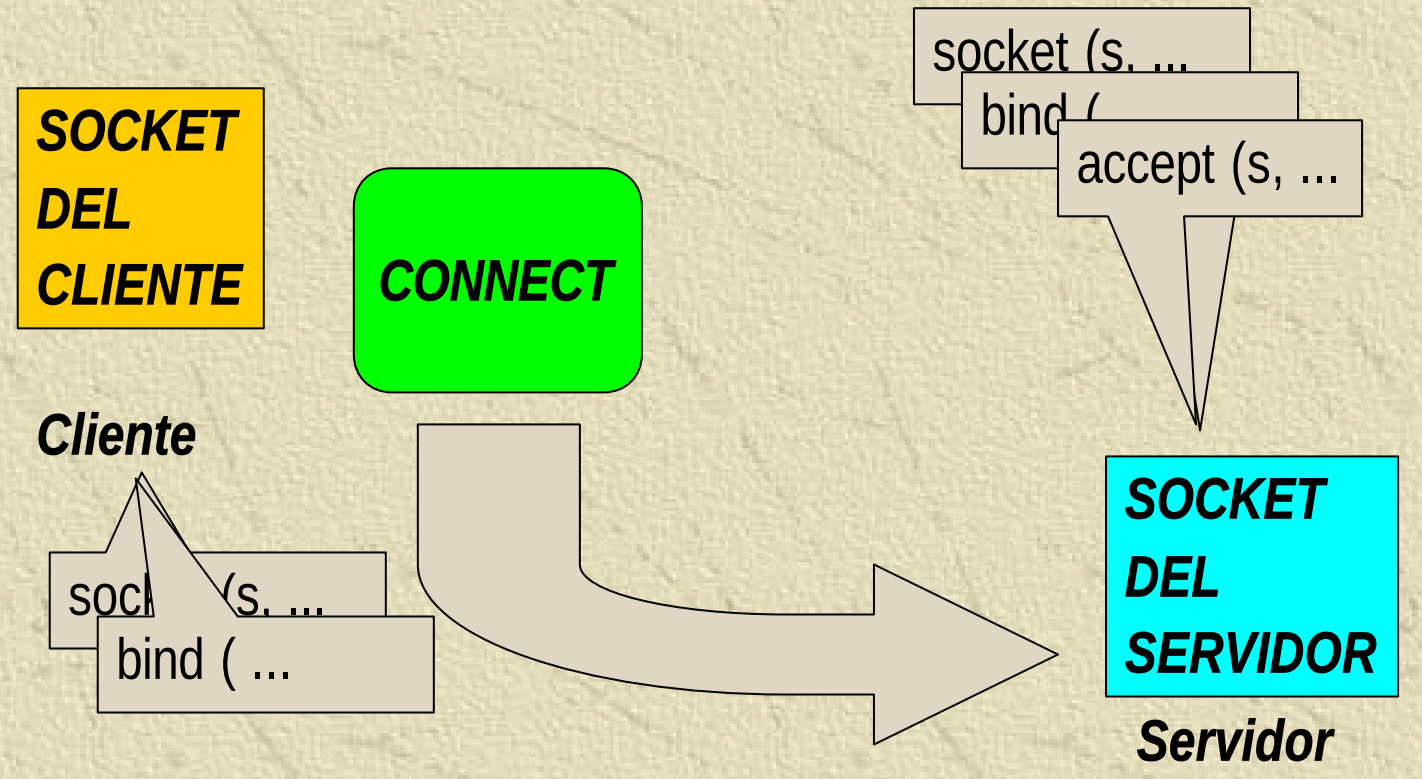
Si la variable «port» no contiene un número, se busca el puerto asociado al nombre simbólico, en «/etc/services» (p.ej.: echo, daytime, ftp, ...).

=~ efectúa un *pattern matching*, !~ es la negación. El patrón va entre / /, \d+ es uno o más dígitos, ^ y $ indican principio y final de la cadena.

```
if ($port !~ /^\d+$/) {
    ($name, $aliases, $port) = getservbyport($port, 'tcp');
}
```

Servidor.pl (3)

La creación del stream de comunicación con sockets tcp es como sigue:



Servidor.pl (4)

```
print "Listening on port $port...\n";
socket (S, AF_INET, SOCK_STREAM, $protocol) ||
        die "socket : $!";
```

«S» es el handler del socket.

«AF_INET» indica que usamos puertos.

Podríamos haber usado sockets UNIX con «AF_UNIX»

«SOCK_STREAM» indica que creamos streams.

También es posible usar datagramas con «SOCK_DGRAM»

Servidor.pl (5)

```
$sockaddr = 'S n a4 x8';
$this = pack($sockaddr, AF_INET, $port, "\0\0\0\0");
```

«sockaddr» es un patrón de bits con la dirección del socket, en formato C, luego hay que empaquetarlo con «pack».

```
bind(S, $this) || die "bind : $!";
```

«bind» enlaza la dirección con el socket

Como un servidor puede atender a más de un cliente, con «listen» se indica cuantas llamadas entrantes se pueden poner en espera.

```
listen(S,10) || die "listen: $!";
```

Servidor.pl (6)

```
select(S);
$| = 1;
select(STDOUT);
for ($con = 1; ; $con++) {
    printf("Waiting for connection %d....\n", $con);
    ($addr = accept(NS,S)) || die $!;
```

«$|» especifica el tamaño del búfer.

Pone el búfer del socket a 1.

Contamos el número de conexiones que se realizan

El proceso se bloquea en «accept» hasta que se recibe una petición de conexión.

Cuando se recibe una conexión, se acepta, y se transfiere a otro socket alternativo «NS», y se libera el original «S»

«NS» se crea en «accept». «$addr» contiene la dirección del cliente.

Servidor.pl (7)

```
select(NS);
$| = 1;
select(STDOUT);
```

«fork» devuelve 0 para el proceso hijo, y el «PID» del hijo, para el padre.

```
if (($child = fork()) == 0) {
```

Desempaquetamos la dirección del proceso cliente, y la imprimimos

```
    ($af,$port, $inetaddr) = unpack($sockaddr, $addr);
    @inetaddr = unpack('C4', $inetaddr);
    print “Connection $con @ Iaddress @inetaddr\n";
```

Servidor.pl (8)

```
while (<NS>) {
        print "Received from client $con: $_";
        print   NS "Server $con: $_";
    }
    close(NS);
    print "Client exits.  Forked server $con exiting...\n";
    exit;
}
```

«NS» es el handle socket en modo stream.

«<NS>» lee del stream y devuelve la próxima línea del socket.

Leemos de él hasta que se llega al fin del archivo.

La línea leída va a «$_».

ECO

Limpieza del proceso hijo

Servidor.pl (9)

```
    close(NS);
}
```

El proceso padre cierra el socket «NS» que ha desviado al proceso hijo.