LLamadas al Sistema Unix

Sistemas Operativos

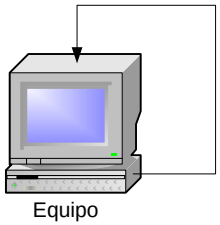
Escuela de Ingeniería Civil Informática

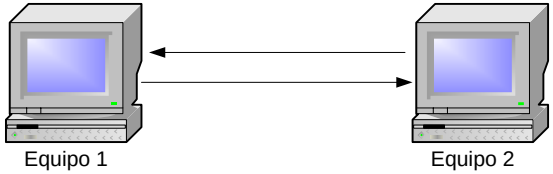
Sockets

UNIVERSIDAD DEL BÍO-BÍO

¿Qué es un Socket?

- Es una interfaz de E/S de datos que permite la intercomunicación entre procesos.
- Los procesos pueden estar ejecutándose en el mismo o en distintos sistemas, unidos mediante una red.





- Como analogía, los sockets permiten la comunicación entre procesos, como los teléfonos permiten la comunicación entre las personas.
- Los sockets se crean dentro de un **dominio de comunicación**, igual que un archivo se crea dentro de un *filesystem* (sistema de archivos).

Dominios de comunicación

- Algunos dominios:

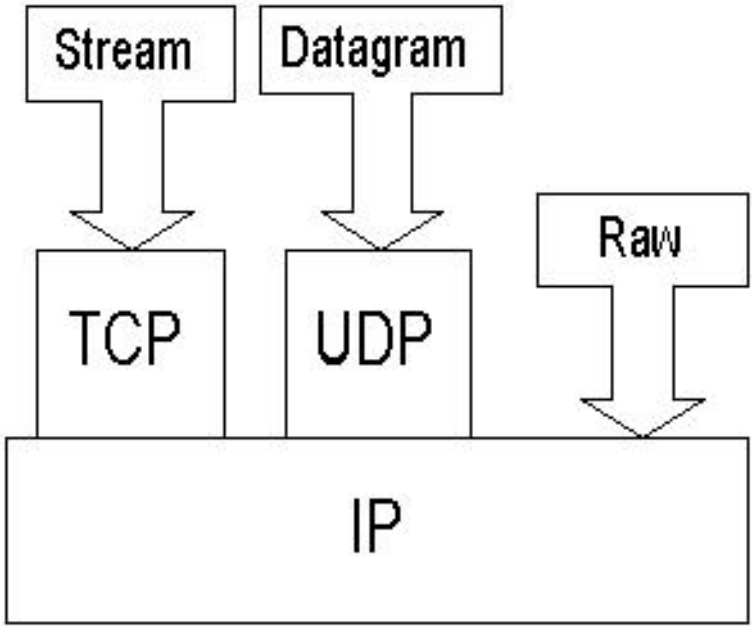
  - **AF_INET** (unidos mediante una red TCP).

  - **AF_UNIX** (en el mismo sistema).

    - Otros dominios distintos de TCP.
      - AF_NS /* protocolos XEROX NS */
      - AF_CCITT /* protocolos CCITT, protocolos X.25, etc. */
      - AF_SNA /* IBM SNA */
      - AF_DECnet /* DECnet */
      - Netbios /* Microsoft */
      - etc.

Tipos de Sockets

- Sockets Stream.
- Sockets Datagram.
- Sockets Raw.



Sockets Stream

- Son los más utilizados, hacen uso del protocolo TCP.
- El protocolo TCP provee un flujo de datos bidireccional, secuenciado, sin duplicación de paquetes y libre de errores.
- Estos sockets también son llamados "**sockets orientados a conexión**" porque es necesario abrir una conexión al comienzo, realizar la comunicación y cerrar la conexión al terminar.
- La especificación del protocolo TCP se puede leer en la RFC-793 .

  RFC: http://www.rfc.net/

Socket Datagram

- Hacen uso del protocolo UDP.
- El protocolo UDP nos provee un flujo de datos bidireccional, pero los paquetes pueden llegar fuera de secuencia, pueden no llegar o contener errores.
- Por lo tanto, el proceso que recibe los datos debe realizar resecuenciamiento, eliminar duplicados y asegurar la confiabilidad.
- Se llaman también "**sockets sin conexión**", porque no hay que mantener una conexión activa, como en el caso de sockets stream.

Socket Datagram

- Son utilizados para transferencia de información paquete por paquete. Ejemplo: DNS.

- Entonces podríamos preguntar, ¿Cómo hacen estos programas para funcionar si pueden perder datos?

  R: Ellos implementan un protocolo encima de UDP que realiza control de errores.

- La especificación del protocolo UDP se puede leer en la RFC-768.

Socket Raw

- No son para el usuario más común, son provistos principalmente para aquellos interesados en desarrollar nuevos protocolos de comunicación o para hacer uso de facilidades ocultas de un protocolo existente.

Byte Order

- En una red pueden haber computadores con procesadores (CPU´s) de distintas tecnologías. Esto ocasiona problemas si los bytes no se representan de la misma forma en todos ellos.

- Un sistema puede almacenar los bytes en la memoria de 2 formas diferentes:
  - Network byte order

int (4bytes)

00000000	00000000	00000000	00000001

+significativo -significativo

  - Host byte order

int

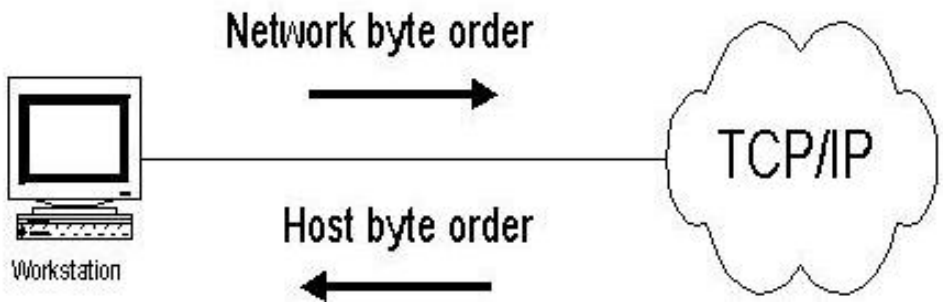
10000000	00000000	00000000	00000000

-significativo +significativo

- El ejemplo muestra como se representa el número “1” de ambas formas.

Byte Order

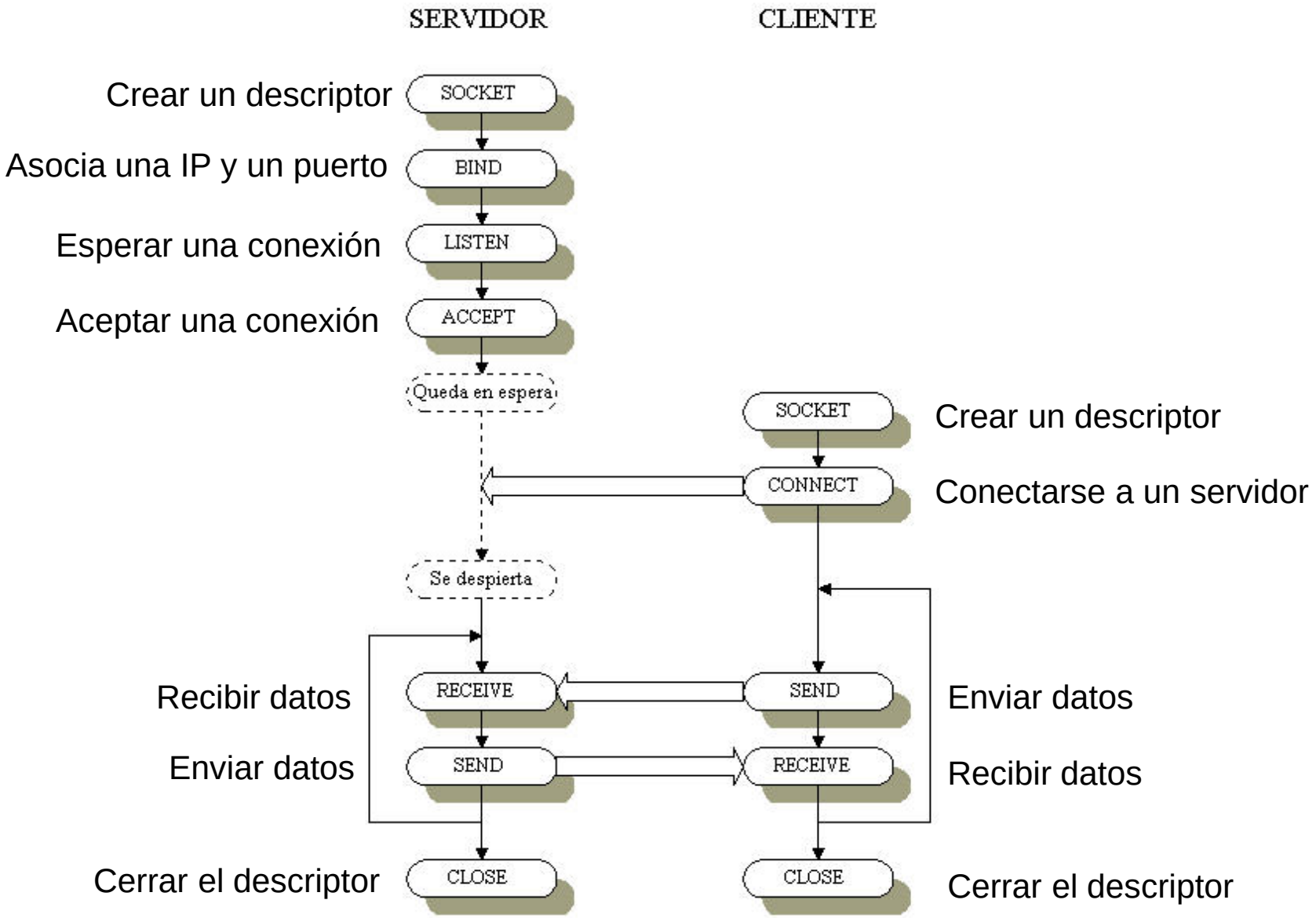
- Para evitar problemas y no enviar los datos al revés, la comunicación por socket establece las siguientes reglas :
  - Todos los bytes que se transmiten hacia la red, sean números IP o datos, deben estar en network byte order.
  - Todos los datos que se reciben de la red, deben convertirse a host byte order.



Byte Order

- Para hacer la implementación independiente de la plataforma, se utilizan las siguientes funciones de conversión:
  - **htons()** - *host to network short* - convierte un short int de host byte order a network byte order.
  - **htonl()** - *host to network long* - convierte un long int de host byte order a network byte order.
  - **ntohs()** - *network to host short* - convierte un short int de network byte order a host byte order.
  - **ntohl()** - *network to host long* - convierte un long int de network byte order a host byte order.
- Para más información : `$man 3 byteorder`

Ejemplo cliente-servidor TCP simple (1 sólo cliente)



Crear un servidor Socket

Pasos para crear un servidor

- Utilizaremos un socket de tipo stream con protocolo TCP.

- Los pasos son:

  - socket() → crear el descriptor de socket
  - bind() → asociar una dirección y un puerto al socket
  - listen() → esperar conexiones de clientes
  - accept() → aceptar una conexión con un cliente
  - recv() / send() → enviar y recibir datos
  - close() → cierra el socket

La función socket()

- La función *socket()* retorna un descriptor asociado a una interfaz de comunicación.

Archivo cabecera	`#include <sys/types.h>` `#include <sys/socket.h>`		
Formato	`int socket(int dominio, int tipo, int protocolo);`		
Salida:	Exito	Fallo	valor en errno
	descriptor del socket	-1	si

dominio : Dominio donde se realiza la conexión. Para nuestro caso será AF_INET.
tipo : Podrá ser SOCK_STREAM o SOCK_DGRAM o SOCK_RAW.
protocolo: 0 (cero, selecciona el protocolo más apropiado).

La función socket()

- **Ejemplos:**

```
#include <sys/types.h>
#include <sys/socket.h>
....
int sockfd;
sockfd =socket ( AF_INET, SOCK_STREAM, 0 );
...
```

Si el parámetro es cero, se selecciona automática,mente el procolo más asdecuado

- También seleccionar el protocolo por su nombre utilizando la función *getprotobyname()*:

```
struct protent pp= getprotobyname("tcp");
s= socket(AF_INET, SOCK_STREAM, pp->p_proto);
```

- Para más información: *man 2 socket*

La función bind()

- La función *bind()* se utiliza para darle un nombre al socket, es decir, una dirección IP y número de puerto por donde escuchará.

Archivo cabecera	`#include <sys/types.h>` `#include <sys/socket.h>`		
Formato	`int bind(int sockfd, struct sockaddr *my_addr, int addrlen)`		
Salida:	Exito	Fallo	valor en errno
	0	-1	si

sockfd : Es el descriptor de socket devuelto por la función socket().

my_addr : Es un puntero a una estuctura de tipo sockaddr que contiene la IP del host local y el número de puerto para a asignar al socket.

addrlen : Debe ser establecido al tamaño de la estuctura sockaddr. sizeof(struct sockaddr).

La estructura sockaddr

- Almacenan el nombre del socket.

```
struct sockaddr {
   unsigned short sa_family; /* AF_* */
   char sa_data[14]; /* Dirección de protocolo*/
};
```

```
struct sockaddr_in {
   short int sin_family; /* AF_INET */
   unsigned short sin_port; /* Número de puerto */
   struct in_addr sin_addr; /* Dirección IP */
  unsigned char sin_zero[8]; /* Relleno con 0 */
};

struct in_addr {
   unsigned long s_addr; /* 4 bytes */
};
```

Otras funciones de conversión

- **inet_addr()**

  Convierte dirección IP en un unsigned long (en network byte order). Retorna -1 si hubo error.

  Ejemplo:

```
struct sockaddr_in ina;
...
ina.sin_addr.s_addr= inet_addr("192.168.1.1");
```

- **inet_ntoa()**

  Convierte un unsigned long (en network byte order), a un string que representa una dirección IP.

  Ejemplo:

```
printf("%s", inet_ntoa(ina.sin_addr));
```

Asignar valores a sockaddr_in

- Considerando la estructura y funciones de conversión anteriores, veamos el siguiente ejemplo:

```
.....
struct sockaddr_in my_addr;
......
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(3490); // Número de puerto por donde escuchará
                                // el servidor.
my_addr.sin_addr.s_addr = inet_addr("132.241.5.10"); // IP de la interface
                                // por donde escuchará el servidor.
bzero( &(my_addr.sin_zero), 8); // Relleno con ceros.
```

- También podemos obtener automáticamente un puerto disponible.

```
my_addr.sin_port = 0;
```

- Podemos automatizar la asignación de la IP, si ponemos el valor **INADDR_ANY** a s_addr, el sistema le asignará la dirección IP local.

```
my_addr.sin_addr.s_addr = htonl (INADDR_ANY);
```

La función listen()

- La función *listen()* habilita el socket para que pueda recibir conexiones. Solo se aplica a sockets tipo SOCK_STREAM.

Archivo cabecera	`#include <sys/socket.h>`		
Formato	`int listen( int sockfd, int backlog)`		
Salida:	Exito	Fallo	valor en errno
	0	-1	si

sockfd : Es el descriptor de socket devuelto por la función socket() que será utilizado para recibir conexiones.

backlog : Es el número máximo de conexiones en la cola de entrada de conexiones. Las conexiones entrantes quedan en estado de espera en esta cola hasta que se aceptan (accept ()).

La función accept()

- La llamada a *accept()* no retornará hasta que se produce una conexión o es interrumpida por una señal.

Archivo cabecera	`#include <sys/types.h>` `#include <sys/socket.h>`		
Formato	`int accept( int sockfd, void *addr, int *addrlen)`		
Salida:	Exito	Fallo	valor en errno
	descriptor del socket para comunicarse con el cliente	-1	si

sockfd : Es el descriptor de socket habilitado para recibir conexiones.

addr : Puntero a una estructura sockadd_in. Aquí se almacenará información de la conexión entrante. Se utiliza para determinar que host está llamando y desde qué número de puerto.

addrlen : Debe ser establecido al tamaño de la estuctura sockaddr. *sizeof(struct sockaddr)*.

La función accept()

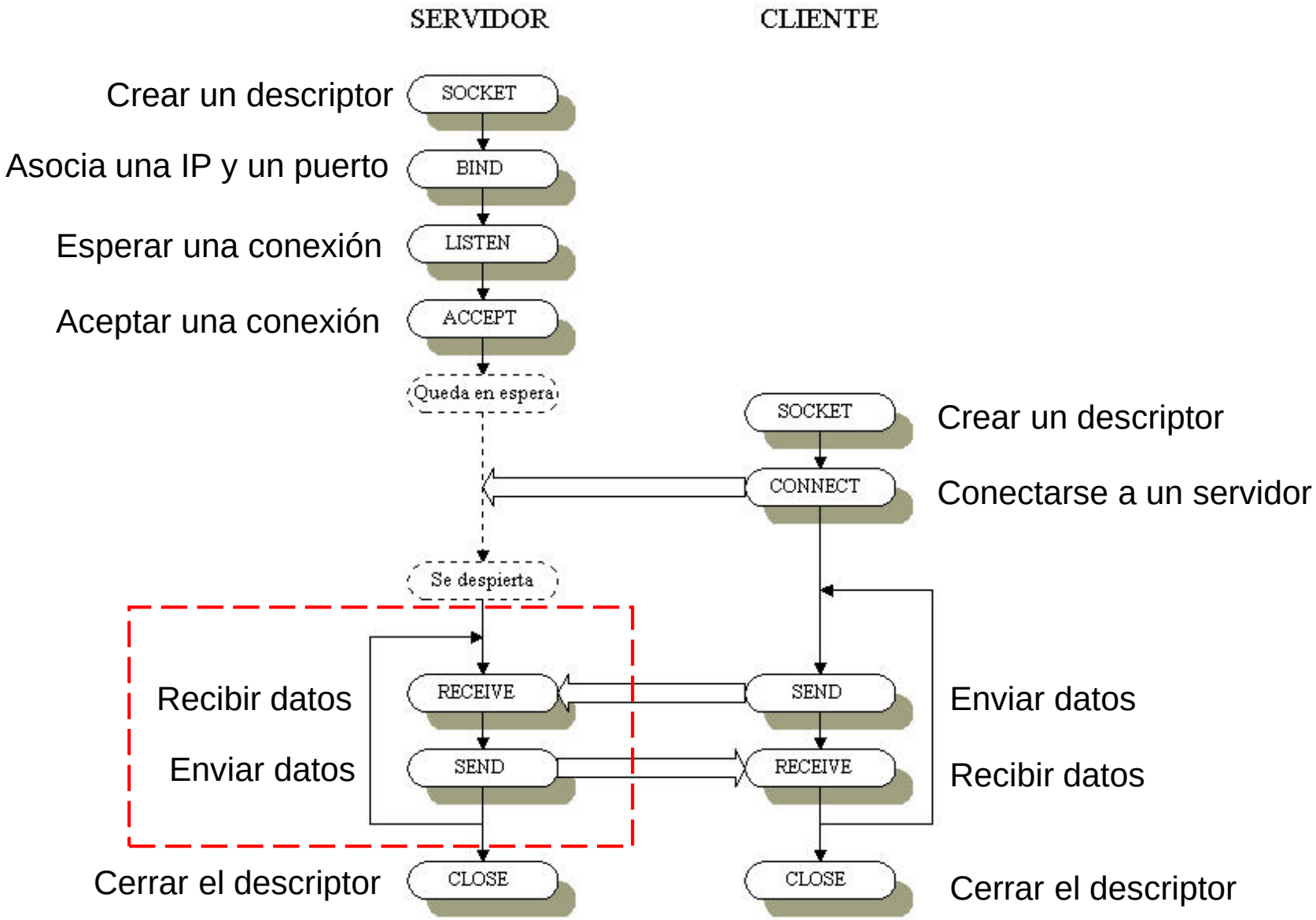
Ejemplo:

```
...
int sockfd, new_sockfd;
struct sockaddr_in my_addr;
struct sockaddr_in remote_addr;
int addrlen;
...
// Creo el socket.
sockfd = socket(AF_INET, SOCK_STREAM, 0 );
...
// Se le asigna un número de puerto al socket por donde el servidor escuchará.
// Antes de llamar a bind() se debe asignar valores a my_addr.
bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr) );

// Se habilita el socket para poder recibir conexiones.
listen (sockfd, 5);

addrlen = sizeof (struct sockaddr );
...
// Se llama a accept() y el servidor queda en espera de conexiones.
new_sockfd = accept(sockfd, &remote_addr, &addrlen);
...
```

Ejemplo cliente-servidor TCP simple (1 sólo cliente)



La función send()

- *send()* enviará la máxima cantidad de datos que pueda manejar y retorna la cantidad de datos enviados, es responsabilidad del programador comparar la cantidad de datos enviados con *len* y si no se enviaron todos los datos, enviarlos en la próxima llamada a send().

Archivo cabecera	`#include <sys/types.h>` `#include <sys/socket.h>`		
Formato	`int send(int sockfd, const void *msg, int len, int flags)`		
Salida:	Exito	Fallo	valor en errno
	0	-1	si

sockfd : Descriptor socket por donde se enviarán los datos.
msg : Puntero a los datos a ser enviados.
len : Longitud de los datos en bytes.
flags : Conjunto de banderas para habilitar la manipulación especial de mensajes.

La función recv()

- Si no hay datos a recibir en el socket, la llamada a recv() no retorna (se bloquea) hasta que llegan datos. Retorna el número de bytes recibidos.

Archivo cabecera	`#include <sys/types.h>` `#include <sys/socket.h>`		
Formato	`int recv (int sockfd, void *buf, int len, unsigned int flags)`		
Salida:	Exito	Fallo	valor en errno
	0	-1	si

sockfd : Descriptor socket por donde se recibirán los datos.
buf : Puntero a un buffer donde se almacenarán los datos recibidos.
len : Longitud del buffer buf.
flags : Conjunto de banderas

El parámetro flags de send() y recv()

- El parámetro flag puede tomar los siguientes valores:
  - **MSG_OOB**: Enviar el mensaje urgente.
  - **MSG_DONTROUTE**: Enviar el mensaje pasando por alto todos los routers. Si no tiene éxito la red devuelve un error.
  - **MSG_DONTWAIT**: No permitir bloqueo, sólo afecta a esta llamada no a todas las que se hagan con el mismo descriptor. Si la llamada se iba a bloquear retorna el valor EWOULDBLOCK en errno.
  - **MSG_NOSIGNAL**: Si el otro equipo (peer) corta la conexión, no emite una señal SIGPIPE local.

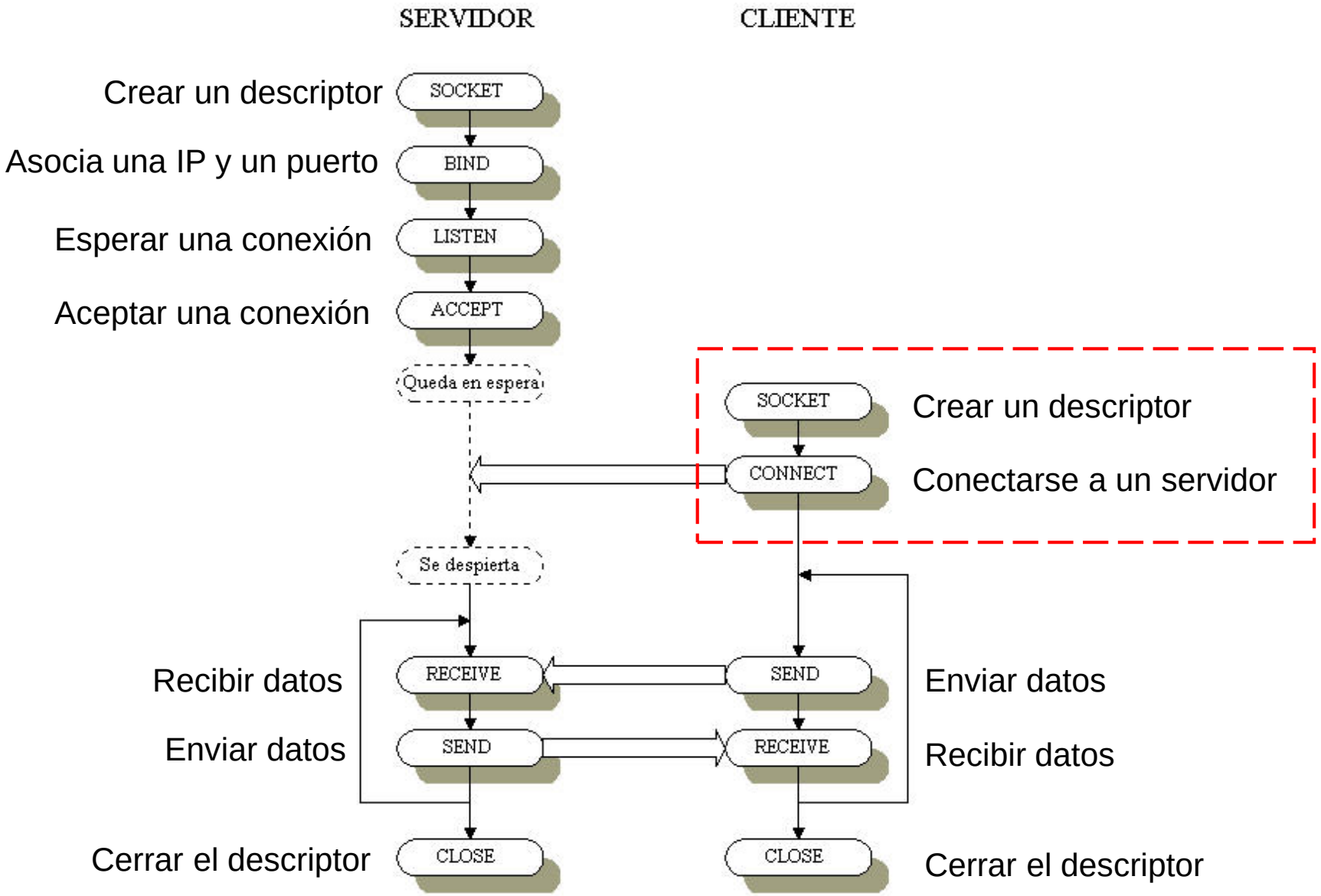
Las funciones close() y shutdown()

- Finalizan la conexión del descriptor de socket.
- ***close (sockfd)*** Después de utilizar close, el socket queda deshabilitado para realizar lecturas o escrituras.
- ***shutdown (sockfd, int how)*** Permite deshabilitar la comunicación en una determinada dirección o en ambas direcciones.
  - how : Puede tomar los siguientes valores :

    0 : Se deshabilita la recepción.
    1 : se deshabilita el envío.
    2 : se deshabilitan la recepción y el envío, igual que en close().

Crear un cliente Socket

Ejemplo cliente-servidor TCP simple (1 sólo cliente)



Las funciones connect()

- Inicia la conexión con el servidor remoto, lo utiliza el cliente para conectarse.

Archivo cabecera	`#include <sys/types.h>` `#include <sys/socket.h>`		
Formato	`int connect(int sockfd, struct sockaddr *serv_addr, int addrlen)`		
Salida:	Exito	Fallo	valor en errno
	0	-1	si

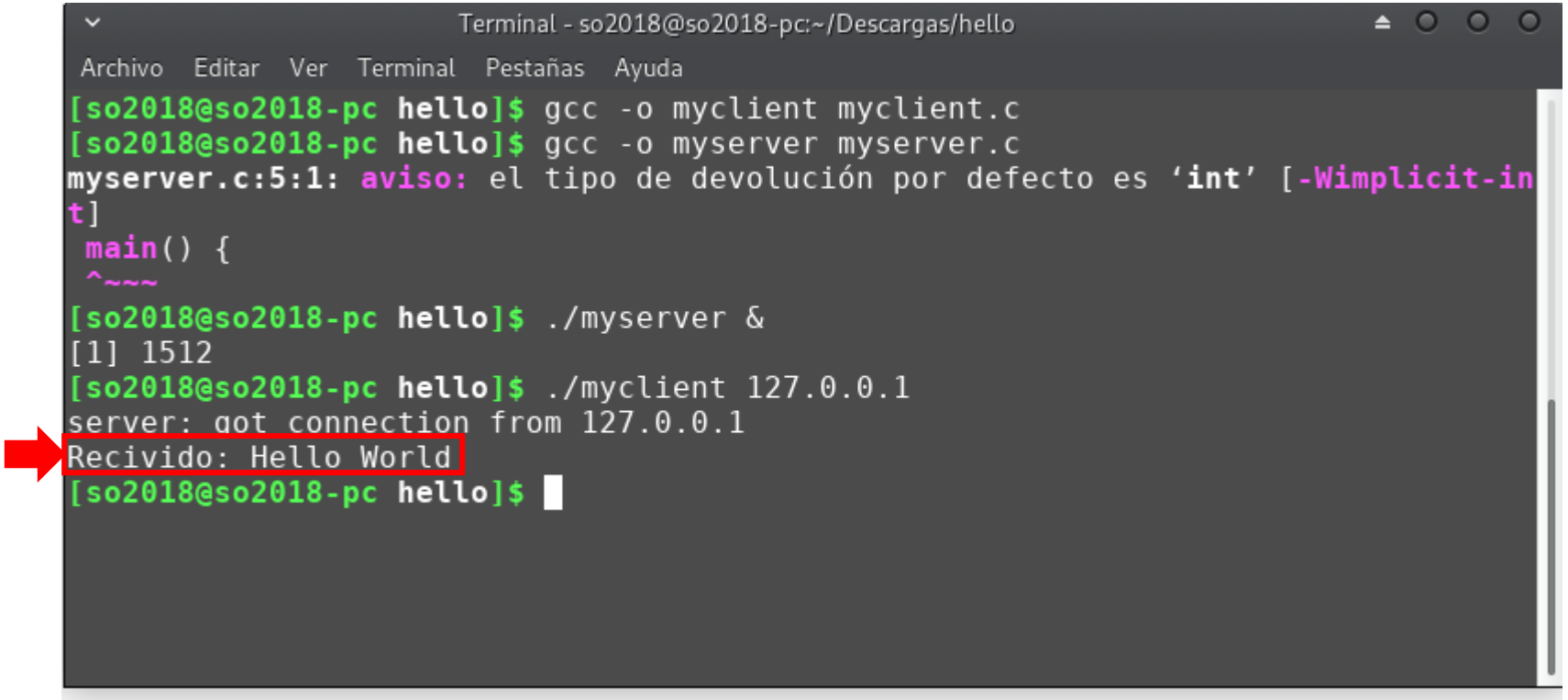
sockfd : Es el descriptor de socket devuelto por la función socket().

serv_addr : Es una estructura sockaddr que contiene la dirección IP y número de puerto destino.

addrlen : Debe ser inicializado al tamaño de struct sockaddr (sizeof (struct sockaddr)).

Ejemplo servidor stream

- Veremos un servidor stream simple, lo único que hace es enviar la frase "Hello World" hacia el cliente.



Ejemplo cliente-servidor TCP concurrente

(varios clientes simultáneamente)