Sistemas Operativos I

IPC: Sockets Continuación

La clase anterior vimos que para crear un socket necesitamos definir primero

+ el dominio: que tipo de conexión física vamos a utilizar (Local o Red)
+ El tipo de conexión orientada a datagramas o a stream de datos
+ Un protocolo, que depende de los dos anteriores y que en general no vamos a especificarlo

Y es nos generaba un socket aunque éste todavía no esta **conectado**.
Para poder transmitir mensajes por el socket deberemos tener un socket **conectado**

Asignar nombres a sockets

```
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

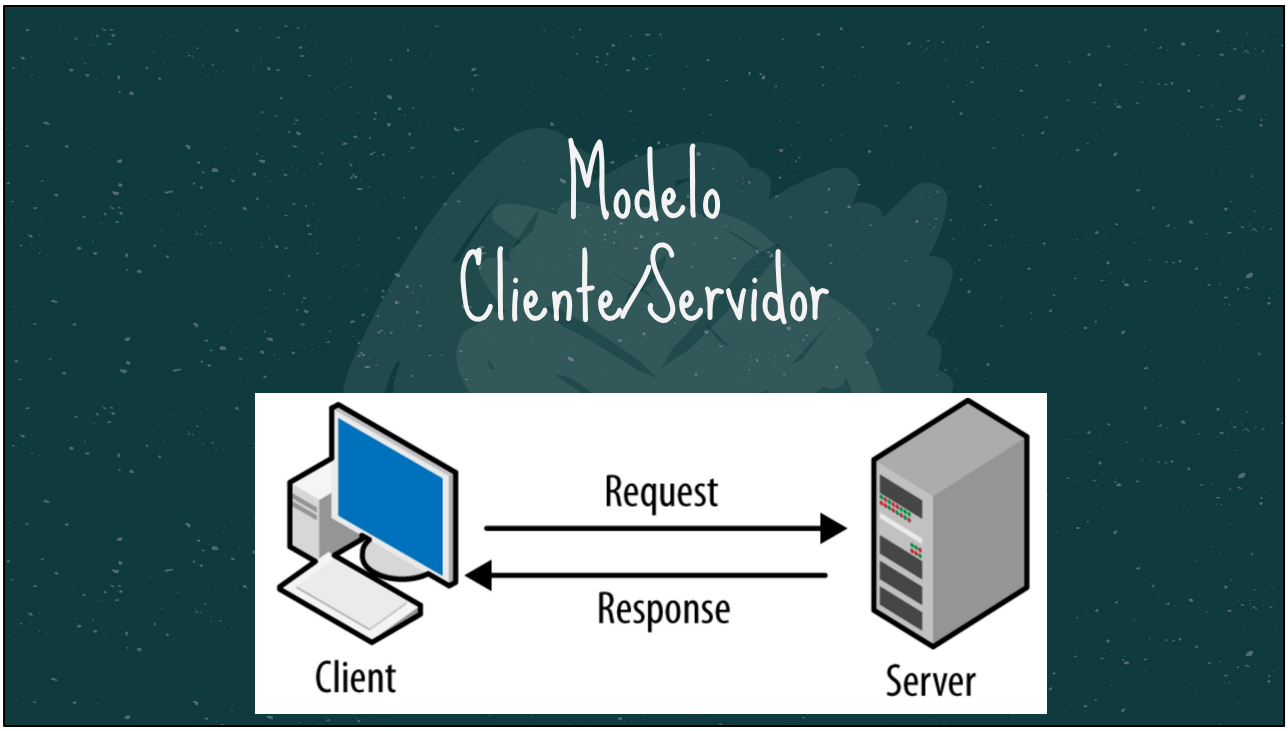
Para asignarle nombres a los sockets utilizamos entonces la función de `bind`, donde se les asigna un nombre.
Esto es para indicar a qué socket queremos enviar mensajes y para esto le tenemos que asignar un nombre.
Es lo que hace por nosotros la función `socketpair` donde en vez de asignarles un nombres le asigna file descriptors, y crea una especia de sockets anonimos.

Conectar Sockets: Direcciones Locales

```
struct sockaddr_un {
    sa_family_t sun_family;               /* AF_UNIX */
    char        sun_path[108];            /* Pathname */
};
```

Sin ninguna sorpresa, las direcciones locales son en realidad el nombre del archivo que se va a utilizar como dispositivo de la conexión.

Introduciremos una función que nos permitirá crear sockets con nombre de manera sencilla llamada `socketes_nombrados`.



La idea consiste en repensar la forma en que programamos.
Vamos a pensar en programar el software asumiendo que hay un proceso ofreciendo un servicio, que vamos a llamar servidor y un proceso que accede a dicho servicio que llamaremos cliente.
Tiene varias ventajas como ser que varios clientes pueden pedir por diferentes servicios del servidor, y la lógica del programa esta dividida en dos, aunque en general el servidor es quien mantiene la mayor lógica del programa mientras que los clientes presentan una interfaz a dicho servicio.

Para el ejemplo del Servidor de Echo vamos a tener 2 procesos ejecutandose:

+ Cliente enviará un mensaje al Servidor, y luego esperará la respuesta del mismo que mostrará en pantalla
+ El Servidor está a la espera de que *algún* cliente le envíe un mensaje, y se lo responderá.

Para eso necesitamos dos sockets, uno para el cliente y uno para el servidor.

SEND y RECV

```c
#include <sys/socket.h>

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
```

```c
#include <sys/socket.h>

ssize_t recv(int sockfd, void *buf, size_t len, int flags);
ssize_t recvfrom(int sockfd, void *restrict buf, size_t len, int flags,
                 struct sockaddr *restrict src_addr,
                 socklen_t *restrict addrlen);
```

Dependiendo del dominio y tipo de socket las funciones tiene un comportamiento un tanto diferente.
Tanto send/recv trabajan sobre sockets conectados.

Ahora veamos cómo montar un simple servidor utilizando sockets orientados a la conexión.
Y para esto vamos a tener que implementar la infraestructura de hacer una conexión cuando antes simplemente intercambiamos mensajes en diferentes procesos.
Ahora el servidor en vez de estar a la espera de mensajes va a tener que estar a la espera de procesos que intenten conectarse, y aceptar dichas conexiones.

Entonces del lado del servidor introduciremos dos funciones: listen(2), accept(2):

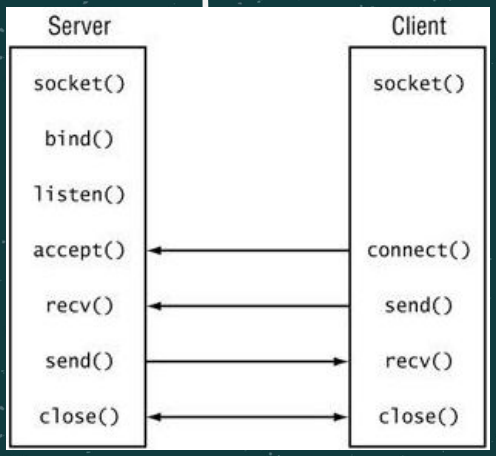
+ La función bloquea el servidor a la espera de clientes que quieran iniciar una conexión
+ La función accept acepta y establece dichas conexiones.

Mientras que del lado del cliente introducimos la función `connect` para conectar sockets.

Recuerden que las los sockets orientados a la conexión establece una conexión antes de empezar a enviar mensajes. Es bastante similar a `bind` pero en este caso espera que se pueda establecer una conexión entre el socket y lo que sea que esté en la dirección que apunta `addr`.

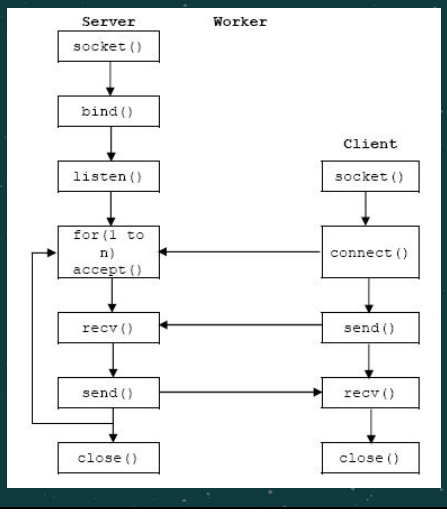
Vamos a hacer lo mismo que antes pero esta vez utilizaremos una conexión!

Conexiones de tipo STREAM



En general el procedimiento será siempre el mismo, desde el lado del servidor se creará un socket, se le asignará una dirección, y se pondrá a la espera de conexiones.
Las conexiones las aceptará con `accept` y comenzará la comunicación con el cliente.
Desde el lado del cliente es simplemente intentar establecer la conexión.

Servidores



Aunque en realidad los servidores tienen un patrón más similar a este. Donde en realidad se quedan esperando a que diferentes clientes aparezcan.