

Taller de IPC

Sistemas Operativos

1er Cuatrimestre de 2018



Menú del Día

- ▶ Conceptos básicos de IPC.
- ▶ Comunicación vía pipes.
- ▶ Comunicación vía sockets.
- ▶ ¡Taller!

Comunicación vía pipes

Recordemos, **pipes**, escritos como “|”.

Por ejemplo, qué sucede si escribimos en **bash**:

```
echo "sistemas es lo más" | wc -c
```

1. Se llama a `echo`, un programa que escribe su parámetro por **stdout**.
2. Se llama a `wc -c`, un programa que cuenta cuántos caracteres entran por **stdin**.
3. Se conecta el **stdout** de `echo` con el **stdin** de `wc -c`.

¿Cuál es el resultado? **20.**

Comunicación vía pipes

Un pipe es un archivo **temporal** y **anónimo**¹ que se aloja en memoria y actúa como un **buffer** para leer y escribir de manera **secuencial**.

Se crea mediante la syscall:

```
int pipe(int pipefd[2]);
```

Luego de ejecutar pipe, tenemos:

- ▶ En pipefd[0] un **file descriptor** que apunta al extremo del pipe en el cual se **lee**.
- ▶ En pipefd[1] otro **file descriptor** que apunta al extremo del pipe en el cual se **escribe**.

¹Si bien el uso más frecuente son los pipes anónimos, ver mkfifo para pipes con nombre

Comunicación vía pipes

Para leer de un **file descriptor** usamos:

```
ssize_t read(int fd, void *buf, size_t count);
```

- ▶ `fd` **file descriptor**.
- ▶ `buf` puntero al **buffer** donde almacenar lo leído.
- ▶ `count` cantidad máxima de bytes a leer.

Se retorna la cantidad de bytes leídos, `-1` en caso de error.

`read` es bloqueante, aunque pueden configurarse ciertos **flags** para que no lo sea.

Más allá de pipes

Un proceso no puede acceder a un pipe anónimo preexistente. Es decir, sólo puede utilizar los pipe que haya creado mediante la llamada a la syscall pipe.

¿Cómo lo utilizo entonces para comunicar dos procesos?

Los **pipes** requieren que los procesos que se comunican tengan un ancestro en común que cree el pipe.

¿Qué pasa si los procesos no se conocen entre sí?

Sockets

- ▶ Un **socket** es el extremo de una conexión y tiene asociado un nombre.
- ▶ Dos procesos que se quieren comunicar entre sí se ponen de acuerdo en dicho nombre.

Comunicación vía sockets

- ▶ Los **sockets** pueden trabajar en distintos dominios.
- ▶ En el dominio UNIX las direcciones son un nombre de archivo y sirven para comunicar dos procesos en una misma máquina.
- ▶ En el dominio IP las direcciones son una dirección IP y un puerto, y sirven para comunicar dos procesos en una misma red.

UDP: Envío de paquetes con sockets

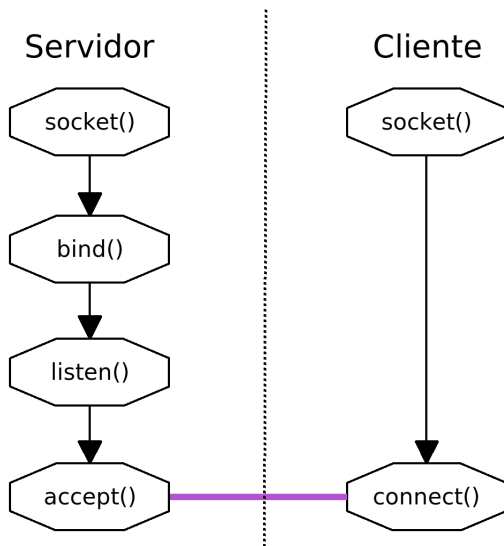
Servidor (recibe mensajes)

1. `int socket(int domain, int type, int protocol);`
Crear un nuevo **socket**.
2. `int bind(int fd, struct sockaddr* a, socklen_t len);`
Asigna una dirección (nombre o IP y puerto) al **socket**.
3.
`size_t recv(int s, void *buf, size_t l, int flags);`
Lee un paquete

Cliente (envía mensajes)

1. `int socket(int domain, int type, int protocol);`
Crear un nuevo **socket**.
2.
`size_t send(int s, void *buf, size_t l, int flag);`
Envía un paquete

TCP: Estableciendo comunicación con sockets



Estableciendo sockets: Servidor

1. `int socket(int domain, int type, int protocol);`
Crear un nuevo **socket**.
2. `int bind(int fd, sockaddr* a, socklen_t len);`
Asigna una dirección (nombre o IP y puerto) al **socket**.
3. `int listen(int fd, int backlog);`
Se asigna una cola y se utiliza el **socket** para recibir conexiones entrantes.
4. `int accept(int fd, sockaddr* a, socklen_t* len);`
Espera la próxima conexión de un cliente en el **socket** (y crea un nuevo socket para atenderla).

Estableciendo sockets: Cliente

1. `int socket(int domain, int type, int protocol);`
Crear un nuevo **socket**.
2. `int connect(int fd, sockaddr* a, socklen_t* len);`
Conectarse a un **socket** remoto que debe estar escuchando.

Comunicación vía sockets: Mensajes

Una vez que un cliente solicita conexión y esta es aceptada por el servidor puede comenzar el intercambio de mensajes con:

```
ssize_t send(int s, void *buf, size_t len, int flags);
```

```
ssize_t recv(int s, void *buf, size_t len, int flags);
```

¿Preguntas?

¿Cómo hacemos si un servidor
se quiere comunicar con dos
clientes a la vez?

Comunicación vía sockets: ¿Bloqueante?

Si usamos llamadas bloqueantes podemos esperar indefinidamente por el primer cliente mientras el segundo tiene mucho que enviarnos y nunca se lo pedimos.

Soluciones (ver ejemplos avanzados):

1. Llamadas no bloqueantes y espera activa.
2. Usar las llamadas al sistema `pselect(2)` o `poll(2)`.
3. Utilizar un proceso (o thread) para atender cada cliente.
(*No la veremos en esta clase.*)

¿Preguntas?