Taller de IPC

Sistemas Operativos

1er Cuatrimestre - 2019

1. Ejercicios

1.1. Mini Shell

Se pide implementar parte de la funcionalidad de un `shell` minimal. El mismo solo soporta comandos de dos formas: **a)** el nombre de programa (más argumentos), por ejemplo, `ls -al`; o **b)** más de un programa comunicados por `|`, por ejemplo, `ls -al | wc | awk '{ print $2 }'`.

- Completar el esqueleto provisto por la cátedra para que el comando `ls -al | wc | awk '{ print $2 }'` se comporte de la misma manera que lo haría en cualquier otro `shell`. No se puede utilizar la función `system` para dicha tarea.
- ¿Por qué es importante cerrar los extremos de los `pipes` que no se utilizan?

1.2. Mini Servidor Telnet

Se tiene el código final de un cliente (MINI-TELNET-CLIENT) que, al ejecutarse, toma cómo parámetro la IP de un servidor y establece una conexión con este a través del protocolo TCP en el puerto 678.

Luego, el cliente interpreta cada línea de entrada estándard como un comando que enviará al servidor, hasta leer el comando "chau" y terminar.

1. Completar el código del servidor MINI-TELNET-SERVER para que acepte la conexión TCP del cliente, lea el comando enviado por este, y lo ejecute.
2. (OPCIONAL) Modificar el cliente para que muestre las salidas de los comandos enviados y ejecutados en el servidor.

Para comenzar, se deberá compilar ambos archivos (MINI-TELNET-SERVER.C y MINI-TELNET-CLIENT.C). Luego, en terminales separadas, ejecutar MINI-TELNET-SERVER y MINI-TELNET-CLIENT. Recomendamos fuertemente leer el apunte y buscar el comportamiento de las funciones en el manual de linux.

Apunte para el ejercicio de Telnet

Sockets

Un *file descriptor*, en particular un *fd* de *socket*, tiene tipo `int`. Recordar de la clase que se puede crear un *socket* usando:

```
int socket(int domain, int type, int protocol);
```

Para trabajar con *sockets* de internet usaremos el `domain AF_INET`. Para TCP, usaremos el `type SOCK_STREAM`. Recordar que en `protocol` en general se utiliza un `0` (ver `/etc/protocols`).

Un socket se cierra con `close(int socket)`.

Direcciones de internet

Para representar una dirección de internet se usa la estructura presentada a continuación:

```
struct sockaddr_in {
    unsigned short sin_family;   /* dominio, usamos AF_INET */
    in_port_t      sin_port;     /* número de puerto */
    struct in_addr sin_addr;     /* dirección IP */
    unsigned char  sin_zero[8];  /* padding (no se usa) */
};
```

Donde la estructura que contiene la dirección IP es la siguiente:

```
struct in_addr {
    in_addr_t s_addr; /* Esto es un número de 32 bits */
};
```

Network byte order

Las estructuras mencionadas arriba necesitan tener el **puerto** y la **dirección IP** almacenadas en un formato conocido como *Network byte order*[1]. Para ello contamos con funciones de conversión:

- `uint16_t htons(uint16_t hostshort)` convierte un *uint16_t* del *host* (máquina local) en un *uint16_t* de la red.
- `uint32_t htonl(uint32_t hostlong)` análoga pero convierte *uint32_t*.

Resolver direcciones IP

Para convertir una cadena de caracteres que contiene una dirección IP (por ejemplo: "127.0.0.1") en una estructura `in_addr` usamos:

```
int inet_aton(const char *cp, struct in_addr *inp);
```

Esta función ya nos deja la dirección IP en formato *Network byte order*.

¡Ojo! Esta función devuelve `0` en caso de error (sí, es al revés que la mayoría de las funciones de sistema).

Conexion TCP

Una conexión TCP desde el lado del servidor requiere de tres pasos: `bind`, `listen` y `accept`. **Identifique a través del manual qué realiza cada uno de estos pasos.**

```
int bind(int s, sockaddr* a, socklen_t len);

int listen(int s, int backlog);

int accept(int s, sockaddr* a, socklen_t* len);
```

[1] Se trata de un estándar *big-endian*

Enviar y recibir paquetes

Una vez que un cliente solicita conexión y esta es aceptada por el servidor pueden 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);
```

Otras funciones útiles

- `ssize_t getline(char **lineptr, size_t *n, FILE *stream);`
  Leer una línea (hasta "`\n`") desde `stream`.
- `int strncmp(const char *s1, const char *s2, size_t n);`
  Comparar dos cadenas `s1` y `s2` de longitud a lo sumo `n`.
- `int system(const char *command)`
  Ejecuta el comando `command` en un shell.

Para el ejercicio opcional

Si desea enviar el resultado del comando de vuelta al cliente, y al mismo tiempo mostrarlo en pantalla por stdin, será necesario utilizar la función `dup2(2)`.