UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
Año 2015 - 1er Cuatrimestre

TALLER DE PROGRAMACIÓN I (75.42)

Resumen de Taller de programación I

AUTOR:
Lafroce Matías - 91378
⟨mlafroce@gmail.com⟩

Índice

1. Sockets

Un socket es un flujo de datos que se utilizar para comunicar procesos entre si. Los sockets tienen un dominio sobre el cual se comunican, como por ejemplo, internet IPv4 e IPv6, o Unix para comunicar procesos dentro del mismo sistema. Además, poseen un tipo de conexión, por ejemplo, si es punto a punto o no, si los paquetes son de longitud fija o variable, etc, y poseen un protocolo sobre el que se realiza la transmisión de datos.

A continuación se describen las primitivas más utilizadas para el manejo de sockets POSIX en C. Nótese que los sockets se manejan igual que los archivos, y que como tales, las acciones sobre ellos se realizan mediante su *file descriptor*.

1.1. int socket(int domain, int type, int protocol)

Crea un nuevo socket y devuelve su número de *socket descriptor* (o -1 si hay un error).

Parámetros

- **domain**: Define si la familia de protocolos de la conexión. Algunos valores usados son: PF_LOCAL (comunicación local), PF_INET (IPv4), PF_INET6 (IPv6).
- **type**: Define el tipo de conexión. Algunos de los valores valores más usados son SOCK_STREAM y SOCK_DGRAM para protocolos TCP y UDP respectivamente.
- **protocol**: Define el protocolo a utilizar, se lo puede dejar en 0 para que se elija el apropiado según el tipo de conexión.

1.2. int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)

Asocia al socket a una dirección. Devuelve 0 en éxito o -1 en caso de error.

Parámetros

- **sockfd**: Número del socket descriptor al que se le quiere asociar la dirección.
- **addr**: Dirección a la que se quiere asociar el socket.

  La estructura que se utiliza generalmente es la siguiente:

```
1  struct sockaddr_in {
2    sa_family_t sin_family; /* address family: AF_INET */
3    in_port_t sin_port;     /* puerto en formato de red (pasarle htons(port)) */
4    in_addr sin_addr;       /*IP a la que se quiere asociar */
5  };
```

- **addrlen**: Tamaño de la estructura: sizeof(struct sockaddr_in); Si el socket se utilizará como cliente, no es necesario bindearlo antes de hacer un connect.

1.3. int listen(int sockfd, int backlog)

Marca el socket como pasivo, es decir, que escuche conexiones entrantes. El socket debe ser del tipo SOCK_STREAM o SOCK_SEQPACKET. Devuelve 0 en éxito o -1 en caso de error.

Parámetros

- **sockfd**: Número del socket descriptor a pasivar.
- **backlog**: Cantidad máxima de conexiones a encolar

1.4. int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)

Genera un socket a partir de la primera de las conexiones encoladas en el socket correspondiente a sockfd. Devuelve el número del file descriptor de la nueva conexión o -1 en caso de error.

Parámetros

- **sockfd**: Socket escuchador. Tiene que estar previamente pasivado con *listen* (Por lo que también es prerequisito haber llamado a *bind*)
- **addr**: Puntero a la estructura en la que se escribe la dirección del cliente que se conecta. Se le puede pasar 0 para ignorar esta información.
- **addrlen**: Tamaño de la estructura sockaddr. Si addr es 0, se recomienda que addrlen sea 0 también.

1.5. int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)

Conecta el socket a la dirección pasada por addr. Devuelve 0 en éxito o -1 en caso de error.

Parámetros

- **sockfd**: Número del socket descriptor que se quiere conectar como “cliente”.
- **addr**: Dirección al que se quiere conectar el socket.
- **addrlen**: Tamaño de la estructura sockaddr.

1.6. int shutdown(int sockfd, int how)

Desactiva toda o parte de la conexión.

Parámetros

- **sockfd**: Número del socket descriptor a desactivar.
- **how**: Especifica si se quiere desactivar la escritura, lectura o ambas

1.7. int close(int sockfd)

Cierra el socket, y libera los recursos correspondientes.

Parámetros

- **sockfd**: Número del socket descriptor a cerrar.

2. Protocolos TCP/UDP

A continuación se describen los protocolos más utilizados para comunicaciones sobre redes: *TCP* y *UDP*, y algunas de sus primitivas.

El protocolo *TCP* se utiliza en conexiones de streaming punto a punto, donde se utiliza un socket para cada punta de cada conexión. Esto implica que si un servidor está conectado a varios clientes, debe tener un socket distinto para cada uno de ellos. Este protocolo además tiene como característico que los paquetes siempre llegan a destino y en el orden en que fueron mandados.

El protocolo *UDP*, por otro lado, se utiliza en conexiones basadas en datagramas, donde las conexiones no son punto a puntos, sino que el servidor realiza un *broadcast* de un mensaje y los clientes lo reciben como pueden. Este protocolo no asegura que todos los paquetes lleguen del servidor al cliente, ni asegura que se mantenga el orden al recibirlos. Es un protocolo más liviano que TCP y se utiliza cuando lo importante no es la fidelidad de los datos, sino la rapida transmisión de los mismos.

Dado que el socket es un archivo con un file descriptor asociado, podemos realizar lectura y escritura como lo haríamos con cualquier otro archivo. Sin embargo, para aprovechar los protocolos ya existentes, se utilizarán las funciones *recv* y *send*. Cabe destacar que las funciones *read* y *write* del standard de Unix funcionan sobre el file descriptor del socket, pero estarían traspasando el protocolo de la conexión, ya que realizan escritura y lectura a bajo nivel.

2.1. ssize_t recv(int sockfd, void *buf, size_t len, int flags)

Lee una cadena de bytes del socket y devuelve la cantidad de bytes que leyó, o -1 en caso de error.

Parámetros

- **sockfd**: Número del socket descriptor a leer.
- **buf**: Buffer sobre el que se va a escribir lo recibido.
- **len**: La cantidad de bytes que se espera leer (no necesariamente se llegan a leer todos).

2.2. ssize_t send(int sockfd, void *buf, size_t len, int flags)

Escribe una cadena de bytes del socket y devuelve la cantidad de bytes que envió, o -1 en caso de error.

Parámetros

- **sockfd**: Número del socket descriptor a escribir.
- **buf**: Buffer sobre el que se va a leer el mensaje a enviar.
- **len**: La cantidad de bytes que se espera escribir (no necesariamente se llegan a enviar todos).
- **flags**: Es recomendable usar MSG_NOSIGNAL para evitar que se emitan señales SIGPIPE cuando la conexión está rota.

3. Ejemplo

3.1. Cliente

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cstring> //Necesario para el memset
#include <cstdio>

#define BACKLOG 20
#define MSG_SIZE 30

int main(int, char**){
    printf("Iniciando el cliente en la direccion 127.0.0.1:8080\n");
    int socketFd = socket(PF_INET, SOCK_STREAM, 0); //Creo el socket

    char serverAddress[] = "127.0.0.1";

    struct sockaddr_in address; //Armo los datos para conectarse
    address.sin_family = AF_INET;
    address.sin_port = htons(8080); //Seteo el puerto, en formato de red
    address.sin_addr.s_addr = inet_addr(serverAddress);
    memset(address.sin_zero, 0, sizeof(address.sin_zero));

    int connected = connect(socketFd, (struct sockaddr *) &address,
        sizeof(struct sockaddr_in)); //Me conecto a la direccion.
    if (connected != 0){
        printf("Falla al conectar\n");
        return connected;
    }

    char message[MSG_SIZE];
    int bytesRecv = 0;

    printf ("Recibiendo el mensaje...\n");
    //Le envio 30 bytes al cliente (un numero arbitrario
    while (bytesRecv < MSG_SIZE && bytesRecv != -1){
        // Agrego offsets si es que no se envia todo el mensaje
        bytesRecv += recv(socketFd, message + bytesRecv,
                  MSG_SIZE - bytesRecv, 0);
        printf("Recibido %d bytes\n", bytesRecv);
    }
    message[29] = 0; //Cierro string

    printf ("Recibo el mensaje %s\n", message);

    shutdown(socketFd, 0); //Dejo de transmitir datos

    close(socketFd); //Cierro file descriptor

    printf("Adios, vuelvas prontos\n");

    return 0;
}
```