

PRÁCTICA 4

“SOCKETS EN C”

1. OBJETIVO

Entender los conceptos básicos de los sockets tanto de manera local como a través de una red.

2. INTRODUCCIÓN

Un socket es una comunicación bidireccional que permite comunicación entre procesos alojados en la misma computadora o en otras. Para crear un socket se deben especificar tres parámetros: comunicación, espacio de nombres y protocolo.

La comunicación se encarga de controlar cómo es transmitida la información y especifica el número asociados para la comunicación. La información mandada es dividida en paquetes, la comunicación se encarga del manejo de estos paquetes y cómo debe ser enviada del emisor al receptor. La transmisión de los paquetes se puede mandar por:

- ✓ Conexión: Garantiza que se van a entregar todos los paquetes en el orden en el que fueron enviados. Si se perdieron paquetes o se re-ordenaron, el receptor solicita su retransmisión al emisor.
- ✓ Datagrama: No garantiza que los paquetes lleguen y mucho menos que lleguen en el mismo orden en el que fueron mandados.

El espacio de nombres (*namespace*) especifica la dirección del socket. La dirección del socket a dónde se va a establecer la conexión.

El protocolo sirve para especificar cómo deben de ser transmitidos los datos; por ejemplo el protocolo TCP/IP.

API de sockets de Berkeley

Debido a la variedad de protocolos, se popularizó el API de sockets de Berkeley como una forma de englobar la variedad de protocolos de red y proporcionar una interface de programación. Con los sockets de Berkeley se pueden crear sockets para trabajar en una sola máquina (POSIX local IPC) y comunicación con otras máquinas (TCP/IP).

La más fundamental estructura para la creación de sockets incluida en el API es `sockaddr`. Su sintaxis es la siguiente:

```
struct sockaddr {
    unsigned short int sa_family;
    char sa_data[14];
};
```

sa_family describe el tipo de dirección almacenada y *sa_data* contiene la dirección actual. Los integrantes de *sa_family* se describe a continuación:

Address family	Protocol family	Descripción
AF_UNIX	PF_UNIX	Sockets de UNIX
AF_INET	PF_INET	TCP/IP (versión 4)
AF_AX25	PF_AX25	Protocolo amateur de radio
AF_IPX	PF_IPX	Protocolo Novell IPX
AF_APPLETALK	PF_APPLETALK	Protocolo Apple Talk DDS

Para el uso de sockets en C, se necesita incluir la cabecera <sys/socket.h>. Debido a que Linux ve todo como archivos, las funciones de lectura y escritura son usadas también en sockets.

La referencia de las funciones se encuentra en:

<http://pubs.opengroup.org/onlinepubs/9699919799/>

Para crear un socket se usa la función `socket` cuya sintaxis se muestra a continuación.

```
int socket(int dominio, int tipo, int protocolo)
```

dominio es usado para especificar el protocolo de red a usar (AF_UNIX, AF_INET, etc.). *tipo* establece la categoría del protocolo ya sea streaming (SOCK_STREAM) o datagrama (SOCK_DGRAM). *protocolo* indica el protocolo por default a usar basado en el *dominio* y el *tipo* antes mencionados.

El servidor se encarga de crear el socket y crear una asociación entre el socket y una dirección que puede ser un archivo —localmente— o una dirección de internet.

Para asociar o ligar el socket a un archivo, se utiliza la función `bind()`:

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

Para que un socket tenga uso, se necesita un servidor que se encargue de escuchar peticiones de un cliente. El cliente puede pedir información del servidor, enviar datos o acceder a alguno de los servicios proporcionados por él.

Después de la asociación, el servidor espera una conexión. Para ello, el servidor debe invocar a la función `listen()`:

`int listen(int sockfd, int backlog);`

sockfd es el descriptor de archivo regresado por la llamada previa de `socket()`.

backlog define el máximo número de conexiones pendientes que son permitidas.

Posteriormente en el servidor, además del socket servidor, debe crearse un segundo socket que se inicializará cuando el cliente realice una conexión. Ese segundo socket en el servidor es el que se “entenderá” con el socket en el cliente.

Del lado del cliente se invoca a la función `connect()` la cual es similar a la función `bind()` del lado del servidor:

*`int connect(int sockfd, struct sockaddr *addr, int addrlen);`*

En el servidor, para aceptar la conexión del cliente, se invoca a la función `accept()`:

*`int accept(int sockfd, struct sockaddr *addr, int *addrlen);`*