Sockets

Creación del socket

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

Crea un punto de conexión para un tipo de comunicación configurada.

El **dominio** indica el ámbito de la comunicación a través de una familia de protocolos. Entre los mas comunes:

AF_UNIX, AF_LOCAL	Comunicación local (servidor y cliente en la misma máquina)	unix
AF_INET	Protocolo de internet IPv4	ip
AF_INET6	Protocolo de internet IPv6	ipv6

El **tipo** indica la semántica de la comunicación, la forma en que se dará la comunicación. Entre los mas comunes:

SOCK_STREAM	Comunicación TCP
SOCK_DGRAM	Comunicación UDP

El **protocolo** indica un número de protocolo en particular a utilizar con un tipo de socket en un dominio dado. Entre los mas comunes:

0, IPPROTO_UDP, o IPPROTO_TCP

Al indicar un valor 0 (cero) el sistema operativo selecciona el protocolo por defecto para el dominio y tipo de socket seleccionado, así que lo mas común es que el tercer parámetro de la función socket se configure con 0 (cero).

Valor de retorno: en caso de éxito retorna un descriptor para el nuevo socket creado. En caso de error -1.

```
//Crear el socket de escucha del servidor
if((server_socket=socket(AF_INET,SOCK_STREAM,0)) ==-1){
   printf("No se pudo crear el socket !!!\n");
   return 1;
}
```

Configurar opciones

int setsockopt(int socket, int nivel, int nombre,const void *valor, socklen_t tamaño)

Configura opciones sobre un socket.

socket: es el descriptor del socket creado con la función socket.
nivel: es el nivel de protocolo para el cual se corresponde la opción. Para setear la opción a nivel de socket se especifica SOL_SOCKET. Pero si la opción debe ser interpretada por ejemplo por TCP, se debe especificar IPPROTO_TCP.
nombre: es la opción que se va a configurar, por ejemplo SO_REUSEADDR habilita a que la dirección local se pueda reutilizar, esto es que una dirección IP y puerto puedan ser "vinculadas" por la función bind mas de una vez.
valor: es el valor que debe tomar la opción (0 es Inactiva – 1 es Activa). Salvo si se especifica la opción SO_LINGER, en cuyo caso se debe proveer un struct linger en vez de un entero.
tamaño: es el tamaño en bytes del valor pasado.

Valor de retorno: en caso de éxito retorna 0 (cero). En caso de error -1.

```
//Configurar opciones del socket
if (setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &habilitar, sizeof(int)) < 0){
   printf("No se pudo configurar opciones del socket !!!\n");
   return 1;
}
```

Configuración y estructuras

Tanto la configuración del cliente como del servidor se almacenan en estructuras. La estructura genérica o estándar del socket es del tipo **struct sockaddr**

```
struct sockaddr {
  uint8_t      sa_len;        /* Tamaño de la estructura */
  sa_family_t  sa_family;     /* AF_xxx */
  char         sa_data[14];   /* Protocolo */
};
```

Esta estructura no tiene un protocolo y formato de dirección especifico. Una vez especificado el protocolo, por ejemplo en nuestro caso utilizamos direcciones IpV4 con el protocolo TCP, se utiliza una estructura del tipo **struct sockaddr_in**

```
struct in_addr {
  in_addr_t   s_addr;           /* 32-bit dirección IPv4 */
};

struct sockaddr_in {
  uint8_t         sin_len;
  sa_family_t     sin_family;
  in_port_t       sin_port;     /* 16-bit TCP o UDP puerto */
  struct in_addr  sin_addr;
  char            sin_zero[8];  /* Sin uso */
};
```

El campo sin_zero solamente está para lograr llegar al tamaño de la estructura genérica de 16 bytes.

A continuación vemos la configuración del servidor:

```
//Configuración del servidor
bzero((char *) &sa, sizeof(struct sockaddr_in));
sa.sin_family     = AF_INET;
sa.sin_port       = htons(atoi(argv[1]));
sa.sin_addr.s_addr= INADDR_ANY;
```

La función **bzero** se utiliza para rellenar con ceros una cantidad de bytes a partir a de una dirección dada, en realidad ya está deprecada y debería utilizarse **memset**. En este caso se utiliza para limpiar la estructura del servidor antes de configurarla.

Se configura la familia de direcciones IpV4 (AF_INET), el puerto y se especifica por cuál dirección IP escuchar, pasándole INADDR_ANY logramos; admitir datos que provengan de cualquiera de las interfaces de red que tenga el servidor y aunque el servidor cambie su IP, seguirá funcionando sin cambios.

Vincular el socket con la configuración del servidor
int bind(int socket, const struct sockaddr *addr,socklen_t addrlen);

```
//Vincular socket con configuracion del servidor
bind(server_socket,(struct sockaddr *)&sa,sizeof(struct sockaddr_in));
```

socket: es el descriptor del socket creado con la función socket.
addr: estructura configurada anteriormente, nótese como se debe castear la estructura especifica de Ipv4 a la estructura genérica de socket.
addrlen: como último parámetro se pasa el tamaño de la estructura que contiene la configuración del servidor.

Valor de retorno: en caso de éxito retorna 0 (cero). En caso de error -1.

Poner al socket en "escucha"
int listen(int socket, int max);

socket: es el descriptor del socket creado con la función socket.
max: cantidad de peticiones a encolar como máximo.
Valor de retorno: en caso de éxito retorna 0 (cero). En caso de error -1.

A través de la función listen marcamos al socket como pasivo, esto es ponerlo a la escucha de peticiones. Las peticiones que lleguen y no puedan ser encoladas serán rechazadas.

```
//Marcar el socket como "Socket Pasivo" y Establecer la máxima
//cantidad de peticiones que pueden ser encoladas
listen(server_socket,MAX_QUEUE);
```

Escucha de solicitudes
int accept(int socket, struct sockaddr *addr, socklen_t *addrlen)
socket: es el descriptor del socket creado con la función socket.
addr: dirección de otra estructura del tipo sockaddr_in distinta de la estructura del servidor.
addrlen: un puntero a una variable socklen_t.
A través de la función **accept** bloqueamos (si el socket no fue marcado como no bloqueante) al servidor a la escucha de peticiones desde los clientes. El accept irá tomando las peticiones que se encuentren pendientes en la cola.

```
//El servidor ser bloquea a la espera de una peticion de conexión
//desde el cliente (connect)
client_socket=accept(server_socket,(struct sockaddr *) &ca, &cl);
```

Valor de retorno: en caso de éxito el descriptor para el socket aceptado. En caso de error -1.

Comunicación

La comunicación se realiza a través de las funciones send y recv.

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);

Cierre
Para el cierre de los respectivos sockets se utiliza la función close.

int close(int fd);

Connect
int connect(int socket, const struct sockaddr *addr, socklen_t addrlen);
socket: es el descriptor del socket creado con la función socket.
addr: dirección de la estructura del tipo sockaddr_in de la configuración del servidor en el cliente.
addrlen: el tamaño de la estructura sockaddr_in.

Valor de retorno: en caso de éxito 0 (cero). En caso de error -1.

Desde el cliente se llamará a la función connect para solicitar la conexión al servidor.

```
if(connect(x,(struct sockaddr *) &sa,sizeof(sa))==-1){
   printf("Solicitud rechazada !!!\n");
   return 1;
}
```

Ejecución del código de ejemplo:

Ejecutar make para compilar ambos programas.
En una terminal ejecutar el servidor con el puerto como parámetro.
En otra terminal ejecutar el cliente con IP y puerto como parámetros, en el ejemplo de la captura se ejecutó con la localhost 127.0.0.1, pero la idea es que se le pase cualquier IP donde corra el servidor y puedan comunicarse. Para obtener la ip de la maquina donde hayan ejecutado o van a ejecutar el servidor es con el comando ifconfig.
Tengan en cuenta que en este ejemplo el servidor se comunica con un solo cliente y luego ambos finalizan, pero la idea de una comunicación cliente – servidor es que el servidor pueda quedar ejecutando indefinidamente (con alguna finalización con señales por ejemplo) y derivando en hilos la atención para cada cliente.

ifconfig

```
root@sodium:/home/sodium/2017/TP3-SOCKETS# ifconfig
eth0      Link encap:Ethernet  direcciónHW 00:0c:29:37:2a:47
          Direc. inet:192.168.113.224  Difus.:192.168.113.255  Másc:255.255
.255.0
          Dirección inet6: fe80::20c:29ff:fe37:2a47/64 Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:21054 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:2648 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:1997871 (1.9 MB)  TX bytes:338488 (338.4 KB)
          Interrupción:19 Dirección base: 0x2024

lo        Link encap:Bucle local
          Direc. inet:127.0.0.1  Másc:255.0.0.0
          Dirección inet6: ::1/128 Alcance:Anfitrión
          ACTIVO BUCLE FUNCIONANDO  MTU:16436  Métrica:1
          Paquetes RX:58 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:58 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:0
          Bytes RX:3339 (3.3 KB)  TX bytes:3339 (3.3 KB)
```

```
root@sodium:/home/sodium/2017/TP3-SOCKETS# make
g++ -c servidor.c
g++ -o servidor servidor.o
g++ -c cliente.c
g++ -o cliente cliente.o
root@sodium:/home/sodium/2017/TP3-SOCKETS# ./servidor 5000
casa
casa
perro
perro
auto
auto
root@sodium:/home/sodium/2017/TP3-SOCKETS#
```

```
root@sodium:/home/sodium/2017/TP3-SOCKETS# ./cliente 127.0.0.1 5000
BIENVENIDO
TIPEE LAS SIGUIENTES PALABRAS
casa
casa
CORRECTO +10 pts
perro
perro
CORRECTO +10 pts
auto
auto
CORRECTO +10 pts
SU PUNTAJE ES: 30
root@sodium:/home/sodium/2017/TP3-SOCKETS#
```