

Introducción a la programación con sockets en C

Enrique Bonet

Departamento de Informática

Universidad de Valencia

Índice

- Introducción
 - Socket
 - Creación de un socket
 - Cierre de un socket
 - Asociación de un socket a un puerto
 - Diferencias entre TCP y UDP
 - Escucha de un puerto TCP
 - Servidor TCP
 - Cliente TCP
 - Recepción / Envío de datos en TCP
 - Recepción de datos en UDP
 - Envío de datos en UDP
 - Comprobación del estado de un socket
-

Introducción

- Para que dos aplicaciones puedan intercambiar información entre sí se necesita:
 - Un protocolo de comunicación común a nivel de red y a nivel de transporte.
 - Una dirección del protocolo de red que identifique a cada uno de los ordenadores.
 - Un número de puerto que identifique la aplicación dentro del ordenador.

Socket (I)

- Concepto abstracto que permite intercambiar información entre dos ordenadores.
- Se define mediante los parámetros:
 - Protocolo de red y de transporte.
 - Dirección de red.
 - Número de puerto.
- Básicamente, es un conector que recibe peticiones de conexión o solicita las mismas.

Socket (II)

- Implementan una arquitectura cliente/servidor.
 - Una aplicación, la aplicación servidor, permanece a la espera de que otras aplicaciones deseen sus servicios:
 - Utilizando un determinado protocolo de red y de transporte.
 - En una determinada dirección de red.
 - En un determinado número de puerto.
 - Otra aplicación, la aplicación cliente, solicita los servicios de la aplicación servidor:
 - Utilizando el mismo protocolo de red y de transporte.
 - Una dirección de red.
 - Un número de puerto.

Creación de un socket (I)

- El primer paso para que cualquier aplicación cliente o servidor pueda comunicarse es crear un socket.
- Se realiza con la función `socket()`.
- Sus parámetros son:
 - Dominio.
 - Tipo.
 - Protocolo.
- El valor devuelto es un entero:
 - ≥ 0 si el socket es creado correctamente.
 - < 0 si se produce un error en la creación.

Creación de un socket (II)

- Dominio: Indica el dominio de comunicación que se desea utilizar:
 - PF_INET: Protocolos de Internet versión 4.
 - PF_INET6: Protocolos de Internet versión 6.
 - PF_IPX: Protocolos IPX (Novell).
 - PF_APPLETALK: Protocolos Appletalk.
 - PF_UNIX o PF_LOCAL: Comunicación local.
 - Optimización de PF_INET para aplicaciones de uso local (conexión entre aplicaciones del propio ordenador).
 - ...

Creación de un socket (III)

- Tipo: Tipo de comunicación deseada.
 - SOCK_STREAM: Conexión bidireccional confiable con el flujo de datos ordenados.
 - SOCK_DGRAM: Mensajes no confiables, sin conexión, con una longitud máxima fija.
 - SOCK_SEQPACKET: Conexión bidireccional confiable con el flujo de datos ordenados y datagramas de longitud máxima fija.
 - ...
- No todos los tipos de comunicación deben estar implementados en todos los dominios.

Creación de un socket (IV)

- Protocolo: Selección del protocolo particular que utilizará el conector que deseamos crear.
 - Generalmente un dominio y tipo solo admite un protocolo particular, por lo que solo puede tomar el valor 0.
 - Dominio PF_INET y tipo SOCK_STREAM:
 - Protocolo de transporte TCP.
 - Dominio PF_INET y tipo SOCK_DGRAM:
 - Protocolo de transporte UDP.

Ejemplo (I)

```
#include <sys/types.h>
#include <sys/socket.h>
...
int s;
...
if ((s=socket(PF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("socket");
    exit(0);
}
```

Ejemplo (II)

```
#include <sys/types.h>
#include <sys/socket.h>
...
int s;
...
if ((s=socket(PF_INET, SOCK_DGRAM, 0))<0)
{
    perror("socket");
    exit(0);
}
```

Cierre de un socket

- Es el último paso cuando se quiere cerrar una conexión.
- El modo de cierre es independiente del dominio, tipo o protocolo que utilice el socket.
- Se realiza con la función `close()`.
- Sus parametros son:
 - Descriptor del socket a cerrar.
- Devuelve:
 - 0 si cierra el socket.
 - <0 si se produce un error.

Ejemplo

```
#include <unistd>
...
int s;
...
if (close(s)<0)
{
    perror("socket");
    exit(0);
}
```

Asociación de un socket a un puerto (I)

- Todo socket que actúe como servidor debe asociarse a un puerto.
 - El cliente necesita conocer en que puerto se encuentra disponible el servicio para dirigirse al mismo.
 - Puerto 21 TCP: FTP (Transferencia de archivos).
 - Puerto 22 TCP: SSH (Conexión segura).
 - Puerto 25 TCP: SMTP (Correo electrónico).
 - Puerto 80 TCP: HTTP (Web).
 - Puerto 443 TCP: HTTPS (Web segura).
 - Puerto 53 UDP: DNS (Servidor de nombres).
 - Puerto 123 UDP: NTP (Sincronización de la hora).

Asociación de un socket a un puerto (II)

- Se realiza con la función `bind()`.
- Su sintaxis es:

```
#include <sys/types.h>
#include <sys/socket.h>
...
int bind(int sockfd, struct sockaddr
*addr, int addrlen);
```
- `struct sockaddr` no se utiliza, se utiliza en su lugar `struct sockaddr_in`:
 - Necesidad de incluir la cabecera:

```
#include <netinet/in.h>
```

Asociación de un socket a un puerto (III)

- `sockfd`: Socket creado con anterioridad.
- `addr`: Puntero a la estructura `sockaddr_in` convertida a `sockaddr` mediante una conversión forzada de tipo (`cast`).
- `addrlen`: Tamaño de la estructura apuntada por el puntero `addr`.
- Devuelve:
 - 0 si sucede correctamente.
 - <0 si ocurre un error.

Asociación de un socket a un puerto (IV)

- La estructura `sockaddr_in` tiene los campos:

```
struct in_addr
{
    unsigned long int s_addr;
};
struct sockaddr_in
{
    int sin_family;
    unsigned short int sin_port;
    struct in_addr sin_addr;
};
```

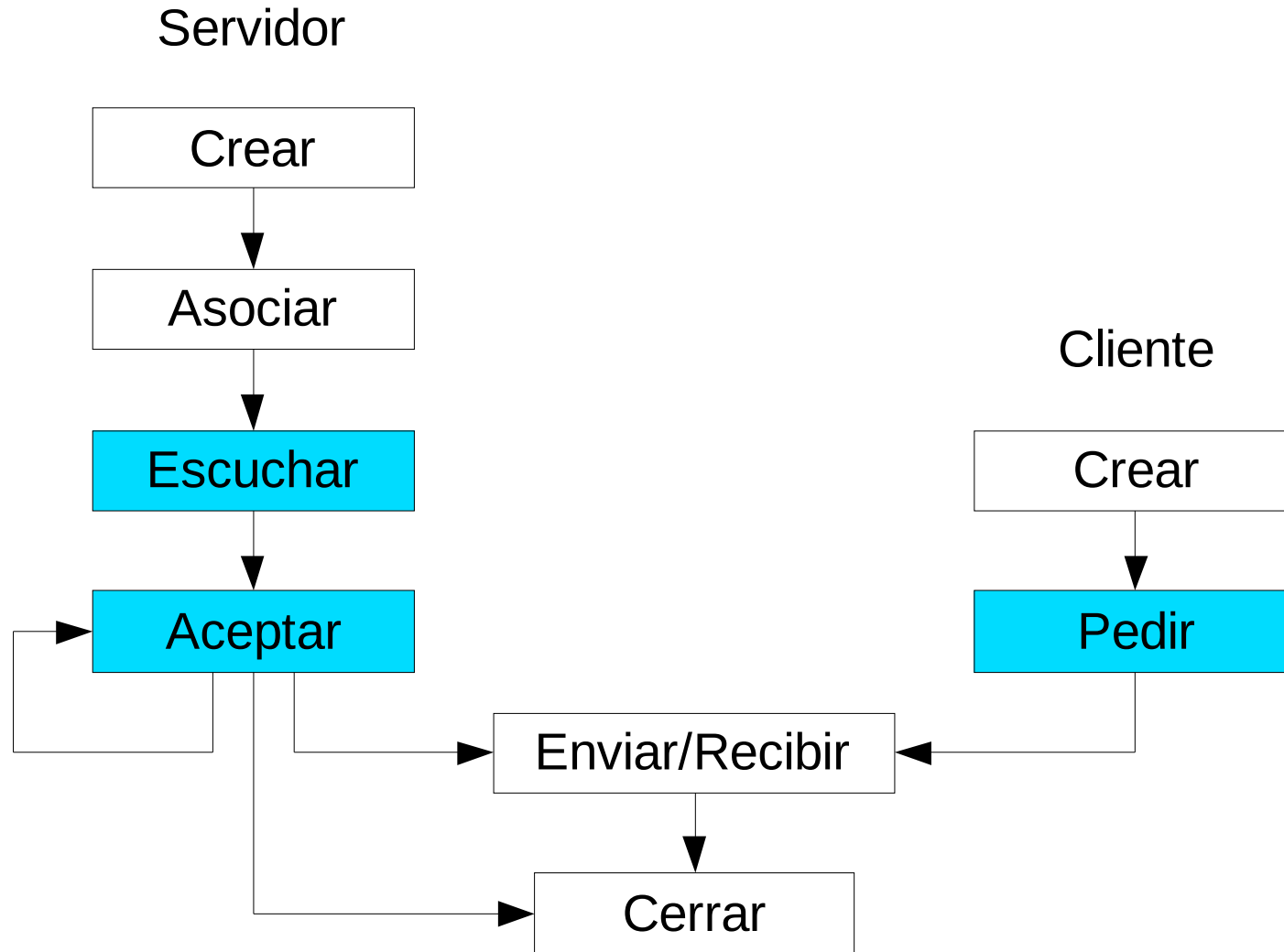
Asociación de un socket a un puerto (V)

- Donde:
 - `sin_family`: Protocolo de comunicación del socket, debe corresponder con el que se uso para crear el socket.
 - `sin_port`: Puerto al que se asocia el socket.
 - `sin_addr.s_addr`: Dirección de red en la que se pone a la escucha el socket (127.0.0.1, 147.156.1.1, etc.).
 - Si se desea que escuche todas las direcciones de red del ordenador se utiliza la constante `INADDR_ANY`.

Ejemplo

```
...
int s;
struct sockaddr_in dir;
...
dir.sin_family=PF_INET;
dir.sin_port=htons(puerto);
dir.sin_addr.s_addr=htonl(INADDR_ANY);
if (bind(s, (struct sockaddr *)&dir,
        sizeof(struct sockaddr_in))!=0)
{
    perror("bind");
    exit(0);
}
```

Diferencias entre TCP y UDP



Escucha de un puerto TCP

- Un socket TCP asociado a una dirección y puerto es puesto en escucha con la función `listen()`.
- Sus parámetros son:
 - Descriptor del socket a poner a la escucha.
 - Tamaño de la cola de aceptación de peticiones.
- Devuelve:
 - 0 si se ejecuta de forma correcta.
 - <0 si se produce un error.

Ejemplo

```
...  
int s;  
...  
if (listen(s, 5) != 0)  
{  
    perror("listen");  
    exit(0);  
}
```

Servidor TCP (I)

- Acepta las peticiones de conexión con `accept()`.
- Sus parámetros son:
 - Socket asociado a la dirección y puerto y puesto a la escucha.
 - Estructura que contendrá la dirección y puerto del cliente del que se acepta la conexión.
 - Puntero al tamaño de la estructura.

Servidor TCP (II)

- Valor devuelto:
 - ≥ 0 Identificador del socket donde se ha aceptado la conexión.
 - Al aceptar una conexión se crea un nuevo socket que es el que atiende la conexión.
 - El socket original que escuchaba la dirección y puerto permanece inalterado después de aceptar la conexión.
 - < 0 Error en la aceptación de la conexión.

Ejemplo

```
...
int sock, sock_conectado, tam;
struct sockaddr_in dir;
...
tam=sizeof(struct sockaddr_in);
if ((sock_conectado=accept(sock, (struct
    sockaddr *)&dir, &tam))<0)
{
    perror("accept");
    exit(0);
}
```

Cliente TCP

- Solicita la conexión mediante `connect()`.
- Sus parámetros son:
 - Descriptor del socket que se utiliza para la conexión.
 - Estructura con la dirección y puerto al que se desea conectarse.
 - Longitud de la estructura anterior.
- Valor devuelto:
 - 0 si la conexión se realiza de forma correcta.
 - <0 si se produce un error.

Ejemplo

```
...
int sock;
struct sockaddr_in dir;
...
dir.sin_family=PF_INET;
dir.sin_port=htons(13);
if (inet_aton("147.156.1.1",&dir.sin_addr)==0)
{
    perror("inet_aton");
    exit(0);
}
if (connect(sock, (struct sockaddr *)&dir, sizeof(struct sockaddr_in))<0)
{
    if (errno==ECONNREFUSED) /* Servicio no disponible */
        ...
    else /* Otro error */
    {
        perror("connect");
        exit(0);
    }
}
```

Recepción / Envío de datos en TCP

- Se suele realizar con las funciones `read()` (para leer datos) y `write()` (para escribir datos).
- Sus parámetros son:
 - Descriptor del socket conectado.
 - Buffer donde almacenar los datos a leer o que contiene los datos a escribir.
 - Tamaño máximo de los datos a leer o tamaño de los datos a escribir.
- Valor devuelto:
 - <0 si sucede un error.
 - ≥ 0 con el número de datos leídos o escritos.
 - Generalmente el valor 0 en lectura indica que se ha cerrado la conexión.

Ejemplo

```
...  
#define TAM 100  
  
...  
int sock, n;  
char buffer[TAM];  
  
...  
if ((n=read(sock, buffer, TAM))<0)  
{  
    perror("read");  
    exit(0);  
}
```

Recepción de datos en UDP

- Se realiza con las funciones `recvfrom()` o `recv()`.
- Sus parámetros son:
 - Descriptor del socket.
 - Buffer donde almacenar los datos a leer.
 - Tamaño máximo de los datos a leer.
 - Opciones de envío, generalmente valor 0 (ninguna opción).
 - Puntero a la estructura de datos que contendrá la dirección y puerto que envió los datos (solo en `recvfrom()`).
 - Puntero a la longitud de la estructura de datos anterior (solo en `recvfrom()`).
- Valor devuelto:
 - <0 si sucede un error.
 - ≥ 0 con el número de datos leídos o escritos.

Envío de datos en UDP

- Se realiza con la función `sendto()`.
- Sus parámetros son:
 - Descriptor del socket.
 - Buffer con los datos a enviar.
 - Tamaño de los datos a enviar.
 - Opciones de envío, generalmente valor 0 (ninguna opción).
 - Puntero a la estructura de datos que contiene la dirección y puerto donde enviar los datos.
 - Longitud de la estructura de datos anterior.
- Valor devuelto:
 - <0 si sucede un error.
 - ≥ 0 número de bytes escritos.

Ejemplo

```
#define TAM 100
...
int sock;
struct sockaddr_in dir;
char buffer[TAM];
...
dir.sin_family=PF_INET;
dir.sin_port=htons(13);
if (inet_aton("147.156.1.1",&dir.sin_addr)==0)
{
    perror("inet_aton");
    exit(0);
}
if (sendto(sock, buffer, strlen(buffer), (struct sockaddr *)&dir,
          sizeof(struct sockaddr_in))<0)
{
    perror("connect");
    exit(0);
}
```


Comprobación del estado de un socket (I)

- Se realiza con la función `select()`.
- Sus parámetros son:
 - Entero con el valor, incrementado en una unidad, del descriptor del socket más alto de cualquiera de los conjuntos que se desea comprobar.
 - Conjunto de sockets a comprobar para lectura.
 - En los sockets de tipo `SOCK_STREAM` también se modifica si llega una petición de conexión a un socket.
 - Conjunto de sockets a comprobar para escritura.
 - Conjunto de sockets a comprobar para excepciones.
 - Limite de tiempo antes de que termine la espera de comprobación.

Comprobación del estado de un socket (II)

- Valor devuelto:
 - <0 si sucede un error.
 - $=0$ Ningún socket ha cambiado su estado.
 - >0 Número de sockets cuyo estado ha cambiado.
- Si algún conjunto no desea comprobarse puede ponerse NULL.
- Si desea que el tiempo de comprobación sea infinito puede ponerse NULL.

Comprobación del estado de un socket (III)

- Los sockets a comprobar se introducen en el conjunto que se declara como:
`fd_set conjunto;`
- Para borrar el conjunto, insertar sockets, etc., se utilizan las funciones:
 - `FD_ZERO(&conjunto)`: Inicializa el conjunto.
 - `FD_SET(socket, &conjunto)`: Añade el socket al conjunto.
 - `FD_CLR(socket, &conjunto)`: Borra el socket del conjunto.
 - `FD_ISSET(socket, &conjunto)`: Comprueba si el socket se encuentra en el conjunto.

Comprobación del estado de un socket (IV)

- El limite de tiempo se comprueba con una estructura:

```
struct timeval{
    unsigned long int tv_sec;
    unsigned long int tv_usec;
};
```

- La función `select()`:
 - Deja en la estructura el tiempo que restaba hasta la finalización del tiempo de espera indicado.
 - Modifica los conjuntos dejando en ellos tan solo los sockets modificados.
 - La función `FD_ISSET()` permite ver cuales existen en el conjunto devuelto.

Ejemplo

```
...
int sock,n;
fd_set conjunto;
struct timeval timeout;
...
FD_ZERO(&conjunto);
FD_SET(sock, &conjunto);
timeout.tv_sec=1;
timeout.tv_usec=0;
if ((n=select(sock+1, &conjunto, NULL, NULL,
    &timeout)) < 0)
{
    perror("select");
    exit(0);
}
```