

¿Qué es un socket?



Sockets

- Un *socket* es una abstracción
- Representa un extremo en una comunicación bidireccional entre dos aplicaciones que se comunican a través de la red.

Sockets

- Diferentes tipos de sockets corresponden a diferentes tipos de protocolos.
- Solo trabajaremos con sockets de TCP/IP
- Sockets de flujo representan el extremo de una conexión TCP
- Sockets de datagrama son un servicio de mejor esfuerzo para el envío individual de datos.
- Un socket TCP/IP se identifica con un número de puerto y una dirección IP

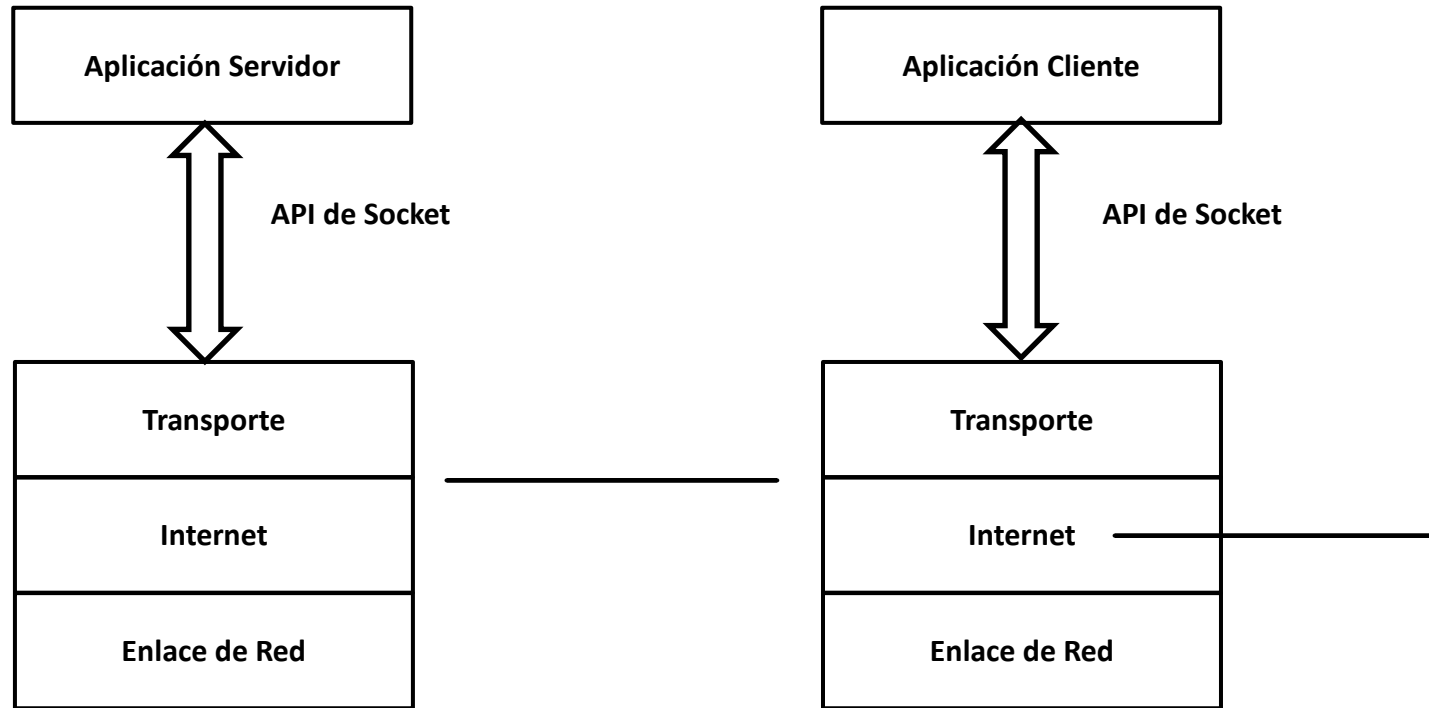
Sockets bloqueantes y no bloqueantes

- No se trata de sockets diferentes realmente, son solo opciones para las formas en las que trabajan
- Los sockets bloqueantes son aquellos que se quedan esperando hasta que existe información para establecer una conexión, leer o escribir un mensaje
- Los sockets no bloqueantes interrogan si hay datos para procesar y en caso de que no se así, continúan con el código
- El socket no bloqueante se definen modificando sus opciones

API de sockets

- Interfaz de programación de aplicaciones
- Conjunto de subrutinas, funciones y procedimientos (o métodos) que ofrece cierta biblioteca para ser utilizado por otro software.

Sockets API



Sockets orientados a conexión
bloqueantes

Socket de flujo bloqueantes

- Es el tipo de socket que utiliza el protocolo TCP y por tanto tiene todas las características relacionadas

API java

<https://docs.oracle.com/javase/8/docs/api>

Clase Socket

- Implementa un socket de flujo del lado del cliente
- Se le llama simplemente socket
- Se encuentra en el paquete **java.net**

Constructores principales de Socket

- **Socket();** crea un socket de flujo desconectado
- **Socket(InetAddress address, int port);** Crea un socket de flujo y lo conecta a un número de puerto en una IP definida
- **Socket(InetAddress address, int port, InetAddress localAddress, int localPort);** Crea un socket de flujo, ligado a una dirección y puerto local y lo conecta a un número de puerto en una IP definida remota

Métodos principales de Socket

- **void bind(SocketAddress bindport)**
- **void close()**
- **void connect(SocketAddress dst)**
- **void connect(SocketAddress dst, int t)**
- **SocketChannel getChannel()**
- **InetAddress getInetAddress()**
- **InputStream getInputStream()**
- **OutputStream getOutputStream()**

Métodos principales de Socket

- **boolean getKeepAlive()**
- **InetAddress getLocalAddress()**
- **int getLocalPort()**
- **boolean getOOBInline()**
- **int getPort()**
- **int getReceiveBufferSize()**
- **boolean getReuseAddress()**
- **int getSendBufferSize()**

Métodos principales de Socket

- `int getSoLinger()`
- `int getSoTimeout()`
- `boolean getTcpNoDelay()`
- `boolean isClosed()`
- `boolean isConnected()`
- `boolean isInputShutdown()`
- `boolean isOutputShutdown()`
- `void setKeepAlive(boolean b)`

Métodos principales de Socket

- **void setOOBInline(boolean b)**
- **void setReuseAddress(boolean b)**
- **void setSoLinger(boolean b, int t)**
- **void setSoTimeout(int t)**
- **void setTcpNoDelay(boolean b)**
- **void shutdownInput()**
- **void shutdownOutput()**

Clase ServerSocket

- Implementa un socket de servidor de flujo
- Una instancia de esta clase espera por solicitudes de conexión en la red
- Se encuentra en el paquete **java.net**

Constructores principales de `ServerSocket()`

- `ServerSocket()`; crea un socket de servidor.
- `ServerSocket(int pto)`; crea un socket de servidor asociado a un puerto.
- `ServerSocket(int pto, int backlog)`; crea un socket de servidor ligado a un puerto con una cola de conexiones específica.
- `ServerSocket(int pto, int backlog, InetAddress dir_local)`; crea un socket de servidor ligado a un puerto con una cola de conexiones específica y una dirección IP local.

Métodos principales de ServerSocket

- **Socket accept()**
- **void bind(SocketAddress local)**
- **void close()**
- **InetAddress getInetAddress()**
- **int getReceiveBufferSize()**
- **boolean getReuseAddress()**
- **int getSoTimeout()**
- **void setReuseAddress(boolean b)**

Métodos principales de ServerSocket

- **`void setSoTimeout(int t)`**

ServerSocket() y Bind()

```
ServerSocket s = new ServerSocket();
```

```
InetSocketAddress dir = new InetSocketAddress(1234);
```

```
s.bind(dir);
```

Ó

```
ServerSocket s = new ServerSocket(1234);
```

Flujos en java

- Paquete **java.io**



- Flujos orientados a byte / orientados a carácter

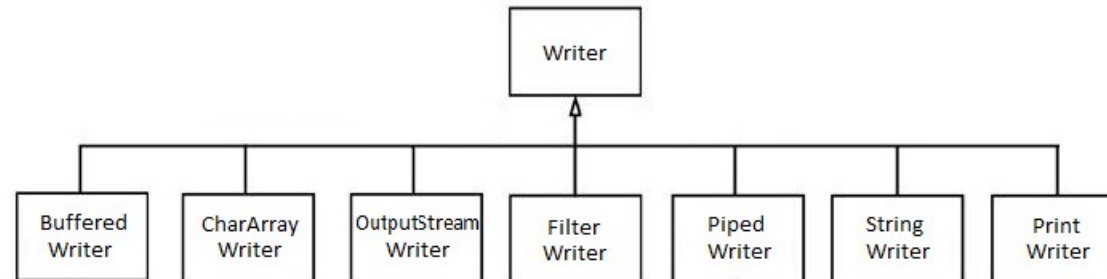
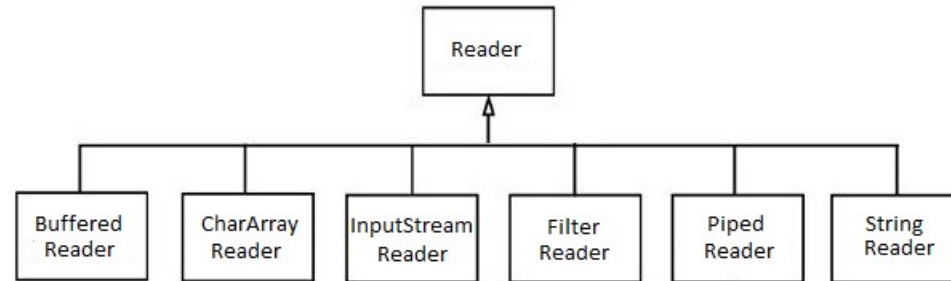
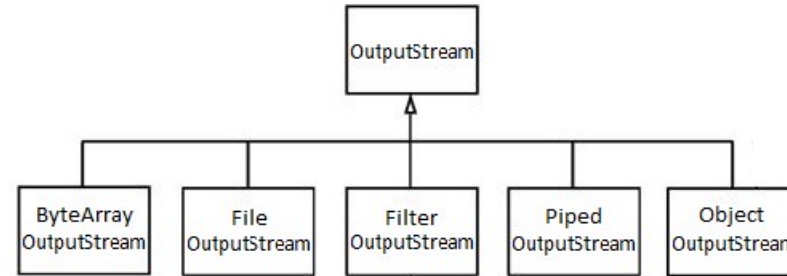
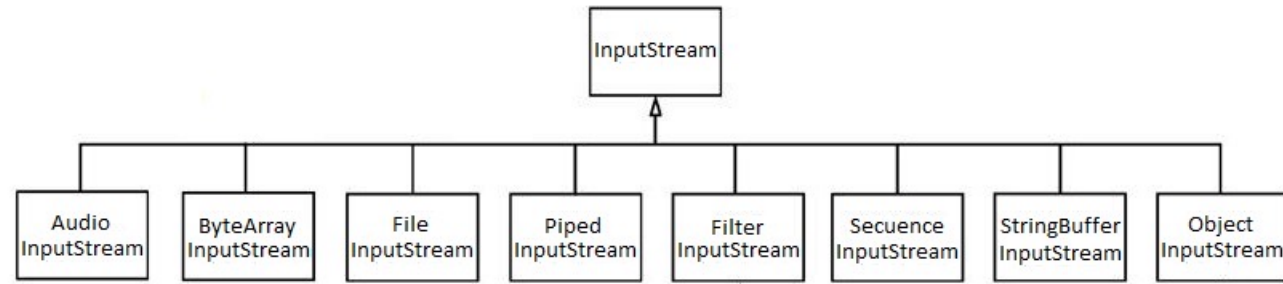
Flujos orientados a byte

- Byte (8bits)
- Más primitivos y portables
- Los demás flujos lo usan
- Flujo de bajo nivel
- **InputStream y OutputStream**

Flujos orientados a carácter

- char (16 bits)
- Codificación unicode
- Ideal para texto plano
- Reader y Writer

Diagrama de clases principales



Lectura y escritura

- Abrir
- Leer o escribir
- Cerrar

Lectura, InputStream

- **int read();** Lee el próximo byte del flujo representado en un entero.
Devuelve -1 si no quedan más datos que leer.
- **int read(byte[] b);** Lee un arreglo de bytes del flujo.
- **int read(byte[] b, int off, int tam);** Lee un arreglo de bytes del flujo, desde y hasta la posición indicada

Lectura, Reader

- `int read()` – Lee el próximo carácter del flujo representado en un entero. Devuelve -1 si no quedan ms datos que leer.
- `int read(char[] cbuf)` – Lee un arreglo de caracteres del flujo.
- `int read(char[] cbuf, int off, int len)` – Lee un arreglo de caracteres del flujo, desde y hasta la posición indicada.

Escritura, OutputStream

- **`void write(int b);`** Escribe un solo byte en el flujo.
- **`void write(byte[] b);`** Escribe un arreglo de bytes en el flujo.
- **`void write(byte[] b, int off, int len);`** Escribe una porción de un arreglo de bytes en el flujo.

Escritura, Writer

- **`void write(int c);`** Escribe un solo carácter en el flujo.
- **`void write(char[] cbuf);`** Escribe un arreglo de caracteres en el flujo.
- **`void write(char[] cbuf, int off, int len);`** Escribe una porción de un arreglo de caracteres en el flujo

Entrada y salida estándar

- Clase **System** dentro de **java.lang**
- **InputStream in (InputStream);** Flujo de entrada estándar.
Típicamente corresponde al teclado.
- **PrintStream out (OutputStream);** Flujo de salida estándar.
Típicamente corresponde a la pantalla.
- **PrintStream err (OutputStream);** Flujo de salida estándar de errores. Típicamente corresponde a la pantalla.
- Pueden ser redirigidos

¿Cómo enviar distintos tipos de datos a través de un socket?

- Texto: Para enviar texto sin importar el tipo de codificación se usan las clases `PrintWriter` y `OutputStreamWriter`.
- Ej.

```
try{
    Socket cl = new Socket("127.0.0.1",1234);
    PrintWriter pw = new PrintWriter(new OutputStreamWriter(cl.getOutputStream()));
    String msj = "un mensaje";
    pw.println(msj);
}catch(Exception e){
    e.printStackTrace();
}
```

¿Cómo enviar distintos tipos de datos a través de un socket?

- Primitivos: Para enviar tipos de dato primitivos (int, float, long, boolean, etc.) se usa la clase `DataOutputStream`.

- Ej.

```
try{
    Socket cl = new Socket("127.0.0.1",1234);
    DataOutputStream dos = new DataOutputStream(cl.getOutputStream());
    String msj = "un mensaje";
    dos.writeUTF(msj);
    dos.writeInt(3);
    dos.writeFloat(2.1f);
}catch(Exception e){
    e.printStackTrace();
}
```


¿Cómo enviar distintos tipos de datos a través de un socket?

- Objetos y primitivos: Para enviar objetos, o tipos de dato primitivos (int, float, long, boolean, etc.) se usa la clase `ObjectOutputStream`.

- Ej.

```
try{
    Socket cl = new Socket("127.0.0.1",1234);
    ObjectOutputStream oos = new ObjectOutputStream(cl.getOutputStream());
    String msj = "un mensaje";
    oos.writeUTF(msj);
    oos.writeInt(3);
    oos.writeFloat(2.1f);
    Dato d = new Dato(1,2.0f,"tres");
    oos.writeObject(d);
}catch(Exception e){
    e.printStackTrace();
}
```

¿Cómo recibir distintos tipos de datos a través de un socket?

- Texto: Para recibir texto sin importar el tipo de codificación se usan las clases `BufferedReader` e `InputStreamReader`.
- Ej.

```
try{
    Socket cl = new Socket("127.0.0.1",1234);
    BufferedReader br = new BufferedReader(new InputStreamReader(cl.getInputStream()));
    String msj = br.readLine();
}catch(Exception e){
    e.printStackTrace();
}
```

¿Cómo recibir distintos tipos de datos a través de un socket?

- Primitivos: Para recibir tipos de dato primitivos (int, float, long, boolean, etc.) se usa la clase `DataInputStream`.
- Ej.

```
try{
    Socket cl = new Socket("127.0.0.1",1234);
    DataInputStream dis = new DataInputStream(cl.getInputStream());
    String v1 = dis.readUTF();
    int v2 = dis.readInt();
    float v3 = dis.readFloat();
}catch(Exception e){
    e.printStackTrace();
}
```

¿Cómo recibir distintos tipos de datos a través de un socket?

- Objetos y primitivos: Para recibir objetos, o tipos de dato primitivos (int, float, long, boolean, etc.) se usa la clase `ObjectInputStream`.

- Ej.

```
try{
    Socket cl = new Socket("127.0.0.1",1234);
    ObjectInputStream ois = new ObjectInputStream(cl.getInputStream());
    String v1 = ois.readUTF();
    int v2 = ois.readInt();
    float v3 = ois.readFloat();
    Dato d = (Dato)ois.readObject();
}catch(Exception e){
    e.printStackTrace();
}
```

Serialización

Proceso mediante el cual solo los datos del objeto (no el código que implementa el objeto) son convertidos a un arreglo de bytes para su envío.

Marhalling

Proceso mediante el cual se serialize el objeto (datos) y también el codebase (código que implementa el objeto).

Ejemplo: servicio de eco

- Revisar los programas CEcoFlujo.java y SEcoFlujo.java desde los recursos

Ejemplo: envío de objetos

- Revisar los programas Cliente_O.java y Servidor_O.java y Objeto.java desde los recursos

Ejemplo: envío de archivos

- Revisar los programas Cenvia.java y Srecibe.java desde los recursos

Tarea

- Modificar el archivo anterior para que permita el envío de múltiples archivos

Sockets en C

Sockets de flujo bloqueantes

Bibliotecas más utilizadas

- **<sys/types.h>**: tipos de datos utilizados(pthread_attr_t, size_t, socklen_t, etc.)
- **<sys/socket.h>**: macros: SOCK_STREAM, SOCK_DGRAM, SOL_SOCKET, etc. Prototipos: socket(), bind(), send(), recv(), accept(), etc.
- **<stdlib.h>**: prototipos: atoi(), malloc(), exit()
- **<stdio.h>**: prototipos: fopen(), fdopen(), fflush(), scanf(), printf(), etc.
- **<netdb.h>**: prototipos: freeaddrinfo(), getaddrinfo(), getnameinfo(), etc.

Estructura sockaddr_in //<netinet/in.h>

```
struct sockaddr_in {  
    short sin_family; // AF_INET (IPv4), AF_UNIX, AF_LOCAL, etc.  
    unsigned short sin_port; // ej. htons(2000)  
    struct in_addr sin_addr; // ver estructura in_addr  
    char sin_zero[8]; // poner en cero's  
};  
  
struct in_addr {  
    unsigned long s_addr; // load with inet_aton()  
};
```

Estructura addrinfo //<netdb.h>

```
struct addrinfo {  
    int ai_flags;          // AI_PASSIVE, AI_CANONNAME, AI_NUMERIC_HOST, etc.  
    int ai_family;        // AF_INET, AF_INET6, AF_UNSPEC, AF_BTH, AF_IRDA, etc.  
    int ai_socktype;      // SOCK_STREAM, SOCK_DGRAM, SOCK_RAW, SOCK_RDM  
    int ai_protocol;      // 0, IPPROTO_TCP, IPPROTO_UDP  
    socklen_t ai_addrlen; // sizeof(ai_addr)  
    struct sockaddr *ai_addr; // struct sockaddr_in/sockaddr_in6  
    char *ai_canonname;     // nombre canónico  
    struct addrinfo *ai_next; // sig. Nodo de lista ligada  
};
```

Nota: ai_flags=PASSIVE && nodo=NULL (en func. getaddrinfo()) para hacer bind()

Función getaddrinfo()

//<sys/types.h>, <sys/socket.h>,
//<netdb.h>

```
int getaddrinfo(const char *nodo, //ej. "www.pc1.net" ó "127.0.0.1"  
               const char *servicio, //ej. "FTP", ó "21"  
               const struct addrinfo *i, // apunta a estructura con info importante  
               struct addrinfo **res); //apuntador a lista ligada con el resultado de la consulta
```

-
- Valor devuelto

0 = éxito

EAI_ADDRFAMILY= El host no tiene una dirección IP en la familia de direcciones

EAI_AGAIN= El nombre de host devolvió una falla temporal (reintentar)

EAI_BADFLAGS=*i.ai_flags* contiene una bandera inválida/está habilitada la bandera
AI_CANNONNAME y el nombre es NULL

EAI_FAIL= Falla permanente

EAI_FAMILY= familia de direcciones no soportada

EAI_NONAME=Nodo o servicio desconocidos, o ambos son NULL, o están puestas las
banderas **AI_NUMERICSERV** o **AI_NUMERICHOST** y uno/ambos de ellos no es numérico

Ejemplo 1 //para un servidor

```
int r;
struct addrinfo i, *lista;
memset(&i,0,sizeof(i));
i.ai_family = AF_INET6; // IPv4 ó IPv6
i.ai_socktype = SOCK_STREAM;
i.ai_flags = AI_PASSIVE; //solo para el servidor o cuando se use bind
if((r=getaddrinfo(NULL,"5678", &i,&lista))!=0){
    fprintf(stderr,"error:%s\n",gai_strerror(r));
    exit(1);
}
// se crea el socket y cuando ya no se necesite la lista se elimina
freeaddrinfo(lista);
```

Ejemplo2 //para un cliente

```
int r;
struct addrinfo i, *lista;
memset(&i,0,sizeof(i));
i.ai_family = AF_UNSPEC; // IPv4 ó IPv6
i.ai_socktype = SOCK_STREAM;
if((r=getaddrinfo("200.1.2.3","5678", &i,&lista))!=0){
    fprintf(stderr,"error:%s\n",gai_strerror(r));
    exit(1);
}
// se crea el socket y cuando ya no se necesite la lista se elimina
freeaddrinfo(lista);
```


Función socket() //<sys/socket.h>

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

```
int sd;
struct addrinfo i, *r, *p;
memset(&i, 0, sizeof(i)); //inicio
i.ai_family = AF_INET6; /* Permite IPv4 or IPv6 */
i.ai_socktype = SOCK_STREAM;
i.ai_flags = AI_PASSIVE; // utilizado para hacer el bind
i.ai_protocol = 0; /* Any protocol */
i.ai_canonname = NULL;
i.ai_addr = NULL;
i.ai_next = NULL;
if ((rv = getaddrinfo(NULL, pto, &i, &r)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    return 1;
} //if
for(p = r; p != NULL; p = p->ai_next) {
    if ((sd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
        perror("server: socket");
        continue;
    } //if
    break;
} //for
```

Familia de direcciones (1/2)

AF_LOCAL	Es otro nombre para AF_UNIX
AF_INET	Protocolo internet DARPA (TCP/IP)
AF_INET6	Protocolo internet versión 6
AF_PUP	Antigua red Xerox
AF_CHAOS	Red Chaos del MIT
AF_NS	Arquitectura Xerox Network System
AF_ISO	Protocolos OSI
AF_ECMA	Red European Computer Manufactures
AF_DATAKIT	Red Datakit de AT&T
AF_CCITT	Protocolos del CCITT, por ejemplo X.25
AF_SNA	System Network Architecture (SNA) de IBM
AF_DECnet	Red DEC

Familia de direcciones (2/2)

AF_IMPLINK	Antigua interfaz de enlace 1822 Interface Message Processor
AF_DLI	Interfaz directa de enlace
AF_LAT	Interfaz de terminales de red de área local
AF_HYLINK	Network System, Córporation Hyperchannel
AF_APPLETALK	Red AppleTalk
AF_ROUTE	Comunicación con la capa de encaminamiento del núcleo
AF_LINK	Acceso a la capa de enlace
AF_XTP	eXpress Transfer Protocol
AF_COIP	Connection-oriented IP (ST II)
AF_CNT	Computer Network Tecnology
AF_IPX	Protocolo Internet de Novell

Tipos de semántica de la comunicación

- **SOCK_STREAM**, sockets de flujo
- **SOCK_DGRAM**, sockets de datagrama
- **SOCK_RAW**, sockets crudos
- **SOCK_SEQPACKET**, conector no orientado a conexión pero fiable de longitud fija (solo en **AF_NS**)
- **SOCK_RDM**, conector no orientado a conexión pero fiable y secuencial (no implementado pero se puede simular a nivel de capa de usuario)

Función bind()

```
#include <sys/socket.h>
```

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

```
int bind(int sd, const struct sockaddr *addr, socklen_t addrlen);
```

- Valor devuelto:

$\left\{ \begin{array}{l} 0 = \text{éxito} \\ -1 = \text{error} \end{array} \right.$

Ejemplo bind()

```
int sd;
struct addrinfo i, *r, *p;
memset(&i, 0, sizeof(i)); //inicio
i.ai_family = AF_INET6; /* Permite IPv4 or IPv6 */
i.ai_socktype = SOCK_STREAM;
i.ai_flags = AI_PASSIVE; // utilizado para hacer el bind
i.ai_protocol = 0; /* Any protocol */
i.ai_canonname = NULL;
i.ai_addr = NULL;
i.ai_next = NULL;
if ((rv = getaddrinfo(NULL, pto, &i, &r)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    return 1;
} //if
for(p = r; p != NULL; p = p->ai_next) {
    if ((sd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
        perror("server: socket");
        continue;
    } //if
    if (bind(sd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sd);
        perror("server: bind");
        continue;
    } //if

    break;
} //for
```

Función listen()

```
#include <sys/socket.h>
```

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

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

//backlog tiene un máximo definido en SOMAXCONN=128 en

/usr/src/linux/net/ipv4/af_inet.c. 5= mal desempeño en webserver

(/usr/src/linux/socket.h en kernels 2.x)

Valor devuelto: { 0= éxito
-1= error

```
if (listen(sd, 128) == -1) {  
    perror("error en func. Listen() \n");  
    close(sd);  
    exit(1);  
}
```

Función accept()

`#include <sys/socket.h>`

int accept (int sd, struct sockaddr *dir, socklen_t *tam_dir)

Valor devuelto: {
>0 = éxito
-1= error

```
for (;;) {  
    char hbuf[NI_MAXHOST], sbuf[NI_MAXSERV];  
    struct sockaddr_storage cdir;  
    socklen_t ctam = sizeof(cdir);  
    cd = accept(sd, (struct sockaddr *)&cdir, &ctam);  
    if (cd == -1) {  
        perror("accept");  
        continue;  
    }  
    if(getnameinfo((struct sockaddr *)&cdir, sizeof(cdir), hbuf,  
        sizeof(hbuf), sbuf, sizeof(sbuf), NI_NUMERICHOST | NI_NUMERICSERV) ==  
        0)  
        printf("cliente conectado desde %s:%s\n", hbuf, sbuf);
```


Función write()

`#include <unistd.h>`

`int write(int sd, const void *buf, size_t tam)`

valor devuelto:

`>0` = #bytes enviados
`-1` = error
`0` = socket cerrado

```
char *msj ="un mensaje";
int n = write(cd,msj, strlen(msj)+1);
if(n<0)
    perror("Error en la función write\n");
else if(n==0){
    perror("Socket cerrado\n");
    exit(1);
}
int v=2;
n = write(cd,&v,sizeof(v));

float v2= 5.1f;
Char b[10];
memset(b,0,sizeof(b));
sprintf(b,"%f",v2);
n=write(cd,b,strlen(b)+1);

...
```

```
struct dato{
    char nombre[30];
    char apellido[25];
    int edad;
};

struct dato *o;
o = (struct dato *)malloc(sizeof (struct dato));

O->nombre="Juan";
O->apellido="Perez";
O->edad=htonl(23);
n = write(cd,(const char*)o,sizeof(struct dato));

...
free(o);
```

Función send()

`#include <sys/socket.h>`

`int send(int sd, const void *buf, size_t tam, int bandera)`

```
char *msj = "un mensaje";
int n = send(cd, msj, strlen(msj)+1, 0);
if (n < 0)
    perror("Error en la función send()\n");
else if (n == 0) {
    perror("Socket cerrado\n");
    exit(1);
}
int v=2;
n = send(cd, &v, sizeof(v), 0);
...
```

```
float v= 5.1f;
Char b[10];
memset(b, 0, sizeof(b));
sprintf(b, "%f", v);
n=send(cd, b, strlen(b)+1, 0);
```

```
struct dato *o;
o = (struct dato *)malloc(sizeof (struct dato));

O->nombre="Juan";
O->apellido="Perez";
O->edad=htonl(23);
n = send(cd, (const char*)o, sizeof(struct dato), 0);
```

0 = prioridad default
MSG_OOB= alta prioridad

Valor devuelto:

>0 = #bytes enviados
-1 = error
0 = socket cerrado

Función read()

```
#include <unistd.h>
```

```
int read(int sd, const void *buf, size_t tam)
```

```
char buf[100];
bzero(buf, sizeof(buf));
int n = read(cd, buf, sizeof(buf));
if(n < 0)
    perror("Error en la función read()\n");
else if(n == 0){
    perror("Socket cerrado\n");
    exit(1);
}
int v;
n = read(cd, &v, sizeof(v));

char b[10];
bzero(b, sizeof(b));
int n = read(cd, b, sizeof(b));
float v1 = atof(b);
...
```

Valor devuelto:

> 0 = #bytes leídos
-1 = error
0 = socket cerrado

```
struct dato{
    char nombre[30];
    char apellido[25];
    int edad;
};

char b[200];
bzero(b, sizeof(b));
n = read(cd, b, sizeof(b));
struct dato *o = (struct dato *)b;
```

Función recv()

```
#include <sys/socket.h>
```

```
int recv(int sd, const void *buf, size_t tam, int bandera)
```

```
char buf[100];  
int n = recv(cd, buf, sizeof(buf), 0);  
if (n < 0)  
    perror("Error en la función recv\n");  
else if (n == 0) {  
    perror("Socket cerrado\n");  
    exit(1);  
}  
int v;  
n = recv(cd, &v, sizeof(v), MSG_OOB);  
...
```

```
char b[10];  
bzero(b, sizeof(b));  
int n =  
recv(cd, b, sizeof(b), 0);  
float v1 = atof(b);
```

Valor devuelto:

>0 = #bytes leídos
-1 = error
0 = socket cerrado

0 = prioridad default
MSG_OOB = alta prioridad

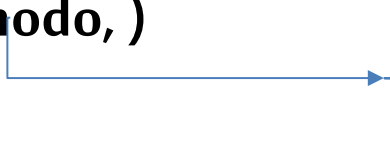
//SO_OOBINLINE

```
char b[200];  
bzero(b, sizeof(b));  
n = recv(cd, b, sizeof(b), 0);  
struct dato *o = (struct dato *)b;
```

Función shutdown()

`#include <sys/socket.h>`

`int shutdown(int sd, int modo,)`



SHUT_RD = deshabilita lectura
SHUT_WR = deshabilita escritura
SHUT_RDWR = deshabilita ambas

Valor devuelto: 
0 = éxito
-1 = error

```
cd = accept(sd, (struct sockaddr *)&cdir, &ctam);  
if (shutdown(cd, SHUT_RD) != 0)  
    perror("No fue posible deshabilitar lectura");
```

Función close()

```
#include <unistd.h>
```

```
int close(int sd)
```

Valor devuelto: $\begin{cases} 0 = \text{éxito} \\ -1 = \text{error} \end{cases}$

Función connect()

```
#include <sys/socket.h>
```

```
#include <sys/types.h>
```

```
int connect(int sd, const struct sockaddr *dir, socklen_t tam_ref);
```

Valor devuelto: $\begin{cases} 0 = \text{éxito} \\ -1 = \text{error} \end{cases}$

Ej. connect()

```
int op = 0;
for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((cd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
        perror("client: socket");
        continue;
    }

    /*if (setsockopt(cd, IPPROTO_IPV6, IPV6_V6ONLY, (void *)&op, sizeof(op)) == -1) {
        perror("setsockopt no soporta IPv6");
        exit(1);
    }*/

    if (connect(cd, p->ai_addr, p->ai_addrlen) == -1) {
        close(cd);
        perror("client: connect");
        continue;
    }

    break;
} //for
```


Ejemplo: servicio de eco

- Revisar los programas `ceco.c` y `seco.c` desde los recursos

Ejemplo: envío de estructura

- Revisar los programas enviaEC.c y enviaES.c desde los recursos

Ejemplo: envío de archivos

- Revisar los programas `enviaArchC.c` y `enviaArchS.c` desde los recursos