# Estructuras y funciones de programación de sockets.

**Introducción.**

En estos apuntes se incluye una breve descripción de las estructuras y funciones cuyo uso puede ser necesario para el desarrollo de las prácticas de programación de sockets planteadas en el presente curso.

Las funciones están clasificadas de acuerdo a las tareas que realizan, tales como creación y cierre de sockets, lectura y escritura, etc. A su vez, cada descripción de las funciones está organizada con la siguiente estructura interna:

- Fichero o ficheros de cabecera que han de incluirse para el uso de la función.
- Prototipo de la función.
- Breve descripción de la misma así como de los parámetros que utiliza.

Una descripción más detallada de estas funciones, así como información sobre cualquier otra función no especificada en estos apuntes, puede encontrase consultando las páginas de manual de Linux mediante el comando man.

**Creación y cierre.**

## socket.

*#include <sys/types.h>*
*#include <sys/socket.h>*

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

La función *socket* crea un extremo de una comunicación y devuelve un descriptor. Los argumentos son:

| | |
|---|---|
| *dominio* | Dominio de comunicaciones, el valor generalmente usado es *AF_INET*, también llamado como *PF_INET*. |
| *tipo* | Especifica la semántica de la comunicación, sus valores más comunes son *SOCK_STREAM* para TCP y *SOCK_DGRAM* para UDP. |
| *protocolo* | Protocolo particular para ser usado con el conector, generalmente valor 0. |

Si tiene éxito, la función devuelve un *int* que indica el valor del conector. En caso de error el valor es -1.

## close.

*#include <unistd.h>*

*int close(int fd);*

La función *close* cierra un descriptor de fichero o socket. Su argumento es:

*fd* Descriptor del fichero o socket a cerrar.

La función *close* devuelve 0 si sucede. En caso de error devuelve el valor -1.

**Asociación a puertos y especificación de propiedades.**

## bind.

*#include <sys/types.h>*
*#include <sys/socket.h>*

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

La función *bind* asocia el socket dado por *sockfd* a la dirección local especificada por *addr* para que el socket quede asignado al puerto especificado en la misma. Sus argumentos son:

*sockfd* Descriptor del socket creado con anterioridad.
*addr* Estructura de datos donde se especifica la dirección y puerto al que se asocia el socket.
*addrlen* Longitud de la estructura de datos anterior.

La función *bind* se suele utilizar solo con sockets de tipo *SOCK_STREAM*. La estructura *sockaddr* no suele ser utilizada, siendo siempre utilizada en su lugar la estructura *sockaddr_in*, cuya declaración puede verse a continuación.

```
struct in_addr
{
        unsigned long int s_addr;
};

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

En esta estructura, cada uno de los campos tiene la siguiente utilidad:

*sin_family* Dominio de comunicaciones del socket, generalmente *AF_INET*.
*port* Puerto al que se asocia el socket.
*sin_addr.s_addr* Dirección IP a la que se asocia el socket. Para permitir conexiones de cualquier dirección utilizar el valor *INADDR_ANY*.

La función *bind* devuelve 0 en caso de éxito. El valor -1 es devuelto si sucede un error.

## listen.

*#include <sys/socket.h>*

*int listen(int s, int backlog);*

La función listen especifica que el socket dado por *s* desea aceptar conexiones. La descripción de sus argumentos es:

*s* Descriptor del socket creado con anterioridad.
*backlog* Longitud máxima de la cola de conexiones pendientes, por compatibilidad con versiones anteriores, el valor máximo que debe especificarse es 5.

La función *listen* devuelve 0 si sucede y -1 en caso de error.

## ioctl.

*#include <sys/ioctl.h>*

*int ioctl(int d, int peticion, …);*

La función *ioctl* manipula los valores de los parámetros de un socket. La función *ioctl* posee un número variable de argumentos según el valor a modificar, por ello solo explicaremos el caso que puede ser necesario utilizar en los programas propuestos. Dicho caso consiste en la modificación del modo de funcionamiento del socket.

Un socket puede funcionar en modo bloqueante, en el cual espera hasta que se produzca una petición solicitada (lectura de datos, escritura de datos, etc.), o bien en modo no bloqueante, en el cual intenta la petición solicitada y si esta disponible la realiza, terminando inmediatamente, sin ningún tipo de espera, en caso contrario. En nuestro caso particular la función *ioctl* toma la forma:

*int ioctl(int d, int peticion, int &tipo);*

Donde los valores de los argumentos son:

*d* Descriptor del socket creado con anterioridad.
*peticion* Propiedad a cambiar, en nuestro caso modo de funcionamiento del socket.
*tipo* Modo de funcionamiento. Indica funcionamiento bloqueante (0) o no

bloqueante (1).

La función *ioctl* devuelve 0 si sucede y -1 falla.

**Aceptación y petición de conexiones.**

## accept.

*#include <sys/types.h>*
*#include <sys/socket.h>*

*int accept(int s, struct sockaddr *addr, int addrlen);*

La función *accept* acepta una petición de conexión al socket especificado por s. Los parámetros son:

| | |
|---|---|
| *s* | Descriptor del socket creado con anterioridad. |
| *addr* | Estructura de datos donde se especificará la dirección y puerto del descriptor que se a conectado a este socket. |
| *addrlen* | Longitud de la estructura de datos anterior. |

La estructura *sockaddr* no suele ser utilizada, siendo siempre utilizada en su lugar la estructura *sockaddr_in*, explicada con anterioridad.

La función devuelve un entero no negativo que es el descriptor del socket aceptado o -1 si sucede un error. El socket original (parámetro s) permanece en cualquier caso inalterado, pudiendo ser utilizado en posteriores llamadas a la función.

## connect.

*#include <sys/types.h>*
*#include <sys/socket.h>*

*int connect(int sockfd, const struct sockaddr *serv_addr, int addrlen);*

La función *connect* solicita poder conectar el socket especificado por *sockfd* a un socket remoto que es especifíca en *serv_addr*. Los parámetros son:

| | |
|---|---|
| *sockfd* | Descriptor del socket creado con anterioridad. |
| *serv_addr* | Estructura de datos donde se especifica la dirección y puerto con el que deseamos establecer la conexión. |
| *addrlen* | Longitud de la estructura de datos anterior. |

La estructura *sockaddr* no suele ser utilizada, siendo siempre utilizada en su lugar la estructura *sockaddr_in*, explicada con anterioridad.

La función devuelve el valor -1 si error o 0 si su llamada tiene éxito.

**Comprobación del estado de un socket.**

## select.

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

int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
        struct timeval *timeout);
```

La función *select* comprueba el estado de un socket. La comprobación afecta a tres conjuntos distintos de descriptores de socket (*readfds*, *writefds* y *exceptfds*). Los parámetros son:

| | |
|---|---|
| *n* | Valor, incrementado en una unidad, del descriptor más alto de cualquiera de los tres conjuntos. |
| *readfds* | Conjunto de sockets que serán comprobados para ver si existen caracteres para leer. Si el socket es de tipo *SOCK_STREAM* y no esta conectado, también se modificará este conjunto si llega una petición de conexión. |
| *writefds* | Conjunto de sockets que serán comprobados para ver si se puede escribir en ellos. |
| *exceptfds* | Conjunto de sockets que serán comprobados para ver si ocurren excepciones. |
| *timeout* | Limite superior de tiempo antes de que la llamada a select termine. Si *timeout* es *NULL*, la función select no termina hasta que se produzca algún cambio en uno de los conjuntos (llamada bloqueante a select). |

La declaración de la estructura timeval es la siguiente:

```
struct timeval
{
        unsigned long int tv_sec; /* Segundos */
        unsigned long int tv_usec; /* Millonesimas de segundo */
};
```

Si no se desea comprobar alguna de las condiciones que proporciona la función *select* (lectura, escritura y excepciones), puede sustituirse el puntero al conjunto por un puntero *NULL*.

Para manejar el conjunto *fd_set* se proporcionan cuatro macros:

```
FD_ZERO(fd_set *set);
FD_SET(int fd, fd_set *set);
FD_CLR(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *est);
```

*FD_ZERO* inicializa el conjunto *fd_set* especificado por *set*.

*FD_SET* y *FD_CLR* añaden o borran un descriptor de socket dado por *fd* al conjunto dado por *set*.

*FD_ISSSET* mira si el descriptor de socket dado por *fd* se encuentra en el conjunto especificado por *set*.

La función devuelve el valor -1 en caso de error y un número, cuyo valor es 0 si se produce el timeout antes de que suceda ninguna modificación en el estado de los descriptores incluidos en los conjuntos, o un valor mayor que 0 si se produce una modificación en algún descriptor. El número mayor que 0 indica el número de descriptores que han sufrido una modificación de forma simultanea, pues basta un solo cambio para salir de la función.

**Lectura y escritura.**

## read.

*#include <unistd.h>*

*int read(int fd, void *buf, int n_bytes);*

La función *read* lee datos del socket especificado por *fd*. Sus parámetros son:

| | |
|---|---|
| *fd* | Descriptor del socket creado con anterioridad. |
| *buf* | Buffer que contendrá los datos leídos. |
| *n_bytes* | Longitud del buffer en bytes, indica además el tamaño máximo en bytes de los datos a leer, pues n_bytes debe ser como máximo igual al tamaño de buf. |

La función devuelve -1 en caso de error y el número de bytes leídos, que pueden ser 0, si tiene éxito la llamada. Generalmente 0 indica el final de los datos en el socket.

## recv.

*#include <sys/types.h>*
*#include <sys/socket.h>*

*int recv(int s, void *buf, int lon, int flags);*

La función *recv* lee datos del socket especificado por *s*. Sus parámetros son:

| | |
|---|---|
| *s* | Descriptor del socket creado con anterioridad. |
| *buf* | Buffer que contendrá los datos leídos |
| *lon* | Longitud del buffer en bytes, indica además el tamaño máximo en bytes de los datos a leer, pues n_bytes debe ser como máximo igual al tamaño de buf. |
| *flags* | Opciones de recepción, generalmente valor 0. |

La función devuelve -1 en caso de error y el número de bytes leídos, que pueden ser 0, si tiene éxito la llamada. Generalmente 0 indica el final de los datos en el socket.

## recvfrom.

*#include <sys/types.h>*
*#include <sys/socket.h>*

*int recv(int s, void *buf, int lon, int flags, struct sockaddr *desde, int *londesde);*

La función *recvfrom* lee datos del socket especificado por *s*. Sus argumentos son:

| | |
|---|---|
| *s* | Descriptor del socket creado con anterioridad. |
| *buf* | Buffer que contendrá los datos leídos |
| *lon* | Longitud del buffer en bytes, indica además el tamaño máximo en bytes de los datos a leer, pues n_bytes debe ser como máximo igual al tamaño de buf. |
| *flags* | Opciones de recepción, generalmente valor 0. |
| *desde* | Estructura de datos que contendrá la dirección IP y el puerto desde el que se han recibido los datos leídos. |
| *londesde* | Longitud de la estructura de datos anterior. |

La estructura *sockaddr* no suele ser utilizada, siendo siempre utilizada en su lugar la estructura *sockaddr_in*, explicada con anterioridad.

La función devuelve el valor -1 si error o el número de bytes leidos si su llamada tiene éxito. Generalmente 0 indica el final de los datos en el socket.

## write.

*#include <unistd.h>*

*int write(int fd, void *buf, int num);*

La función *write* escribe hasta *num* bytes de datos al socket especificado por *fd*. Sus parámetros son:

| | |
|---|---|
| *fd* | Descriptor del socket creado con anterioridad. |
| *buf* | Buffer que contiene los datos a escribir. |
| *num* | Número de bytes a escribir en el socket. |

La función devuelve -1 en caso de error y el número de bytes realmente escritos si tiene éxito. Es necesario tener en cuenta que la función no tiene porque poder escribir todos los bytes solicitados en una sola llamada.

## send.

*#include <sys/types.h>*
*#include <sys/socket.h>*

*int send(int s, const void *buf, int num, int flags);*

La función *send* escribe hasta *num* bytes de datos al socket especificado por *s*. Sus parámetros son:

| | |
|---|---|
| *s* | Descriptor del socket creado con anterioridad. |
| *buf* | Buffer que contiene los datos a escribir en el socket. |
| *num* | Número de bytes a escribir en el socket. |
| *flags* | Opciones de envío, generalmente valor 0. |

La función devuelve -1 en caso de error y el número de bytes realmente escritos si tiene éxito. Es necesario tener en cuenta que la función no tiene porque poder escribir todos los bytes solicitados en una sola llamada.

## sendto.

*#include <sys/types.h>*
*#include <sys/socket.h>*

*int sendto(int s, const void *buf, int num, int flags, const struct sockaddr *to, int tolen);*

La función *sendto* escribe hasta *num* bytes de datos mediante el socket especificado por *s*. Sus parámetros son:

| | |
|---|---|
| *s* | Descriptor del socket creado con anterioridad. |
| *buf* | Buffer que contiene los datos a escribir en el socket. |
| *num* | Número de bytes a escribir en el socket. |
| *flags* | Opciones de envío, generalmente valor 0. |
| *to* | Estructura de datos que contiene la dirección IP y el puerto al que se desean escribir los datos. |
| *tolen* | Longitud de la estructura de datos anterior. |

La estructura *sockaddr* no suele ser utilizada, siendo siempre utilizada en su lugar la estructura *sockaddr_in*, explicada con anterioridad.

La función devuelve -1 en caso de error y el número de bytes realmente escritos si tiene éxito. Es necesario tener en cuenta que la función no tiene porque poder escribir todos los bytes solicitados en una sola llamada.

**Conversión entre formatos de representación de direcciones IP.**

## inet_aton.

*#include <sys/socket.h>*
*#include <netinet/in.h>*
*#include <arpa/inet.h>*

*int inet_aton(const char *cp,struct in_addr *inp);*

La función *inet_aton* convierte la dirección de Internet dada por *cp* desde la notación estándar de números y puntos (por ejemplo 147.156.1.1) a la representación binaria en orden de bytes de red y la guarda en la estructura a la que apunta *inp*. Sus parámetros son:

*cp* Cadena de caracteres con la dirección Internet a convertir.
*inp* Estructura que contendrá la dirección convertida.

La estructura *in_addr* ha sido explicada con anterioridad.

La función devuelve 0 en caso de error y un valor distinto de 0 si la dirección proporcionada es válida.

## inet_ntoa.

*#include <sys/socket.h>*
*#include <netinet/in.h>*
*#include <arpa/inet.h>*

*char *inet_ntoa(struct in_addr in);*

La función *inet_ntoa* convierte la dirección de Internet representada en formato binario en orden de bytes de red a una cadena de caracteres en la notación estándar de números y puntos. Sus parámetros son:

*in* Estructura que contiene las dirección a convertir.

La función devuelve un puntero a la cadena de caracteres con la dirección Internet en la notación estándar de números y puntos. Dicho puntero hace referencia a una variable estática que es sobrescrita en cada llamada a la función.

**Conversión entre formatos de representación de datos en el computador y en la red.**

## htonl.

*#include <netinet/in.h>*

*unsigned long int htonl(unsigned long int hostlong);*

La función *htonl* convierte el entero largo (32 bytes) dado por *hostlong* desde el orden de bytes del hosts al orden de bytes de la red.

## htons.

*#include <netinet/in.h>*

*unsigned short int htons(unsigned short int hostshort);*

La función *htons* convierte el entero corto dado (16 bytes) dato por *hostshort* desde el orden de bytes del hosts al orden de bytes de la red.

## ntohl.

*#include <netinet/in.h>*

*unsigned long int ntohl(unsigned long int netlong);*

La función *ntohl* convierte el entero largo (32 bytes) dado por *netlong* desde el orden de bytes de la red al orden de bytes del hosts.

## ntohs.

*#include <netinet/in.h>*

*unsigned short int ntohs(unsigned short int netshort);*

La función ntohs convierte el entero corto (16 bytes) dado por *netshort* desde el orden de bytes de la red al orden de bytes del hosts.