Sockets

Ing Florencia Ferrigno

2016

UTN - FRBA

Que es un socket

- Recordemos que en los sistemas UNIX **todo es un archivo**. Es decir que cuando se realiza cualquier operacion de I/O (input / output) se hace a traves de un **file descriptor**.

Que es un socket

- Recordemos que en los sistemas UNIX **todo es un archivo**. Es decir que cuando se realiza cualquier operacion de I/O (input / output) se hace a traves de un **file descriptor**.
- Que es un file descriptor?

Que es un socket

- Recordemos que en los sistemas UNIX **todo es un archivo**. Es decir que cuando se realiza cualquier operacion de I/O (input / output) se hace a traves de un **file descriptor**.
- Que es un file descriptor? Es simplemente un valor entero asociado a un archivo abierto **pero** un archivo puede ser: un pipe, una FIFO, una conexion de red, un archivo en el disco, la placa de audio,etc.

Que es un socket

- Recordemos que en los sistemas UNIX **todo es un archivo**. Es decir que cuando se realiza cualquier operacion de I/O (input / output) se hace a traves de un **file descriptor**.
- Que es un file descriptor? Es simplemente un valor entero asociado a un archivo abierto **pero** un archivo puede ser: un pipe, una FIFO, una conexion de red, un archivo en el disco, la placa de audio,etc.
- **Socket** es un canal de comunicacion bidireccional entre dos procesos que puede manejarse mediante un **file descriptor**

Tipos de sockets

- Existen varios tipos de sockets: sin embargo nosotros vamos a concentrarnos en los **Internet sockets** que son los que manejan las direcciones de internet definidas por DARPA.

Tipos de sockets

- Existen varios tipos de sockets: sin embargo nosotros vamos a concentrarnos en los **Internet sockets** que son los que manejan las direcciones de internet definidas por DARPA.
- Dentro de los internet sockets existen distintos tipos, nosotros nos vamos a concentrar en solo dos tipos:
  1. **SOCK_DGRAM**
  2. **SOCK_STREAM**

Tipos de sockets: SOCK_DGRAM

- Son para los protocolos que **no estan orientados a la conexion**, no son mensajes secuenciados y aquellos mensajes que son enviados pueden o no llegar a destino. Y en caso de llegar, pueden o no llegar en el mismo orden que fueron enviados.

Tipos de sockets: SOCK_DGRAM

- Son para los protocolos que **no estan orientados a la conexion**, no son mensajes secuenciados y aquellos mensajes que son enviados pueden o no llegar a destino. Y en caso de llegar, pueden o no llegar en el mismo orden que fueron enviados.
- Por que se dice que no estan orientandos a la conexion? Simplemente por que para poder establecer la comunicacion con la otra parte no es necesario tener una conexion abierta.

Tipos de sockets: SOCK_DGRAM

- Son para los protocolos que **no estan orientados a la conexion**, no son mensajes secuenciados y aquellos mensajes que son enviados pueden o no llegar a destino. Y en caso de llegar, pueden o no llegar en el mismo orden que fueron enviados.
- Por que se dice que no estan orientandos a la conexion? Simplemente por que para poder establecer la comunicacion con la otra parte no es necesario tener una conexion abierta.
- El protocolo utilizado para este tipo de sockets es el **UDP (User Datagram Protocol)**.

Tipos de sockets: SOCK_STREAM

- Se dice de los protocolos que estan **orientados a la conexion**. Es decir que para poder establecer una comunicacion es necesario que los dos extremos esten definidos.

Tipos de sockets: SOCK_STREAM

- Se dice de los protocolos que estan **orientados a la conexion**. Es decir que para poder establecer una comunicacion es necesario que los dos extremos esten definidos.
- Este tipo de protocolos cuenta con dos caracteristicas:
  - **Secuenciamiento**: significa que garantiza que los datos arriben de la misma manera en la que fueron enviados.
  - **Control de errores**: es capaz de detectar la corrupcion de datos, descartarlos y disponer de la retransmision de los mismos.
- Como se consigue esto? los **stream sockets** utilizan el protocolo **TCP (Transfer Control Protocol)** que se encarga que los paquetes lleguen de manera secuencial y libre de errores.

Resumen

- **SOCK_STREAM**: Acepta secuecia de caracteres (streams) orientadas a la conexion, secuenciadas, con control de errores y full duplex (cada extremo puede transmitir y recibir datos al mismo tiempo). Protocolo utilizado: **TCP**
- **SOCK_DGRAM**: Acepta mensajes sin conexion, sin orden secuencial, orientado a paquetes de tamaño fijo. Protocolo utilizado: **UDP**

Byte Order

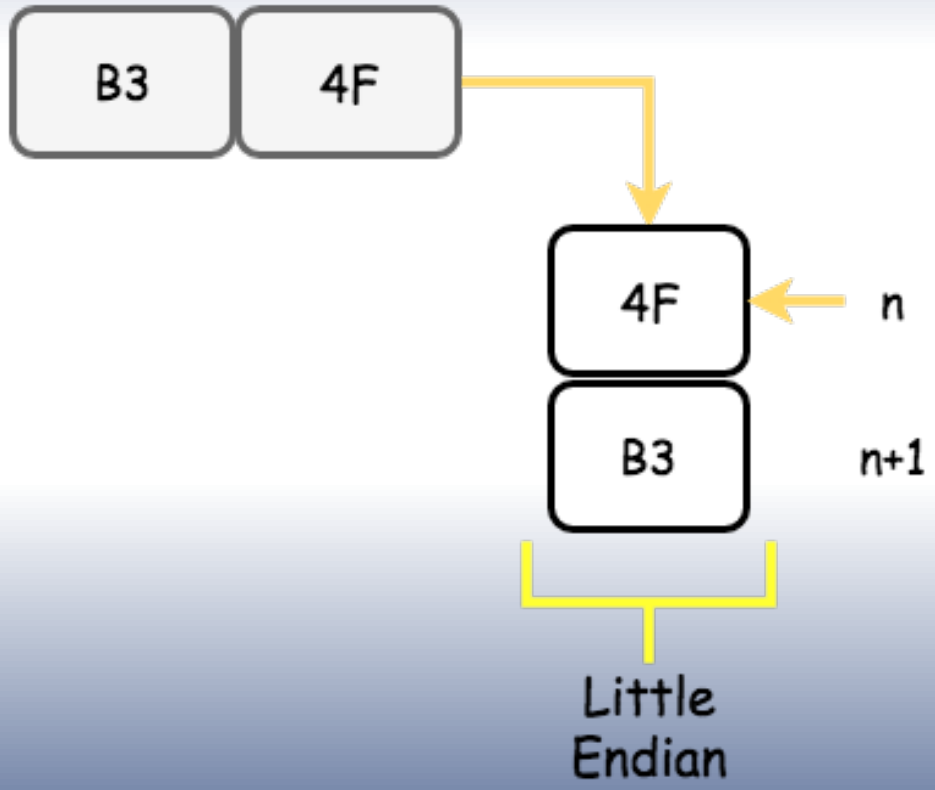
Existen dos maneras de almacenar los datos que contienen mas de un byte de informacion.

Byte Order

Existen dos maneras de almacenar los datos que contienen mas de un byte de informacion.

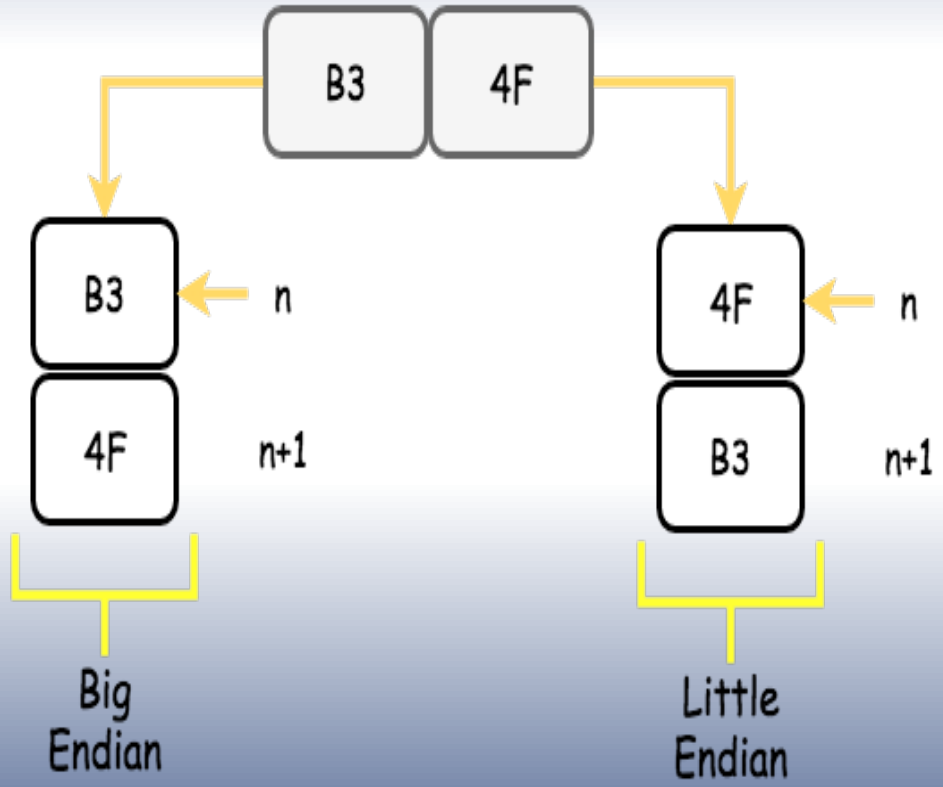
Byte Order

Existen dos maneras de almacenar los datos que contienen mas de un byte de informacion.



Byte Order

Existen dos maneras de almacenar los datos que contienen mas de un byte de informacion.



Byte Order

- Al momento de diseñar los protocolos de red, se adopto como formato Big Endian para ordenar y transmitir su informacion.

Byte Order

- Al momento de diseñar los protocolos de red, se adopto como formato Big Endian para ordenar y transmitir su informacion.
- Como este formato es el adoptado por todas redes tambien se lo conoce como: **Network Byte Order**

Byte Order

- Al momento de diseñar los protocolos de red, se adopto como formato Big Endian para ordenar y transmitir su informacion.
- Como este formato es el adoptado por todas redes tambien se lo conoce como: **Network Byte Order**
- En el mundo de las computadoras, no existe ningun acuerdo. Mientras los procesadores Intel almacenan su informacion en formato Little Endian, los procesadores Motorola lo hacen en Big Endian. El formato (independientemente de cual sea) utilizado por las computadoras es conocido como: **Host Byte Order** y dependera de la arquitectura de donde corra nuestro programa.

UTN.BA

Byte Order

- Al momento de diseñar los protocolos de red, se adopto como formato Big Endian para ordenar y transmitir su informacion.
- Como este formato es el adoptado por todas redes tambien se lo conoce como: **Network Byte Order**
- En el mundo de las computadoras, no existe ningun acuerdo. Mientras los procesadores Intel almacenan su informacion en formato Little Endian, los procesadores Motorola lo hacen en Big Endian. El formato (independientemente de cual sea) utilizado por las computadoras es conocido como: **Host Byte Order** y dependera de la arquitectura de donde corra nuestro programa.
- Por este motivo es necesario poder convertir el formato **Host Byte Order** en **Network Byte Order** antes de transmitirlo.

Berkley API Socket

- Fue diseñada para trabajar con una variedad de protocolos de red y brindar un unica interfaz de programacion para uso de quien desarrollar aplicaciones de red

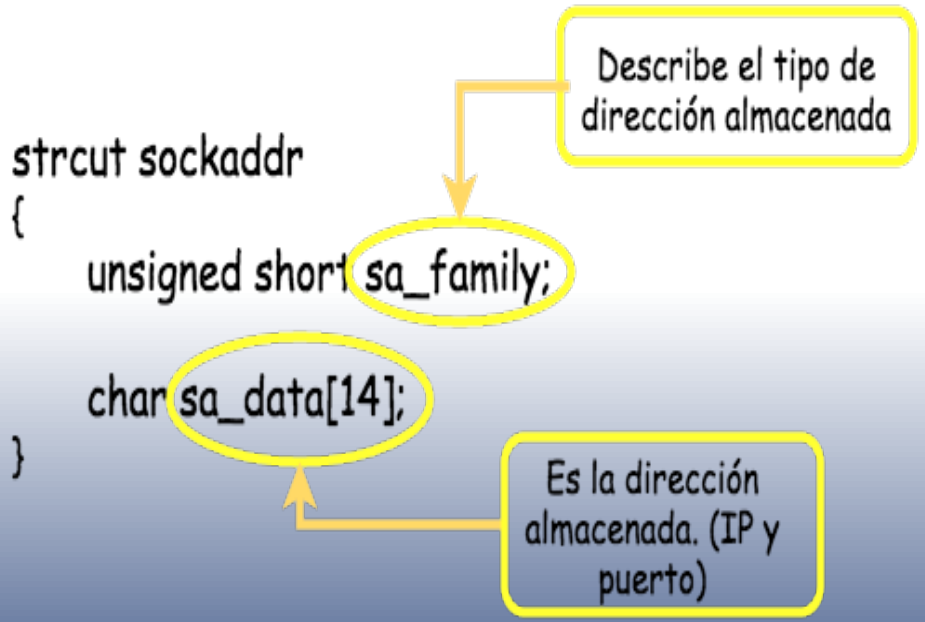
Berkley API Socket

- Fue diseñada para trabajar con una variedad de protocolos de red y brindar un unica interfaz de programacion para uso de quien desarrollar aplicaciones de red
- Existe una estructura fundamental de datos de la API: **sockaddr** que almacena la direcion de red.

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

Berkley API Socket

- Fue diseñada para trabajar con una variedad de protocolos de red y brindar un unica interfaz de programacion para uso de quien desarrollar aplicaciones de red
- Existe una estructura fundamental de datos de la API: **sockaddr** que almacena la direcion de red.



Familia de direcciones

Flia de direcciones	Flia de protocolos	Descripción
AF_UNIX	PF_UNIX	Sockets de entorno UNIX
AF_INET	PF_INET	TCP/IP version 4
AF_AX25	PF_AX25	Protocolo AX25 para radioaficionados
AF_IPX	PF_IPX	Protocolo Novell IPx
AF_APPLETALK	PF_APPLETALK	Protocolo Appletalk DDS

Estructura sockaddr_in

- Con la intención de hacer mas sencillo el trabajo de los programadores se creo otra estructura que llamaron: **sockaddr_in**

Estructura sockaddr_in

- Con la intención de hacer mas sencillo el trabajo de los programadores se creo otra estructura que llamaron: **sockaddr_in**
- Por la manera en que fue definida, vamos a poder declarar un puntero a **sockaddr_in** y castearlo a la **sockaddr** y viceversa.

```
strcut sockaddr_in
{
    short int sin_family;
    unsigned shor int sin_port;
    struct in_addr sin_addr;
    unsigned char sin_zero[8];
}
```

Campos de sockaddr_in

- **sin_family**: se corresponde con el campo **sa_family** de la estructura **sockaddr**

Campos de sockaddr_in

- **sin_family**: se corresponde con el campo **sa_family** de la estructura **sockaddr**
- **sin_port**: es el puerto con el cual se va a trabajar. Debe ser almacenado en **Network Byte Order**

Campos de sockaddr_in

- **sin_family**: se corresponde con el campo **sa_family** de la estructura **sockaddr**
- **sin_port**: es el puerto con el cual se va a trabajar. Debe ser almacenado en **Network Byte Order**
- **sin_addr**: es la direccion IP con la cual se va a trabajar. Debe ser almacenada en **Network Byte Order**

Campos de sockaddr_in

- **sin_family**: se corresponde con el campo **sa_family** de la estructura **sockaddr**
- **sin_port**: es el puerto con el cual se va a trabajar. Debe ser almacenado en **Network Byte Order**
- **sin_addr**: es la direccion IP con la cual se va a trabajar. Debe ser almacenada en **Network Byte Order**
- **sin_zero**:Es un campo de relleno, y debe ser seteado todo a cero. Utilizaremos por ejemplo la funcion:
  - **void * memset(void *s, int c,size_t n)**

Conversiones de byte order

- Cuando nos encontramos ante la necesidad de cargar la IP y el puerto en estas estructuras debemos asegurarnos que sea en **Network Byte Order**. Como lo hacemos?

Conversiones de byte order

- Cuando nos encontramos ante la necesidad de cargar la IP y el puerto en estas estructuras debemos asegurarnos que sea en **Network Byte Order**. Como lo hacemos?
- Contamos con un set de funciones que se van a ocupar de chequear si es necesario hacer la conversion, de esta manera nuestro codigo sera portable a cualquier plataforma.

Conversiones de byte order

- Cuando nos encontramos ante la necesidad de cargar la IP y el puerto en estas estructuras debemos asegurarnos que sea en **Network Byte Order**. Como lo hacemos?
- Contamos con un set de funciones que se van a ocupar de chequear si es necesario hacer la conversion, de esta manera nuestro codigo sera portable a cualquier plataforma.
- Hay dos tipos de datos que podremos convertir: **short** y **long**, las funciones que vamos a ver tambien trabajan con las variaciones **unsigned** de estos tipos de datos.

Conversiones de byte order

- Cuando nos encontramos ante la necesidad de cargar la IP y el puerto en estas estructuras debemos asegurarnos que sea en **Network Byte Order**. Como lo hacemos?
- Contamos con un set de funciones que se van a ocupar de chequear si es necesario hacer la conversion, de esta manera nuestro codigo sera portable a cualquier plataforma.
- Hay dos tipos de datos que podremos convertir: **short** y **long**, las funciones que vamos a ver tambien trabajan con las variaciones **unsigned** de estos tipos de datos.
- Las conversiones de **Host Byte Order** a **Network Byte Order** seran a traves de funciones cuyo nombre comienzan con **h** -por host- seguida de un **to**, luego **n** -por network- y por utlimo **s** -por short-. Entonces el nombre de la funcion sera: **htons()**

Funciones de conversion

- **htons()**: Host to Network short
- **htonl()**: Host to Network long
- **ntohs()**: Network to Host short
- **ntohl()**: Network to Host short

Recordar siempre poner los bytes en Network Byte Order ANTES de presentarlos en la red

Direcciones IP

- El formato de las direcciones IP con el cual estamos familiarizados es: **10.12.110.57**. Esta direccion vamos a tener que almacenarla en el campo **sin_addr** de la structura **sockaddr_in**

Direcciones IP

- El formato de las direcciones IP con el cual estamos familiarizados es: **10.12.110.57**. Esta direccion vamos a tener que almacenarla en el campo **sin_addr** de la structura **sockaddr_in**
- Sin embargo dicho campo es otra estructura del tipo **in_addr**. Que hay dentro de esa estructura? Simplemente un campo del tipo:
  - uint32_t s_addr

Direcciones IP

- El formato de las direcciones IP con el cual estamos familiarizados es: **10.12.110.57**. Esta direccion vamos a tener que almacenarla en el campo **sin_addr** de la structura **sockaddr_in**
- Sin embargo dicho campo es otra estructura del tipo **in_addr**. Que hay dentro de esa estructura? Simplemente un campo del tipo:
  - uint32_t s_addr
- En dicho campo es donde vamos a almacenar la direccion IP. Sin embargo vamos a tener que convertir nuestra cadena de caracteres compuesta por numeros y puntos en un unsigned long: **inet_addr(char *)**

inet_addr()

- Recibe un string que representa la direccion IP
- Devuelve dicha direccion en **Network Byte Order**, por lo tanto no sera necesario llamar a la funcion **htonl()**
- Si la direccion que recibio como argumento no es una direccion IP valida, retornara: **INADDR_NONE** que generalmente es un valor de **-1**
- Hay que tener especial cuidado con este valor de retorno, ya que el valor **-1** corresponde a la direccion IP 255.255.255.255. Direccion IP de broadcast, por lo tanto si no se desea transmitir un mensaje a "todo el mundo", sera necesario hacer el control de errores pertinente.

Direcciones IP

- Supongamos que definimos la siguiente estructura donde vamos a almacenar la direccion IP en cuestion: **struct sockaddr_in ina;**

Direcciones IP

- Supongamos que definimos la siguiente estructura donde vamos a almacenar la direccion IP en cuestion: **struct sockaddr_in ina;**
- Mediante la funcion encargada de convertir un string en un unsigned long en Network Byte Order:
- **ina.sin_addr.s_addr = inet_addr("10.12.110.57")**

Direcciones IP

- Otra forma de convertir una cadena de numeros y puntos a formato network es mediante:
  - int inet_aton(const char *cp, struct in_addr *inp);
  - Convierte la direccion IP apuntada por **cp** en formato network y la almacena en **inp**
  - En caso de no ser una direccion valida retorna **cero**

```
struct sock_addr my_addr;
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(MYPORT);
inet_aton("10.12.110.57",&(my_addr.sin_addr));
memset(&(my_addr.sin_zero),'\0',8);
```

Direcciones IP

- Que pasa ahora, si recibimos una direccion IP a traves de la red (network byte order) y queremos guardarla o mostrarla en pantalla de modo mas legible, es decir como una cadena de numeros y puntos

Direcciones IP

- Que pasa ahora, si recibimos una direccion IP a traves de la red (network byte order) y queremos guardarla o mostrarla en pantalla de modo mas legible, es decir como una cadena de numeros y puntos
- Existe otra funcion que hace justamente el camino inverso:
  - char * inet_ntoa(struct in_addr in);
  - convierte la direccion IP pasada como argumento (en network byte order), a una cadena de caracteres
  - **ntoa** es por network to ascii
  - NOTA: Esta funcion toma como argumento una **struct in_addr**, no un long. La funcion retorna un puntero a char, quien apunta a un vector estatico global. Por lo tanto cada vez que se llame a esta funcion, la misma sobreescribira la ultima direccion IP que se haya pedido. Si necesitamos guardar dicha direccion, entonces debemos hacer uso de las funciones de string para almacenarla en otra variable (strcpy)

socket()

- **int socket(int domain, int type, int protocol)**
- Retorno: file descriptor del socket que se acaba de abrir. En caso de error retornara **-1** y la variable **errno** tendra almacenado el error, el cual podremos ver mediante la funcion **perror()**
- Argumentos:
  - **domain**: AF_INET
  - **type**: SOCK_DGRAM o SOCK_STREAM
  - **protocol**: siempre en 0 para que el socket decida cual es el mejor protocolo basandose en el tipo seleccionado.
- **Headers**:
  - **sys/types.h**
  - **sys/socket.h**

bind()

- Una vez que tenemos el descriptor del socket, vamos a necesitar asociar al mismo a un puerto en nuestra maquina.

bind()

- Una vez que tenemos el descriptor del socket, vamos a necesitar asociar al mismo a un puerto en nuestra maquina.
- Esto solo sera necesario si nosotros vamos a desarrollar un servidor, es decir nos vamos a quedar *escuchando* a ver si recibimos mensajes

bind()

- Una vez que tenemos el descriptor del socket, vamos a necesitar asociar al mismo a un puerto en nuestra maquina.
- Esto solo sera necesario si nosotros vamos a desarrollar un servidor, es decir nos vamos a quedar *escuchando* a ver si recibimos mensajes
- El numero de puerto sera usado por el kernel para poder asociar los paquetes entrantes con un determinado proceso y su socket descriptor.
- **int bind(int sockd, struct sockaddr *myaddr, int addrlen)**
  - Argumentos:
    - **sockfd**: descriptor del socket
    - **myaddr**: puntero a la estructura que contiene la informacion sobre la direccion, puerto e IP.
    - **addrlen**: puede ser seteado a *sizeof(struct sockaddr)*
  - Retorno: Devolvera -1 en caso de error y seteara la variable errno al valor de error.
  - Headers: los mismos que para la funcion socket

Como obtener nuestra IP

- En el caso de estar desarrollando un proceso cliente, tanto el puerto y la direccion IP propios no son los datos mas relevantes de nuestro desarrollo.

Como obtener nuestra IP

- En el caso de estar desarrollando un proceso cliente, tanto el puerto y la direccion IP propios no son los datos mas relevantes de nuestro desarrollo.
- Sin embargo necesitamos cargar la estructura que represente nuestra direccion IP y nuestro puerto. Como hacemos eso?

Como obtener nuestra IP

- En el caso de estar desarrollando un proceso cliente, tanto el puerto y la direccion IP propios no son los datos mas relevantes de nuestro desarrollo.
- Sin embargo necesitamos cargar la estructura que represente nuestra direccion IP y nuestro puerto. Como hacemos eso?
- La manera de obtener nuestra IP sera:
  - **my_addr.sin_port = 0**: Al asignarle cero al puerto, estamos indicandole al kernel que elija algun puerto que tenga libre para nuestro proceso.
  - **my_addr.sin_addr.s_addr = INADDR_ANY**: esta etiqueta no es mas que ceros. Lo que le estamos indicando al kernel es que almacene ahi la direccion IP de la maquina donde esta corriendo nuestro proceso.

Numero de port

- Todos los valores menores a 1024 son numeros reservados de puertos, tambien conocidos como **well-known ports**
- Cualquier valor por encima de los 1024 sera un valor valido para ser utilizados en nuestros desarrollos, hasta el valor 65535

Numero de port

- Todos los valores menores a 1024 son numeros reservados de puertos, tambien conocidos como **well-known ports**
- Cualquier valor por encima de los 1024 sera un valor valido para ser utilizados en nuestros desarrollos, hasta el valor 65535
- Puede ocurrir que al correr nuestra aplicacion, al hacer la llamada a **bind()** falle, y muestre este mensaje: **"Address already in use"**.
- Habra momentos que no hara falta hacer una llamada a esta funcion. Si nos estamos conectando a una maquina remota, no nos importara que puerto local estamos usando.

connect()

- **int connect(int sockfd, struct sockaddr * serv_addr, int addrlen)**

connect()

- **int connect(int sockfd, struct sockaddr * serv_addr, int addrlen)**
- Argumentos:
  - **sockfd**:socket descriptor
  - **serv_addr**: es la estructura que contiene la direccion IP y puerto destino al cual nos queremos conectar
  - **addrlen**: sizeof(struct sockaddr)
- Retorno: -1 en caso de error
- Headers:
  - **sys/types.h**
  - **sys/socket.h**

connect()

- **int connect(int sockfd, struct sockaddr * serv_addr, int addrlen)**
- Argumentos:
  - **sockfd**:socket descriptor
  - **serv_addr**: es la estructura que contiene la direccion IP y puerto destino al cual nos queremos conectar
  - **addrlen**: sizeof(struct sockaddr)
- Retorno: -1 en caso de error
- Headers:
  - **sys/types.h**
  - **sys/socket.h**

NOTA:Cuando usamos connect, no necesitamos hacer bind(). La funcion bind() sera utilizada por aquellos desarrollos que oficiaran de server. Mientras que connect() sera la llamada que deban hacer los desarrollos en modo cliente. Al hacer connect(), sera el kernel quien nos asigne un puerto libre

listen()

- Si tenemos que hacer un desarrollo donde nuestra aplicacion es quien debe esperar que se conecten a ella, la secuencia que vamos a seguir sera:

listen()

- Si tenemos que hacer un desarrollo donde nuestra aplicacion es quien debe esperar que se conecten a ella, la secuencia que vamos a seguir sera:
  - bind()
  - listen()
  - accept()
- **int listen(int sockfd, int backlog)**
  - sockfd: socket descriptor
  - backlog: cantidad de conexiones permitidas en la cola de conexiones entrantes.
  - Valor de retorno: -1 en caso de error

accept()

- Desde una mquina remota se intentaran conectar a nuestro proceso en el puerto al cual estamos haciendo **listen()**. Estas conexiones entrantes seran encoladas a la espera de una aceptacion
- Al hacer **accept()** se toma esa conexion pendiente y se obtendra un nuevo socket para usar con esta nueva conexion, mientras el socket original queda escuchando por nuevas conexiones.

accept()

- **int accept(int sockfd, void * addr, int * addrlen)**
  - sockfd: socket descriptor
  - addr: puntero a la estructura **sockaddr_in**, donde se guardara la informacion de la conexion entrante (IP y puerto del cliente remoto)
  - addrlen: sizeof(struct sockaddr_in)
  - Valor de retorno: devolvera un valor entero que representa el socket descriptor nuevo que relaciona ambos puntos. En caso de error devolvera -1

close()

- **close(int sockfd)**
- Una vez realizada esta llamada, el socket queda cerrado y por lo tanto no se podran recibir ni enviar mensajes.
- Quien quiera enviarnos algo una vez cerrado el socket recibira un mensaje de error.

send() - SOCK_STREAM

- **int send(int sockfd, const void *msg, int len, int flags);**
  - **sockfd**: socket descriptor
  - **msg**: puntero al inicio del mensaje que se desea enviar
  - **len**: longitud del mensaje a enviar
  - **flags**: simplemente seteamos este valor a cero
  - **Valor de retorno**: cantidad de bytes enviados. Este valor puede ser menor que la cantidad de bytes que el mensaje contiene, en este caso sera responsabilidad del programador enviar el resto del mensaje. Si el mensaje es menor a 1K se envia todo el mensaje. En caso de error retorna -1

recv() - SOCK_STREAM

- **int recv(int sockfd, const void *msg, int len, int flags);**
  - **sockfd**: socket descriptor
  - **msg**: puntero al inicio del buffer donde se va a almacenar el mensaje
  - **len**: tamaño del buffer
  - **flags**: simplemente seteamos este valor a cero
  - **Valor de retorno**: Cantidad de bytes recibidos, -1 en caso de error. Si el valor de retorno es 0, significa que del otro lado cerraron la conexion.

sendto() - SOCK_DGRAM

- **int sendto(int sockfd, const void* msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);**
  - **struct sockaddr *to**: puntero a la estructura que contiene la direccion IP y puerto del remoto al cual nos queremos conectar
  - **tolen**: tamaño de dicha estructura
  - **Valor de retorno**: cantidad de bytes enviados, -1 en caso de error.

recvfrom() - SOCK_DGRAM

- **int recvfrom(int sockfd, const void* msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);**
  - **struct sockaddr *to**: puntero a la estructura que contiene la direccion IP y puerto del remoto al cual nos queremos conectar
  - **tolen**: tamaño de dicha estructura
  - **Valor de retorno**: cantidad de bytes enviados, -1 en caso de error.

Domain Name Service -DNS-

- Es la manera de dar una forma mas amigable a la direccion IP de una maquina.
- Por ejemplo si nosotros ponemos *google.com*, sera el DNS el encargado de traducir ese nombre a una direccion IP para que nosotros podamos acceder a dicho servicio.
- Nosotros vamos a poder traducir un nombre a una direccion IP mediante:
  - **struct hostent * gethostbyname(const char*name)**
  - Esta funcion retorna un puntero a esta estructura o NULL en caso de error. Pero NO queda en errno el mensaje de error.

struct hostent

```
struct hostent {
char * h_name; //nombre del host
char ** h_aliases; //vector con los nombres alternativos
int h_addrtype //tipo de direccion. Generalmente AF_INET
int h_length; //longitud de la direccion en bytes
char **h_addr_list;//lista de las direcciones del host. la primer direccion
puede ser accedida como h_addr
}
```