TCP/IP Y SOCKETS

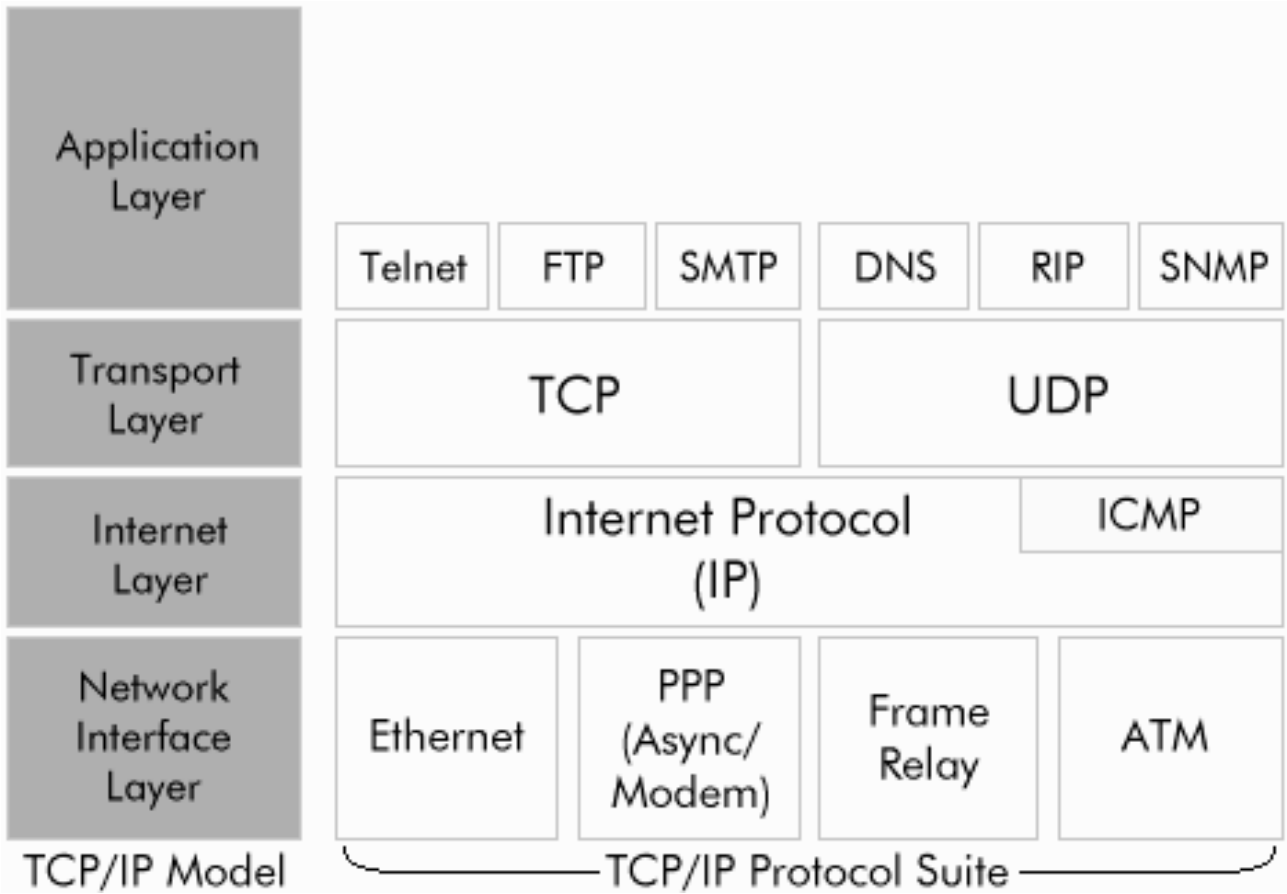
Héctor Menéndez

AIDA Research Group
Computer Science Department
Universidad Autónoma de Madrid

7 de febrero de 2013

Index

TCP/IP


Application Layer
Transport Layer
Internet Layer
Network Interface Layer
Telnet
FTP
SMTP
DNS
RIP
SNMP
TCP
UDP
Internet Protocol (IP)
ICMP
Ethernet
PPP (Async/ Modem)
Frame Relay
ATM
TCP/IP Model
TCP/IP Protocol Suite

TCP/IP: Descripción

- Para poder realizar una conexión se necesita la IP y el puerto.
- Las posibles rutas son:

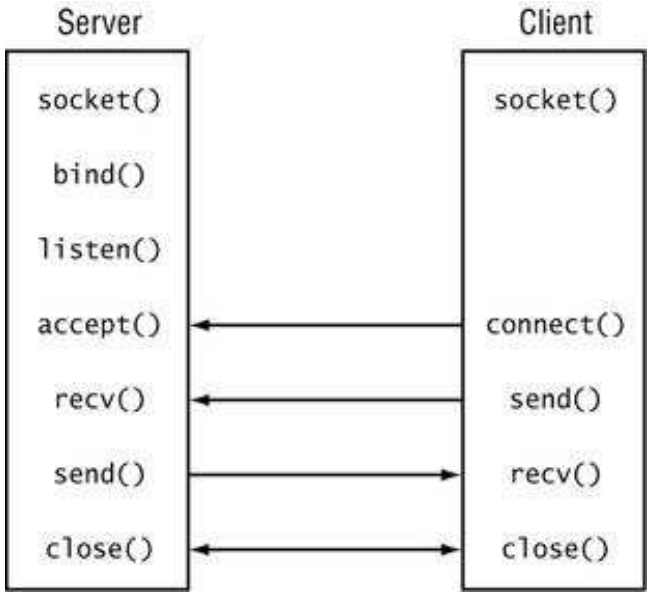
$$[0-255].[0-255].[0-255].[0-255] : [0-65535]$$

- Para poder realizar estas conexiones, el sistema utiliza un recurso conocido como **sockets**.

Sockets: Descripción

- Interfaz común entre todos los protocolos disponibles de un Sistema Operativo Linux/UNIX.
- Se utilizan para enviar y recibir información a través de la red.

Conexión por Sockets



Conexión por Sockets: Cliente

- `socket()`: Crea las colas de envío y recepción de datos.
- `connect()`: Conecta el socket a un destino.
- `send()`: Envía la información.
- `recv()`: Recibe la información.
- `close()`: Cierra la conexión.

Conexión por Sockets: Servidor

- `socket()`: Crea las colas de envío y recepción de datos.
- `bind()`: Conecta las colas a la red.
- `listen()`: Se queda escuchando a la espera de clientes.
- `accept()`: Acepta la conexión de un cliente.
- `send()`: Envía la información.
- `recv()`: Recibe la información.
- `close()`: Cierra la conexión.

La función `socket()`

Listing 1: Definitición de la función `socket()`

```
#include<sys/socket.h>
#include<resolv.h>
int socket(int domain, int type, int protocol)
```

- Hace eficaz al receptor de mensajes y comienza el proceso completo de envío y recepción de mensajes de otra computadora.
- Crea un descriptor para acceder a las computadoras de la red (al igual que `open()` crea un descriptor para acceder a los archivos y dispositivos de nuestro Sistema Operativo).
- Necesita información que determine a qué capa se quiere acceder.

La función `socket()`: Domain

Valor	Uso	Ejemplo
`PF_INET`	Protocolos de Internet IPv4	TCP/IP.
`PF_LOCAL`	Canales con nombres locales	registro sistema, cola impresión.
`PF_IPX`	Orientado al paquete no a la conexión	Servidores Novell Netware.
`PF_INET6`	IPv6	TCP/IP.

La función `socket()`: Type

Valor	Uso	Ejemplo
`SOCK_STREAM`	Fiable, flujo de datos secuencial	Transmisión TCP.
`SOCK_RDM`	Fiable, datos en paquetes	Obsoleto.
`SOCK_DGRAM`	No fiable, datos en paquete	UDP.
`SOCK_RAW`	No fiable	Paquetes de bajo nivel.

La función `socket()`: Protocol

Valor	Uso
`IPPROTO_TCP`	Protocolo TCP
`IPPROTO_UDP`	Protocolo UDP
`IPPROTO_SCTP`	Protocolo STCP
`0` (default)	Protocolo por defecto con la combinación domain-type elegida

La función socket(): Ejemplo

Listing 2: Definición de la función socket()

```
int sd;
sd=socket(PF_INET,SOCK_STREAM,0)
```

- La función socket() devuelve un descriptor (sd) o -1 en caso de error.
- Cuando devuelve error, coloca el código en errno. Lo valores pueden ser, entre otros:
  - EPROTONOSUPPORT: Protocolo no soportado.
  - EACCES: Permiso denegado.
  - EINVAL: Protocolo desconocido.
- Crea un extremo de la comunicación.

La función `bind()`

Listing 3: Definición de la función `bind()`

```
#include<sys/socket.h>
#include<sys/types.h>
#include<resolv.h>
int bind(int sockfd, const struct sockaddr *addr,
 socklen_t addrlen);
```

- La función `bind()` devuelve 0 si todo sale bien o `-1` en caso de error.
- Cuando devuelve error, coloca el código en `errno`. Lo valores pueden ser, entre otros:
  - `EADDRINUSE`: La dirección dada está actualmente en uso.
  - `EACCES`: Permiso denegado.
  - `EINVAL`: La longitud de `addrlen` es errónea.

La función `bind()`

Listing 3: Definición de la función `bind()`

```
#include<sys/socket.h>
#include<sys/types.h>
#include<resolv.h>
int bind(int sockfd, const struct sockaddr *addr,
 socklen_t addrlen);
```

- Cuando un socket es creado con `socket()`, existe en un espacio de nombrs (o familia de direcciones) pero no tiene una dirección asignada. Esta función asigna la dirección especificada por `addr` al socket referido por el descriptor `sockfd`. `addrlen` especifica el tamaño en bytes de la estructura de la dirección a la que apunta `addr`.
- Esta operación tradicionalmente se denomina “asignar un nombre a un socket”.

La función connect()

Listing 4: Definición de la función connect()

```
#include<sys/socket.h>
#include<sys/types.h>
#include<resolv.h>
int connect(int sockfd, const struct sockaddr *addr,
  socklen_t addrlen);
```

- La función connect() devuelve 0 si todo sale bien o -1 en caso de error.
- Cuando devuelve error, coloca el código en errno. Lo valores pueden ser, entre otros:
  - EAFNOSUPPORT: La dirección dada no está soportada.
  - EACCES: Permiso denegado.
  - EISCONN: El socket ya está conectado.

La función connect()

Listing 4: Definición de la función connect()

```
#include<sys/socket.h>
#include<sys/types.h>
#include<resolv.h>
int connect(int sockfd, const struct sockaddr *addr,
 socklen_t addrlen);
```

- Conecta el socket referenciado en sockfd a la dirección especificada por addr. El valor addrlen especifica el tamaño de addr:
  - Identifica el destino IP.
  - Habilita el canal.
  - Informa al destino de dónde debe enviar las respuestas.
- Realiza bind de forma implícita.

La estrutura sockaddr

Listing 5: Definición de la estructura sockaddr

```
struct sockaddr {
    sa_family_t  sa_family;
    char         sa_data[14];
}
```

- sa_family: Familia utilizada.
- sa_data: resto de la información.

La estrutura sockaddr_in

Listing 6: Definición de la estructura sockaddr_in

```
struct sockaddr_in {
    sa_family_t sin_family;
    unsigned short int sin_port;
    struct in_addr sin_addr;
    unsigned char __pad[];
}
```

- sin_family: Familia utilizada.
- sin_port: Puerto utilizado.
- in_addr: Dirección utilizada.
- __pad[]: Resto de información.

Funciones para la ordenación de bytes

Listing 7: Definición de las funciones de conversión

```
#include <arpa/inet.h>
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

- Cuando se trabaja con envíos de bytes en red, es necesarios ponerlos de forma apropiada. Hay dos tipos de orden: *big-endian* y *little-endian*. El orden en red es big-endian, el del sistema depende del procesador.
- `htonl`, `htons`, `ntohl`, `ntohs`: convierten el orden en bytes de los valores de orden host a orden de red. Las dos primeras de host a red, las demás alrevés. las acabadas en s convierten 16 bytes, las acabadas en l 32.

Funciones para la ordenación de bytes

Listing 8: Definición de la estructura `sockaddr_in`

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int inet_aton(const char *cp, struct in_addr *inp);
char *inet_ntoa(struct in_addr in);
```

- `inet_aton`: transforma la notación punto (###.###.###.###) al binario ordenado en red. Devuelve 0 si falla y distinto de cero si la dirección es válida.
- `inet_ntoa`: transforma un binario IP ordenado en red a un ASCII en notación punto decimal.

La estrutura `listen()`

Listing 9: Definición de la estructura `listen()`

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

- En caso de éxito, se devuelve cero. En caso de error, se devuelve -1 y se pone en errno un valor apropiado.
  - `EADDRINUSE`: Otro conector ya se encuentra escuchando en el mismo puerto.
  - `ENOTSOCK`: El argumento `sockfd` no es un conector.
  - `EBADF`: El argumento `sockfd` no es un descriptor válido.

La estrutura `listen()`

Listing 9: Definición de la estructura `listen()`

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

- Espera conexiones en un conector (socket).
- El parámetro `backlog` define la longitud máxima a la que puede llegar la cola de conexiones pendientes.
- Si una petición de conexión llega estando la cola llena, el cliente puede recibir un error con una indicación de ECONNREFUSED o, si el protocolo subyacente acepta retransmisiones, la petición puede no ser tenida en cuenta, de forma que un reintento tenga éxito.

La estrutura accept()

Listing 10: Definición de la estructura accept()

```
#include <sys/types.h>
#include <sys/socket.h>
#include<resolv.h>
int accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen);
```

- La llamada devuelve -1 ante un error. Si tiene éxito, devuelve un entero no negativo que es el descriptor del conector aceptado. Los valores de errno pueden ser, entre otros:
  - `EINVAL`: El conector no está escuchando conexiones.
  - `ENOTSOCK`: El descriptor referencia a un fichero, no a un conector.
  - `EAGAIN`: El conector está marcado como no-bloqueante y no hay conexiones que aceptar.

La estrutura accept()

Listing 10: Definición de la estructura accept()

```
#include <sys/types.h>
#include <sys/socket.h>
#include<resolv.h>
int accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen);
```

- Acepta una conexión sobre un conector (socket).
- La función accept se usa con conectores orientados a conexión (SOCK_STREAM, SOCK_SEQPACKET y SOCK_RDM).
- Extrae la primera petición de conexión de la cola de conexiones pendientes, le asocia un nuevo conector con, practicamente, las misma propiedades que sockfd y reserva un nuevo descriptor de fichero para el conector, el cuál es el valor devuelto por la llamada.

La estrutura accept()

Listing 10: Definición de la estructura accept()

```
#include <sys/types.h>
#include <sys/socket.h>
#include<resolv.h>
int accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen);
```

- Si no hay conexiones pendientes en la cola y el conector no está marcado como "no bloqueante", accept bloqueará al invocador hasta que se presente una conexión. Si el conector está marcado como no bloqueante y no hay conexiones pendientes en la cola, accept devolverá `EAGAIN`.

La estrutura `send()`

Listing 11: Definición de la estructura `send()`

```
#include <sys/types.h>
#include <sys/socket.h>
 ssize_t send(int sockfd, const void *msg, size_t len,
  int flags);
```

- La llamada devuelven el numero de caracteres enviados, o -1 si ha ocurrido un error.
  - `EINVAL`: Se ha pasado un argumento inválido.
  - `ENOTSOCK`: El descriptor no es un conector.
  - `EAGAIN`: El conector está marcado como no-bloqueante y y la operación solicitada lo bloquearía.

La estrutura `send()`

Listing 11: Definición de la estructura `send()`

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t send(int sockfd, const void *msg, size_t len,
 int flags);
```

- Envía un mensaje de un conector (socket).
- Sólo puede ser usado cuando un conector está en un estado connected.
- La longitud del mensaje viene dada por len. Si el mensaje es demasiado largo para pasar automáticamente a través del protocolo inferior, se devuelve el error `EMSGSIZE` y el mensaje no es transmitido.

La estrutura send()

Listing 11: Definición de la estructura send()

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t send(int sockfd, const void *msg, size_t len,
 int flags);
```

- El parámetro flags es una palabra de opciones y puede contener las siguientes opciones:
  - MSG_OOB: Enviar datos fuera de orden(out-of-band).
  - MSG_DONTROUTE: No usar un "gateway" para enviar el paquete, enviar sólo a los ordenadores que se encuentren en redes conectadas directamente.
  - MSG_CONFIRM: Le dice a la capa de enlace que se produjo el proceso de redirección: tienes una confirmación positiva del otro lado. Sólo válida para conectores SOCK_DGRAM y SOCK_RAW y actualmente sólo está implementada en IPv4 e IPv6.

La estrutura `recv()`

Listing 12: Definición de la estructura `recv()`

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buf, size_t len,
 int flags);
```

- devuelve el número de bytes recibidos o -1 en caso de error. El valor de retorno es 0 cuando el enlace se ha cerrado.
  - `EBADF`: sockfd es un descriptor inválido.
  - `EFAULT`: El buffer apunta a una dirección fuera del que corresponde al proceso.
  - `EINVAL`: Se han pasado argumentos inválidos.

La estrutura `recv()`

Listing 12: Definición de la estructura `recv()`

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buf, size_t len,
 int flags);
```

- Recibe un mensaje desde un socket. Se usa normalmente cuando el socket está conectado.
- Si no hay datos a recibir en el socket , se bloquea hasta que llegan datos, aunque se puede establecer al socket como no bloqueante.
- `sockfd`: Descriptor socket por donde se recibirán los datos.
- `buf`: Puntero a un buffer donde se almacenarán los datos recibidos.
- `len`: Longitud del buffer buf.

La estrutura `recv()`

Listing 12: Definición de la estructura `recv()`

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buf, size_t len,
 int flags);
```

- El parámetro flags es una palabra de opciones y puede contener las siguientes opciones:
  - `MSG_OOB`: Recibir datos fuera de orden (out-of-band).
  - `MSG_DONTWAIT`: Habilita operaciones no-bloqueantes.

La estrutura `close()`

Listing 13: Definición de la estructura `close()`

```
#include <unistd.h>
int close(int sockfd);
```

- Cierra la conexión.
- Devuelve 0 en caso de éxito y -1 en caso de error. Sitúa el error en errno:
  - `EBADF`: `sockfd` no es un descriptor válido.

Ejemplo

Listing 14: Ejemplo

```
int main(int count, char *argv[])
{       //Invocacion: programa IP_dest puerto_dest mensaje
        int sockfd;
        struct sockaddr_in dest;
        char buffer[MAXBUF];
        // Crear un socket y asignar un numero de puerto
        sockfd = socket(PF_INET, SOCK_STREAM,0);
        bzero(&dest,sizeof(dest));
        dest.sin_family = AF_INET;
        dest.sin_port = htons(atoi(argv[2]));
        inet_aton(argv[1],&dest.sin_addr.s_addr);
        // Conectar a un servidor y enviar la peticion
        if (connect (sockfd,(const struct sockaddr *) &dest, sizeof(dest)) != 0)
                printf("connect()_failed");
        sscanf( argv[3], "%s\n",buffer);
        send(sockfd, buffer,strlen(buffer),0);
        //Vaciar el buffer y leer la respuesta CORTA.
        bzero(buffer,MAXBUF);
        recv(sockfd,buffer,MAXBUF-1,0);
        printf("%s",buffer);
        close(sockfd);
        return 0;
}
```

Para profundizar

- ¿Tienes dudas sobre el funcionamiento de una función?
- Usa `man` y el nombre de la función en tu terminal.

Sockets UDP

- El protocolo UDP no está orientado a conexión por lo que varios detalles de implementación cambian.
- El servidor no se queda escuchando una conexión sino un paquete, así mismo, el cliente no se conecta al servidor, sino que le envío información que puede no llegar.
- Para realizar estas funcionalidades se utilizan las funciones `sendto` y `recvfrom`.

La estrutura sendto()

Listing 15: Definición de la estructura sendto()

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t sendto(int sockfd, const void *buf, size_t len,
  int flags, const struct sockaddr *dest_addr,
  socklen_t addrlen);
```

- Envía un mensaje a un destinatario especificado en `dest_addr` (su longitud viene dada por `addrlen`).
- `sockfd` es el descriptor del socket.
- `buf` es el mensaje cuya longitud viene dada por `len`.
- Devuelve la longitud de la información enviada en caso de éxito y -1 en caso de error. Sitúa el error en errno.
- Tiene bind implícito.

La estrutura `recvfrom()`

Listing 16: Definición de la estructura `recvfrom()`

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recvfrom(int sockfd, void *buf, size_t len,
  int flags, struct sockaddr *src_addr,
  socklen_t *addrlen);
```

- Recibe un mensaje de un destinatario que será especificado en `dest_addr` (su longitud vendrá dada por `addrlen` aunque hay que pasarle la longitud de la estructura original) tras recibir el mensaje.
- `sockfd` es el descriptor del socket.
- `buf` es el lugar donde se escribirá el mensaje cuya longitud viene dada por `len`.
- Devuelve la longitud de la información enviada en caso de éxito y -1 en caso de error. Sitúa el error en errno.

Abrir Socket UDP

Listing 17: Abrir Socket UDP

```
int Abre_Socket_Udp ()
{
        struct sockaddr_in Direccion;
        int Descriptor;

        Descriptor = socket (AF_INET, SOCK_DGRAM, 0);
        Direccion.sin_family = AF_INET;
        Direccion.sin_port = htons(80);
        Direccion.sin_addr.s_addr = htonl(INADDR_ANY);

        bind (Descriptor,(struct sockaddr *)&Direccion,
                        sizeof (Direccion)) == -1)

}
```

Enviar Socket UDP

Listing 18: Enviar Socket UDP

```
int Lee_Socket_Udp (
        int fd, struct sockaddr *Remoto, socklen_t *Longitud_Remoto,
        char *Datos, int Longitud_Datos)
{
        int Leido = 0;
        int Aux = 0;

        while (Leido < Longitud_Datos)
        {
                Aux = recvfrom (fd, Datos + Leido, Longitud_Datos - Leido, 0,
                        Remoto, Longitud_Remoto);

                if (Aux > 0)
                        Leido = Leido + Aux;
                else
                {
                        if (Aux == 0)
                                return Leido;
                        if (Aux == -1)
                                return -1;
                }
        }
        return Leido;
}
```

Recibir Socket UDP

Listing 19: Recibir Socket UDP

```
int Escribe_Socket_Udp (int fd, struct sockaddr *Remoto,
        socklen_t Longitud_Remoto, char *Datos, int Longitud_Datos)
{
        int Escrito = 0;
        int Aux = 0;

        while (Escrito < Longitud_Datos)
        {
                Aux = sendto (fd, Datos + Escrito, Longitud_Datos - Escrito, 0,
                        Remoto, Longitud_Remoto);
                if (Aux > 0)
                        Escrito = Escrito + Aux;
                else
                {
                        if (Aux == 0)
                                return Escrito;
                        else
                return -1;
                }
        }
        return Escrito;
}
```