Programación Básica de Sockets en Unix para Novatos

BracaMan

Tabla de contenidos

Introducción

¿Estás tratando de aprender programación en C de sockets? ¿O piensas que es una tarea demaciado difícil? Si es así, debes leer este tutorial básico para aprender las ideas y conceptos básicos y así, empezar a trabajar con sockets. Luego de leer este tutorial no esperes ser un "maestro" en la programación de sockets. Sólo lo serás si practicas y lees mucho.

Diferentes tipos de sockets en Internet

En primer lugar deberé explicar qué es un socket. De un modo muy simple, se puede decir que es una manera de hablar con otra computadora. Para ser más preciso, es una manera de hablar con otras computadoras usando descriptores de archivos estándard de Unix. En Unix, todas las acciones de entrada y salida son desempeñadas escribiendo o leyendo en uno de estos descriptores de archivo, los cuales son simplemente un número entero, asociado a un archivo abierto que puede ser una conexión de red, una terminal, o cualquier otra cosa[1]

Ahora bien, sobre los diferentes tipos de sockets en Internet, hay muchos tipos pero sólo se describirán dos de ellos - Sockets de Flujo (`SOCK_STREAM`) y Sockets de Datagramas (`SOCK_DGRAM`).

Y "¿cual es la diferencia entre estos dos tipos?", podrías preguntarte. He aquí la respuesta:

Sockets de Flujo

Están libres de errores: Si por ejemplo, enviáramos por el socket de flujo tres objetos "A, B, C", llegarán a destino en el mismo orden -- "A, B, C". Estos sockets usan TCP ("Transmission Control Protocol[2]") y es éste protocolo el que nos asegura el orden de los objetos durante la transmisión.

Sockets de Datagramas

Estos usan UDP ("User Datagram Protocol[3]"), y no necesitan de una conexión accesible como los Sockets de Flujo -- se construirá un paquete de datos con información sobre su destino y se lo enviará afuera, sin necesidad de una conexión.

Mucho más podría ser explicado aquí sobre estas dos clases de sockets, pero creo que esto es suficiente como para captar el concepto básico de socket. Entender qué es un socket y algo sobre estos dos tipos de sockets de Internet es un buen comienzo, pero lo más importante será saber cómo trabajar con ellos. Esto se apredерá en las próximas secciones.

Estructuras.

El propósito de esta sección no es enseñar el concepto de *estructuras* en programación, sino enseñar cómo se usan estas en la programación en C de Sockets. Si no sabes lo que es una estructura, mi consejo es leer un buen tutorial de C. Por el momento, digamos simplemente que una estructura es un tipo de dato que es un conglomerado,

o sea, que puede contener otros tipos de datos, los cuales son agrupados todos juntos en un único tipo definido por el programador.

Las estructuras son usadas en la programación de sockets para almacenar información sobre direcciones. La primera de ellas es struct sockaddr, la cual contiene información del socket.

```
struct sockaddr
{
   unsigned short sa_family; /* familia de la dirección */
   char sa_data[14]; /* 14 bytes de la dirección del protocolo */
};
```

Pero, existe otra estructura, struct sockaddr_in, la cual nos ayuda a hacer referencia a los elementos del socket.

```
struct sockaddr_in
{
   short int sin_family; /* Familia de la Dirección */
   unsigned short int sin_port; /* Puerto */
   struct in_addr sin_addr; /* Dirección de Internet */
   unsigned char sin_zero[8];
   /* Del mismo tamaño que struct sockaddr */
};
```

Nota

sin_zero puede ser seteada con ceros usando las funciones `memset()` o `bzero()` (Ver los ejemplos más abajo).

La siguiente estructura no es muy usada pero está definida como una unión.

Como se puede ver en los dos ejemplos de abajo (ver la sección de nombre *Un ejemplo de Servidor de Flujos* y la sección de nombre *Un ejemplo de Cliente de Flujos*), cuando se declara, por ejemplo "client" para que sea del tipo sockaddr_in, luego se hace `client.sin_addr = (...)`.

De todos modos, aquí está la estructura:

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

Finalmente, creo que es mejor hablar sobre la estructura hostent. En el ejemplo de Cliente de Flujos (ver la sección de nombre *Un ejemplo de Cliente de Flujos*), se puede ver cómo se usa esta estructura, con la cual obtenemos información del nodo remoto[4].

Aquí se puede ver su definición:

```
struct hostent
{
   char *h_name; /* El nombre oficial del nodo.  */
```

```
   char **h_aliases; /* Lista de Alias.  */
   int h_addrtype; /* Tipo de dirección del nodo.  */
   int h_length; /* Largo de la dirección. */
   char **h_addr_list; /* Lista de direcciones de el nombre del servi-
dor. */
   #define h_addr  h_addr_list[0]  /* Dirección, para la compatibili-
dad con anteriores. */
};
```

Esta estructura está definida en el archivo `netdb.h`.

Al principio, es posible que estas estructuras nos confundan mucho. Sin embargo, luego de empezar a escribir algunas líneas de código, y luego de ver los ejemplos que se incluyen en este tutorial, será mucho más fácil entenderlas. Para ver cómo se pueden usar estas estructuras, recomiendo ver los ejemplos de la sección de nombre *Un ejemplo de Servidor de Flujos* y la sección de nombre *Un ejemplo de Cliente de Flujos*.

Conversiones.

Existen dos tipos de ordenamiento de bytes: *bytes más significativos*, y *bytes menos significativos*. Este es llamado "Ordenamiento de Bytes para Redes [5] ", y hasta algunas máquinas utilizan este tipo de ordenamiento para guardar sus datos, internamente.

Existen dos tipos a los cuales seremos capaces de convertir: short y long [6]. Imaginémonos que se quiere convertir una variable larga de Ordenación de Bytes para Nodos a una de Ordenación de Bytes para Redes. ¿Qué haríamos? Existe una función llamada `htonl()` que haría exactamente esta conversión. Las siguientes funciónes son análogas a esta y se encargan de hacer este tipo de conversiones:

- `htons()` -> "Nodo a variable corta de Red"
- `htonl()` -> "Nodo a variable larga de Red"
- `ntohs()` -> "Red a variable corta de Nodo"
- `ntohl()` -> "Red a variable larga de Nodo"

Estarás pensando ahora para qué necesitamos todas estas funciones, y el porqué de estos ordenamientos. Bien, cuando se termine de leer este documento todas estas dudas se aclararán (aunque sea un poco). Todo lo que necesitarás es leer y precticar mucho.

Una cosa importante, es que `sin_addr` y `sin_port`, de la estructura sockaddr_in, deben ser del tipo Ordenación de Bytes para Redes. Se verá, en los ejemplos, las funciones que aquí se describen para realizar estas conversiones, y a ese punto se entenderán mucho mejor.

Direcciones IP

En C, existen algunas funciones que nos ayudarán a manipular *direcciones IP*. En esta sección se hablará de las funciones `inet_addr()` y `inet_ntoa()`.

Por un lado, la función `inet_addr()` convierte una dirección IP en un entero largo sin signo (unsigned long int), por ejemplo:

```
(...)

dest.sin_addr.s_addr = inet_addr("195.65.36.12");

(...)

/*Recordar que esto sería así, siempre que tengamos una
estructura "dest" del tipo sockaddr_in*/
```

Por otro lado, `inet_ntoa()` convierte a una cadena que contiene una dirección IP en un entero largo. Por ejemplo:

```
(...)

char *ip;

ip=inet_ntoa(dest.sin_addr);

printf("Address is: %s\n",ip);

(...)
```

Se deberá recordar también que la función `inet_addr()` devuelve la dirección en formato de Ordenación de Bytes para Redes por lo que no necesitaremos llamar a `htonl()`.

Funciónes Importantes

En esta sección, (en la cual se nombrarán algunas de las funciones más utilizadas para la programación en C de sockets), se mostrará la sintáxis de la función, las librerías necesarias a incluír para llamar llamarla, y algunos pequeños comentarios. Además de las que se mencionan aquí, existen muchas funciones más, aunque sólo decidí incluír estas. Tal vez sean incluídas en una futura versión de este documento[7]. Nuevamente, para ver ejemplos sobre el uso de estas funciones, se podrán leer la sección de nombre *Un ejemplo de Servidor de Flujos* y la sección de nombre *Un ejemplo de Cliente de Flujos*, en las cuales hay código fuente de un *Cliente de Flujos* y un *Servidor de Flujos*.

`socket()`

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

int socket(int domain,int type,int protocol);
```

Analicemos los argumentos:

- **domain**

  Se podrá establecer como `AF_INET` (para usar los protocolos ARPA de Internet), o como `AF_UNIX` (si se desea crear sockets para la comunicación interna del sistema). Estas son las más usadas, pero no las únicas. Existen muchas más, aunque no se nombrarán aquí.

- **type**

  Aquí se debe especificar la clase de socket que queremos usar (de Flujos o de Datagramas). Las variables que deben aparecer son `SOCK_STREAM` o `SOCK_DGRAM` según querramos usar sockets de Flujo o de Datagramas, respectivamente.

- **protocol**

  Aquí, simplemente se puede establecer el protocolo a 0.

La función `socket()` nos devuelve un descriptor de socket, el cual podremos usar luego para llamadas al sistema. Si nos devuelve *-1*, se ha producido un error (obsérvese que esto puede resultar útil para rutinas de chequeo de errores).

bind()

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

int bind(int fd, struct sockaddr *my_addr,int addrlen);
```

Analicemos los argumentos:

-

fd

Es el descriptor de archivo socket devuelto por la llamada a `socket()`.

-

my_addr

es un puntero a una estructura sockaddr

-

addrlen

contiene la longitud de la estructura sockaddr a la cual apunta el puntero `my_addr`. Se debería establecerla como `sizeof(struct sockaddr)`.

La llamada `bind()` es usada cuando los puertos locales de nuestra máquina están en nuestros planes (usualmente cuando utilizamos la llamada `listen()`). Su función escencial es asociar un socket con un puerto (de nuestra máquina). Análogamente `socket()`, devolverá -1 en caso de error.

Por otro lado podremos dar hacer que nuestra dirección IP y puerto sean elegidos automáticamente:

```
server.sin_port = 0;  /* bind() will choose a random port*/

server.sin_addr.s_addr = INADDR_ANY; /* puts server's IP automatically */
```

Un aspecto importante sobre los puertos y la llamada `bind()` es que todos los puertos menores que 1024 son reservados. Se podrá establecer un puerto, siempre que esté entre 1024 y 65535 (y siempre que no estén siendo usados por otros programas).

connect()

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

int connect(int fd, struct sockaddr *serv_addr, int addrlen);
```

Analicemos los argumentos:

-

fd

Debería setearse como el archivo descriptor del socket, el cual fue devuelto por la llamada a `socket()`.

-

serv_addr

Es un puntero a la estructura sockaddr la cual contiene la dirección IP destino y el puerto.

•

addrlen

Análogamente de lo que pasaba con `bind()`, este argumento debería establecerse como `sizeof(struct sockaddr)`.

La función `connect()` es usada para conectarse a un puerto definido en una dirección IP. Devolverá -a si ocurre algún error.

listen()

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

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

Veamos los argumentos de `listen()`:

•

fd

Es el archivo descritor del socket, el cual fue devuelto por la llamada a `socket()`

•

backlog

Es el número de conecciones permitidas.

La función `listen()` se usa si se está esperando conexiones entrantes, lo cual significa, si se quiere, alguien que se quiera conectar a nuestra máquina.

Luego de llamar a `listen()`, se deberá llamar a `accept()`, para así aceptar las conexiones entrantes. La secuencia resumida de llamadas al sistema es:

1. `socket()`
2. `bind()`
3. `listen()`
4. `accept()`/* En la próxima sección se explicará como usar esta llamada */

Como todas las funciones descriptas arriva, listen() devolverá -1 en caso de error.

accept()

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

int accept(int fd, void *addr, int *addrlen);
```

Veamos los argumentos de la función:

- **fd**

  Es el archivo descriptor del socket, el cual fue devuelto por la llamada a `listen()`.

- **addr**

  Es un puntero a una estructura sockaddr_in en la cual se pueda determinar qué nodo nos está contactando y desde qué puerto.

- **addrlen**

  Es la longitud de la estructura a la que apunta el argumento `addr`, por lo que conviene establecerlo como `sizeof(struct sockaddr_in)`, antes de que su dirección sea pasada a `accept()`.

Cuando alguien intenta conectarse a nuestra computadora, se debe usar `accept()` para conseguir la conexión. Es muy fácil de entender: alguien sólo podrá conectarse (asóciese con `connect()`) a nuestra máquina, si nosotros aceptamos (asóciese con `accept()`);-)

A continuación, Se dará un pqueño ejemplo del uso de `accept()` para obtener la conexión, ya que esta llamada es un poco diferente de las demás.

```
(...)

sin_size=sizeof(struct sockaddr_in);
/* En la siguiente línea se llama a accept() */
if ((fd2 = accept(fd,(struct sockaddr *)&client,&sin_size))==-1){
printf("accept() error\n");
exit(-1);
}

(...)
```

A este punto se usará la variable `fd2` para añadir las llamadas `send()` y `recv()`.

send()

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

int send(int fd,const void *msg,int len,int flags);
```

Y sobre los argumentos de esta llamada:

- **fd**

  Es el archivo descriptor del socket, con el cual se desea enviar datos.

- **msg**

  Es un puntero apuntando al dato que se quiere enviar.

- **len**

  es la longitud del dato que se quiere enviar (en bytes).

- **flags**

  deberá ser es establecido a 0[8] .

El propósito de esta función es enviar datos usando sockets de flujo o sockets conectados de datagramas. Si se desea enviar datos usando sockets no conectados de datagramas debe usarse la llamada `sendto()`. Al igual que todas las demás llamadas que aquí se vieron, `send()` devuelve -1 en caso de error, o el número de bytes enviados en caso de éxito.

recv()

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

int recv(int fd, void *buf, int len, unsigned int flags);
```

Veamos los argumentos:

- **fd**

  Es el descriptor del socket por el cual se leerán datos.

buf

Es el búfer en el cual se guardará la información a recivir.

len

Es la longitud máxima que podrá tener el búfer.

flags

Por ahora, se deberá establecer como 0.

Al igual de lo que se dijo para `send()`, esta función es usada con datos en sockets de flujo o sockets conectados de datagramas. Si se deseara enviar, o en este caso, recibir datos usando sockets *desconectados* de Datagramas, se debe usar la llamada `recvfrom()`. Analogamente a `send()`, `recv()` devuelve el número de bytes leídos en el búfer, o -1 si se produjo un error.

recvfrom()

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

int recvfrom(int fd,void *buf, int len, unsigned int flags
             struct sockaddr *from, int *fromlen);
```

Veamos los argumentos:

- **fd**

  Lo mismo que para `recv()`

- **buf**

  Lo mismo que para `recv()`

- **len**

  Lo mismo que para `recv()`

- **flags**

  Lo mismo que para `recv()`

from

Es un puntero a la estructura sockaddr.

- **fromlen**

Es un puntero a un entero local que debería ser inicializado a `sizeof(struct sockaddr)`.

Analogamente a lo que pasaba con `recv()`, `recvfrom()` devuelve el número de bytes recibidos, o -1 en caso de error.

close()

```
#include <unistd.h>

close(fd);
```

La función `close()` es usada para cerrar la conexión de nuestro descriptor de socket. Si llamamos a `close()` no se podrá escribir o leer usando ese socket, y se alguien trata de hacerlo recibirá un mensaje de error.

shutdown()

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

int shutdown(int fd, int how);
```

Veamos los argumentos:

- **fd**

Es el archivo descritor del socket al que queremos aplicar esta llamada.

- **how**

Sólo se podrá establecer uno de estos nombres:

- **0**

Prohibido recibir.

-

1

Prohibido enviar.

•

2

Prohibido recibir y enviar.

Es lo mismo llamar a `close()` que establecer `how` a 2. `shutdown()` devolverá 0 si todo ocurre bien, o -1 en caso de error.

gethostname()

```
#include <unistd.h>

int gethostname(char *hostname, size_t size);
```

Veamos de qué se tratan los argumentos:

•

hostname

Es un puntero a un vector que contiene el nombre del nodo actual.

•

size

La longitud del vector que contiene al nombre del nodo (en bytes).

La función `gethostname()` es usada para adquirir el nombre de la máquina local.

Algunas palabras sobre dns

Esta sección fue creada ya que el lector debería saber qué es un *DNS*.

DNS son las siglas de *"Domain Name Service*[9]*"* y, básicamente es usado para conseguir direcciones IP. Por ejemplo, necesito saber la dirección IP del servidor queima.ptlink.net[10] y usando el DNS puedo obtener la dirección IP 212.13.37.13[11].

Esto es importante en la medida de que las funciónes que ya vimos (como `bind()` y `connect()`) son capaces de trabajar con direcciones IP.

Para mostrar cómo se puede obtener la dirección IP de un servidor, por ejemplo de queima.ptlink.net[12], utilizando C, el autor ha realizado un pequeño ejemplo:

```
/* <---- EL CÓDIGO FUENTE EMPIEZA AQUÍ ----> */

#include <stdio.h>
```

```
#include <netdb.h>   /* Este es el archivo de cabecera necesitado por geth-
ostbyname() */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>


int main(int argc, char *argv[])
{

   struct hostent *he;

   if (argc!=2) {
      printf("Usage: %s <hostname>\n",argv[0]);
      exit(-1);
   }

   if ((he=gethostbyname(argv[1]))==NULL) {
      printf("gethostbyname() error\n");
      exit(-1);
   }

   printf("Hostname : %s\n",he->h_name);
      /* muestra el nombre del nodo */
   printf("IP Address: %s\n",
          inet_ntoa(*((struct in_addr *)he->h_addr)));
      /* muestra la dirección IP */

}

/* <---- CÓDIGO FUENTE TERMINA AQUÍ ---> */
```

Un ejemplo de *Servidor de Flujos*

En esta sección, se describirá un bonito ejemplo de un servidor de flujos. El código fuente tiene muchos comentarios para que así, al leerlo, no nos queden dudas.

Empecemos:

```
/* <---- El CÓDIGO FUENTE COMIENZA AQUÍ ----> */

/* Estos son los ficheros de cabecera usuales */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 3550 /* El puerto que será abierto */
#define BACKLOG 2 /* El número de conexiones permitidas */

main()
{
```

```
int fd, fd2; /* los archivos descriptores */

struct sockaddr_in server;
/* para la información de la dirección del servidor */

struct sockaddr_in client;
/* para la información de la dirección del cliente */

int sin_size;

/* A continuación la llamada a socket() */
if ((fd=socket(AF_INET, SOCK_STREAM, 0)) == -1 ) {
   printf("error en socket()\n");
   exit(-1);
}

server.sin_family = AF_INET;

server.sin_port = htons(PORT);
/* ¿Recuerdas a htons() de la sección "Conversiones"? =) */

server.sin_addr.s_addr = INADDR_ANY;
/* INADDR_ANY coloca nuestra dirección IP automáticamente */

bzero(&(server.sin_zero),8);
/* escribimos ceros en el reto de la estructura */


/* A continuación la llamada a bind() */
if(bind(fd,(struct sockaddr*)&server,
        sizeof(struct sockaddr))==-1) {
   printf("error en bind() \n");
   exit(-1);
}

if(listen(fd,BACKLOG) == -1) {  /* llamada a listen() */
   printf("error en listen()\n");
   exit(-1);
}

while(1) {
   sin_size=sizeof(struct sockaddr_in);
   /* A continuación la llamada a accept() */
   if ((fd2 = accept(fd,(struct sockaddr *)&client,
                     &sin_size))==-1) {
      printf("error en accept()\n");
      exit(-1);
   }

   printf("You got a connection from %s\n",
          inet_ntoa(client.sin_addr) );
   /* que mostrará la IP del cliente */

   send(fd2,"Bienvenido a mi servidor.\n",22,0);
   /* que enviará el mensaje de bienvenida al cliente */
```

```
      close(fd2); /* cierraclose a fd2 */
   }
}

/* <---- EL CÓDIGO FUENTE TERMINA AQUÍ ----> */
```

Un ejemplo de *Cliente de Flujos*

Todo será análogo a lo visto en la sección anterior.

```
/* <---- El CÓDIGO FUENTE COMIENZA AQUÍ ----> */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
/* netbd.h es necesitada por la estructura hostent ;-) */

#define PORT 3550
/* El Puerto Abierto del nodo remoto */

#define MAXDATASIZE 100
/* El número máximo de datos en bytes */

int main(int argc, char *argv[])
{
   int fd, numbytes;
   /* archivos descriptores */

   char buf[MAXDATASIZE];
   /* en donde es almacenará el texto recibido */

   struct hostent *he;
   /* estructura que recibirá información sobre el nodo remoto */

   struct sockaddr_in server;
   /* información sobre la dirección del servidor */

   if (argc !=2) {
      /* esto es porque nuestro programa sólo necesitará un
      argumento, (la IP) */
      printf("Usage: %s <IP Address>\n",argv[0]);
      exit(-1);
   }

   if ((he=gethostbyname(argv[1]))==NULL){
      /* llamada a gethostbyname() */
      printf("gethostbyname() error\n");
      exit(-1);
   }

   if ((fd=socket(AF_INET, SOCK_STREAM, 0))==-1){
      /* llamada a socket() */
```

```
      printf("socket() error\n");
      exit(-1);
   }

   server.sin_family = AF_INET;
   server.sin_port = htons(PORT);
   /* htons() es necesaria nuevamente ;-o */
   server.sin_addr = *((struct in_addr *)he->h_addr);
   /*he->h_addr pasa la información de "*he" a "h_addr" */
   bzero(&(server.sin_zero),8);

   if(connect(fd, (struct sockaddr *)&server,
      sizeof(struct sockaddr))==-1){
      /* llamada a connect() */
      printf("connect() error\n");
      exit(-1);
   }

   if ((numbytes=recv(fd,buf,MAXDATASIZE,0)) == -1){
      /* llamada a recv() */
      printf("recv() error\n");
      exit(-1);
   }

   buf[numbytes]='\0';

   printf("Server Message: %s\n",buf);
   /* muestra el mensaje de bienvenida del servidor =) */

   close(fd);   /* cerramos fd =) */

}

/* <---- EL CÓDIGO FUENTE TERMINA AQUÍ ----> */
```

Últimas Palabras

En esta sección el autor de este documento, BracaMan, añadirá algunas palabras. Se lo puede encontrar mandando un correo electrónico a <BracaMan@clix.pt>[13]. Su número ICQ es 41476410 y se puede visitar URL en http://www.BracaMan.net[14]

Soy un simple humano, y como tal me equivoco, por lo que es casi seguro que haya errores en este documento. Y al decir errores me refiero a errores en el uso del Inglés[15] (no sería raro dado que este no es mi lenguaje nativo), como también errores técnicos. Sería muy bueno que si se encuentran errores en este tutorial, yo sea notificado vía email.

Sin embargo, debe entenderse que esta es simplemente la primera versión del documento, por lo que es natural que no esté muy completa (de hecho pienso que así es), y también es natural que existan errores estúpidos. Sea como sea, puedo asegurar que los fragmentos de código fuente que se encuentran en este documento funcionan perfectamente. Si se necesitara ayuda en lo que a esto respecta se puede contactarme mandando un email a <BracaMan@clix.pt>[16].

Agradecimientos especiales a: Ghost_Rider (mi viejo y querido camarada), Raven (por dejarme escribir este tutorial) y para todos mis amigos =)

En cuanto al traductor de este documento al español, su nombre es Sebastián Gurin, y se puede entrar en contacto con él mandándole un mail a cancerbero_sgx@users.sourceforge.net[17]. Él fue también quien ha dado formato usando DocBook -- SGML, versión 3.1 por lo que, si se encuentran errores tanto en el formato como en la traducción, se deberían enviar comentarios a la dirección de correo electrónico anterior.

Notas

1. Recuérdese uno de los primeros conceptos que aprendemos al usar UNIX, "En un sistema Unix, todo es un archivo". Nota del T.
2. Protocolo de Control de Transmisión. Nota del T.
3. Protocolo de los Datagramas de Usuario. Nota del T.
4. "*nodo*" será la traducción del término en inglés "*host*", que se usará a lo largo de este texto. Nota del T.
5. que es la traducción al español de Network Byte Order. El término también es conocido como *"Big-Endian Byte Order"*. Tener en cuenta que, si no se habla de Ordenamiento de Bytes para Redes, se hablará de Ordenamiento de Bytes para Nodos. Nota del T.
6. los cuales ocupan 2 y 4 bytes de memoria, respectivamente.
7. Se puede indagar más sobre estas y las demás funciones relacionadas a Sockets en UNIX, leyendo las páginas de manual correspondientes a cada una. Nota del Traductor.
8. El argumento "*flags*", también contenido en otras funciones de manejo de sockets, no es una característica propia del mensaje, y puede omitirse. En este caso, hace diferencia entre los distintos modos de enviar un paquete, además de otras cosas. Para más información, se puede leer el manual de send() (man 2 send). Nota del T.
9. que se traduce como "Servicio de Nombres de Dominio". Nota del T
10. http://queima.ptlink.net
11. http://212.13.37.13
12. http://queima.ptlink.net
13. mailto:BracaMan@clix.pt
14. http://www.BracaMan.net
15. Seguramente también los haya en la traducción ;-). Nota del T.
16. mailto:BracaMan@clix.pt

17. mailto:cancerbero_sgx@users.sourceforge.net