COMENZANDO CON LOS SOCKETS EN ANSI C

Sagrini : elhacker.net

1.) Introdución

Me decidí a escribir este manual porque mucha gente en el foro tiene algunas duda con el uso de sockets en Ansi C. Todas las dudas en el foro. Comento que necesitaréis algo de experiencia en C, no os voy a explicar lo que es "int main ()".

Las liberías a usar son:

```
#include <arpa/inet.h>       //struct sockaddr_in
#include <sys/socket.h>      //Socket, Connect...
```

2) Funciones principales I

2.1) Función socket ();

La primera función que vamos a "estudiar" es socket. Esta función inicializa el socket para su uso... Lo que hace es meter en una variable tipo INT un descriptor para usarlo en otras funciones. Uso:

```
int sockfd = socket (2, 1, 0);
```

Bueno, la cosa es declarar una variable y pasarla como return a la función socket. ¡Distintos tipos de argumentos se pueden ver en Google! Estos ahora mismo son suficientes, pero el saber no mata...

2.2) Función close ();

Esto se usa para cerrar el socket una vez usado. Si lo dejamos abierto, podríamos tener algunos problemas luego...

Es muy sencillo. Le pasas como argumento el descriptor de socket. En nuestro caso es "sockfd".

Evidentemente se debe cambiar en cada uno...

```
int sockfd = socket (2, 1, 0);
close (sockfd);
```

2.3) Estructura struct sockaddr // sockaddr_in

En esta estructura se usa para guardar los datos de las conexiones (Ej: ip, puerto, familia...). Lo explicaremos todo...
Bueno, se declara así.

```
struct sockaddr_in host;
struct sockaddr_in client;
```

"Struct sockaddr" se usa por compatibilidad...
Bueno, ya vereis... Una vez declarada la estructura y el descriptor de socket, rellenamos la estructura...

```
host.sin_addr.s_addr = 0;          // Así rellenamos la dirección
                                        con nuestra IP en orden de
                                        bits de red. Para escuchar.
host.sin_port = htons (31337);     // HTONS pasa un numero (puerto)
                                        a orden de bits de red. Es
                                        para que el PC lo entienda.
host.sin_family = AF_INET;         // Esta es la familia de Ipv4.
                                        Yo la suelo dejar así.
memset (host.sin_zero, 0, 8);      // Rellena el resto de la
                                        estructura con ceros. Así.
```

Esta se usa para escuchar. Para conectar...

```
host.sin_addr.s_addr = arpa_inet ("127.0.0.1"); // IP
host.sin_port = htons (31337);     // HTONS pasa un numero (puerto)
                                        a orden de bits de red. Es
                                        para que el PC lo entienda.
host.sin_family = AF_INET;         // Esta es la familia de Ipv4.
memset (host.sin_zero, 0, 8);      // Rellena el resto con ceros.
```

Estas estructuras son muy importantes, y contienen los datos que usamos.

2.4) Función connect ();

Esto se usa para conectarnos a cualquier IP. Usa las estructuras antes vistas. ¿Os acordais de lo que dije antes de la compatibilidad? Es para esto...

Sus argumentos son:

```
connect (sockfd, (struct sockaddr*)&host, sizeof (struct sockaddr*));
```

Poco a poco. “sockfd” es el descriptor que se usa para guardar el descriptor con el que mandamos y recibimos datos.
Lo siguiente es un “typedef”. Es hacer una variable otro tipo de variable. Se hace por compatibilidad (por tercera vez). Se le pasa una dirección a una estructura, no una estructura completa...
Lo tercero es el tamaño de la estructura. Se hace fácil con sizeof. Es por seguridad.

2.5) Función bind ();

Esto es diferente a conectar. ¡No os lieis!
Esto se usa para preparar un socket para escuchar en nuestra máquina. Usa tanto las estructuras de antes como lo de la compatibilidad.
Sus argumentos son:

```
bind (sockfd, (struct sockaddr*)&host, sizeof (struct sockaddr*));
```

Son exactamente iguales que los de connect. Sólo se elimina “newsock”, pues aquí seguimos usando “sockfd”. Aún necesitamos usar dos funciones más.

2.6) Función listen ();

Esto es otro paso para escuchar. Sus argumentos son:

```
listen (sockfd, 1);
```

“Sockfd” es el socket que hemos usado antes. Lo segundo es el número de conexiones que puede aceptar. En este manual lo dejamos en una.

2.7) Función accept ();

Esto es el último paso para aceptar una conexión. Aquí otros PCs se conectan.

```
    int a = sizeof (struct sockaddr*);
    struct sockaddr_in client;
    int newsock = accept (sockfd, (struct sockaddr*)&client, &a);
```

En esta función hay que declarar “a”, que contiene el tamaño de “struct sockaddr*”. A “accept”, que se le pasa la dirección de la variable, no la variable.
Ahora se usa “newsock”, que luego usamos para enviar y recibir datos. A partir de aquí podéis cerrar “sockfd”, pues ya no lo usamos más.
Client se usa para guardar los datos del quien se conecta. Luego podeis escribir las IPs de quien se conecta a vosotros, el puerto que usa, etc... Se declara aparte. Aclaro que aquí el programa se para hasta que alguien se conecta.

2.0) Código de demostración.

```
#include <stdio.h>          //Funciones basicas
entrada/salida
#include <string.h>         //StrCmp
#include <stdlib.h>         //Exit y otras
#include <arpa/inet.h>      //struct sockaddr_in
#include <sys/socket.h>     //Socket, Connect...
struct sockaddr_in host, client;    //Declaraciones
int a=sizeof (struct sockaddr);
int newsock;
int sockfd;
int main (int argc, char *argv [])
{
    printf ("Code 1.0  By Sagrini (2010)\n");
    if ((sockfd=socket (2, 1, 0))==1)
    {
        printf ("Error abriendo socket...\n\n");
        return 1;
    }
    host.sin_port=htons(31337);
    host.sin_family=AF_INET;
    host.sin_addr.s_addr=0;
    memset (host.sin_zero, 0, 8);
    if(bind(sockfd,(struct
sockaddr*)&host,sizeof(host))==1)
    {
        printf ("Error haciendo binding...\n\n");
        return 1;
    }
    if(listen(sockfd,5)==1)
    {
```

```
        printf ("Error escuchando...\n\n");
        return 1;
    }
    if((newsock=accept(sockfd, (struct
sockaddr*)&client, &a))==1)
    {
        printf ("Error esperando conectando...\n\n");
        return -1;
    }
    printf ("Got connection from %s:%d\n", inet_ntoa
(client.sin_addr), ntohs (client.sin_port));
    close (sockfd);
    close (newsock);
    return 0;
}
```

3) Funciones principales II

3.1) Función recv ();

Otra funcion muy importante es RECV. Es sencillamente para recibir datos.. Se usa una vez conectado o con algun cliente conectado a ti.

```
char buffer [1024];
recv (newsock, &buffer, sizeof (buffer), 0);
```

Bueno, la cosa es pasarle el descrpitor de socket, un buffer, su tamaño, y un cero. Para el tamaño se usa sizeof. Es una medida de seguridad.

3.2) Función send ();

La siguiente funcion es SEND. Evidentemente se usa para mandar datos. También se usa una vez conectado o con algun cliente conectado a ti.

```
char buffer [1024];
send (newsock, &buffer, sizeof (buffer), 0);
```

Usa los mismos argumentos que recv (). Se usan conjuntamente.

3.3) Función inet_addr ();

Bueno, esta función se usa para pasar una IP a orden de bits de red. La hemos visto antes en la estructura sockaddr_in. Su argumento es una cadena con una IP. Devuelve otra cadena.

3.4) Función htons ();
Bueno, esta función se usa para pasar un puerto a orden de bits de red. La hemos visto antes en la estructura sockaddr_in. Su argumento es un número.

3.5) Función inet_ntoa ();
Esta función es contraria a inet_addr. Pasa los bytes en orden de red a cadena. Su argumento es un puntero a la estructura struct sockaddr*.sin_addr.

3.5) Función inet_ntoa ();
Esta función es contraria a htons. Pasa los bytes en orden de red a número. Su argumento es un puntero a la estructura struct sockaddr*.sin_port.

3.0) Código de demostración

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <time.h>
int socketfd, newsocket;
char *filename;
void shutup (int signal)
{
    FILE *log;
    log=fopen (filename, "a+");
    times ();
    printf ("Shutting down...\n\n");
    fprintf (log, "Shutting down...\n\n");
    fclose (log);
    close (newsocket);
    close (socketfd);
    exit (0);
}
int times ()
{
    FILE *log;
    time_t now=time (0);
    struct tm *ahora;
    char buffer [40];
    ahora=localtime ((const time_t*)&now);
    strftime (buffer, 40, "%d/%m/%Y %H:%M:%S" ,
ahora);
    log=fopen (filename, "a+");
    printf ("%s   ", buffer);
```

```
    fprintf (log,"%s   ", buffer);
    fclose (log);
    return 0;
}
int main (int argc, char *argv [])
{
    time_t now=time (0);
    struct tm *ahora;
    char hora [40];
    ahora=localtime ((const time_t*)&now);
    strftime (hora, 40, "%d/%m/%Y %H:%M:%S" , ahora);
    printf ("SmallServ 2.0  By Sagrini  Sagrini
2010  %s\n", hora);
    if (getuid()!=0)
    {
        printf ("This proccess must be run by
root.\n\n");
        return 1;
    }
    if (argc<3)
    {
        printf ("Use: %s <PORT> <LOG>
[MODE]\nMode:\t\t0) Fork the proccess to background
[DEFAULT].\n\t\t1) Run in the terminal.\n\n", argv
[0]);
        return 1;
    }
    int cont;
    FILE *log;
    struct sockaddr_in client, host;
    char buffer [1024];
    int size=sizeof (client);
    filename = argv [2];
    socketfd=socket (2, 1 ,  0);
    host.sin_family=AF_INET;
    host.sin_port=htons (atoi (argv [1]));
    host.sin_addr.s_addr=0;
    bind (socketfd, (struct sockaddr*)&host, sizeof
(struct sockaddr));
    listen (socketfd, 3);
    log=fopen (filename, "a+");
    times ();
    if (argv [3] != NULL && atoi (argv [3]) == 1)
        printf ("\nStarting up...\n\n");
    fprintf (log, "Starting up...\n\n");
    fclose (log);
    signal (SIGTERM, shutup);
    signal (SIGINT, shutup);

    if (argv [3] == NULL || atoi (argv [3]) == 0)
        daemon (1, 0);
    while (1)
```

```
    {
        newsocket=accept (socketfd, (struct
sockaddr*)&client, &size);
        log=fopen (filename, "a+");
        times ();
        printf ("Got connection from %s:%d\n",
inet_ntoa (client.sin_addr), ntohs
(client.sin_port));
        fprintf (log, "Got connection from %s:%d\n",
inet_ntoa (client.sin_addr), ntohs
(client.sin_port));
        fclose (log);
        cont=recv (newsocket, &buffer, 1024, 0);
        while (cont>2)
        {
            printf ("%d", cont);
            log=fopen (filename, "a+");
            times ();
            buffer [cont1]='\0';
            printf ("RECV %d bytes: %s ", cont2,
buffer);
            fprintf (log, "RECV %d bytes: %s\n",
cont2, buffer);
            fclose (log);
            cont=recv (newsocket, &buffer, 1024, 0);
        }
        log=fopen (filename, "a+");
        times ();
        printf ("Finishing connection from %s:
%d\n\n", inet_ntoa (client.sin_addr), ntohs
(client.sin_port));
        fprintf (log, "Finishing connection from %s:
%d\n\n", inet_ntoa (client.sin_addr), ntohs
(client.sin_port));
        fclose (log);
        close (newsocket);
    }
    close (socketfd);
    return 0;
}
```

Este code es el mismo que antes, pero completo. Es el código fuente original.

4) Sockets en Windows

4.1) Función WSAStartup ();

Microsoft cambia algunas cosas a la hora de programar en Windows. Así de malotes son...
Bueno, una de esas cosas es que a la hora de usar sockets, tenemos que inicializarlos de otra

forma. Primero, en nuestro code después del típico "if (argc!=3)" tiene que ir algo así.

```
WSADATA wsa;
WSAStartup(MAKEWORD(2,2),&wsa);
```

Esto es para implementar algunas cosillas. No me enrollaré en eso. Basta saber que se usa en todos los programas con sockets.

4.2) Linkeando al compilar

Otra cosita que fastidia un poco es que a la hora de compilar tenemos que linkear a una librería. En mi PC se hace así.

```
gcc o code.exe code.c lwsock32
```

En otros Windows cambia. Toca mirar a Google...
PD: Hay otras formas más elegantes de hacerlo, pero yo lo hago así.

4.3) Declarar y cerrar sockets.

Bueno, seguimos. Otro detalle es que a la hora de declarar los descriptores de socket tenemos que usar el tipo SOCKET. Se escribe así, en mayúscula. No me he molestado en mirarlo, no merece la pena, pero creo que es lo mismo que "int". De todos modos de otra forma da error...
Y a la hora de cerrar tenemos que usar la función closesocket (); Es exactamente igual que close (); Sólo cambia el nombre (¿Sólo?).

4.4) Recv y Send

Último detalle. En las funciones RECV y SEND pasamos la dirección de buffer así:

```
recv (newsock, &buffer, sizeof (buffer), 0);
send (newsock, &buffer, sizeof (buffer), 0);
```

Pero los de Windows quieren que lo hagamos de otra forma...

```
recv (newsock, buffer, sizeof (buffer), 0);
send (newsock, buffer, sizeof (buffer), 0);
```

Esto se debe a un cambio en la declaración. Si pasamos lo primero mandamos un puntero a un puntero a un char. En cambio de la segunda forma es un puntero a un char. Caprichosos...

4.0) Código de demostración.

```
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
SOCKET sockfd, newsock;
char *filename;
int times ()
{
        time_t now=time (0);
        struct tm *ahora;
        char buffer [40];
        ahora=localtime ((const time_t*)&now);
        strftime (buffer, 40, "%d/%m/%Y %H:%M:%S", ahora);
        printf ("%s   ", buffer);
        return 0;
}
int main (int argc, char *argv [])
{
        WSADATA wsa;
        WSAStartup(MAKEWORD(2,2),&wsa);
        time_t now=time (0);
        struct tm *ahora;
        char hora [40];
        ahora=localtime ((const time_t*)&now);
        strftime (hora, 40, "%d/%m/%Y %H:%M:%S", ahora);
        printf ("SmallServ 2.0  By Sagrini  Sagrini 2010  %s\n",
hora);
        if (argc != 2)
        {
                printf ("Use: %s <port>\n\n", argv [0]);
                return 1;
        }
        int cont;
        struct sockaddr_in client, host;
        char buffer [1024];
        int size=sizeof (client);
        sockfd=socket (2, 1 , 0);
        host.sin_family=AF_INET;
        host.sin_port=htons (atoi (argv [1]));
        host.sin_addr.s_addr=0;
        bind (sockfd, (struct sockaddr*)&host, sizeof (struct
sockaddr));
        listen (sockfd, 3);
```

```
        times ();
        printf ("Starting up...\n\n");

        while (1)
        {
                newsock=accept (sockfd, (struct sockaddr*)&client,
&size);
                times ();
                printf ("Got connection from %s: %d\n", inet_ntoa
(client.sin_addr), ntohs (client.sin_port));
                cont=recv (newsock, buffer, 1024, 0);
                while (cont>2)
                {
                        printf ("%d", cont);
                        times ();
                        buffer [cont1]='\0';
                        printf ("RECV %d bytes: %s\n", cont2,
buffer);
                        cont=recv (newsock, buffer, 1024, 0);
                }
                times ();
                printf ("Finishing connection from %s:%d\n\n",
inet_ntoa (client.sin_addr), ntohs (client.sin_port));
                closesocket (newsock);
        }
        closesocket (sockfd);
        return 0;
}
```

Si os dais cuenta, he borrado algunas funciones. Eso se debe a que al pasar el code a Windows me daba pereza cambiar algunas cosas... Todo es culpa de la tele jeje...

5.) Final

Bueno, esta es la parte que nadie lee en la que el autor se despide y dice los agradecimientos. Gracias por leerme jeje...

Primero, gracias a los que están en el foro echándome una manita de vez en cuando. Gracias al staff y a el-brujo, por el gran trabajo que estáis haciendo. Gracias a los users, algunos colaboran mejor que otros, pero sois parte de la comunidad... Gracias a mi familia, que no puede faltar, y a mis amigos. Evidentemente no leerán esto nunca, pero bueno... Muchas gracias por leer. Espero vuestros comentarios. Toda crítica bienvenida. ¡¡¡ GRACIAS !!!

Sagrini en elhacker.net : sagrini@elhacker.net