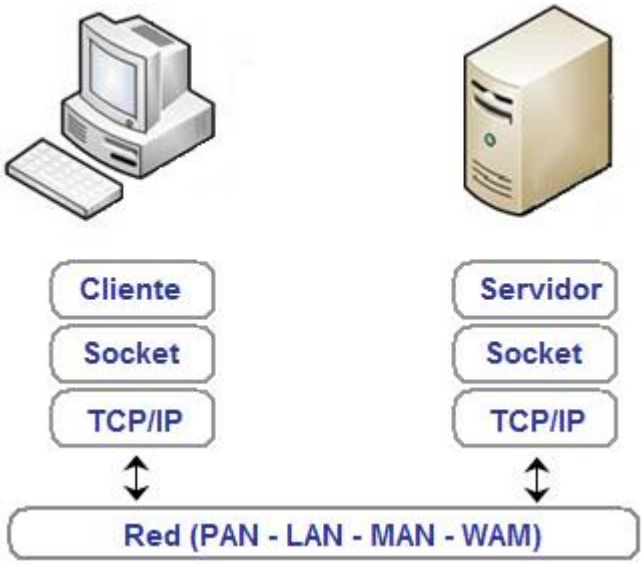
HDC

C Programming

Sockets

Muy buenas a todos, hoy nos toca seguir programando en C, ya estamos bien avanzados, confío en que hayáis practicado bastante y se haya entendido todo. Hoy vamos a aprender algo que a mí personalmente me parece muy interesante, estoy hablando de los **sockets**.



Un socket es un proceso por el cual dos programas en máquinas diferentes pueden **intercambiar información**. Sería algo parecido (y más que parecido) a lo que aprendimos con las clases en las que estudiábamos los diferentes servicios (ftp, ssh, etc). De hecho podemos crear un programa con sockets que se comunique con cualquiera de estos servicios. También podremos crear un "cliente-servidor" que haga lo que nosotros queramos. Lo que vamos a aprender ahora está enfocado a Windows, aunque en Linux es muy parecido.

Vamos a empezar viendo como podríamos hacer un cliente para ftp y más tarde haremos un "cliente-servidor" para ver como funciona todo.

“Pero, ya tenemos Filezilla ¿Para que queremos hacer nuestro propio servidor?”

Bueno, Manolo. Es cierto que ya tenemos un cliente para Filezilla, pero vamos a hacer el nuestro propio para entender como funcionan los **sockets**, además muchas veces es más satisfactorio usar tus propias herramientas que algo que ha hecho otra persona.

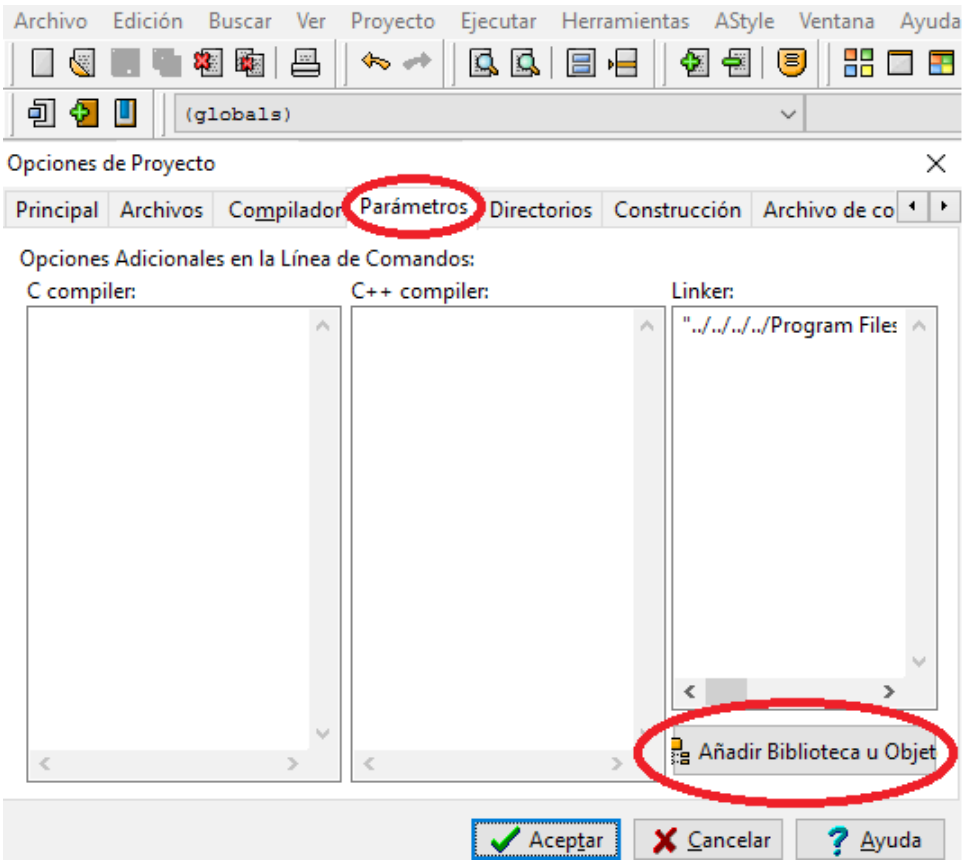
Vamos a empezar, como siempre lo primero que hacemos cuando programamos en C es añadir las librerías y el método principal.

```
1.  #include <stdio.h>
2.  #include <string.h>
```

```
3.  #include <winsock2.h>
4.  #include <windows.h>
5.
6.  int main(){
7.
8.  }
```

Las dos primeras librerías las conocemos, la tercera y la cuarta son las que se usan para añadir sockets en Windows (a esto se le llama **winsock**). Para que todo esto funcione tenemos que incluir la biblioteca en nuestro proyecto. Para esto vamos a pulsar **CONTROL+H**, vamos a la sección de parámetros y le damos a añadir biblioteca u objeto. Ahí buscamos **libws2_32.a**, en mi caso estaba en la ruta:

"*C:/Program Files (x86)/Dev-Cpp/MinGW64/x86_64-w64-mingw32/lib*".



Lo siguiente que vamos a hacer es iniciar un **winsock** dentro de la función principal.

```
1.  WSADATA wsa; // Llama a Winsock.
2.
3.      int sock; // Define un socket.
4.      struct sockaddr_in direccion; // Tipo de dato en el que guardaremos la
    información del servidor.
```

```
5.
6.      WSAStartup(MAKEWORD(2,0), &wsa);  //  Define la versión de Winsock que
    vamos a utilizar en wsa (lo definimos al principio) (2,0).
```

Sockaddr_in es un tipo de dato usado para guardar la información del servidor al cual nos vamos a conectar. Lo siguiente que vamos a hacer crear el **socket** que hemos definido, para ello vamos a usar la función:

sock(familia del protocolo, tipo de socket, protocolo);

Familias del protocolo.

- **AF_INET**: Establece comunicación con cualquier tipo de máquina en internet.
- **AF_UNIX**: Establece comunicación con una máquina local que corra bajo Linux.

Tipo de socket.

- **SOCKET_STREAM**: Tipo de socket usado para TCP.
- **SOCKET_DGRAM**: Tipo de socket usado para UDP.

Protocolo.

- **PPROTO_TCP**: Usa el protocolo TCP.
- **IPPROTO_UDP**: Usa el protocolo UDP.

Teniendo en cuenta esa tabla en nuestro caso quedaría así.

```
sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
```

Lo siguiente que vamos a hacer es elegir la ip del servidor y el puerto, el puerto será el 21 y la ip la introducirá el usuario.

```
1.      int puerto = 21;
2.      char ip[30];
3.
4.      printf("Ip del servidor FTP al que se quiere conectar: ");
5.      scanf("%s", &ip);
```

Una vez hecho esto podemos pasarle los valores a nuestro **sockaddr_in** llamado **dirección**, que era donde íbamos a guardar toda la información.

```
1.      direccion.sin_family=AF_INET;
2.      direccion.sin_port=htons(puerto);
3.      direccion.sin_addr.s_addr=inet_addr(ip);
```

Los datos que estamos modificando de nuestra ***dirección*** son, en la primera línea **sin_family**, que es la familia del protocolo. En la segunda línea **sin_port**, que es el puerto por el que nos vamos a conectar, **htons** sirve para decir que ese int que creamos es un puerto. Por último modificamos el dato **sin_addr.s_addr**, que es donde se guarda la IP, **inet_addr** sirve para decir que ese string que teníamos es una IP.

Tranquilidad que ya hemos pasado la parte más dura y esta parte siempre es de la misma forma, ahora nos vamos a pasar a la parte más interesante, nos vamos a conectar y vamos a comenzar a enviar y a recibir datos. **Para conectarnos** vamos a usar la función:

connect(nombre del socket, (struct sockaddr*)&nombre de la estructura, sizeof(nombre de la estructura))

En nuestro caso quedaría así.

```
connect(sock, (struct sockaddr*)&direccion,sizeof(direccion));
```

Además de esto nos quedan tres funciones que entender, sería una para enviar datos y otra para recibirlos, estas dos las meteremos en un bucle para que el programa pueda interactuar con el FTP, y otra para cerrar el socket definitivamente.

send(nombre del socket, variable que envías, sizeof(variable que envías), 0);

recv(nombre del socket, variable que envías, sizeof(variable que envías), 0);

closesocket(nombre del socket);

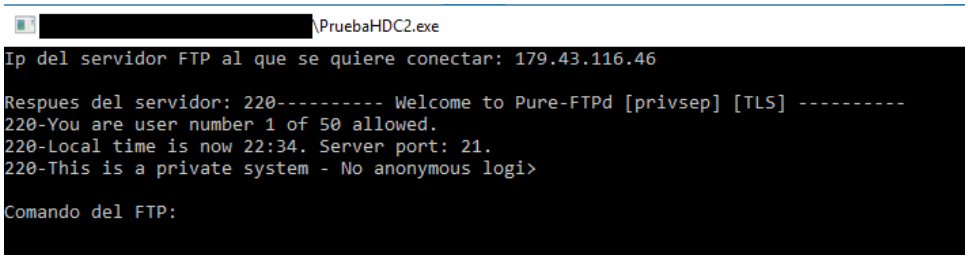
Esta última parte quedaría de esta forma.

```
1.      while( 1==1){
2.
3.          printf("\n\nComando del FTP: ");
4.          scanf("%s", &mensajes);
5.          send(sock, mensajes, sizeof(mensajes),0);
6.
7.          recv(sock, mensajer, sizeof(mensajer),0);
8.          printf("\nRespues del servidor: %s", mensajer);
9.
10.     }
11.
12.     closesocket(sock);
```

Habría que añadir al principio dos variables string, una llamada mensajes y otra llamada mensajer, por lo tanto el código completo quedaría así.

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #include <winsock2.h>
4.  #include <windows.h>
5.
6.  int main(){
7.
8.      WSADATA wsa; // Llama a Winsock.
9.
10.     int puerto = 21;
11.     char ip[30];
12.     char mensajes[200];
13.     char mensajer[200];
14.
15.     int sock; // Define un socket.
```

```
16.     struct sockaddr_in direccion; // Tipo de dato en el que guardaremos la
    información del servidor.
17.
18.     WSAStartup(MAKEWORD(2,0),&wsa);  //  Define la versión de Winsock que vamos
    a utilizar en wsa (lo definimos al principio) (2,0)
19.
20.     sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
21.
22.     printf("Ip del servidor FTP al que se quiere conectar: ");
23.     scanf("%s", &ip);
24.
25.     direccion.sin_family=AF_INET;
26.     direccion.sin_port=htons(puerto);
27.     direccion.sin_addr.s_addr=inet_addr(ip);
28.
29.     connect(sock, (struct sockaddr*)&direccion,sizeof(direccion));
30.
31.     while( 1==1 ){
32.
33.         printf("\n\nComando del FTP: ");
34.         scanf("%s", &mensajes);
35.         send(sock, mensajes, sizeof(mensajes),0);
36.
37.         recv(sock, mensajer, sizeof(mensajer),0);
38.         printf("\nRespues del servidor: %s", mensajer);
39.
40.     }
41.
42.     closesocket(sock);
43. }
44.
```



"Este cliente FTP no será funcional, es solo un ejemplo para comentar como funcionan los sockets."

Ahora que hemos aprendido como funciona la parte del **cliente** en los sockets nos vamos a centrar en hacer la parte del **servidor**.

"¿También vamos a hacer un servidor para fpt?"

No, Manolo, aunque sería interesante solo queremos aprender como funcionan los sockets, vamos a hacer un servidor al que nos podamos conectar y con el cual nos podamos comunicar, pero ahora solo vamos a hacer un chat de texto. El servidor se define de la misma forma que el cliente, la diferencia es que vamos a usar esta función:

bind(nombre del socket, (struct sockaddr*)&nombre de la estructura, sizeof(nombre de la estructura));

Funcionaría algo así como la función **connect** que usábamos en el cliente, y para poder poner el programa en modo escucha usaremos esta función:

Listen(nombre del socket, número máximo de clientes);

Por último para aceptar las conexiones utilizaremos la siguiente función:

(sock=accept(nombre del socket,0,0));

Por lo demás el servidor funciona de la misma manera que un cliente, solo hay que enviar, recibir información y tratarla según lo que quieras hacer. Sabiendo todo esto el servidor quedaría de la siguiente forma:

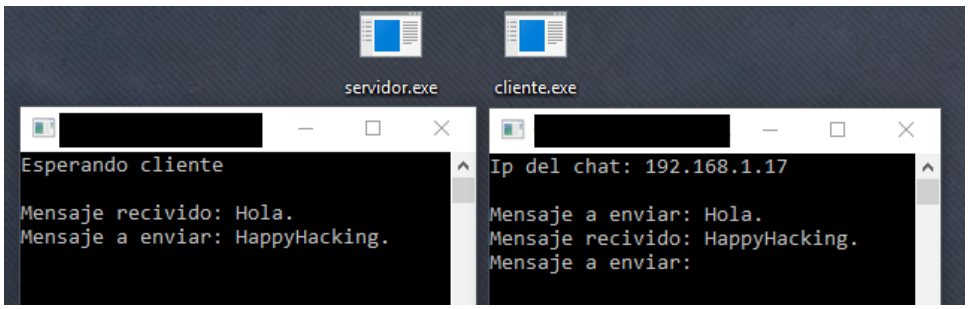
```c
#include <stdio.h>
#include <string.h>
#include <winsock2.h>
#include <windows.h>


int main(){

    WSADATA wsa;

    int puerto = 30;
    char mensajes[200];
    char mensajer[200];

    int sock;
    struct sockaddr_in direccion;

    WSAStartup(MAKEWORD(2,0),&wsa);

    sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);

    direccion.sin_family=AF_INET;
    direccion.sin_addr.s_addr=INADDR_ANY; // INADDR_ANY es lo mimo que poner local host (127.0.0.1).
    direccion.sin_port=htons(puerto);

    bind(sock, (struct sockaddr*)&direccion,sizeof(direccion));

    listen(sock,1);

    printf("Esperando cliente\n\n");

```

```
32.     (sock=accept(sock,0,0));
33.
34.
35.     while(1==1){
36.
37.         recv(sock, mensajer, sizeof(mensajer),0);
38.         printf("\nMensaje recivido: %s", mensajer);
39.
40.         printf("Mensaje a enviar: ");
41.         scanf("%s", &mensajes);
42.         send(sock, mensajes, sizeof(mensajes),0);
43.     }
44.
45.     closesocket(sock);
46. }
```

El cliente para el chat con este servidor quedaría de la siguiente forma:

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #include <winsock2.h>
4.  #include <windows.h>
5.
6.
7.  int main(){
8.
9.      WSADATA wsa;
10.
11.     int puerto = 30;
12.     char ip[30];
13.     char mensajes[200];
14.     char mensajer[200];
15.
```

```
16.     int sock;
17.     struct sockaddr_in direccion;
18.
19.     WSAStartup(MAKEWORD(2,0),&wsa);
20.
21.     sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
22.
23.     printf("Ip del chat: ");
24.     scanf("%s", &ip);
25.
26.     direccion.sin_family=AF_INET;
27.     direccion.sin_port=htons(puerto);
28.     direccion.sin_addr.s_addr=inet_addr(ip);
29.
30.     connect(sock, (struct sockaddr*)&direccion,sizeof(direccion));
31.
32.
33.
34.     while(1==1){
35.
36.         printf("\nMensaje a enviar: ");
37.         scanf("%s", &mensajes);
38.         send(sock, mensajes, sizeof(mensajes),0);
39.
40.         recv(sock, mensajer, sizeof(mensajer),0);
41.         printf("Mensaje recivido: %s", mensajer);
42.
43.     }
44.
45.     closesocket(sock);
46. }
```



Como bonus para esta clase de sockets, me gustaría explicar de forma sencilla como funciona el **tratamiento de archivos**. Para entender esto lo único que haremos será un programa que lea un archivo de texto, lo imprima por pantalla, y lo vuelque a otro archivo diferente. Vamos a crear una variable de tipo FILE, lo haremos de la siguiente forma:

FILE *nombre variable;

La función para abrir un fichero es la siguiente:

fopen("nombre del archivo", "OpenType");

El **OpenType** es la forma en la que se va a abrir el archivo, las posibles opciones son las siguientes.

- *"r" : Abrir archivo modo lectura.*
- *"w" : Abrir archivo modo escritura, se crea un nuevo archivo o se sobrescribe.*
- *"a" : Abrir modo escritura, escribe al final del archivo, si no existe se crea.*
- *"r+" : Abrir archivo modo lectura y escritura, el fichero debe existir.*
- *"w+" : Abrir archivo modo lectura y escritura, se crea un nuevo archivo o se sobrescribe.*

Igual que con los sockets, los archivos también hay que cerrarlos al acabar de tratarlos, para ellos usamos la función.

fclose(archivo);

Para hacer el programa que queríamos hacer vamos a abrir dos archivos, uno para leer y otro para escribir, después veremos que haremos con ellos.

```
1.  FILE *archivo1;
2.  FILE *archivo2;
3.  char nombreArchivo1[20];
```

```
4.      char nombreArchivo2[30] = "copia";
5.
6.      printf("Nombre del archivo a copiar: ");
7.      scanf("%s", &nombreArchivo1);
8.
9.      strcat(nombreArchivo2, nombreArchivo1);
10.
11.     archivo1 = fopen(nombreArchivo1, "r");
12.     archivo2 = fopen(nombreArchivo2, "w");
```

Ahora solo nos queda aprender como leer y como escribir en un fichero, para eso vamos a usar las siguientes dos funciones:

fscanf(archivo, "%tipo de dato", variable en la que se guarda);
fprintf(archivo, "%tipo de dato", variable que escribiremos);

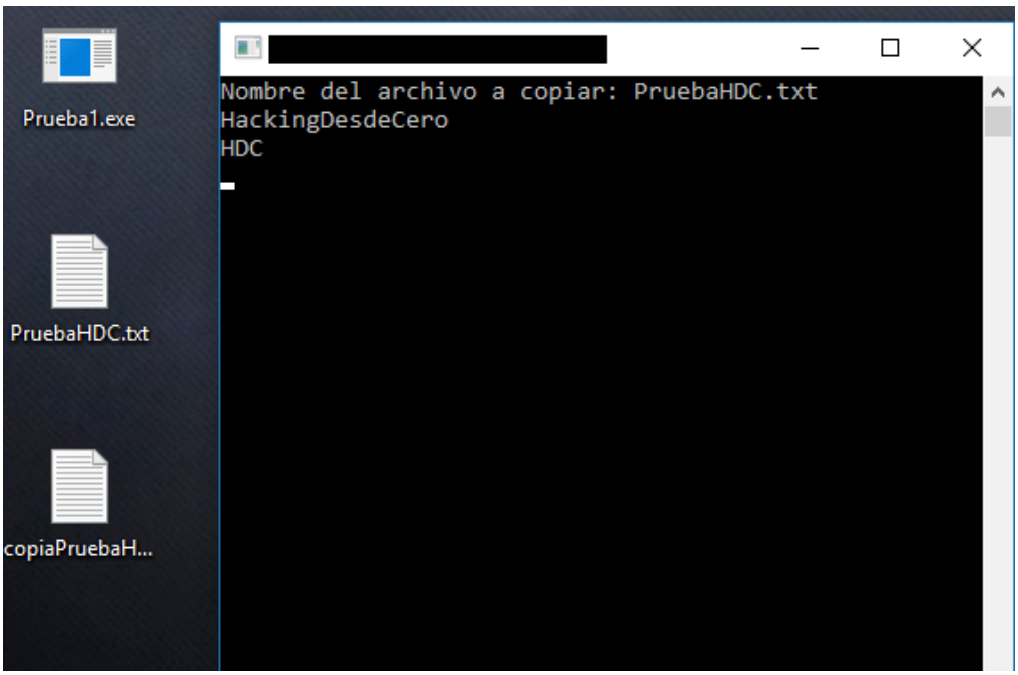
Para que lea todas las líneas vamos a hacer un bucle, el archivo acaba cuando el resultado de **fscanf** es -1.

```
1.      while(fscanf(archivo1, "%s", linea) != -1){
2.
3.          printf("%s\n", linea);
4.          fprintf(archivo2, "%s\n", linea);
5.
6.      }
```

Le vamos a añadir un **retardo** para que se pueda ver como se imprime por pantalla, el programa al final quedaría de esta forma.

```
1.  #include <stdio.h>
2.  #include <string.h>
3.
4.  int main(){
5.
6.      FILE *archivo1;
7.      FILE *archivo2;
8.      char nombreArchivo1[20];
9.      char nombreArchivo2[30] = "copia";
```

```
10.     char linea[200];
11.
12.     printf("Nombre del archivo a copiar: ");
13.     scanf("%s", &nombreArchivo1);
14.
15.     strcat(nombreArchivo2, nombreArchivo1);
16.
17.     archivo1 = fopen(nombreArchivo1, "r");
18.     archivo2 = fopen(nombreArchivo2, "w");
19.
20.     while(fscanf(archivo1, "%s", linea) != -1){
21.
22.         printf("%s\n", linea);
23.         fprintf(archivo2, "%s\n", linea);
24.
25.     }
26.
27.     sleep(10000);
28.
29.     fclose(archivo1);
30.     fclose(archivo2);
31.
32. }
```



Ya hemos acabado por ahora, para aprender a programar bien es muy importante practicar mucho. ¿Cómo estaría un chat que guarda los logs? ¿O tal vez un programa que haga fuerza bruta por diccionario a un servidor SSH? Hay muchas cosas que podéis hacer, pero lo importante es que practiquéis.

Pueden seguirme en Twitter: @RoaddHDC

Contactarse por cualquier duda a: r0add@hotmail.com

Para donaciones, pueden hacerlo en bitcoin en la dirección siguiente:

1HqpPJbbWJ9H2hAZTmpXnVuoLKkP7RFSvw

También recomiendo que se unan al foro: underc0de.org/foro

Rollth.