PROGRAMACIÓN DE SOCKETS EN ENTORNO UNIX

P.1.2

Laboratorio de Redes y Servicios

INTEGRANTES DEL GRUPO:

Nombre: MANUEL MONTOYA CATALÁ

Nombre: ANDRÉS BEATO OLLERO

ÍNDICE

INTRODUCCIÓN

El problema del bloqueo

Muchas de las funciones que hemos visto relacionadas con sockets son bloqueantes, es decir, bloquean el flujo de ejecucion del programa a la espera de un determinado evento. Por ejemplo accept() espera a que un cliente se conecte a nuesta maquina o recv() espera a que recibamos algun paquete.
El problema de esto es que solo podemos estar a la espera de un unico evento determinado, no podriamos por ejemplo estar a la espera de recibir datos (recv()) de 2 sockets diferentes al mismo tiempo.

Podriamos configurar los sockets como "no bloqueantes" usando la funcion fcntl(). De esta manera si por ejemplo intentamos leer con recv() y no nos ha llegado ningun paquete, la funcion devolvera -1 y el programa continuara. Sin embargo esta es una mala practica ya que nuestro programa estara continuamiente ejecutando codigo, consumiendo mucho tiempo de CPU para al final no leer nada.

Asi pues lo ideal seria poder estar escuchando a diferentes evento al mismo tiempo en diferentes sockets y que cuando se produzca uno, se nos indique y podamos atenderlo. Para suplir esta necesidad tenemos la funcion select(), que sera la base de nuesto control y gestion de sockets a partir de ahora.

SELECT(): CONTROL Y GESTION DE SOCKETS

La función **select()** permite monitorizar eventos en diferentes descriptores simultáneamente.
Se trata de una funcion bloqueante que puede estar esperando la aparición de 3 tipos de sucesos:

- Lectura de un descriptor dado.
- Escritura de un descriptor dado.
- Exepcion en un descriptor dado.

Asi pues ahora vamos a ver como indicamos a esta funcion, los sockets y sus eventos asociado a los que tiene que estar escuchando. Para ello primero vamos a ver la declaracion de la funcion:
int select(int numfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);

Podemos desglosar sus parametros en 3 partes:
1) int numfds:
Aqui tenemos que poner el mayor descriptor de fichero fd mas 1, es decir $numfds = fdmax + 1$
Probablemente el select() cree un nuevo descriptor de fichero con ese numero (guess)
2) fd_set *readfds, *writefds, *exceptfds:
Estas son las estructuras de informacion donde indicaremos para cada socket, a que evento del mismo tenemos que estar escuchando (de entre los 3 posibles: Lectura, escritura o excepción).
A nivel interno son un array de bits que indica para cada descriptor de fichero, si se esta escuchando (1) o no (0) al evento que representa dicha estructura (lectura, escritura o exepcion). Asi pues nosotros dispondremos de 3 estructuras "fd_set", una para cada tipo de evento en donde hemos indicado a que sockets estamos escuchando; al llamar a select() le pasamos dichas "fd_set" y la funcion bloquea el flujo del programa a la espera de alguno de los eventos que hemos puesto a la escucha.

Array fd_leer

descriptor					12							5		3			
bits	....	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Existen una serie de funciones (macros) con las que podemos operar con estas estructuras para anadir o quitar descriptores de fichero, estas funciones son:

FD_ZERO(fd_set *set): Pone a 0 el "set" dado. Por lo que no habra ningun fd indicado en el
FD_SET(int fd, fd_set *set): Anade un fd (socket en nuestro caso) al set
FD_CLR(int fd, fd_set *set): Quita un fd (socket en nuestro caso) al set
FD_ISSET(int fd, fd_set *set) : Comprueba si un fd esta activo en el set dado.

Cuando se produzca uno de los eventos, la funcion select() **MODIFICARA** las estructuras readfds,writefds y exceptfds que le hemos pasado, poniendo un 1 en aquello/s descriptores que han ocasionado el evento. Por lo tando, a continuacion del select() usamos FD_ISSET() para comprobar cual ha sido el evento que ha tenido lugar y actuar en consecuencia.
Como select() **MODIFICARA** las estructuras, si una vez atendido el evento queremos volver a estar escuchando a los mismos eventos de los mismos descriptores de fichero, no podemos volver a pasarle las estructuras tal cual ya que no seran las mismas que en la llamada anterior a la funcion.

Normalmente lo que se hace es tener las estructuras duplicadas, una *"master"* que contiene los eventos a los que queremos escuchar y que no pasamos a la funcion select() y otra *"temp"* que es una copia de la *"master"* que si que pasamos a la funcion select(). Tras atender a cada evento, lo que debemos hacer es copiar el "*master*" en el *"temp"* y volver a pasar los slaves a la funcion select().

Si alguno de estos eventos no nos interesa utilizamos NULL como parametro.

3) struct timeval *timeout:
Dado que el select() es una funcion bloqueante, el flujo del programa no seguira hasta que ocurra alguno de los eventos a los que escucha. Podemos indicar un "timeout" pasado el cual salimos de la funcion select() para por ejemplo enviar la informacion "still waitiong".

La estructura timeval tiene los siguientes atributos con los que indicamos el timeout:

```
struct timeval {
        int tv_sec;             // seconds
        int tv_usec;            // microseconds
};
```

Con tv_sec indicamos los segundo del timeout y con tv_usec los microsegundos, si bien como en entornos Unix las ranuras de tiempo suelen ser de 100 ms, el timeout minimo podria ser este independientemente de lo que indiquemos en la estructura. Si no queremos timeout indicamos NULL.

1. CHAT BROADCAST CENTRALIZADO

En esta primera aproximación, hemos creado un chat centralizado donde cada usuario (cliente) se conecta a un servidor hacia el cual envía todos sus mensajes. El servidor, al recibir un mensaje de uno de los clientes, lo reenvía al resto de usuarios. Tanto el cliente como el servidor deben ser capaces de atender eventos provenientes de diferentes fuentes (descriptores de fichero fd) por lo cual no podemos utilizar funciones bloqueantes, en su lugar, ambos programas hacen uso de la función select():

- El servidor está a la espera de un evento en cualquiera de sus clientes
- El cliente está a la espera de un evento del servidor o del teclado (fd_STDIN = 0)

A grandes rasgos, al ejecutarse el servidor, este se pone a la escucha en un determinado puerto usando ya la función select(), cuando le llega una conexión al socket la acepta e indica a la próxima llamada a la función select() que se escuche a ese socket() modificando la estructura "fd_set master_read_fds".
Cuando un usuario envía un mensaje al servidor, la función select() indica al servidor de quien es y este lo reenvía al resto de usuarios. Por su parte el usuario simplemente tiene que conectarse al servidor y enviarle paquetes que serán retransmitidos por este al resto de usuarios, cuando recibe un mensaje por teclado lo envía al servidor y cuando lo recibe del servidor lo escribe en pantalla.

Las mejoras que hemos realizado sobre la proposición inicial son:

1- El cliente escribe la fecha y hora del mensaje recibido:
   Cuando un cliente recibe un mensaje del servidor obtiene la hora del sistema y la imprime antes del mensaje, de esta manera sabemos a qué hora nos llegan los mensajes.
2- Los usuarios se registran con un nickname:
   Cuando un usuario se conecta al servidor, este guarda su IP y puerto para identificarle y le pide que le envíe un nickname con el cual identificarse en el chat. Cuando dicho cliente envía un mensaje al servidor, este obtiene su nickname guardado a partir del descriptor del socket del usuario y lo añade al principio del mensaje a enviar. De esta manera sabemos quién escribe cada mensaje.
3- Cuando el servidor se desconecta, los usuarios finalizan su ejecución.

1.1 Cliente:

El código del cliente es:

```
#include "sockets_cab.h"

int main(int argc, char **argv) {
  int errores,i,j; // IP or name of the server we will connect to
  char * Server_port; // Port of the server we will connect to;
  char * Server_id; // Port of the server we will connect to;
  int nbytes, len;
  struct   sockaddr_in Server_addr ;  // Structure with the data about socket
  int  descSocket ;           // Socket descriptor
  int fdmax;    // maximum file descriptor number
  char send_buff[256];        // Send buffer
  char recv_buff[256];        // Send buffer
  struct hostent *he;        // Holds the addr of Server in case we are given an URL

  time_t fechaActual ;
  struct tm * fechaPtr;

  fd_set master_read_fds, read_fds;  // Master and temp file descriptor read list for select()
  fd_set master_write_fds, write_fds;  // Master and temp file descriptor write list for select()
  fd_set master_exept_fds, exept_fds;  // Master and temp file descriptor exept list for select()



  // ********* Process the given parameters  ********
  if (argc < 3) {    // If we are not given the basic parameters
    printf("Not enough parameters \n");
    exit(-1);
  }
```

```
 Server_id = argv[1];    // Get the server addr
  if ((Server_id[0] < '0')||(Server_id[0] > '9')){ // If we are given a name and not IP
    if ((he =gethostbyname(argv[1])) == NULL) {
      herror("gethostbyname");
      exit(1);
    }
    Server_id = inet_ntoa(*((struct in_addr *)he->h_addr)) ;
  }
  Server_port = argv[2];


  // ***** Open the socket and checks it went well *****
  descSocket = socket(AF_INET, SOCK_STREAM, 0); // Socket IP, TCP
  if (descSocket== -1) {
    fprintf(stderr, "CLIENTE: No se ha podido abrir el socket \n") ;
    exit(-1) ;
  }
  fprintf(stderr, "CLIENTE: Socket abierto \n") ;

  // Initialize the Server addr structure to connect server
  bzero((char *) &Server_addr, sizeof(Server_addr));  // Set everything to 0
  Server_addr.sin_family = PF_INET;       //
  Server_addr.sin_addr.s_addr= inet_addr(Server_id);  // IP of the machine we will connect with
  Server_addr.sin_port = htons(atoi(Server_port));  // Port of the machine we will connect to

  // Connect to the server
  errores = connect(descSocket,(struct sockaddr *) &Server_addr, sizeof(Server_addr));
  if ( errores ==-1) {
    fprintf(stderr, "CLIENTE: No se ha podido conectar con servidor\n") ;
    exit(-1) ;
  }
  printf("CLIENTE: Conexion establecida con %s en el puerto %s \n", Server_id, Server_port) ;


//*****************************************************************************************
  // Now we are connected to the Server. Since recv() is a bloquing function and we have to be
  // able to send data any time we want we will use the function select() with the sockets and
  // the keyboard (STDIN). This way we can read from the keyboard and the socket

  // Reset the fds event structures
  FD_ZERO(&master_read_fds);
  FD_ZERO(&master_write_fds);
  FD_ZERO(&master_exept_fds);

  // Add the Socket and STDIN to the master set
  FD_SET(descSocket, &master_read_fds);
  FD_SET(STDIN, &master_read_fds);

  // keep track of the biggest file descriptor
  fdmax = descSocket; // so far, it's this one

  while(1){
    read_fds = master_read_fds;
    write_fds = master_write_fds;
    exept_fds = master_exept_fds;
    errores = select(fdmax+1, &read_fds, NULL, NULL, NULL) ;
    if (errores == -1) {
      perror("select");
      exit(1);
    }

    // run through the existing connections looking for data to read
    for(i = 0; i <= fdmax; i++) {
      if (FD_ISSET(i, &read_fds)) {   // Check for reading on every socket
        if (i == descSocket) {  // SERVER SOCKET CASE !!!!!
          nbytes = recv(i, recv_buff, sizeof(recv_buff), 0); // Read data
          if (nbytes<= 0) {    // Connection closed
      //*************************************************************************
      //************************** SERVER DOWN **********************************
      //*************************************************************************
            printf("Servidor desconectado. Cerramos conexion \n");
            if (nbytes< 0)  {
              perror("recv");
            }
          close(i); // bye!
          exit(0);
```

```
          }
          else {        // we got some data from a client
      //*************************************************************************
      //************************** PRINT RECEIVED MESSAGE ************************
      //*************************************************************************
            fechaActual = time(0) ;    // Get the time and print it
            fechaPtr = gmtime(&fechaActual) ;
            printf("%i/%i/%i (%i:%i:%i) ", fechaPtr->tm_mday, fechaPtr->tm_mon + 1,
            fechaPtr->tm_year + 1900, fechaPtr->tm_hour, fechaPtr->tm_min, fechaPtr->tm_sec);

            recv_buff[nbytes] = 0;
            printf("%s \n", recv_buff);  // Print the message

            }
        }// Received data from server
        if (i == STDIN) {  // RECEIVED DATA FROM KEYBOARD
      //*************************************************************************
      //****************** SEND KEYBOARD MESSAGE TO SERVER **********************
      //*************************************************************************

          // We only get here when [ENTER] is pressed
          fgets(send_buff, sizeof(send_buff), stdin);

          len = strlen(send_buff) - 1;
          if (send_buff[len] == '\n'){
            send_buff[len] = '\0';
          }

          if (send(descSocket, send_buff, len, 0) == -1) {
            perror("send");
          }

        }   // Received data from keyboard
      }    // If its the origin of the event
    }       // For every possible socket
  }      // While(1)
    return 0;
    close(descSocket);
}
```

1.2 Servidor:

```
#include "sockets cab.h"
#define NUM USERS 64
int main(int argc, char **argv) {
  char * Server_port;
  int i,j;
  struct sockaddr_in Server_addr, Clien_addr;
  int   listenSocket;    //Socket (fd) used to listen for connections
  int   newfd; // newly accept()ed socket descriptor
  int   aux_int = 0 ;
  int fdmax;    // maximum file descriptor number

  fd_set master_read_fds, read_fds;  // Master and temp file descriptor read list for select()
  fd_set master_write_fds, write_fds;  // Master and temp file descriptor write list for select()
  fd_set master_exept_fds, exept_fds;  // Master and temp file descriptor exept list for select()


  char buf[256];  //buffer for client data
  char buf2[256];  //buffer for client data
  int nbytes;
  socklen_t addrlen;

  char   Client_IPs[NUM_USERS][16];    // IP's of the users
  int   Client_Ports[NUM_USERS];    // Ports of the users
  char   Client_nicks[NUM_USERS][16];  // Nicknames of the users

  for (i = 0; i < NUM_USERS; i++){
    Client_IPs[i][0] = 0;
    Client_Ports[i] = 0;
    Client_nicks[i][0] = 0;
   }

  char *get_nickname_s = {"Log in with nickname: "};
```

```
char *got_nickname_s = {"Nickname Set: "};
// ********* Process the given parameters  ********
if (argc < 2) {
  printf("Not enough parameters \n");
  exit(-1);
}
Server_port = argv[1];

// Open the socket and checks it went well
listenSocket = socket(AF_INET, SOCK_STREAM, 0); // Socket IP, TCP
if (listenSocket < 0) {
  fprintf(stderr, "SERVIDOR: no se ha podido abrir el socket \n") ;
  exit(-1) ;
}
// Initialize the server addr structure
bzero ((char*) &Server_addr, sizeof(Server_addr));
Server_addr.sin_family = AF_INET;
Server_addr.sin_addr.s_addr= htonl(INADDR_ANY);
Server_addr.sin_port = htons(atoi(Server_port));

// Stablish TCP port of the server
if (bind(listenSocket,(struct sockaddr *) &Server_addr,sizeof(Server_addr))< 0) {
  fprintf(stderr, "SERVIDOR: error al vincular la direccion local\n") ;
  exit(-1) ;
}
// Listen to incoming TCP conenctions (5 max in the queue)
listen(listenSocket, 15);
printf("Servidor escuchando en el puerto %s \n", Server_port);

//*********************************************************************************************
// Now we are listening to incoming connections, instead of using the bloquing function accept()
// straight away we will use select(). This way we can still attend new connections and respond to
// the ones we have already made.

// Reset the fds event structures
FD_ZERO(&master_read_fds);
FD_ZERO(&master_write_fds);
FD_ZERO(&master_exept_fds);

// add the listenSocket to the master set
FD_SET(listenSocket, &master_read_fds);

// keep track of the biggest file descriptor
fdmax = listenSocket; // so far, it's this one


while (1){
  read_fds = master_read_fds;
  write_fds = master_write_fds;
  exept_fds = master_exept_fds;

  if (select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1) {
    perror("select");
    exit(1);
  }
  // run through the existing connections looking for data to read
  for(i = 0; i <= fdmax; i++) {
    if (FD_ISSET(i, &read_fds)) {   // Check for reading on every socket


      if (i == listenSocket) {  // NEW CONECTION CASE !!!!!
    //*************************************************************************
    //*********************** NEW CLIENT CONNECTION ***************************
    //*************************************************************************
        addrlen = sizeof(Clien_addr);
        newfd = accept(listenSocket, (struct sockaddr *)&Clien_addr,&addrlen);
        if (newfd == -1) {
          perror("accept");
        }
        else {
          FD_SET(newfd, &master_read_fds); // add to master set
          if (newfd > fdmax) {    // keep track of the maximum fd
            fdmax = newfd;
          }
                    // get IP and port of the user
          strcpy(Client_IPs[newfd],inet_ntoa(((struct in_addr)Clien_addr.sin_addr)));
          Client_Ports[newfd] = (int)Clien_addr.sin_port;
```

```
            printf("Conexion establecida con %s en el puerto %i \n",
                      Client_IPs[newfd], Client_Ports[newfd]);
            if (send(newfd, get_nickname_s, strlen(get_nickname_s), 0) == -1) {
              perror("send");
            }
          }
        }
        else {  // RECEIVED DATA FROM CLIENT
          nbytes = recv(i, buf, sizeof(buf), 0); // Read data

          if (nbytes<= 0) {        // Connection closed
      //*************************************************************************
      //******************** CLIENT CLOSED CONNECTION ***************************
      //*************************************************************************

            printf("Cliente %s [%i] se ha desconectado \n",Client_IPs[i], Client_Ports[i]);
            if (nbytes< 0)  {
              perror("recv");
            }
            close(i); // bye!
            FD_CLR(i, &master_read_fds); // remove from master set
            Client_nicks[i][0] = 0;    // Erase nickname
          }
          else {        // we got some data from a client

            if (Client_nicks[i][0] == 0) {   // If the info we received is to set

      //*************************************************************************
      //************************ GET CLIENTS NICKNAME ***************************
      //*************************************************************************
              buf[nbytes] = 0;
              strcpy(Client_nicks[i],buf);

              strcpy(buf ,got_nickname_s);
              strcpy(buf + strlen(buf) ,Client_nicks[i]);

              if (send(i, buf, strlen(buf), 0) == -1) {
                perror("send");
              }
            }else{

      //*************************************************************************
      //******************** BROADCAST CLIENT MESSAGE ***************************
      //*************************************************************************
            buf2[0] = '[';
            buf[nbytes] = 0;

            strcpy(buf2 + 1,Client_nicks[i]);  // Add the nickname to the message
            aux_int = strlen(buf2);
            buf2[aux_int ] = ']';
            buf2[aux_int +1] = ' ';
            buf2[aux_int +2] = 0;

            strcpy(buf2 + strlen(buf2) ,buf);

            for(j = 0; j <= fdmax; j++) {  // send to everyone!

              if (FD_ISSET(j, &master_read_fds)) {

                if (j != listenSocket && j != i) {// except the listener and ourselves
                  if (send(j, buf2, strlen(buf2), 0) == -1) {
                    perror("send");
                  }
                }
              }
            }} // Else
          }
        }   // Received data from client
      }    // If its the origin of the event
    }       // For every possible socket
  }      // While(1)
    return 0;
    close(listenSocket);
    close(newfd);
}
```

2. CHAT HIBRIDO PEER TO PEER

Es esta segunda implementación del chat tenemos un diseño hibrido en el cual se dispone de un Servidor cuya función principal es la de proporcionar el listado de Usuarios conectados (Peers).
Cuando un nuevo Peer se conecta al servidor, este enviará sus datos al mismo (IP, Puerto a la escucha y nick) para que el servidor tenga constancia de los mismos cuando un Peer le pida la lista de usuarios.

El servidor deberá enviar dicha lista a los clientes que la soliciten para que estos puedan conectarse a otros usuarios directamente. Esta lista contendrá 3 parámetros por cada cliente:

- IP: IP del cliente
- Puerto a la escucha: Dado que los clientes se conectaran unos con otros, estos tambien deben tener un puerto a la escucha
- Nickname: Identificador único para cada cliente conectado

Un cliente se puede identificar unívocamente tanto por el par {IP + Puerto} como por el {nickname}.
Una vez el cliente se identifica al servidor mediante su nickname y le pasa el puerto al que está escuchando, este puede ejecutar una serie de comandos para crear/destruir conexiones con Peers y para obtener información sobre sus conexiones actuales. Los comandos empiezan con el carácter '$'. Si la palabra tecleado por un Peer no es un comando, esta será un mensaje que se enviará a todos los Peers a los que esté conectado.

2.1 Funciones y utilidades

Para hacer la programación más fácil y legible hemos implementado las siguientes funciones:

int listen_TCP_Port (int Port, int max_conn);
Esta función se pone a escuchar al puerto Port aceptando un número máximo de conexiones pendientes en buffer de max_conn. Devuelve el descriptor de socket o -1 si ha habido error.

int connect_machine (char *IP, int Port):
Esta función se conecta a la maquina con dirección IP y puerto Port dados como parámetros.
Devuelve el descriptor de socket o -1 si ha habido error.

int sendall(int socket_fd, char *buf, int len, int FLAGS):
Dado que la función send() no garantiza el envió total de la información que le pasamos,
esta función sí que lo hace, recibiendo los mismos parámetros que la función send().

El código de estas funciones es:

```
int listen_TCP_Port (int Port, int max_conn){
        struct sockaddr_in Server_addr ;     // Structure with the data about socket
        int     listenSocket ;                                  // Socket descriptor
        int errores;
                // Open the socket and checks it went well
        listenSocket = socket(AF_INET, SOCK_STREAM, 0); // Socket IP, TCP
        if (listenSocket < 0) {
                fprintf(stderr, "SERVIDOR: no se ha podido abrir el socket \n") ;
                exit(-1) ;
        }
        // Initialize the server addr structure
        bzero ((char*) &Server_addr, sizeof(Server_addr));
        Server_addr.sin_family = AF_INET;
        Server_addr.sin_addr.s_addr= htonl(INADDR_ANY);
        Server_addr.sin_port = htons(Port);

        // Bind address to machine
        errores = bind(listenSocket,(struct sockaddr *) &Server_addr,sizeof(Server_addr));
        if (errores < 0) {
                fprintf(stderr, "Error al vincular la direccion local\n") ;
                return -1;
        }
```

```
        // Listen to incoming TCP conenctions (5 max in the queue)
        errores = listen(listenSocket,  max_conn);
        printf("Somos %s escuchando en el puerto %i \n",
                inet_ntoa(((struct in_addr)Server_addr.sin_addr)) ,Port);

        if (errores < 0) {
                fprintf(stderr, "Error al ponerse a escuchar\n") ;
                return -1 ;
        }
        return listenSocket;
  }

int connect_machine (char *IP, int Port){
        struct  sockaddr_in machine_addr ;    // Structure with the data about socket
        int     descSocket ;                                  // Socket descriptor
        int errores;

// ***** Open a socket and checks it went well *****
        descSocket = socket(AF_INET, SOCK_STREAM, 0); // Socket IP, TCP
        if (descSocket== -1) {
                fprintf(stderr, "No se ha podido abrir el socket \n") ;
                return -1 ;
        }
        fprintf(stderr, "Socket abierto \n") ;

        // Initialize the Server addr structure to connect server
        bzero((char *) &machine_addr, sizeof(machine_addr));       // Set everything to 0
        machine_addr.sin_family = PF_INET;                         //
        machine_addr.sin_addr.s_addr= inet_addr(IP);// IP of the machine we will connect with
        machine_addr.sin_port = htons(Port); // Port of the machine we will connect to

        // Connect to the machine
        errores = connect(descSocket,(struct sockaddr *) &machine_addr, sizeof(machine_addr));
        if ( errores ==-1) {
                fprintf(stderr, "No se ha podido conectar con servidor\n") ;
                return -1;
        }
        printf("Conexion establecida con %s en el puerto %i \n", IP, Port) ;

                 return descSocket;
  }

int sendall(int socket_fd, char *buf, int len, int FLAGS) {
        int bytes_sent = 0;    // how many bytes we've sent
        int n = 0;
        while(bytes_sent < len) {
                n = send(socket_fd, buf + bytes_sent, len - bytes_sent, FLAGS);
                if (n == -1) {
                        break;
                }
                bytes_sent += n;
        }
        if (n != -1) {
                return bytes_sent;
        }else {
                return -1;
        }
}
```

2.2 Servidor

El servidor al iniciarse se pone a escuchar al puerto dado como parámetro.
Cuando un usuario se conecta al servidor, este guardara sus datos en las estructuras:

```
char Client_IPs[NUM_USERS][16];          // IP's of the users
int Client_Ports[NUM_USERS];             // Ports of the users
char Client_nicks[NUM_USERS][16];        // Nicknames of the users

int num_connected;
```

La posición dentro de los arrays está dada por el número de descriptor de socket del usuario.
Al conectarse el usuario al servidor:

- El servidor guarda su IP en la estructura y espera a recibir el puerto al que el cliente esta a la escucha. Cuando el cliente se lo envía, el servidor lo guarda y le envía un mensaje pidiéndole un nickname. Cuando el servidor lo recibe, comprueba que no haya ningún usuario con dicho nickname, de haberlo vuelve a solicitar un nuevo nickname, en caso contrario, lo guarda y asocia a la IP y puerto obtenidos anteriormente.

- La única otra función que esta implementada en el servidor es la $show_users. Cuando el servidor recibe esta cadena, genera la lista de usuarios conectados y se la envía al cliente que la solicita según el formato:

num_users Dir1 Port1 Nick1, Dir2 Port2 Nick2...

Sera tarea del cliente tratar el mensaje de acuerdo con este formato.

Si un cliente se desconecta del servidor, este elimina sus datos de la estructura.
El código del servidor es:

```
#include "sockets_cab.h"
void get_word_array(char *cadena, int *num_palabras);
char words_list[20][20];
char aux_string[20];

int main(int argc, char **argv) {
char * Server_port;
int i,j;
struct sockaddr_in Clien_addr;
int      listenSocket;            //Socket (fd) used to listen for connections
int      newfd; // newly accept()ed socket descriptor
int      aux_int = 0 ;
int flag_aux;
int num_palabras;

int num_users = 0;               // Number of users in the system.

fd_set master_read_fds, read_fds; // Master and temp file descriptor read list for select()
fd_set master_write_fds, write_fds;       // Master and temp file descriptor write list for select()
fd_set master_exept_fds, exept_fds;       // Master and temp file descriptor exept list for select()

int fdmax;               // maximum file descriptor number


char buf[256];  //buffer for client data
char buf2[256];  //buffer for client data
int nbytes;
socklen_t addrlen;

char Client_IPs[NUM_USERS][16];           // IP's of the users
int Client_Ports[NUM_USERS];              // Ports of the users
char Client_nicks[NUM_USERS][16]; // Nicknames of the users

char *get_nickname_s = {"Log in with nickname: "};
char *got_nickname_s = {"Nickname Set: "};
char *invalid_nick_s = {"Invalid nickname, try again "};


// ********* Process the given parameters  ********
if (argc < 2) {
printf("Not enough parameters \n");
exit(-1);
}
Server_port = argv[1];

// Initialice variables

for (i = 0; i < NUM_USERS; i++){
Client_IPs[i][0] = 0;
Client_Ports[i] = 0;
Client_nicks[i][0] = 0;
}

// Set the machine listening to the gien port.
listenSocket = listen_TCP_Port (atoi(Server_port), 20);

// Reset the fds event structures
FD_ZERO(&master_read_fds);
FD_ZERO(&master_write_fds);
FD_ZERO(&master_exept_fds);

// add the listenSocket to the master set
FD_SET(listenSocket, &master_read_fds);
```

```
// keep track of the biggest file descriptor
fdmax = listenSocket; // so far, it's this one


while (1){
read_fds = master_read_fds;
write_fds = master_write_fds;
exept_fds = master_exept_fds;

if (select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1) {
perror("select");
exit(1);
}
// run through the existing connections looking for data to read
for(i = 0; i <= fdmax; i++) {
if (FD_ISSET(i, &read_fds)) {     // Check for reading on every socket


if (i == listenSocket) {  // NEW CONECTION CASE !!!!!
//*************************************************************************
//************************ NEW CLIENT CONNECTION ***************************
//*************************************************************************
        addrlen = sizeof(Clien_addr);
        newfd = accept(listenSocket, (struct sockaddr *)&Clien_addr,&addrlen);
        if (newfd == -1) {
                perror("accept");
        }
        else {
                num_users++;
                FD_SET(newfd, &master_read_fds); // add to master set
                if (newfd > fdmax) {              // keep track of the maximum fd
                        fdmax = newfd;
                }
                strcpy(Client_IPs[newfd],inet_ntoa(((struct in_addr)Clien_addr.sin_addr))) ;// IP of the user
                printf("Conexion establecida con %s en el puerto %i \n",
                        Client_IPs[newfd], (int)Clien_addr.sin_port);
        }
}

else {   //
//*************************************************************************
//********************* RECEIVED DATA FROM CLIENT **************************
//*************************************************************************
        nbytes = recv(i, buf, sizeof(buf), 0); // Read data
        buf[nbytes] = 0;

        if (nbytes<= 0) {                                 // Connection closed
//********************* CLIENT CLOSED CONNECTION ***************************

                printf("Cliente %s con dir %s [%i] se ha desconectado \n",
                        Client_nicks[i], Client_IPs[i], Client_Ports[i]);
                if (nbytes< 0)  {
                        perror("recv");
                }
                close(i); // bye!
                num_users--;
                FD_CLR(i, &master_read_fds); // remove from master set
                Client_nicks[i][0] = 0;           // Erase
                Client_IPs[i][0] = 0;             // Erase
                Client_Ports[i] = 0;              // Erase
        }
        else {                            // we got some data from a client

                if (Client_Ports[i] == 0){        // CHECK WE GOT THE PORT

// ********************* GET CLIENTS LISTENING PORT ****************************
                        Client_Ports[i] = atoi(buf);

                        if (sendall(newfd, get_nickname_s, strlen(get_nickname_s), 0) == -1) {
                                perror("sendall");
                        }
                }

                else if (Client_nicks[i][0] == 0) { // CHECK WE GOT THE NICKNAME

// ************************* GET CLIENTS NICKNAME ******************************

                        flag_aux = 0;
                        for (j = 0; j <= fdmax; j++){ // Check uniqueness of nickname
                                if (strcmp(Client_nicks[j], buf) == 0){
                                        flag_aux = 1;
                                        break;
                                }
                        }
                        if (flag_aux == 1){       // Not valid nickname
                                if (sendall(i, invalid_nick_s , strlen(invalid_nick_s ), 0) == -1) {
                                        perror("sendall");
```

```
                                }
                        } else {
                                strcpy(Client_nicks[i],buf);

                                strcpy(buf ,got_nickname_s);
                                strcpy(buf + strlen(buf) ,Client_nicks[i]);

                                if (sendall(i, buf, strlen(buf), 0) == -1) {
                                        perror("sendall");
                                }
                                printf("Usuario %s, con dir %s [%i] registrado \n",
                                        Client_nicks[i], Client_IPs[i], Client_Ports[i]);
                        }
                }

//**************************************************************************
//************************ GET COMMAND   ***********************************
//**************************************************************************
                else if (buf[0] == '$'){
                                get_word_array(buf+1, &num_palabras);
                        //      printf("Primer comando %s\n", words_list[0]);
//************************ SHOW USERS   ************************************
// Send de users as: num users Dir1 Port1 Nick1, Dir2 Port2 Nick2...
// We dont send the user who asks
                                if (strcmp(words_list[0],"show_users")==0){
                                        printf("Lista pedida \n");
                                        aux_int  = 0;
                                        conversor_IaS(num_users - 1,aux_string); // Convertto string
                                        strcpy(buf2 + aux_int,aux_string );
                                        aux_int += strlen(aux_string );
                                        buf2[aux_int++] = ' ';
                                        for (j = 0; j <= fdmax; j++) {
                                                if (j != i) {                           // If the user isnt
the one who asked

                                                if (Client_IPs[j][0] != 0) {      // If it exists
                                        //              printf("Encontrado usuario %s, %s
\n",Client_IPs[j],Client_nicks[j]);
                                                        strcpy(buf2 + aux_int, Client_IPs[j]);
                                                        aux_int += strlen(Client_IPs[j]);
                                                        buf2[aux_int++] = ' ';


                                                        conversor_IaS(Client_Ports[j],aux_string); // Convert
Port to string
                                                        strcpy(buf2 + aux_int, aux_string);
                                                        aux_int += strlen(aux_string);
                                                        buf2[aux_int++] = ' ';


                                                        strcpy(buf2 + aux_int, Client_nicks[j]);
                                                        aux_int += strlen(Client_nicks[j]);
                                                        buf2[aux_int++] = ' ';


                                                }
                                        }}
                                        buf2[aux_int] = 0;
                                if (sendall(i, buf2 , strlen(buf2), 0) == -1) {
                                        perror("sendall");
                                }

                                }
                        }
                        else {
//**************************************************************************
//************************ PLAIN TEXT RECEIVED FROM CLIENT *******************
//**************************************************************************


        } }// Else
}       // Received data from client
}               // If its the origin of the event
}                       // For every possible socket
}                       // While(1)
return 0;
close(listenSocket);
close(newfd);
}




void get_word_array(char *cadena, int *num_palabras){
int i = 0, num_p = 0;
char *p;
p = cadena;
while (*p != 0) {
```

```
if (*p == ' '){
if (i != 0){  // To avoid multiple ' ' in a row error
words_list[num_p][i] = 0;
i = 0;
num_p++;
}
} else {
words_list[num_p][i] =*p ;
i++;
}
p++;
}

words_list[num_p][i] = 0;
num_p++;
*num_palabras = num_p;
}
```

2.3 Cliente

En esta implementación los peers también deben tener un puerto a la escucha para poder conectarse unos a otros. Así pues al iniciarse el cliente, este se pone a escuchar al puerto dado y se conecta al servidor, enviándole también el puerto de escucha. Después el servidor le pedirá el nickname y cuando el cliente proporcione uno libre, el Servidor lo registrará.
Los comandos que puede utilizar el cliente son:

$show_users: Ensena lista de usuarios:
El servidor nos enviara un listado con {IP + Puerto} + Nickname de todos los usuarios.
Para cumplir con la especificacion de la practica el servidor enviara primero el mensaje con:
num dir_ip1 port1 dir_ip2 port2 ... dir_ipN portN nickname1 nickname2... nicknameN
Cuando el usuario consigue dicha info la guarda en los arrays.

$connect {IP + Puerto} o Nickname. Conectarse a un usuario para hablarle:
Un usuario se puede conectar a cualquiera de ellos mediante la orden $connect {IP + Puerto} o Nickname. Dicho mensaje hara que el usuairo se conecte al peer deseado.
Los usuarios al conectarse se intercambian los nickname primero.

$connect_all: Conectarse a todos los usuarios a los que no esté conectado.

$discconect {IP + Puerto} o Nickname. Desconectarse de dicho usuario

$user MENSAJE: Enviar un mensaje a un solo usuario.
Si estamos conectados a varios usuarios, los mensajes que enviemos serán enviados a todos ellos, para enviar un mensaje a sólo un usuario dado, escribimos su Nick precedido por '$'.

$exit: Cierra el cliente.

El código del cliente es:

```
#include "sockets_cab.h"

#define IDLE 0
#define SHOW_USERS 1
#define CONNECT_ALL 1
char *get_nickname_s = {"Log in with nickname: "};
char *got_nickname_s = {"Nickname Set: "};
char *invalid_nick_s = {"Invalid nickname, try again "};

char words_list[20][20];
int num_palabras;

char nickname[30];

// USERS WE ARE CONNECTED TO ORDERED BY "FILE DESCRIPTOR NUMBER"
char Client_IPs[NUM_USERS][16];           // IP's of the users
int Client_Ports[NUM_USERS];              // Ports of the users
```

```
char Client_nicks[NUM_USERS][16]; // Nicknames of the users

// USERS ONLINE IN THE SYSTEM
char Peer_IPs[NUM_USERS][16];             // IP's of the users
int Peer_Ports[NUM_USERS];                // Ports of the users
char Peer_nicks[NUM_USERS][16];   // Nicknames of the users

int num_users = 0;                                          // Number of users in the system.

void get_word_array (char *cadena, int *num_palabras);
int connect_user (char *user);

int fdmax;              // maximum file descriptor number

int main(int argc, char **argv) {
int errores,i,j,k; // IP or name of the server we will connect to
char * Server_port; // Port of the server we will connect to;
char * Server_id; // Port of the server we will connect to;
char * Our_Port;
int order_state = IDLE;

char got_nickname =0;     // Flag que nos dice si tenemos nickname o no.
int nbytes, len;
int      descSocket,listenSocket, newfd ;                                    // Socket descriptor
struct sockaddr_in Server_addr, Clien_addr;
socklen_t addrlen;
int server_socket;
int aux_int = 0;

time_t fechaActual ;
struct tm * fechaPtr;

char send_buff[256];                            // Send buffer
char recv_buff[256];                            // Send buffer

struct hostent *he;                             // Holds the addr of Server in case we are given an URL


fd_set master_read_fds, read_fds; // Master and temp file descriptor read list for select()
fd_set master_write_fds, write_fds;       // Master and temp file descriptor write list for select()
fd_set master_exept_fds, exept_fds;       // Master and temp file descriptor exept list for select()


// ********* Process the given parameters  ********
if (argc < 4) {           // If we are not given the basic parameters
printf("Not enough parameters \n");
exit(-1);
}

Server_id = argv[1];              // Get the server addr
if ((Server_id[0] < '0')||(Server_id[0] > '9')){ // If we are given a name and not IP
if ((he =gethostbyname(argv[1])) == NULL) {
herror("gethostbyname");
exit(1);
}
Server_id = inet_ntoa(*((struct in_addr *)he->h_addr)) ;
}
Server_port = argv[2];
Our_Port = argv[3];


// Initialice variables

for (i = 0; i < NUM_USERS; i++){
Client_IPs[i][0] = 0;
Client_Ports[i] = 0;
Client_nicks[i][0] = 0;

Peer_IPs[i][0] = 0;
Peer_Ports[i] = 0;
Peer_nicks[i][0] = 0;
}

nickname[0]  = 0;

//  CONNECT TO THE SERVER !!!!!
server_socket = connect_machine (Server_id, atoi(Server_port));
fdmax = server_socket;
// Send our listening port
if (sendall(server_socket,Our_Port,strlen(Our_Port), 0) == -1) {
perror("sendall");
}

// LISTEN FOR OTHER USERS TO CONNECT IN THE SAME PORT AS THE SERVER
listenSocket = listen_TCP_Port (atoi(Our_Port), 20);

if (listenSocket > fdmax){
fdmax = listenSocket;
```

```
}

// Reset the fds event structures
FD_ZERO(&master_read_fds);
FD_ZERO(&master_write_fds);
FD_ZERO(&master_exept_fds);

// Add the Socket, listen and STDIN to the master set
FD_SET(server_socket, &master_read_fds);
FD_SET(STDIN, &master_read_fds);
FD_SET(listenSocket, &master_read_fds);


while(1){
read_fds = master_read_fds;
write_fds = master_write_fds;
exept_fds = master_exept_fds;

if (select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1) {
perror("select");
exit(1);
}

// run through the existing connections looking for data to read
for(i = 0; i <= fdmax; i++) {
if (FD_ISSET(i, &read_fds)) {     // Check for reading on every socket

        if (i == listenSocket) {  // NEW CONECTION CASE !!!!!
//*************************************************************************
//************************ NEW PEER CONNECTION ****************************
//*************************************************************************
                addrlen = sizeof(Clien_addr);
                newfd = accept(listenSocket, (struct sockaddr *)&Clien_addr,&addrlen);
                if (newfd == -1) {
                        perror("accept");
                }
                else {
                        FD_SET(newfd, &master_read_fds); // add to master set
                        if (newfd > fdmax) {              // keep track of the maximum fd
                                fdmax = newfd;
                        }
                        strcpy(Client_IPs[newfd],inet_ntoa(((struct in_addr)Clien_addr.sin_addr))) ;// IP of the
user
                        Client_Ports[newfd] = (int)Clien_addr.sin_port;            // Port of the user qe are
connected to

                        if (sendall(newfd,nickname,strlen(nickname), 0) == -1) {    // Send our nickname
                                perror("sendall");
                        }
                }
        }

        else if (i == server_socket) {    // SERVER SOCKET !!!!!

//*************************************************************************
//******************* RECEIVE INFO FROM SERVER ****************************
//*************************************************************************
                nbytes = recv(i, recv_buff, sizeof(recv_buff), 0); // Read data
                recv_buff[nbytes] = 0;

                if (nbytes<= 0) {                 // Connection closed

                        printf("Servidor desconectado. Cerramos conexion \n");
                        if (nbytes< 0)  {
                                perror("recv");
                        }
                close(i); // bye!
                exit(0);

                }
                else {            // PRINT SERVER MESSAGE
                        if (order_state == SHOW_USERS){
                                        // printf("%s \n", recv_buff);    // Print the message
                                        get_word_array (recv_buff, &num_palabras);
                                        num_users = atoi(words_list[0]);
                                        for (j = 0; j < num_users; j++){
                                                strcpy(Peer_IPs[j], words_list[1 + 3*j]);
                                                Peer_Ports[j] = atoi(words_list[2 + 3*j]);
                                                strcpy(Peer_nicks[j], words_list[3 + 3*j]);
                                        }
                                        for (j = 0; j < num_users; j++){
                                        printf("Usuario %s, con dir %s [%i]  \n",
                                                Peer_nicks[j], Peer_IPs[j], Peer_Ports[j]);
                                        }
                                order_state = IDLE;
                        } else {
                                fechaActual = time(0) ;           // Get the time and print it
                                fechaPtr = gmtime(&fechaActual) ;
```

```
                                printf("%i/%i/%i (%i:%i:%i) ", fechaPtr->tm_mday, fechaPtr->tm_mon + 1,
                                fechaPtr->tm_year + 1900, fechaPtr->tm_hour, fechaPtr->tm_min, fechaPtr-
>tm_sec);

                                recv_buff[nbytes] = 0;
                                printf("%s \n", recv_buff);       // Print the message

                                if (nickname[0]==0){
                                        if (recv_buff[0] == 'N'){// If we got valid nickname
                                                strcpy(nickname,recv_buff+strlen(got_nickname_s));
                                                // printf("Really got %s \n",nickname);
                                        }
                                }
                }
                }
        }// Received data from server

        else if (i == STDIN) {    // RECEIVED DATA FROM KEYBOARD

//**************************************************************************
//******************* RECEIVE INFO FROM STDIN ******************************
//**************************************************************************
                // We only get here when [ENTER] is pressed
                fgets(send_buff, sizeof(send_buff), stdin);

                len = strlen(send_buff) - 1;
                if (send_buff[len] == '\n'){
                        send_buff[len] = '\0';
                }
// ********************  COMMANDS ****************************
                // printf("recibida cadena \n");
                if (send_buff[0] == '$'){
                        // Get the command and parameters
                        get_word_array (send_buff+1, &num_palabras);

//***********************  $SHOW_USERS ***************************

                        if (strcmp(words_list[0],"show_users")==0){
                                if (sendall(server_socket, send_buff, len, 0) == -1) {
                                        perror("send");
                                }
                                // printf("Lista pedida \n");
                                order_state = SHOW_USERS;
                        }
//***********************  $SHOW_CONNECTED ***************************

                        if (strcmp(words_list[0],"show_connected")==0){

                                for (j = 0; j < NUM_USERS; j++){
                                        if (Client_nicks[j][0] != 0){
                                                printf("Conectado al Usuario %s, con dir %s [%i]  \n",
                                                Client_nicks[j], Client_IPs[j], Client_Ports[j]);
                                        }
                                }
                        }

//***********************  $CONNECT ***************************
                        else if (strcmp(words_list[0],"connect")==0){

                                if (num_palabras == 2) {
                                        descSocket = connect_user (words_list[1]);
                                }
                                if (num_palabras == 3) {
                                        descSocket= connect_machine (words_list[1], atoi(words_list[2]));
                                        strcpy(Client_IPs[descSocket],words_list[1]) ;  // IP of the user
                                        Client_Ports[descSocket] = atoi(words_list[2]); // Port of the user qe
are connected to
                                }
                                // Send nickname first as it s the protocol.
                                if (sendall(descSocket,nickname,strlen(nickname), 0) == -1) {       // Send our
nickname
                                        perror("sendall");
                                }
                                if (descSocket > fdmax){
                                        fdmax = descSocket;
                                }
                                FD_SET(descSocket, &master_read_fds); // add to master set
                        }
        //***********************  $CONNECT_ALL ***************************
                        else if (strcmp(words_list[0],"connect_all")==0){

                                for (j = 0; j < num_users; j++){  // For every available peer
                                        aux_int = 0;

                                        for (k = 0; k <= fdmax; k++){     // For every possible socket
                                                if (strcmp(Peer_nicks[j],Client_nicks[k])==0){     // If we are
not already connected
                                                        aux_int = 1;
```

```
                                                        break;
                                                }
                                        }
                                        if (aux_int == 0){        // If we are not connected to that peer
                                                descSocket = connect_user (Peer_nicks[j]);

                                        // Send nickname first as it s the protocol.
                                        if (sendall(descSocket,nickname,strlen(nickname), 0) == -1) {       //
Send our nickname
                                                perror("sendall");
                                        }
                                        if (descSocket > fdmax){
                                                fdmax = descSocket;
                                        }
                                        FD_SET(descSocket, &master_read_fds); // add to master set


                                        }
                                }
                        }
        //**********************  $DISCONNECT ***************************
                        else if (strcmp(words_list[0],"connect")==0){

                        }

        //**********************  $EXIT ***************************
                        else if (strcmp(words_list[0],"exit")==0){
                                exit(0);
                        }
        //**********************  $USER [Mensaje] ***************************
                        else {
                                for (j = 0; j <=fdmax; j++){      // For evey connected peer
                                        if (strcmp(words_list[0],Client_nicks[j]) == 0){ // If it s its nick
                                        aux_int = 1 + strlen(words_list[0]);
                                                if (sendall(j, send_buff + aux_int, len- aux_int, 0) == -1) {
                                                        perror("send");
                                                }
                                        }

                                }
                        }
                }
                else {

// SEND KEYBOARD MESSAGE TO SERVER IF ITS THE NICKNAME
                        if (sendall(server_socket, send_buff, len, 0) == -1) {
                                perror("send");
                        }
// SEND KEYBOARD MESSAGE TO OTHER USERS CONNECTED
                        for(j = 0; j <= fdmax; j++) {     // sendall to everyone!

                                if (FD_ISSET(j, &master_read_fds)) {

                                        if ((j != listenSocket) && (j != i) && (j != server_socket)) {// except
the listener and ourselves
                                                //      printf("Enviado al puerto %i",Client Ports[j]);
                                                if (sendall(j, send_buff, strlen(send_buff), 0) == -1) {
                                                        perror("sendall");
                                                }
                                        }
                                }
                        }
                }

        }       // Received data from keyboard
        else {
//**************************************************************************
//******************* RECEIVE INFO FROM PEER ****************************
//**************************************************************************
                        nbytes = recv(i, recv_buff, sizeof(recv_buff), 0); // Read data
                        recv_buff[nbytes] = 0;

                        if (nbytes<= 0) {                                  // Connection closed
        //********************* CLIENT CLOSED CONNECTION ****************************

                                printf("Usuario %s con dir %s [%i] se ha desconectado \n",
                                        Client_nicks[i], Client_IPs[i], Client_Ports[i]);
                                if (nbytes< 0)  {
                                        perror("recv");
                                }
                                close(i); // bye!
                                FD_CLR(i, &master_read_fds); // remove from master set
                                Client_nicks[i][0] = 0;           // Erase nickname
                                Client_IPs[i][0] = 0;             // Erase nickname
                                Client_Ports[i] = 0;              // Erase nickname
                        }
                        else {                              // we got some data from a client
```

```
                if (Client_nicks[i][0] == 0) {
// ************************ GET CLIENTS NICKNAME *****************************
                                strcpy(Client_nicks[i],recv_buff);
                                printf("Conectado al Usuario %s, con dir %s [%i]  \n",
                                                Client_nicks[i], Client_IPs[i], Client_Ports[i]);
                                }
                        else {
// ************************ PRINT CLIENT MESSAGE *****************************

                                fechaActual = time(0) ;           // Get the time and print it
                                fechaPtr = gmtime(&fechaActual) ;
                                printf("%i/%i/%i (%i:%i:%i) ", fechaPtr->tm_mday, fechaPtr->tm_mon + 1,
                                        fechaPtr->tm_year + 1900, fechaPtr->tm_hour, fechaPtr->tm_min, fechaPtr-
>tm_sec);
                                printf("%s ", Client_nicks[i]);
                                recv_buff[nbytes] = 0;
                                printf("[%s] \n", recv_buff);     // Print the message

                        }}
        }
}               // If its the origin of the event
}                       // For every possible socket
}                       // While(1)
return 0;
close(descSocket);
}


int connect_user (char *user){
int i, desc;
for ( i = 0; i < fdmax; i++){             // Search available users
if (strcmp(user,Peer_nicks[i])==0){
desc = connect_machine (Peer_IPs[i], Peer_Ports[i]);
if (desc != -1){
        strcpy(Client_IPs[desc],Peer_IPs[i]) ;  // IP of the user
        Client_Ports[desc] = Peer_Ports[i]; // Port of the user qe are connected to
}
return desc;
}
}
printf("Usuario no existe \n");
return -1;
}

void get_word_array (char *cadena, int *num_palabras){
int i = 0, num_p = 0;
char *p;
p = cadena;

while (*p != 0) {
if (*p == ' '){
if (i != 0){  // To avoid multiple ' ' in a row error
        words_list[num_p][i] = 0;
        i = 0;
        num_p++;
}
} else {
words_list[num_p][i] =*p ;
i++;
}
p++;
}

words_list[num_p][i] = 0;
num_p++;
*num_palabras = num_p;


}
```