Sartenejas, 2 de Diciembre de 2013
Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Curso de Redes I CI-4815
Trimestre Septiembre Diciembre 2013

Grupo 45
Integrantes:

- Johanna Chan Carnet: 08-10218
- Carlos Rodriguez Carnet: 06-40189

Informe
Proyecto I (Socket C)

Cada usuario se conectará usando un programa cchat y una vez que esté conectado, los mensajes que escriba en su pantalla serán enviados al servidor schat, quien a su vez los reenviará a todos los usuarios que estén conectados a la misma sala en ese momento

La sintaxis para la invocación del servidor será la siguiente:

schat [-p <puerto>] [-s <sala>]

Donde:

- <**puerto**> Es el número de puerto que el servidor utilizará para colocar un socket para esperar por las solicitudes de conexión que el servidor puede recibir, estas solicitudes podrán ser originadas por diferentes programas cchat.
- <**sala**> Es el nombre de la sala de chat por defecto que tendrá el servidor. Si no se especifica ningún nombre, la sala por defecto llevará el nombre de actual los programas cchat que se conecten al schat se suscribirán automáticamente a esta sala al iniciar su conexión.

La sintaxis para la invocación del cchat será la siguiente:

cchat [-h <host>] [-p <puerto>] [-n <nombre>][-a <archivo>]

Donde:

- <**host**> Es el nombre o dirección IP del computador donde está corriendo el programa schat.
- <**puerto**> Es el número de puerto que el programa schat utilizará para recibir las conexiones de los diferentes programas cchat.
- <**nombre**> Es el nombre de usuario que será usado en todos los mensajes que el usuario envíe al servidor y que el servidor enviará a todos los otros usuarios, incluyéndolo a el mismo.
- <**archivo**> Es el nombre y dirección relativa o absoluta de un archivo de texto en el que en cada línea habrá un comando. Al cchat terminar de ejecuatr los comandos presentes en el archivo, debe permanecer a la espera de nuevos comandos por el teclado. A menos que, en el archivo este presente el comando fue .

Los parámetros de entrada pueden ser escritos en cualquier orden.

.

El usuario del programa cchat podrá usar una serie de comandos que escribirá por pantalla y que serán enviados al programa schat.

La lista de comandos que podrá usar es la siguiente:

- **sal:** Este comando hace que el usuario pueda ver en su pantalla una lista de las salas de chat que el servidor posee.
- **usu:** Este comando hace que el usuario pueda ver en su pantalla una lista actualizada de todos los usuarios que están suscritos en el servidor, incluyéndolo a el mismo
- **men** <**mensaje**>**:** Este comando envía el mensaje a todos los usuarios que están conectados al mismo servidor en la sala de chat a la que está suscrito el usuario.
- **sus** <**sala**>**:** El usuario se suscribe a la sala de chat sala.
- **des:** Este comando de-suscribe al usuario de la sala o salas a las que este suscrito

- **cre** <**sala**> El usuario crea la sala en el servidor.
- **eli** <**sala**> El usuario elimina la sala del servidor.
- **fue:** Este comando permite terminar la ejecución del programa de introducción de comandos y la ejecución del programa cchat.

Aspectos Importantes

- Despues de un analisis exhaustivo y un numero de pruebas considerable, se puede constatar que el proyecto funciona como fue pautado en el enunciado, sin ninguna anomalia.

- Todas las pruebas se hicieron utilizando como puertos 20810 y 20640. Sin embargo no se coloco una restriccion para que fuesen estrictamente estos puertos (en el enunciado no se especifico).

  Restricciones que se consideraron:

  - Se establecio 1024, como tamano maximo a los mensajes enviados y recibidos, por parte del cliente o el servidor.

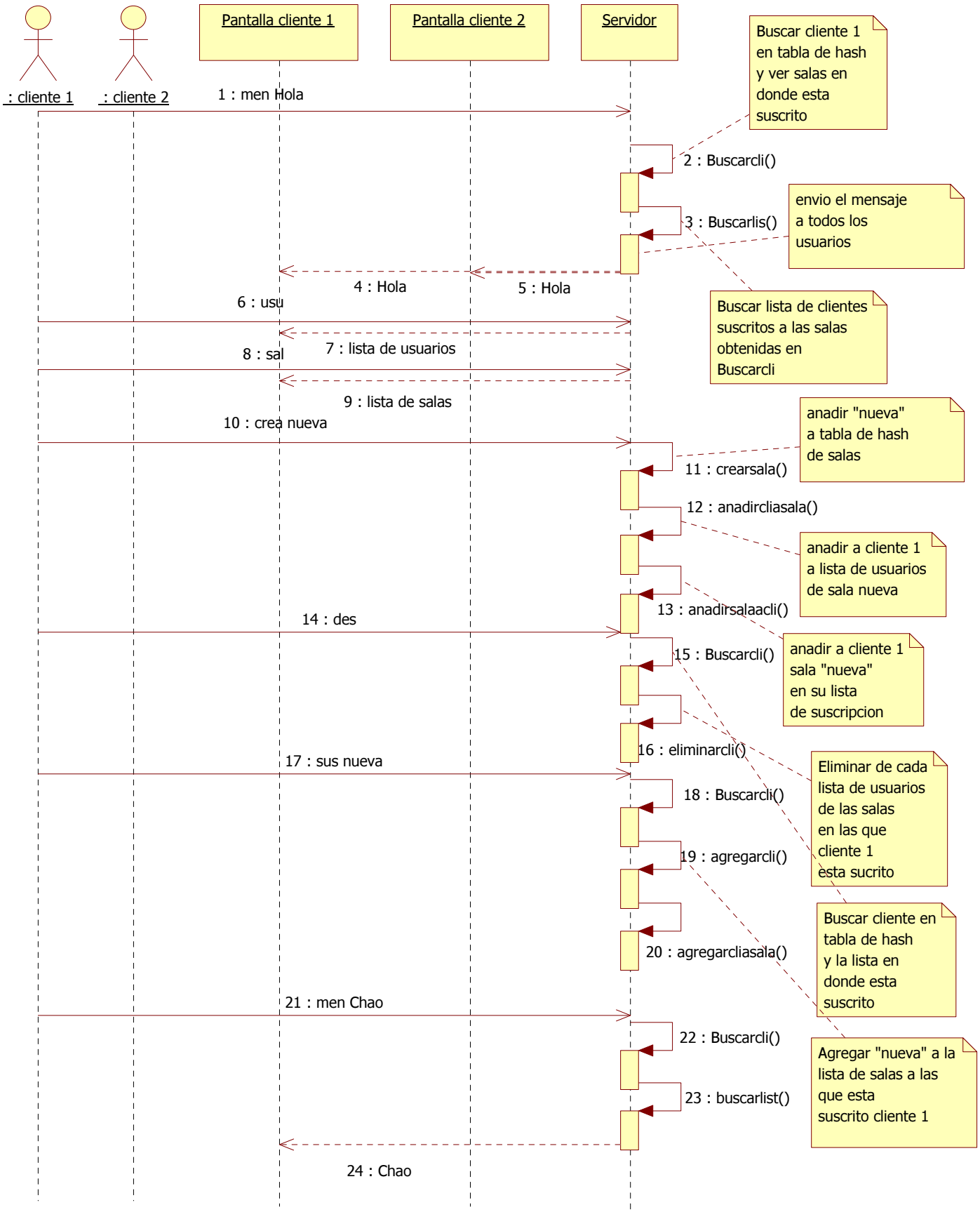
  - Se establecio 20, como numero maximo de usuarios.

  - Se establecion 200 como tamano maximo para las salas.

Diagrama de Secuencia

A continuacion se muestra un diagrama de secuencia en el que estan dos clientes en una misma sala, llamada “actual” y uno de los clientes interactua con el servidor.

- **men Hola** → el cliente 1 envia el mensaje Hola a quienes esten suscritos a las mismas salas que el.
- **usu** → el cliente 1 solicita la lista de usuarios en el sistema.
- **sal** → el cliente 1 solicita los nombres de las salas en el sistema.
- **cre nueva** → el cliente 1 crea una sala nueva, denominada “nueva”.
- **des** → el cliente 1 se desinscribe de todas las salas del sistema.
- **sus nueva** → el cliente 1 se suscribe a la sala “nueva”.
- **men chao** → el cliente 1 envia un mensaje a los usuarios que estan suscritos a las salas en la que el esta inscrito, en este caso es solo “nueva”, por lo tanto se envia el mensaje a el mismo.


: cliente 1
: cliente 2
Pantalla cliente 1
Pantalla cliente 2
Servidor
1 : men Hola
Buscar cliente 1 en tabla de hash y ver salas en donde esta suscrito
2 : Buscarcli()
envio el mensaje a todos los usuarios
3 : Buscarlis()
4 : Hola
5 : Hola
6 : usu
Buscar lista de clientes suscritos a las salas obtenidas en Buscarcli
7 : lista de usuarios
8 : sal
9 : lista de salas
10 : crea nueva
anadir "nueva" a tabla de hash de salas
11 : crearsala()
12 : anadircliasala()
anadir a cliente 1 a lista de usuarios de sala nueva
13 : anadirsalaacli()
14 : des
15 : Buscarcli()
anadir a cliente 1 sala "nueva" en su lista de suscripcion
16 : eliminarcli()
17 : sus nueva
18 : Buscarcli()
Eliminar de cada lista de usuarios de las salas en las que cliente 1 esta sucrito
19 : agregarcli()
Buscar cliente en tabla de hash y la lista en donde esta suscrito
20 : agregarcliasala()
21 : men Chao
22 : Buscarcli()
Agregar "nueva" a la lista de salas a las que esta suscrito cliente 1
23 : buscarlist()
24 : Chao

```
/**
 ************************************************************************************
 *                                                                                  *
 *                                                                                  *
 *                            88                                                    *
 *                            88                          ,d                        *
 *                            88                          88                        *
 *      ,adPPYba,   ,adPPYba,  88,dPPYba,   ,adPPYYba,  MM88MMM        ,adPPYba,    *
 *      I8[    ""  a8"     ""  88P'    "8a  ""     `Y8    88          a8"     ""    *
 *       `"Y8ba,   8b          88       88  ,adPPPPP88    88          8b            *
 *      aa    ]8I  "8a,   ,aa  88       88  88,    ,88    88,    888  "8a,   ,aa    *
 *      `"YbbdP"'   `"Ybbd8"'  88       88  `"8bbdP"Y8    "Y888  888   `"Ybbd8"'    *
 *                                                                                  *
 *                                                                                  *
 *                                                                                  *
 ************************************************************************************
 ******-> A U T O R E S <-*******
 *                             *
 * Johanna Chan        08-10218 *
 * Carlos Rodriguez    06-40189 *
 *                             *
 *******************************
 **/

#include "funciones.h"
#include "libSockets.h"
#define max_usuarios 20

/*
 * Tabla de hash que almacena la informacion referente a los usuarios.
 * Cada usuario posee una lista de salas a las que esta suscrito.
 */

  HASHTABLES * t_usuarios;

/*
 * Tabla de hash que almacena la informacion referente a las salas.
 * Cada sala posee una lista de usuarios que estan suscritos a ella.
 * En los nodos pila los descriptores no son necesarios asi que se almacena
 * el valor 0 en todos ellos. Los valores numSalas representan la cantidad
 * de usuarios asiciados y las listas salas representan listas de usuarios.
 */
  HASHTABLES * t_salas;

  void * manejar_cliente (void * arg);

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

    /* se inicializan las tablas de hash */
    createHashTable (&t_usuarios);
    createHashTable (&t_salas);

    int socket_servidor;               /*socket que tiene como funcion recibir y encolar clientes*/
    struct sockaddr_in server ;        /*Sirve solo para recibir y encolar clientes*/
    logo_servidor();
    DatosServidor Datos = Verificacion_servidor(argc, argv);

    socket_servidor = socket(AF_INET,SOCK_STREAM,0);    /* Se procede a abrir el socket que retornara un entero
    y este sera nuestro
                                                         * descriptor.
                                                         * utilizamos AF_INET para garantizar que el cliente
                                                        pueda ubicarse en otro
                                                         * ordenador.
                                                         * SOCK_STREAM indica que es un socket orientado a
                                                        conexion.
                                                         *  0 corresponde al protocolo a utilizar, usualmente
                                                        se utiliza 0.
                                                         */

    if(socket_servidor == -1){  /*Se verifica que la creacion del socket sea exitosa.*/
```

```
      printf("Error\n");
      exit(1);                          /*y sale del programa*/
    }                                   /*Si el socket se creo sin ningun problema, podemos pasar a hacer la
    asociacion entre
                                         *el socket y la direccion del servidor*/

    server.sin_family = AF_INET;                /*Se especifica la familia del protocolo (tipo de conexion).*/
    server.sin_addr.s_addr = INADDR_ANY;        /*Direccion que atendera el servidor, con INADDR_ANY especifico
    que puede ser
                                                 *cualquier direccion, proveniente de cualquier cliente.*/
    server.sin_port=htons(Datos.puerto);        /*Asignacion del puerto, el especificado en la entrada estandar.
    Sera 20810*/

    if(bind(socket_servidor,(struct sockaddr *)&server,sizeof(server)) == -1){  /*Asociacion del socket a la
    direccion del servidor.*/
      printf("Error\n");                                                        /*Ocurrio un error
      haciendo la asociacion entre el
                                                                               *socket y la direccion del
                                                                               servidor.*/
      exit(1);
    }


    if (listen (socket_servidor, 1) == -1) {                          /*Una vez hecha la asociacion procedemos
    a hacer la asociacion
                                                                       *se procede a avisarle al sistema que
                                                                       puede comenzar a
                                                                       *escuchar peticiones.*/
    printf ("Error\n");                                               /*Ocurrio un error*/
    }

    /* se anhade la sala indicada como parametro de entrada */
    add (Datos.sala, 0, 0, NULL, t_salas);

    pthread_t hilo[max_usuarios];
    int socket_cliente[max_usuarios * 2];
    int num_sockets = 0;
    Nombre_sala n[max_usuarios];
    Cabecera admitido[max_usuarios];

    /* mientras hayan entrado menos de max_usuarios usuarios, se siguen recibiendo solicitudes de conexion */
    while (num_sockets < max_usuarios)
    {
      /* se inician los sockets */
      socket_cliente[num_sockets] = accept(socket_servidor, 0, 0);         /*Se acepta la peticion, se genera un
      nuevo socket
                                                                            *que es por el cual el servidor
                                                                           "hablara" con el cliente.*/
      if (socket_cliente[num_sockets]  == -1)
        printf ("Error\n");
      socket_cliente[num_sockets + max_usuarios] = accept(socket_servidor, 0, 0);         /*Se acepta la
      peticion, se genera un nuevo socket
                                                                             *que es por el cual
el servidor "hablara" con el cliente.*/
      if (socket_cliente[num_sockets+ max_usuarios]  == -1)
        printf ("Error\n");

      /* se recibe el nombre que utilizara el cliente */
      read (socket_cliente[num_sockets], &n[num_sockets], sizeof(Nombre_sala));
      int no_repetido = add (n[num_sockets].nombre, socket_cliente[num_sockets], socket_cliente[num_sockets +
      max_usuarios], Datos.sala, t_usuarios);

      /* si no hay otro usuario con ese nombre, se le informa que ha sido aceptado */
      if (no_repetido != FALSE)
      {
        subAdd (Datos.sala, n[num_sockets].nombre, t_salas);
        admitido[num_sockets].id = TRUE;
        write (socket_cliente[num_sockets], &admitido[num_sockets], sizeof(Cabecera));
        int ch;
        NODOPILA * aux = buscar (n[num_sockets].nombre, t_usuarios);
```

```
        if( (ch = pthread_create( &hilo[num_sockets], NULL, &manejar_cliente, (void * ) aux)))
          printf("ERROR AL CREAR HILO\n");
        num_sockets++;
      }
      else
      {
        admitido[num_sockets].id = FALSE;
        write (socket_cliente[num_sockets], &admitido[num_sockets], sizeof(Cabecera));
        close (socket_cliente[num_sockets]);
      }
    }
    return 1;
  }


/*
 * Hilo que se encarga de la comunicacion con un cliente.
 *
 * @param arg NODOPILA donde se encuentra la informacion del cliente.
 *
 */

  void * manejar_cliente (void * arg) {
    NODOPILA * aux = (NODOPILA *) arg;
    int socket_cliente = aux->desc;
    char * nombre = aux->name;
    int i;
    do
    {
      i = servidor_recibe_mensaje (socket_cliente, t_usuarios, nombre, t_salas);
    }while (i);
    pthread_exit (NULL);
  }
```

```
/**
 ************************************************************************************************
 *                                                                                              *
 *                                                                                              *
 *                                88                                                            *
 *                                88                          ,d                                *
 *                                88                          88                                *
 *        ,adPPYba,   ,adPPYba,  88,dPPYba,   ,adPPYYba,  MM88MMM        ,adPPYba,             *
 *       a8"     ""  a8"     ""  88P'    "8a  ""     `Y8    88          a8"     ""            *
 *       8b          8b          88       88  ,adPPPPP88    88          8b                    *
 *       "8a,   ,aa  "8a,   ,aa  88       88  88,    ,88    88,    888  "8a,   ,aa            *
 *        `"Ybbd8"'   `"Ybbd8"'  88       88  `"8bbdP"Y8    "Y888  888   `"Ybbd8"'            *
 *                                                                                              *
 *                                                                                              *
 *                                                                                              *
 ************************************************************************************************
 ******-> A U T O R E S <-*******
 *                            *
 * Johanna Chan        08-10218 *
 * Carlos Rodriguez    06-40189 *
 *                            *
 ******************************
 **/

#include "funciones.h"
#include "libSockets.h"

void * recibir (void * arg);

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

  struct sockaddr_in server ;                           /*Sirve solo para recibir y encolar clientes*/
  int socket_cliente;                                   /*Sirve para comunicarse con el servidor*/
  int socket_cliente2;
  struct sockaddr_in server2 ;                           /*Sirve solo para recibir y encolar clientes*/
  char buffer[1024];
  memset( buffer, 0, 1024*sizeof(char) );               /*Se inicializa el buffer*/

  logo_cliente();
  DatosCliente Datos = Verificacion_cliente(argc, argv);

  socket_cliente = socket(AF_INET,SOCK_STREAM,0);       /* procedemos a abrir el socket que retornara un entero
y este sera
                                                         * nuestro descriptor.
                                                         * utilizamos AF_INET para garantizar que el cliente
pueda ubicarse
                                                         * en otro ordenador.
                                                         * SOCK_STREAM indica que es un socket orientado a
conexion.
                                                         * 0 corresponde al protocolo a utilizar, usualmente se
utiliza 0.*/

  if(socket_cliente == -1){                             /* Se verificamos que la creacion del socket sea
exitosa.*/
    printf("Error creando socket\n");
    exit(1);                                            /* y sale del programa*/
  }                                                     /* Si el socket se creo sin ningun problema, podemos
pasar a hacer la
                                                         * asociacion entre el socket y la direccion del
servidor*/

  server.sin_family = AF_INET;                          /* Se especifica la familia del protocolo (tipo de
conexion).*/
  server.sin_addr.s_addr=inet_addr("127.0.0.1");        /* Direccion que atendera el servidor, con INADDR_ANY
especifico que
                                                         * puede ser cualquier direccion, proveniente de
cualquier cliente.*/
  server.sin_port=htons(Datos.puerto);                  /* Asignacion del puerto, el especificado en la entrada
estandar. Sera 20810*/
```

```
  if(connect(socket_cliente,(struct sockaddr *)&server,sizeof(server)) == -1){  /*Asociacion del socket a la direccion del servidor.*/
    printf("Error asociando\n");                        /* Ocurrio un error haciendo la asociacion entre el socket y la direccion del
                                                         * servidor.*/
    exit(1);
  }

  socket_cliente2 = socket(AF_INET,SOCK_STREAM,0);       /* procedemos a abrir el socket que retornara un entero y este sera
                                                         * nuestro descriptor.
                                                         * utilizamos AF_INET para garantizar que el cliente pueda ubicarse
                                                         * en otro ordenador.
                                                         * SOCK_STREAM indica que es un socket orientado a conexion.
                                                         * 0 corresponde al protocolo a utilizar, usualmente se utiliza 0.*/


                                                         /* Se crea un segundo socket para que reciba mensajes de otros usuarios a traves del servidor */
  if(socket_cliente2 == -1){                             /* Se verificamos que la creacion del socket sea exitosa.*/
    printf("Error creando socket\n");
    exit(1);                                            /* y sale del programa*/
  }                                                     /* Si el socket se creo sin ningun problema, podemos pasar a hacer la
                                                         * asociacion entre el socket y la direccion del servidor*/

  server2.sin_family = AF_INET;                          /* Se especifica la familia del protocolo (tipo de conexion).*/
  server2.sin_addr.s_addr=inet_addr("127.0.0.1");        /* Direccion que atendera el servidor, con INADDR_ANY especifico que
                                                          * puede ser cualquier direccion, proveniente de cualquier cliente.*/
  server2.sin_port=htons(Datos.puerto);                  /* Asignacion del puerto, el especificado en la entrada estandar. Sera 20810*/



  if(connect(socket_cliente2,(struct sockaddr *)&server2,sizeof(server)) == -1){  /*Asociacion del socket a la direccion del servidor.*/
    printf("Error asociando\n");                        /* Ocurrio un error haciendo la asociacion entre el socket y la direccion del
                                                         * servidor.*/
    exit(1);
  }

  /* se envia el nombre de usuario al servidor */
  enviar_mi_nombre (socket_cliente, Datos.nombre);

  /* se crea el hilo que se encargara de recibir mensajes del servidor enviados por otros usuarios */
  pthread_t hilo_recibir;
  int ch;
  if( (ch = pthread_create( &hilo_recibir, NULL, &recibir, &socket_cliente2)))
                        printf("ERROR INTERNO\n");

  /* se leen las instrucciones contenidas en el archivo */
  leer_archivo (socket_cliente, Datos.archivo);

  /* se reciben mensajes por la entrada estandar y se envian las instrucciones correspondientes al servidor */
  char mensaje_enviar[1024];
  while (TRUE)
  {
    gets (mensaje_enviar);
    verificar_mensaje (socket_cliente, mensaje_enviar);
```

```
  }

    return 1;

}

/*
 * Hilo que recibe mesajes del servidor.
 *
 * @param arg Puntero al socket a traves del cual se recibiran los mensajes */
void * recibir (void * arg)
{
  int * socket = (int *) arg;
  Manda_mensaje m;
  int i;

  /* se reciben los mensajes que envie el servidor */
  do
  {
    i = read (*(socket), &m, sizeof(Manda_mensaje));
    printf ("\nUsuario '%s' de la sala '%s' dice: %s\n\n", m.nombre, m.sala, m.mensaje);
  }while (i);
  pthread_exit (NULL);
}
```

```
/**
 ************************************************************************************************************************
 *****************
 *
 *
 *
 *
 *         ad88                                    88
 *
 *        d8"                                      ""
 *
 *        88
 *
 *      MM88MMM  88       88  8b,dPPYba,    ,adPPYba,  88   ,adPPYba,   8b,dPPYba,    ,adPPYba,  ,adPPYba,
,adPPYba,      *
 *        88     88       88  88P'   `"8a  a8"     ""  88  a8"     "8a  88P'   `"8a  a8P_____88  I8[    ""
a8"     ""     *
 *        88     88       88  88       88  8b          88  8b       d8  88       88  8PP"""""""   `"Y8ba,
8b             *
 *        88     "8a,   ,a88  88       88  "8a,   ,aa  88  "8a,   ,a8"  88       88  "8b,   ,aa  aa    ]8I  888
"8a,   ,aa     *
 *        88      `"YbbdP'Y8  88       88   `"Ybbd8"'  88   `"YbbdP"'   88       88   `"Ybbd8"'  `"YbbdP"'  888
`"Ybbd8"'      *
 *
 *
 *
 *
 ************************************************************************************************************************
 *****************
 ******-> A U T O R E S <-*******
 *                             *
 * Johanna Chan        08-10218 *
 * Carlos Rodriguez    06-40189 *
 *                             *
 *******************************
 **/

#define KNRM  "\x1B[0m"
#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <pthread.h>
#include <sys/dir.h>
#include <dirent.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <getopt.h>
#include <netdb.h>
#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <math.h>
#include "funciones.h"
#include "libSockets.h"

 /*
  *
  * Esta funcion se encarga de verificar si la cadena introducida corresponde al
  * nombre de un archivo, el nombre de un archivo puee estar constituido por
```

```
 * caracteres o digitos, es por eso que se utilizo la funcion isalnum
 * correspondiente a la libreria ctype.
 *
 * @param cadena es el string que se desea analizar
 * @return un booleano, true si la cadena corresponde al nombre de un archivo.
 *
 */

int EsNombreArchivo(char * cadena) {
  int boolean, i;
  i = 0;
  boolean = 0;
  while(i != strlen(cadena)){
    if(cadena[i] != '/'){
      i++;
      boolean = 1;
    }else{
      printf(" La cadena de caracteres no corresponde al nombre de un archivo.\n");
      boolean = 0;
      break;
    }
  }
  return boolean;
}


/*
 *
 * EsNombre es una funcion que se encarga de verificar si la cadena de
 * caracteres introducida es una palabra, esto lo logra mediante la funcion
 * isalpha() que retorna un 0 si no lo es.
 *
 * @param nombre es el string a ser analizado.
 * @return 1 si es una palabra, 0 de lo contrario.
 *
 */

int EsPalabra(char * nombre){
  int i;
  i = 0;
  int boolname;
  boolname = 1;
  while(i<strlen(nombre)-1){
    if(isalpha(nombre[i])== 0){
      boolname = 0;
      break;
    }
    i++;
  }
  return boolname;
}


/*
 *
 * EsNumero es una funcion que se encarga de verificar si la cadena de
 * caracteres introducida corresponde a un numero.
 *
 * @param numero, es la cadena de caracteres a ser analizada.
 * @return 1 si es un numero, 0 de lo contrario.
 */

int EsNumero(char * numero){
  int i,boolean;
  i = 0;
  boolean = 0;
  while(i != strlen(numero)){
    if(isdigit(numero[i]) != 0 ){
      i++;
      boolean = 1;
```

```
    }else{
      boolean = 0;
      break;
    }
  }
  return boolean;
}


/*
 *
 * ContarPalabra es una funcion que se encarga de abrir un archivo y contar las
 * apariciones de una determinada palabra, esta funcion recibe como parametros
 * 2 cadenas de caracteres, la primera es el nombre del archivo a abrir y la
 * segunda es la palabra a la que se quiere contabilizar el numero de
 * apariciones.
 *
 * @param socket_cliente es el socket al que se enviara la informacion de cada linea.
 * @param NombreArchivo es el nombre del archivo que se va a examinar.
 * @param Palabra es la palabra que se desea contar a lo largo del archivo.
 * @param nombre nombre del usuario.
 * @return el numero de veces que aparece Palabra en el archivo NombreArchivo.
 *
 */

int ContarPalabras(int socket_cliente, char * NombreArchivo, char * Palabra, char* nombre){
  char *s2 = ".,:;`'\"+-_(){}[]<>*&^%$#@!?~/|\\= \t\r\n1234567890#";
  char LineaEntrada[1024];    /* Contiene una linea del archivo que se abrio.*/
  memset( LineaEntrada, 0, 1024*sizeof(char) );/*inicializacion del arreglo "buffer"*/
  FILE *Entrada;
  int ocur = 0;
  char *ptr;  /* Se almacena la palabra (token) delimitada por los caracteres de la cadena s2 */
  if((Entrada = fopen(NombreArchivo, "r")) != NULL){
    while (fgets(LineaEntrada, sizeof(LineaEntrada), Entrada) != NULL){
      escribir_cliente(socket_cliente, LineaEntrada, nombre);
      ptr = strtok( LineaEntrada, s2 );       /* Primera llamada => Primer token, separo las palabras por los
delimitadores. */
      if(ptr != NULL){
        if(strcmp(ptr, Palabra)==0){  /* Comparo si el token es igual a la palabra que busco.*/
          ocur++;
        }
      }
      while( ptr != NULL ){   /* Posteriores llamadas */
        ptr = strtok( NULL, s2 );
        if(ptr != NULL){
          if(strcmp(ptr, Palabra)==0){
            ocur++;
          }
        }
      }
    }
  }else{
    printf("ERROR\n   - El Archivo %s no se pudo abrir.\n", NombreArchivo);
  }
  return ocur;
}


/*
 *
 * msjdivision es una funcion que se encarga de recibir un string y separarlo en partes,
 * y de esta forma obtener el comando y el mensaje enviado a traves del socket.
 *
 * @param msjcompleto es el string completo que se pretende dividir.
 * @return una estructura del tipo MSJdiv que almacena el comando y el resto del mensaje.
 *
 */

MSJdiv msjdivision(char *msjcompleto){
  MSJdiv New;
```

```
    char s1[1024];
    char buffer[1024];
    memset( s1, 0, 1024*sizeof(char) );/*inicializacion del arreglo*/
    memset( buffer, 0, 1024*sizeof(char) );/*inicializacion del arreglo*/
    strcpy(s1, msjcompleto);
    char s2[2] = " ";
    char *ptr;
    ptr = strtok( s1, s2 );    // Primera llamada => Primer token
    New.comando =(char*)malloc(strlen(ptr)*sizeof(char));
    strcpy(New.comando, ptr);
    while( (ptr = strtok( NULL, s2 )) != NULL ){    // Posteriores llamadas
      New.resto =(char*)malloc(1024*sizeof(char));
      strcat(buffer, ptr);
      strcat(buffer, " ");
      strcpy(New.resto, buffer);
    }
    printf("New.comando contiene %s\n", New.comando);
    printf("New.resto contiene %s\n", New.resto);
    printf("buffer contiene %s\n", buffer);
    return New;
  }



 /*
  *
  * ImprimirAyuda, como su nombre lo dice imprime la ayuda para la buena utilizacion
  * del programa.
  *
  */

  void ImprimirAyuda_servidor(){
    printf("\n");
    printf(" La sintaxis para la invocación del servidor será la siguiente: \n\n");
    printf(" schat [-p <puerto>] [-s <sala>]\n\n");
    printf(" Donde: \n\n");
    printf("\t<puerto> Es el número de puerto que el servidor utilizará para colocar un socket para\n");
    printf("\t         esperar por las solicitudes de conexión que el servidor puede recibir, estas\n");
    printf("\t         solicitudes podrán ser originadas por diferentes programas cchat.\n\n");
    printf("\t <sala>  Es el nombre de la sala de chat por defecto que tendrá el servidor. No incluye\n");
    printf("\t         numeros.\n\n");

    }


 /*
  *
  * ImprimirAyuda, como su nombre lo dice imprime la ayuda para la buena utilizacion
  * del programa.
  *
  */
  void ImprimirAyuda_cliente(){
    printf(" La sintaxis para la invocación del cchat será la siguiente:\n\n");
    printf(" cchat [-h <host>] [-p <puerto>] [-n <nombre>][-a <archivo>]\n\n");
    printf(" Donde:\n\n");
    printf("\t  <host>   Es el nombre o dirección IP del computador donde está corriendo el programa\n\n");
    printf("\t           schat.\n\n");
    printf("\t <puerto>  Es el número de puerto que el programa schat utilizará para recibir las conexiones\n");
    printf("\t           de los diferentes programas cchat.\n\n");
    printf("\t <nombre>  Es el nombre de usuario que será usado en todos los mensajes que el usuario\n");
    printf("\t           envíe al servidor y que el servidor enviara a todos los otros usuarios,
incluyéndolo\n");
    printf("\t           a el mismo.\n\n");
    printf("\t <archivo> Es el nombre y dirección relativa o absoluta de un archivo e texto en el que en\n");
    printf("\t           cada línea habrá un comando para cchat. Al cchat terminar de ejecuatr los comandos\n");
    printf("\t           presentes en el archivo, debe permanecer a la espera de nuevos comandos por el teclado.
\n");
    printf("\t           A menos que, en el archivo este presente el comando fue .\n\n");
  }
```

```c
/*
 *
 * Esta funcion se encarga de verificar que se introdujo en la entrada estandar
 * si fue un schat [-p <puerto>] [-s <sala>], si la entrada es valida,
 * retorna
 *
 * @param NumArg corresponde al numero de argumentos introducidos por el usuario.
 * @argu es el arreglo de caracteres que contienen las opciones y argumentos.
 * @return una estructura del tipo DatosServidor que almacena la informacion
 * colocada en la entrada estandar.
 *
 */

DatosServidor Verificacion_servidor(int NumArg, char *argu[]){
  printf("\n");
  DatosServidor New;
  int c,s,p,h;
  c = 0;
  s = 0; /*indica si el usuario efectivamente coloco el nombre correspondiente a la sala*/
  p = 0; /*indica si el usuario efectivamente coloco el numero correspondiente a la puerto*/
  h = 0; /*indica si el usuario solicito ayuda*/
  opterr = 0;
  while((c = getopt(NumArg, argu, "hp:s:")) != -1){ /*"p:s:" Son los comandos aceptados*/

    switch(c){

      case 'h':
        if(NumArg ==2 && p == 0 && s == 0){
          h = 1;
          ImprimirAyuda_servidor();
          break;
        }else{
          printf(" Para utilizar el comando -h, debe escribir en la linea de comandos solamente pargrep -h.
\n");
          break;
        }

      case 'p':
        if(NumArg == 5 || NumArg == 3){
          if(optarg){
            if(EsNumero(optarg) == 1){
              p = 1;
              New.puerto = atoi(optarg);
              break;
            } else {
              printf("\"%s\" No es valido para la opcion -p.\n", optarg);
              break;
            }
          }
        }

        case 's':
          if(NumArg == 5){
            if(optarg){
              if(EsPalabra(optarg) == 1){
                s = 1;
                New.sala = malloc(sizeof(char)*strlen(optarg));
                New.sala = optarg;
                break;
              } else {
                printf("\"%s\" No es valido para la opcion -s.\n", optarg);
                break;
              }
            }
          }

        /*
        * en optopt se guardan los comandos invalidos o que les falta el argumento.
        */
```

```
        case '?':
          if ((optopt != 'p') && (optopt != 's') && (optopt != 'h')){
            fprintf (stderr, "El comando -%c es invalido.\n", optopt);
          }else if (isprint (optopt)){
            fprintf (stderr, " El Comando '-%c' requiere un argumento.\n", optopt);
          }else{
            fprintf (stderr,"Comando Invalido `\\x%x'.\n",optopt);
          }
          break;
    }
  }

  if(p == 0){ // si no especifico el puerto
    if(h == 0){
          printf("\nLo que acaba de escribir es inválido.\nSi desea ayuda ingrese ./schat -h\n\n");
    }
    exit(1);
  }
  if(s == 0){
    New.sala = malloc(sizeof(char)*strlen("actual"));
    New.sala = "actual";
  }
  return New;
}

/*
 *
 * Esta funcion se encarga de verificar que se introdujo en la entrada estandar
 * si fue un cchat [-h host] [-p <puerto>] [-n <nombre>] [-a <archivo>]
 *
 * @param NumArg corresponde al numero de argumentos introducidos por el usuario.
 * @argu es el arreglo de caracteres que contienen las opciones y argumentos.
 * @return una estructura del tipo DatosServidor que almacena la informacion
 * colocada en la entrada estandar.
 *
 */

DatosCliente Verificacion_cliente(int NumArg, char *argu[]){
  printf("\n");
  DatosCliente New;   //Estructura definida en el archivo funciones.h para guardar los argumentos.
  int c,h,p,n,a,o;
  c = 0;
  h = 0; //host
  p = 0; //puerto
  n = 0; //nombre
  a = 0; //archivo
  o = 0; //ayuda
  opterr = 0;
  while((c = getopt(NumArg, argu, "h:p:n:a:o")) != -1){ //"p:s:" Son los comandos aceptados
    switch(c){
      case 'o':
        if(NumArg ==2 && h == 0 && p == 0 && n == 0 && a == 0){
          o = 1;
          ImprimirAyuda_cliente();
          break;
        }else{
          printf(" Para utilizar el comando -h, debe escribir en la linea de comandos solamente pargrep -h.
\n");//OJO
          break;
        }


  case 'h':
        if(NumArg == 9 ){
          if(optarg){
            printf("La IP es %s\n", optarg);
                h = 1;
                New.host = (char*)malloc(sizeof(char)*strlen(optarg));
                New.host = optarg;
                break;
```

```
          }
        }

        case 'p':
                printf("estoy en p con %s\n", optarg);
          if(NumArg == 9){
            if(optarg){
              if(EsNumero(optarg) == 1){
                        p = 1;
                        printf("El numero de puerto es %s\n", optarg);
                        New.puerto = atoi(optarg);
                        break;
              } else {
                        printf("\"%s\" No es valido para la opcion -p.\n", optarg);
                        break;
              }
            }
            printf("sali del case p\n");
          }
          break;

        case 'n':
          if(NumArg == 9){
            if(optarg){
              if(EsPalabra(optarg) == 1){
                        n = 1;
                        printf("El nombre que introdujo es %s\n", optarg);
                        New.nombre = malloc(sizeof(char)*strlen(optarg));
                        New.nombre = optarg;
                        break;
              } else {
                        printf("\"%s\" No es valido para la opcion -n.\n", optarg);
                        break;
              }
            }
          }

        case 'a':
          if(NumArg == 9){
            if(optarg){
              if(EsNombreArchivo(optarg) == 1){
                        a = 1;
                        printf("EL nombre del archivo es %s\n", optarg);
                        New.archivo = malloc(sizeof(char)*strlen(optarg));
                        New.archivo = optarg;
                        break;
              } else {
                        printf("\"%s\" No es valido para la opcion -a.\n", optarg);
                        break;
              }
            }
          }

        /*
         * en optopt se guardan los comandos invalidos o que les falta el argumento.
         */
        case '?':
          if ((optopt != 'p') && (optopt != 's') && (optopt != 'h')){
            fprintf (stderr, "El comando -%c es invalido.\n", optopt);
          }else if (isprint (optopt)){
            fprintf (stderr, " El Comando '-%c' requiere un argumento.\n", optopt);
          }else{
            fprintf (stderr,"Comando Invalido `\\x%x'.\n",optopt);
          }
          break;
    }
  }

  if(h == 0 || p == 0 || n == 0 || a == 0){ // si especifico el numero de jugadores y pidio ayuda
    if(o == 0){
```

```
            printf("\nLo que acaba de escribir es inválido.\nSi desea ayuda ingrese ./schat -o\n\n");
      }
      exit(1);
    }
    return New;
  }


 /*
  *
  * Esta funcion se encarga de imprimir el logo del programa del servidor.
  *
  */

  void logo_servidor(){
    printf("\n");
    printf("%s                                    hhhhhhh                                        tttt      \n",
    KBLU);
    printf("%s                                    h:::::h                                     ttt:::t      \n",
    KBLU);
    printf("%s                                    h:::::h                                     t:::::t      \n",
    KBLU);
    printf("%s                                    h:::::h                                     t:::::t      \n",
    KBLU);
    printf("%s    sssssssssss       cccccccccccccccc h::::h hhhhh         aaaaaaaaaaaaa   ttttttt:::::ttttttt\n",
    KBLU);
    printf("%s  ss:::::::::::s    cc:::::::::::::::c h::::hh:::::hhh      a::::::::::::a  t:::::::::::::::::t\n",
    KBLU);
    printf("%sss::::::::::::s  c:::::::::::::::::c h::::::::::::::hh    aaaaaaaaa:::::a t:::::::::::::::::t\n",
    KBLU);
    printf("%ss::::::ssss:::::sc:::::::ccccccc:::::c h:::::::hhh::::::h           a::::a tttttt:::::::tttttt\n",
    KBLU);
    printf("%s s:::::s  ssssss c::::::c     ccccccc h::::::h   h::::::h    aaaaaaa:::::a       t:::::t      \n",
    KBLU);
    printf("%s   s::::::s      c:::::c             h:::::h     h:::::h  aa::::::::::::a       t:::::t      \n",
    KBLU);
    printf("%s      s::::::s   c:::::c             h:::::h     h:::::h a::::aaaa::::::a       t:::::t      \n",
    KBLU);
    printf("%sssssss   s:::::s c::::::c     ccccccc h:::::h     h:::::ha::::a    a:::::a       t:::::t
    tttttt\n", KBLU);
    printf("%ss:::::ssss::::::sc:::::::ccccccc:::::c h:::::h     h:::::ha::::a    a:::::a
    t::::::tttt:::::t\n", KBLU);
    printf("%ss:::::::::::::s  c:::::::::::::::::c h:::::h     h:::::ha:::::aaaa::::::a
    tt:::::::::::::t\n", KBLU);
    printf("%s s:::::::::::ss    cc:::::::::::::::c h:::::h     h:::::h a::::::::::aa:::a
    tt::::::::::::tt\n", KBLU);
    printf("%s  sssssssssss        cccccccccccccccc hhhhhhh     hhhhhhh  aaaaaaaaaa  aaaa          ttttttttttt
    \n", KBLU);
    printf("%s\n", KWHT);
  }


 /*
  *
  * Esta funcion se encarga de imprimir el logo del programa del cliente.
  *
  */

  void logo_cliente(){

    printf("%s                                      hhhhhhh                                        tttt
    \n",KRED);
    printf("%s                                      h:::::h                                     ttt:::t
    \n",KRED);
    printf("%s                                      h:::::h                                     t:::::t
    \n",KRED);
    printf("%s                                      h:::::h                                     t:::::t
    \n",KRED);
    printf("%s    ccccccccccccccc    ccccccccccccccc h::::h hhhhh         aaaaaaaaaaaaa   ttttttt:::::ttttttt
    \n",KRED);
```

```
    printf("%s  cc:::::::::::::::c  cc:::::::::::::::c h::::hh:::::hhh      a::::::::::::a  t:::::::::::::::::t
    \n",KRED);
    printf("%s c:::::::::::::::::c c:::::::::::::::::c h:::::::::::::hh    aaaaaaaaa:::::a t:::::::::::::::::t
    \n",KRED);
    printf("%sc:::::::ccccccc:::::cc:::::::ccccccc:::::c h:::::::hhh::::::h           a::::a tttttt:::::::tttttt
    \n",KRED);
    printf("%sc::::::c     ccccccc::::::c     ccccccc h::::::h   h::::::h    aaaaaaa:::::a       t:::::t
    \n",KRED);
    printf("%sc:::::c             c:::::c             h:::::h     h:::::h  aa::::::::::::a       t:::::t
    \n",KRED);
    printf("%sc:::::c             c:::::c             h:::::h     h:::::h a::::aaaa::::::a       t:::::t
    \n",KRED);
    printf("%sc::::::c     ccccccc::::::c     ccccccc h:::::h     h:::::ha::::a    a:::::a       t:::::t
    tttttt\n",KRED);
    printf("%sc:::::::ccccccc:::::cc:::::::ccccccc:::::c h:::::h     h:::::ha::::a    a:::::a
    t::::::tttt:::::t\n",KRED);
    printf("%s c:::::::::::::::::c c:::::::::::::::::c h:::::h     h:::::ha::::aaaa::::::a
    tt:::::::::::::t\n",KRED);
    printf("%s  cc:::::::::::::::c  cc:::::::::::::::c h:::::h     h:::::h a:::::::::::aa:::a
    tt:::::::::::tt\n",KRED);
    printf("%s    ccccccccccccccc    ccccccccccccccc hhhhhhh     hhhhhhh  aaaaaaaaaa  aaaa
    ttttttttttt  \n",KRED);
    printf("%s\n",KWHT);
  }


 /*
  *
  * leer_servidor es una funcion que se encarga de leer la informacion contenida
  * en el socket, este se especifica en el argumento de la funcion.
  *
  * @socket_cliente corresponde al descriptor del socket de donde se desea leer el mensaje.
  * @return una estructura que contiene el mensaje y su tamano
  *
  */

  MSJ leer_servidor(int socket_cliente) {
    MSJ New;                                  /*Estructura que almacena el mensaje y su tamaño*/
    char Datos[1024];                         /* Buffer donde guardaremos los caracteres a leer*/
    memset( Datos, 0, 1024*sizeof(char) );    /*Inicializacion del arreglo*/
    int Aux = read (socket_cliente, Datos, sizeof(Datos));      /*Se procede a leer del socket*/

    if (Aux == 0) {                                             /*Si no se ha podido escribir,
                                                                  *se verifica la condicion de socket cerrado*/
      printf ("Socket cerrado\n");
    } else if (Aux == -1) {
        printf ("Error\n");
    } else if(Aux >0) {
      New.mensaje = Datos;
      New.numero = Aux;
    }
    return New;
  }


 /*
  *
  * leer_cliente es una funcion que se encarga de leer la informacion contenida
  * en el socket, este se especifica en el argumento de la funcion.
  *
  * @socket_cliente corresponde al descriptor del socket de donde se desea leer el mensaje.
  * @return una estructura que contiene el mensaje y su tamano
  *
  */

  MSJ leer_cliente(int socket_cliente) {
    MSJ New;
    char Datos[1024]; /* Buffer donde guardaremos los caracteres */
      memset( Datos, 0, 1024*sizeof(char) );//inicializacion del arreglo "buffer"
    int Aux = read (socket_cliente, Datos, sizeof(Datos));
```

```
    /* Si no hemos podido escribir caracteres,
    comprobamos la condici??n de socket cerrado */
    if (Aux == 0) {
      printf ("Socket cerrado\n");
    } else if (Aux == -1) {
        printf ("Error\n");
    } else if(Aux >0) {
      New.mensaje = Datos;
      New.numero = Aux;
    }
    return New;
  }



 /*
  *
  * escribir_servidor es una funcion que se encarga de escribir informacion en el socket especificado
  * en el argumento de la funcion.
  *
  * @param socket_cliente correspondel al descriptor del socket en donde se desea escribir.
  * @param mensaje es el mensaje que se desea enviar
  * @paramm nombre corresponde al nombre del usuario que esta escribiendo el mensaje
  * @return una estructura con el mensaje enviado y su tamano.
  *
  */

  MSJ escribir_servidor(int socket_cliente, char* mensaje, char* nombre) {
    MSJ New;                                            /*Estructura que almacena el mensaje y su tamaño*/
    int tamano = strlen(nombre) + strlen(mensaje)+2;    /*tamano del buffer, el tamano del usuario mas el
                                                         *tamano del mensaje mas el espacio que separa el
                                                         *nombre del usuario y el mensaje y el caracter de fin
                                                        de linea*/
    char Datos[tamano];
    memset( Datos, 0, tamano*sizeof(char) );            /*inicializacion del arreglo "buffer"*/
    strcat(Datos, nombre);                              /*Concatenacion del nombre*/
    strcat(Datos, " ");                                 /*Concatenacion del espacio separador*/
    strcat(Datos, mensaje);                             /*Concatenacion del mensaje*/
    Datos[tamano]='\0';                                 /*Colocacion del caracter fin de linea*/

    int Aux = write(socket_cliente, Datos, sizeof(Datos));      /*Se procede a escribir en el socket*/
    if (Aux == 0) {                                             /*Si el socket en donde se desea escribir esta
    cerrado*/
      printf ("Socket cerrado\n");
    } else if (Aux == -1) {                                     /*Si ocurrio un error en el socket*/
        printf ("Error\n");
    } else if(Aux >0) {
      New.mensaje = Datos;
      New.numero = Aux;
    }
    return New;
  }



 /*
  *
  * escribir_cliente es una funcion que se encarga de escribir informacion en el socket especificado
  * en el argumento de la funcion.
  *
  * @param socket_cliente correspondel al descriptor del socket en donde se desea escribir.
  * @param mensaje es el mensaje que se desea enviar
  * @paramm nombre corresponde al nombre del usuario que esta escribiendo el mensaje
  * @return una estructura con el mensaje enviado y su tamano.
  *
  */

  MSJ escribir_cliente(int socket_cliente, char* mensaje, char* nombre) {
    MSJ New;
    int tamano = strlen(nombre) + strlen(mensaje)+2;
    char Datos[tamano];
```

```
    memset( Datos, 0, tamano*sizeof(char) );//inicializacion del arreglo "buffer"
    strcat(Datos, nombre);
    strcat(Datos, " ");
  strcat(Datos, mensaje);
  Datos[strlen(mensaje)+1]='\0';
  int Aux = write(socket_cliente, Datos, sizeof(Datos));
  if (Aux == 0) {
    printf ("Socket cerrado\n");
  } else if (Aux == -1) {
      printf ("Error\n");
  } else if(Aux >0) {
    New.mensaje = Datos;
    New.numero = Aux;
  }
  return New;
}


/*
 *
 * Verifica si la cadena de caracteres mensaje es un comando valido.
 * De ser corrrecto, lo ejecuta.
 *
 * @param socket Socket a traves del cual se enviara el comando al servidor.
 * @param mensaje Cadena de caracteres que contiene un comando a ejecutar por el cliente.
 *
 */
void verificar_mensaje (int socket, char * mensaje) {
  char inicio[4];
  int i;
  /* se extraen los primeros 3 caracteres, correspondientes al comando */
  for (i = 0; i < 3; i++) {
    inicio[i] = *(mensaje);
    mensaje++;
  }
  mensaje++;

  /* se ejecuta la instruccion segun sea el comando */
  if (strcmp (inicio, "sal") == 0){
    cliente_escribe_mensaje (socket, Id_sal, NULL, 0);
  }

  else if (strcmp (inicio, "usu") == 0){
    cliente_escribe_mensaje (socket, Id_usu, NULL, 0);
  }

  else if ((strcmp (inicio, "men") == 0) && (strlen(mensaje) > 0)){
    cliente_escribe_mensaje (socket, Id_men, mensaje, strlen (mensaje));
  }

  else if ((strcmp (inicio, "sus") == 0) && (strlen(mensaje) > 0)){
    cliente_escribe_mensaje (socket, Id_sus, mensaje, strlen (mensaje));
  }

  else if (strcmp (inicio, "des") == 0){
    cliente_escribe_mensaje (socket, Id_des, NULL, 0);
  }

  else if ((strcmp (inicio, "cre") == 0) && (strlen(mensaje) > 0)){
    cliente_escribe_mensaje (socket, Id_cre, mensaje, strlen (mensaje));
  }

  else if ((strcmp (inicio, "eli") == 0) && (strlen(mensaje) > 0)){
    cliente_escribe_mensaje (socket, Id_eli, mensaje, strlen (mensaje));
  }

  else if (strcmp (inicio, "fue") == 0){
    cliente_escribe_mensaje (socket, Id_fue, NULL, 0);
  }
```

```
    /* si el comando no es valido, se notifica al usuario */
    else{
      printf ("\nERROR! Comando no valido\n");
    }
  }


  /*
   *
   * Lee un archivo de texto y ejecuta los comandos
   * que hayan en el.
   *
   * @param socket El socket a traves del cual enviara los comandos al servidor schat.
   * @param archivo Nombre del archivo a abrir.
   *
   */
  void leer_archivo (int socket, char * archivo)
  {
    char mensaje_enviar[1024];
    FILE * Entrada;

    /* se abre el archivo */
    if((Entrada = fopen(archivo, "r")) != NULL)
    {
      int i;
      /* mientras no se haya llegado al final se lee una linea y se llama a la funcion que lo ejecuta */
      while (fgets (mensaje_enviar,1024, Entrada) != NULL){
        i = strlen (mensaje_enviar);

        /* se elimina el salto de linea del final de la linea */
        if (i > 3){
        mensaje_enviar[i - 1] = '\0';
        }

        /* se envia la linea a la funcion que la analiza para luego enviarla al servidor */
        verificar_mensaje (socket, mensaje_enviar);
      }
      printf ("\n");
    }
    /* en caso de error, se notifica al usuario */
    else
      printf ("\nERROR al abrir el archivo.\n");
  }
```

```
/**
 ************************************************************************************************************************
 *****************
 *
 *
 *
 *
 *         ad88                                        88
 88              *
 *        d8"                                          ""
 88              *
 *        88
 88              *
 *      MM88MMM  88       88  8b,dPPYba,    ,adPPYba,  88   ,adPPYba,   8b,dPPYba,    ,adPPYba,  ,adPPYba,
 88,dPPYba,      *
 *        88     88       88  88P'   `"8a  a8"     ""  88  a8"     "8a  88P'   `"8a  a8P_____88  I8[    ""
 88P'    "8a     *
 *        88     88       88  88       88  8b          88  8b       d8  88       88  8PP"""""""   `"Y8ba,
 88       88     *
 *        88     "8a,   ,a88  88       88  "8a,   ,aa  88  "8a,   ,a8"  88       88  "8b,   ,aa  aa    ]8I  888
 88       88     *
 *        88      `"YbbdP'Y8  88       88   `"Ybbd8"'  88   `"YbbdP"'   88       88   `"Ybbd8"'  `"YbbdP"'  888
 88       88     *
 *
 *
 *
 *
 ************************************************************************************************************************
 *****************
 ******-> A U T O R E S <-*******
 *                             *
 * Johanna Chan        08-10218 *
 * Carlos Rodriguez    06-40189 *
 *                             *
 *******************************
 **/

#define KNRM  "\x1B[0m"
#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <pthread.h>
#include <sys/dir.h>
#include <dirent.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <getopt.h>
#include <netdb.h>
#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <math.h>


/*structura que se encarga de almacenar los parametros de la sintaxis.
 * Campo p: corresponde al numero de puerto habilitado por el servidor.
 * Campo s: corresponde al nombre de la sala.
 * La utiliza el cliente
 */
```

```
struct DatosC {
  char* host;
  int   puerto;
  char* nombre;
  char* archivo;
};typedef struct DatosC DatosCliente;

/* Estructura que se encarga de almacenar los parametros de la sintaxis.
 * Campo p: corresponde al numero de puerto habilitado por el servidor.
 * Campo s: corresponde al nombre de la sala.
 * La utiliza el servidor
 */
  struct DatosS {
    int puerto;
    char* sala;
  };
  typedef struct DatosS DatosServidor;

/* Estructura que se encarga de almacenar el tamaño del mensaje y el mensaje
 */
  struct tammensaje {
    char* mensaje;
    int   numero;
  };
  typedef struct tammensaje MSJ;

  struct MensajeDividido {
    char* comando;
    char* resto;
  };
  typedef struct MensajeDividido MSJdiv;

 /*
  *
  * Esta funcion se encarga de verificar si la cadena introducida corresponde al
  * nombre de un archivo, el nombre de un archivo puee estar constituido por
  * caracteres o digitos, es por eso que se utilizo la funcion isalnum
  * correspondiente a la libreria ctype.
  *
  * @param cadena es el string que se desea analizar
  * @return un booleano, true si la cadena corresponde al nombre de un archivo.
  *
  */

  extern int EsNombreArchivo(char * cadena);

 /*
  *
  * EsNombre es una funcion que se encarga de verificar si la cadena de
  * caracteres introducida es una palabra, esto lo logra mediante la funcion
  * isalpha() que retorna un 0 si no lo es.
  *
  * @param nombre es el string a ser analizado.
  * @return 1 si es una palabra, 0 de lo contrario.
  *
  */

  extern int EsPalabra(char * nombre);


 /*
  *
  * EsNumero es una funcion que se encarga de verificar si la cadena de
  * caracteres introducida corresponde a un numero.
  *
  * @param numero, es la cadena de caracteres a ser analizada.
  * @return 1 si es un numero, 0 de lo contrario.
  */
```

```
 extern int EsNumero(char * numero);


/*
 *
 * ContarPalabra es una funcion que se encarga de abrir un archivo y contar las
 * apariciones de una determinada palabra, esta funcion recibe como parametros
 * 2 cadenas de caracteres, la primera es el nombre del archivo a abrir y la
 * segunda es la palabra a la que se quiere contabilizar el numero de
 * apariciones.
 *
 * @param socket_cliente es el socket al que se enviara la informacion de cada linea.
 * @param NombreArchivo es el nombre del archivo que se va a examinar.
 * @param Palabra es la palabra que se desea contar a lo largo del archivo.
 * @param nombre nombre del usuario.
 * @return el numero de veces que aparece Palabra en el archivo NombreArchivo.
 *
 */

 extern int ContarPalabras(int socket_cliente, char * NombreArchivo, char * Palabra, char* nombre);


/*
 *
 * msjdivision es una funcion que se encarga de recibir un string y separarlo en partes,
 * y de esta forma obtener el comando y el mensaje enviado a traves del socket.
 *
 * @param msjcompleto es el string completo que se pretende dividir.
 * @return una estructura del tipo MSJdiv que almacena el comando y el resto del mensaje.
 *
 */

 extern MSJdiv msjdivision(char *msjcompleto);


/*
 *
 * ImprimirAyuda, como su nombre lo dice imprime la ayuda para la buena utilizacion
 * del programa.
 *
 */

 extern void ImprimirAyuda_servidor();


/*
 *
 * ImprimirAyuda, como su nombre lo dice imprime la ayuda para la buena utilizacion
 * del programa.
 *
 */

 extern void ImprimirAyuda_cliente();


/*
 *
 * Esta funcion se encarga de verificar que se introdujo en la entrada estandar
 * si fue un schat [-p <puerto>] [-s <sala>], si la entrada es valida,
 * retorna
 *
 * @param NumArg corresponde al numero de argumentos introducidos por el usuario.
 * @argu es el arreglo de caracteres que contienen las opciones y argumentos.
 * @return una estructura del tipo DatosServidor que almacena la informacion
 * colocada en la entrada estandar.
 *
 */

 extern DatosServidor Verificacion_servidor(int NumArg, char *argu[]);
```

```
/*
 *
 * Esta funcion se encarga de verificar que se introdujo en la entrada estandar
 * si fue un cchat [-h host] [-p <puerto>] [-n <nombre>] [-a <archivo>]
 *
 * @param NumArg corresponde al numero de argumentos introducidos por el usuario.
 * @argu es el arreglo de caracteres que contienen las opciones y argumentos.
 * @return una estructura del tipo DatosServidor que almacena la informacion
 * colocada en la entrada estandar.
 *
 */

 extern DatosCliente Verificacion_cliente(int NumArg, char *argu[]);


/*
 *
 * Esta funcion se encarga de imprimir el logo del programa del servidor.
 *
 */

 extern void logo_servidor();


/*
 *
 * Esta funcion se encarga de imprimir el logo del programa del cliente.
 *
 */

 extern void logo_cliente();


/*
 *
 * leer_servidor es una funcion que se encarga de leer la informacion contenida
 * en el socket, este se especifica en el argumento de la funcion.
 *
 * @socket_cliente corresponde al descriptor del socket de donde se desea leer el mensaje.
 * @return una estructura que contiene el mensaje y su tamano
 *
 */

 extern MSJ leer_servidor(int socket_cliente);


/*
 *
 * leer_servidor es una funcion que se encarga de leer la informacion contenida
 * en el socket, este se especifica en el argumento de la funcion.
 *
 * @socket_cliente corresponde al descriptor del socket de donde se desea leer el mensaje.
 * @return una estructura que contiene el mensaje y su tamano
 *
 */

 extern MSJ leer_cliente(int socket_cliente);


/*
 *
 * escribir_cliente es una funcion que se encarga de escribir informacion en el socket especificado
 * en el argumento de la funcion.
 *
 * @param socket_cliente correspondel al descriptor del socket en donde se desea escribir.
 * @param mensaje es el mensaje que se desea enviar
 * @paramm nombre corresponde al nombre del usuario que esta escribiendo el mensaje
 * @return una estructura con el mensaje enviado y su tamano.
 *
```

```
 */

 extern MSJ escribir_servidor(int socket_cliente, char* mensaje, char* nombre);


/*
 *
 * escribir_cliente es una funcion que se encarga de escribir informacion en el socket especificado
 * en el argumento de la funcion.
 *
 * @param socket_cliente correspondel al descriptor del socket en donde se desea escribir.
 * @param mensaje es el mensaje que se desea enviar
 * @paramm nombre corresponde al nombre del usuario que esta escribiendo el mensaje
 * @return una estructura con el mensaje enviado y su tamano.
 *
 */

 extern MSJ escribir_cliente(int socket_cliente, char* mensaje, char* nombre);

 /*
  *
  * Verifica si la cadena de caracteres mensaje es un comando valido.
  * De ser corrrecto, lo ejecuta.
  *
  * @param socket Socket a traves del cual se enviara el comando al servidor.
  * @param mensaje Cadena de caracteres que contiene un comando a ejecutar por el cliente.
  *
  */

 extern void verificar_mensaje (int socket, char *mensaje);

 /*
  *
  * Lee un archivo de texto y ejecuta los comandos
  * que hayan en el.
  *
  * @param socket El socket a traves del cual enviara los comandos al servidor schat.
  * @param archivo Nombre del archivo a abrir.
  *
  */

 extern void leer_archivo (int socket, char * archivo);
```

```
/**
 ***************************************************************************************************************
 ***************
 *
 *
 *
 *
 *      88                                  88                                  88          88
 *
 *      88                                  88           ,d                     88          88
 *
 *      88                                  88           88                     88          88
 *
 *      88,dPPYba,   ,adPPYYba,  ,adPPYba,  88,dPPYba,  MM88MMM  ,adPPYYba,  88,dPPYba,   88   ,adPPYba,
,adPPYba,      *
 *      88P'    "8a  ""     `Y8  I8[    ""  88P'    "8a   88     ""     `Y8  88P'    "8a  88  a8P_____88
a8"     ""      *
 *      88       88  ,adPPPPP88   `"Y8ba,   88       88   88     ,adPPPPP88  88       d8  88  8PP"""""""
8b              *
 *      88       88  88,    ,88  aa    ]8I  88       88   88,    88,    ,88  88b,   ,a8"  88  "8b,   ,aa  888
"8a,   ,aa      *
 *      88       88  `"8bbdP"Y8  `"YbbdP"'  88       88   "Y888  `"8bbdP"Y8  8Y"Ybbd8"'   88   `"Ybbd8"'  888
`"Ybbd8"'      *
 *
 *
 *
 *
 ***************************************************************************************************************
 ***************
 ******-> A U T O R E S <-*******
 *                            *
 * Johanna Chan        08-10218 *
 * Carlos Rodriguez    06-40189 *
 *                            *
 ******************************
 **/

#include "hashtable.h"

void liberarPila (NODOPILA * n);

/*
 * Inicializa una tabla de hash
 *
 * @param t apuntador al apuntador de la tabla de hash
 * @return TRUE si puede inicializar la tabla, FALSE en caso contrario.
 *
 */

  int createHashTable (HASHTABLES ** t) {
    if (!((*t) = (HASHTABLES *) malloc (sizeof (HASHTABLES))))
      return FALSE;
    int i;
    for (i = 0; i < TAMHASH; i++)
    {
      PILA * p;
      if (!(p = (PILA *) malloc (sizeof (PILA))))
        return FALSE;
      NODOPILA * inicio;
      if (!(inicio = (NODOPILA *) malloc (sizeof (NODOPILA))))
        return FALSE;
      p->inic = inicio;
      p->tam = 0;
      (*t)->tabla[i] = p;
    }
    (*t)->tam = 0;
    return TRUE;
  }
```

```
/*
 * Agrega un cliente a la tabla de hash.
 *
 * @param name Nombre del cliente.
 * @param desc Primer descriptor asociado.
 * @param desc2 Segundo descriptor asociado.
 * @param sala Sala a la que se asocia el cliente inicialmente. Puede ser NULL.
 *
 * @return TRUE si logra finalizar, FALSE en caso contrario.
 *
 */

  int add (char * name, int desc, int desc2, char * sala, HASHTABLES * t) {
    /* se verifica que no haya otro cliente con el mismo nombre */
    NODOPILA * n = buscar (name, t);
    if (n != NULL)
      return FALSE;

    /* Se crea un nodo y se llena con la informacion necesaria */
    NODOPILA * aux;
    if (!(aux = (NODOPILA *) malloc(sizeof(NODOPILA))))
      return FALSE;
    aux->name = name;
    aux->desc = desc;
    aux->desc2 = desc2;
    if (sala != NULL)
    {
      aux->numSalas = 1;

      /* se crea un nodo para la subpila y se llena con la informacion necesaria */
      NODOSUBPILA * aux2;
      if (!(aux2 = (NODOSUBPILA *) malloc (sizeof (NODOSUBPILA))))
        return FALSE;

      /* se asocia el sobnodo al nodo antes creado */
      aux2->nombre = sala;
      aux2->sig= aux->salas;
      aux->salas = aux2;
    }
    else
    {
      aux->numSalas = 0;
      aux->salas = NULL;
    }

    /* se busca la posicion de la tabla donde ira el cliente */
    int h = hashcode (name) % TAMHASH;

    /* se asocia el nodo con la informacion del cliente con la tabla */
    aux->sig = t->tabla[h]->inic->sig;
    t->tabla[h]->inic->sig = aux;
    t->tabla[h]->tam++;
    t->tam++;
    return TRUE;
  }


/*
 * Asocia una sala al cliente de nombre name.
 *
 * @param name Nombre del cliente.
 * @param sala Nombre de la sala.
 * @param t Apuntador a la tabla de hash.
 *
 * @return TRUE si logra finalizar, FALSE en caso contrario.
 *
 */

  int subAdd (char * name, char * sala, HASHTABLES * t) {
    /* se busca el cliente al que sera asociada la sala */
```

```
    NODOPILA * n = buscar (name, t);
    if (n == NULL)
      return FALSE;

    /* se verifica que no haya una sala con el mismo nombre */
    NODOSUBPILA * s = subbuscar (n, sala);
    if (s != NULL)
      return FALSE;

    /* se crea un subnodo donde se carga la informacion de la sala */
    NODOSUBPILA * aux;
    if (!(aux = (NODOSUBPILA *) malloc(sizeof(NODOSUBPILA))))
      return FALSE;

    /* se asocia el subnodo al cliente */
    aux->nombre = sala;
    aux->sig = n->salas;
    n->salas = aux;
    n->numSalas++;
    return TRUE;
  }


/*
 * Elimina a un cliente de la tabla de hash.
 *
 * @param name Nombre del cliente.
 * @param t Apuntador a la tabla de hash.
 *
 * @return TRUE si logra eliminarlo. FALSE si no pertenecia a la tabla.
 *
 */

  int del (char * name, HASHTABLES * t) {
    /* si la tabla esta vacia, se retorna FALSE */
    if (t->tam == 0)
      return FALSE;

    /* se busca la posicion de la tabla donde puede estar el cliente */
    int h = hashcode(name) % TAMHASH;
    int i = t->tabla[h]->tam;
    if (i == 0)
      return FALSE;
    NODOPILA * aux = t->tabla[h]->inic;
    int j = 1;

    /* se revisa la lista para buscar al cliente */
    while ((j <= i) && strcmp (name, aux->sig->name) != 0)
    {
      aux = aux->sig;
      j++;
    }
    if (j > i)
      return FALSE;
    /* si se consigue se libera su lista de salas asociadas */
    liberarPila (aux->sig);
    NODOPILA * aux2 = aux->sig;
    aux->sig = aux2->sig;
    /* se libera la memoria */
    free (aux2);
    t->tabla[h]->tam--;
    t->tam--;
    return TRUE;
  }


/*
 * Elimina una sala asociada al cliente de nombre name.
 *
 * @param name Nombre del cliente.
```

```
 * @param sala Nombre de la sala a eliminar.
 * @param t apuntador a la tabla de hash.
 *
 * @return TRUE si logra eliminarlo. FALSE si la sala no esta asociada al cliente.
 *
 */

  int subDel (char * name, char * sala, HASHTABLES * t) {
    /* se busca el cliente al que esta asociada la sala */
    NODOPILA * n = buscar (name, t);
    if (n == NULL)
      return FALSE;
    if (n->numSalas == 0)
      return FALSE;

    /* si la sala esta de primera en la lista de salas asociadas, se elimina facilmente */
    NODOSUBPILA * aux = n->salas;
    if (strcmp (sala, aux->nombre) == 0)
    {
      n->salas = aux->sig;
      free (aux);
      n->numSalas--;
      return TRUE;
    }
    int i = 1;

    /* se busca la sala en la lista de salas asociadas al cliente */
    while ((i < n->numSalas) && strcmp (aux->sig->nombre, sala) != 0)
    {
      i++;
      aux = aux->sig;
    }
    if (i == n->numSalas)
      return FALSE;
    NODOSUBPILA * aux2 = aux->sig;
    aux->sig = aux2->sig;

    /* se libera la memoria */
    free(aux2);
    n->numSalas--;
    return TRUE;
  }


/*
 * Busca un cliente en la tabla de hash.
 *
 * @param n Nombre del cliente.
 * @param t apuntador a la tabla de hash.
 *
 * @return El nodo asociado al cliente.
 *
 */

  NODOPILA* buscar (char * n, HASHTABLES * t) {
    if (t->tam == 0)
      return NULL;

    /* se busca la posicion donde debe estar el cliente */
    int h = hashcode(n) % TAMHASH;
    int i = t->tabla[h]->tam;
    if (i > 0)
    {
      int j = 1;
      NODOPILA * aux = t->tabla[h]->inic->sig;

      /* se busca al cliente */
      while ((j <= i) && (strcmp (n, aux->name) != 0))
      {
        aux = aux->sig;
```

```
      j++;
    }

    /* si no se consigue se retorna NULL */
    if (j > i)
      return NULL;
    else
      return aux;
   }
   return NULL;
 }


/*
 * Crea una representacion numerica de una cadena de caracteres.
 * Multiplica el valor de cada caracter segun la tabla ascii
 * por la posicion que ocupa el caracter en la cadena.
 *
 * @param c Cadena de caracteres a convertir a entero.
 *
 * @return Representacion en entero de la cadena de caracteres.
 *
 */

 int hashcode (char * c) {
    int l = strlen (c);
    int h = 0;
    int i;
    for (i =  0; i < l; i++)
    {
      h+= *(c + i) * (l - i);
    }
    return h;
 }


/*
 * Busca una sala asociada a un cliente.
 *
 * @param n Nodo asociado al cliente en una tabla de hash.
 * @param s Nombre de la sala a buscar.
 *
 * @return Nodo que asocia la sala al cliente en una tabla de hash.
 *
 */

 NODOSUBPILA* subbuscar (NODOPILA * n, char * s) {
    if (n->numSalas == 0)
        return NULL;
    int i = n->numSalas;
    int j = 1;
    NODOSUBPILA * aux = n->salas;

    /* se busca la sala en la lista de salas asociada al cliente */
    while ((j <= i) && strcmp (s, aux->nombre))
    {
        aux = aux->sig;
        j++;
    }

    /* si no se consigue se retorna NULL */
    if (j > i)
        return NULL;
    else
        return aux;
 }


/*
 * Libera un nodo de una pila perteneciente a una tabla de hash.
```

```
 *
 * @param n Nodo a liberar.
 *
 */

 void liberarPila (NODOPILA * n) {
     int i = n->numSalas;
     if (i == 1)
         free (n->salas);
     else if (i > 1)
     {
         int j = 2;
         NODOSUBPILA * aux = n->salas->sig;
         NODOSUBPILA * aux2 = n->salas;
         while (j < i)
         {
             free (aux2);
             aux2 = aux;
             aux = aux->sig;
         }
         free(aux2);
         free (aux);
         n->numSalas = 0;
     }
 }
```

```
/**
 *
 ********************************************************************************************************************
 **************
 *
 *
 *
 *
 *      88                                  88                                  88          88
 88              *
 *      88                                  88           ,d                     88          88
 88              *
 *      88                                  88           88                     88          88
 88              *
 *      88,dPPYba,   ,adPPYYba,  ,adPPYba,  88,dPPYba,  MM88MMM  ,adPPYYba,  88,dPPYba,   88   ,adPPYba,
 88,dPPYba,      *
 *      88P'    "8a  ""     `Y8  I8[    ""  88P'    "8a   88     ""     `Y8  88P'    "8a  88  a8P_____88
 88P'    "8a     *
 *      88       88  ,adPPPPP88   `"Y8ba,   88       88   88     ,adPPPPP88  88       d8  88  8PP"""""""
 88       88     *
 *      88       88  88,    ,88  aa    ]8I  88       88   88,    88,    ,88  88b,   ,a8"  88  "8b,   ,aa  888
 88       88     *
 *      88       88  `"8bbdP"Y8  `"YbbdP"'  88       88   "Y888  `"8bbdP"Y8  8Y"Ybbd8"'   88   `"Ybbd8"'  888
 88       88     *
 *
 *
 *
 ********************************************************************************************************************
 ****************
 *
 *
 * CONTENIDO : Archivo de cabecera de la libreria hashtable. Implementa una tabla de hash con distintas
 funciones.              *
 *
 *
 ******-> A U T O R E S <-
 ********************************************************************************************************************
 *                              *
 * Johanna Chan        08-10218 *
 * Carlos Rodriguez    06-40189 *
 *                              *
 ********************************
 **/

#ifndef HASHTABLE
#define HASHTABLE

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

#define TRUE 1
#define FALSE -1
#define TAMHASH 7

/*
 * Nodo de la estructura subpila.
 *
 * Contiene un apuntador al siguiente
 * elemento de una subpila y un apuntador
 * a una cadena de caracteres (nombre de la sala).
 */

  typedef struct t_nodoSubPila {
    char * nombre;
    struct t_nodoSubPila * sig;
  }NODOSUBPILA;
```

```
/*
 * Nodo de una pila.
 *
 * Contiene un apuntador al siguiente elemento
 * de una pila, un apuntador a una subpila (salas
 * a las que esta subscrito el usurio),
 * un apuntador a una cadena de caracteres (nombre
 * del usuario) y un descriptor asociado al usuario.
 */

  typedef struct t_nodoPila {
    char * name;
    int desc;
    int desc2;
    int numSalas;
    struct t_nodoSubPila * salas;
    struct t_nodoPila * sig;
  }NODOPILA;


/*
 * Nodo inicial de una pila.
 *
 * Contiene el numero de elementos en
 * dicha pila y un apuntador al comienzo
 * de la pila.
 */

  typedef struct t_pila {
    int tam;
    struct t_nodoPila * inic;
  }PILA;


/*
 * Tabla de hash.
 *
 * Contiene el numero de elementos en la
 * tabla de hash y un arreglo de apuntadores
 * a pilas de tamanio TAMHASH. Este tamanio es
 * un numero primo para minimizar las "colisiones"
 * producidas en una tabla de hash.
 */

  typedef struct t_hashtable {
    struct t_pila * tabla[TAMHASH];
    int tam;
  }HASHTABLES;


/*
 * Inicializa una tabla de hash
 *
 * @param t apuntador al apuntador de la tabla de hash
 *
 * @return TRUE si puede inicializar la tabla, FALSE en caso contrario.
 *
 */

  extern int createHashTable (HASHTABLES ** t);

/*
 * Agrega un cliente a la tabla de hash.
 *
 * @param name Nombre del cliente.
 * @param desc Primer descriptor asociado.
 * @param desc2 Segundo descriptor asociado.
 * @param sala Sala a la que se asocia el cliente inicialmente. Puede ser NULL.
```

```
 *
 * @return TRUE si logra finalizar, FALSE en caso contrario.
 *
 */

  extern int add (char * name, int desc, int desc2, char * sala, HASHTABLES * t);

/*
 * Asocia una sala al cliente de nombre name.
 *
 * @param name Nombre del cliente.
 * @param sala Nombre de la sala.
 * @param t Apuntador a la tabla de hash.
 *
 * @return TRUE si logra finalizar, FALSE en caso contrario.
 *
 */

  extern int subAdd (char * name, char * sala, HASHTABLES * t);

/*
 * Elimina a un cliente de la tabla de hash.
 *
 * @param name Nombre del cliente.
 * @param t Apuntador a la tabla de hash.
 *
 * @return TRUE si logra eliminarlo. FALSE si no pertenecia a la tabla.
 *
 */

  extern int del (char * name, HASHTABLES * t);

/*
 * Elimina una sala asociada al cliente de nombre name.
 *
 * @param name Nombre del cliente.
 * @param sala Nombre de la sala a eliminar.
 * @param t apuntador a la tabla de hash.
 *
 * @return TRUE si logra eliminarlo. FALSE si la sala no esta asociada al cliente.
 *
 */

  extern int subDel (char * name, char * sala, HASHTABLES * t);

/*
 * Crea una representacion numerica de una cadena de caracteres.
 * Multiplica el valor de cada caracter segun la tabla ascii
 * por la posicion que ocupa el caracter en la cadena.
 *
 * @param c Cadena de caracteres a convertir a entero.
 *
 * @return Representacion en entero de la cadena de caracteres.
 *
 */

  extern int hashcode (char * n);

/*
 * Busca un cliente en la tabla de hash.
 *
 * @param n Nombre del cliente.
 * @param t apuntador a la tabla de hash.
 *
 * @return El nodo asociado al cliente.
 *
 */

  extern NODOPILA* buscar (char * n, HASHTABLES * t);
```

```
/*
 * Busca una sala asociada a un cliente.
 *
 * @param n Nodo asociado al cliente en una tabla de hash.
 * @param s Nombre de la sala a buscar.
 *
 * @return Nodo que asocia la sala al cliente en una tabla de hash.
 *
 */

  extern NODOSUBPILA* subbuscar (NODOPILA * n, char * s);

#endif
```

```
/**
 *
 ***************************************************************************************************************
 ***************
 *
 *
 *
 *
 *      88  88  88            ad88888ba                              88
 *
 *      88  ""  88           d8"     "8b                             88                    ,d
 *
 *      88      88           Y8,                                     88                    88
 *
 *      88  88  88,dPPYba,   `Y8aaaaa,     ,adPPYba,    ,adPPYba,  88   ,d8   ,adPPYba,  MM88MMM  ,adPPYba,
,adPPYba,      *
 *      88  88  88P'    "8a    `""""8b,  a8"     "8a  a8"     ""  88 ,a8"   a8P_____88    88     I8[    ""
a8"     ""      *
 *      88  88  88       d8          `8b  8b       d8  8b          8888[     8PP"""""""    88      `"Y8ba,
8b              *
 *      88  88  88b,   ,a8"  Y8a     a8P  "8a,   ,a8"  "8a,   ,aa  88`"Yba,  "8b,   ,aa    88,    aa    ]8I  888
"8a,   ,aa      *
 *      88  88  8Y"Ybbd8"'    "Y88888P"    `"YbbdP"'    `"Ybbd8"'  88   `Y8a  `"Ybbd8"'    "Y888  `"YbbdP"'  888
`"Ybbd8"'      *
 *
 *
 *
 ***************************************************************************************************************
 ******************
 *
 *
 *
 * CONTENIDO : Archivo de cabecera de la libreria libSockets. Implementa distintas funciones a utilizar por
cchat y schat         *
 *
 *
 ******-> A U T O R E S <-
 ***************************************************************************************************************
 *                              *
 * Johanna Chan        08-10218 *
 * Carlos Rodriguez    06-40189 *
 *                              *
 ********************************
 **/

#include "libSockets.h"

/*
 * Envia el nombre de un cliente a traves de un socket.
 *
 * @param socket Socket a traves del cual se enviara el nombre.
 * @param nombre Nombre del cliente.
 *
 */

  void enviar_mi_nombre (int socket, char * nombre) {
    /* se declara y rellena la estructura que se utilizara para enviar el nombre */
    Nombre_sala n;
    strcpy (n.nombre, nombre);

    /* se envia la informacion */
    write (socket, &n, sizeof(Nombre_sala));
    Cabecera i;

    /* se lee la respuesta del servidor */
    read (socket, &i, sizeof(Cabecera));

    /* si ya existe un cliente con el mismo nombre, se notifica al usuario y se cierra el programa */
    if (i.id == FALSE)
```

```
    {
      printf ("\nERROR: ya existe un usuario con el mismo nombre conectado al servidor\n");
      close (socket);
      exit (0);
    }
  }


/*
 * Envia una instruccion al servidor.
 * La accion a ejecutar depende
 * de la instruccion que se haya ingresado.
 *
 * @param socket Socket a traves del cual se enviara la instruccion al servidor.
 * @param id Identificador del tipo de instruccion.
 * @param msj Mensaje que complementa la instruccion.
 * @param tam Tamanho del mensaje.
 *
 */

  void cliente_escribe_mensaje (int socket, int id, char * msj, int tam) {
    /* Se declara y rellena la cabecera */
    Cabecera cab;
    cab.id = id;

    /* Se envia la cabecera */
    write (socket, &cab, sizeof(cab));

    /* se realiza la opcion necesaria */
    switch (id)
    {
      case Id_sal:
        cliente_recibe_lista (0, socket);
        break;

      case Id_usu:
        cliente_recibe_lista (1, socket);
        break;

      /* en estos casos solo seenvia el complemento de la instruccion.El resto es hecho por el servidor */
      case Id_men:
      case Id_sus:
      case Id_cre:
      case Id_eli:
        write (socket, msj, tam);
        break;

      case Id_des:
        break;

      case Id_fue:

        /* si el usuario introdujo fue, se cierra el socket y la aplicacion */
        close (socket);
        exit (0);
    }

    /* se espera a que el servidor termine de interpretar la instruccion */
    read (socket, &cab, sizeof (Cabecera));
  }


/*
 * Envia un mensaje a todos los usuarios que pertenezcan a alguna
 * sala a la que pertenece el usuario que envia el mensaje.
 *
 * @param m Estructura con informacion necesaria acerca del mensaje.
 * @param t_usuarios Tabla de hash que contiene a los usuarios.
 * @param t_salas Tabla de hash que contiene a las salas.
 * @param name Nombre del cliente que envia el mensaje.
```

```
 *
 */

  void enviar_mensaje (Manda_mensaje m, HASHTABLES * t_usuarios, HASHTABLES * t_salas, char * name) {
    /* se busca al cliente para acceder a las salas a las que esta suscrito */
    NODOPILA * aux = buscar (name, t_usuarios);
    NODOSUBPILA * aux2 = aux->salas;
    NODOPILA * usuario;
    NODOPILA * auxx;
    NODOSUBPILA * auxx2;
    int i, j;

    /* para cada sala se busca la lista de usuarios suscritos a ella */
    for (i = 0; i < aux->numSalas; i++)
    {
      auxx = buscar (aux2->nombre, t_salas);
      strcpy (m.sala, auxx->name);
      auxx2 = auxx->salas;

      /* se envia el mensaje a cada usuario perteneciente a la sala */
      for (j = 0; j < auxx->numSalas; j++)
      {
        usuario = buscar (auxx2->nombre, t_usuarios);
        write (usuario->desc2, &m, sizeof(Manda_mensaje));
        auxx2 = auxx2->sig;
      }
      aux2 = aux2->sig;
    }
  }

/*
 * Recibe una instruccion de un cliente y decide que operacion debe realizar
 * en funcion de la instruccion recibida.
 *
 * @param socket Socket a traves del cual se comunica el servidor con el cliente.
 * @param tU Tabla de hash de los usuarios.
 * @param name Nombre del usuario que envia la instruccion.
 * @param tS Tabla de hash de las salas.
 *
 * @return TRUE si logra realizar su funcion. FALSE en caso contrario.
 *
 */

  int servidor_recibe_mensaje (int socket, HASHTABLES * tU, char * name, HASHTABLES * tS) {
    Cabecera cab;
    /* se lee la cabecera de la instruccion (el tipo de instruccion) */
    read (socket, &cab, sizeof (Cabecera));
    int i;

    /* se decide que operacion realizar en funcion de la instruccion que desea ejecutar el cliente */
    switch (cab.id)
    {

      /* caso instruccion sal */
      case Id_sal:
      {
        servidor_envia_lista (socket, tS);
        break;
      }

      /* caso instruccion usu */
      case Id_usu:
      {
        servidor_envia_lista (socket, tU);
        break;
      }

      /* caso instruccion men */
      case Id_men:
      {
```

```
      Nombre_sala s;
      memset (s.nombre, 0, 200);
      read (socket, &s, sizeof (Nombre_sala));
      Manda_mensaje m;
      strcpy (m.nombre, name);
      strcpy(m.mensaje, s.nombre);
      enviar_mensaje (m, tU, tS, name);
      break;
    }

    /* caso instruccion sus */
    case Id_sus:
    {
      Nombre_sala s;
      memset (s.nombre, 0, 200);
      read (socket, &s, sizeof (Nombre_sala));
      char * c = strdup (s.nombre);
      i = subAdd (name, c, tU);
      if (i == TRUE)
        subAdd (s.nombre, name, tS);
      break;
    }

    /* caso instruccion des */
    case Id_des:
    {
      NODOPILA * aux = buscar (name, tU);
      NODOSUBPILA * aux2;
      while (aux->numSalas > 0)
      {
        aux2 = aux->salas;
        i = subDel (aux2->nombre, name, tS);
        if (i == TRUE)
          subDel (name, aux2->nombre, tU);
      }
      break;
    }

    /* caso instruccion cre */
    case Id_cre:
    {
      Nombre_sala s;
      memset (s.nombre, 0, 200);
      read (socket, &s, sizeof (Nombre_sala));
      char * c = strdup (s.nombre);
      add (c, 0, 0, NULL, tS);
      break;
    }

    /* caso instruccion eli */
    case Id_eli:
    {
      Nombre_sala s;
      memset (s.nombre, 0, 200);
      read (socket, &s, sizeof (Nombre_sala));
      char * c = strdup (s.nombre);
      NODOPILA * aux = buscar (c, tS);
      if (aux != NULL)
      {
        NODOSUBPILA * aux2;
        while (aux->numSalas > 0)
        {
          aux2 = aux->salas;
          subDel (aux2->nombre, c, tU);
          subDel (c, aux2->nombre, tS);
        }
        del (c, tS);
      }
      break;
    }
```

```
      /* caso instruccion fue */
      case Id_fue:
      {
        NODOPILA * aux = buscar (name, tU);
        NODOSUBPILA * aux2;
        while (aux->numSalas > 0)
        {
          aux2 = aux->salas;
          subDel (aux2->nombre, name, tS);
          subDel (name, aux2->nombre, tU);
        }
        del (name, tU);
        close (socket);
        return 0;
      }
    }

    /* se informa al cliente que termino de interpretar la instruccion */
    write (socket, &cab, sizeof(Cabecera));
    return FALSE;
  }


/*
 * Recibe la lista de salas o usuarios segun sea el caso.
 * Se utiliza luego de que el cliente ejecuta la instruccion sal o usu.
 *
 * @param id Identificador de la instruccion utilizada. 0 para sal, 1 para usu.
 * @param socket Socket a traves del cual el cliente se comunica con el servidor.
 *
 */

  void cliente_recibe_lista (int id, int socket) {
    Cant_usu_sala cant;
    Nombre_sala nombre;

    /* Se lee cantidad de usuarios / salas a recibir */
    read (socket, &cant, sizeof (Cant_usu_sala));
    int i;
    if (id == 0)
      printf ("Salas:\n");
    else
      printf ("Usuarios:\n");

    /* se recibe el nombre de cada usuario / sala en el servidor */
    for (i = 0; i < cant.cantidad; i++)
    {
      strcpy (nombre.nombre, "");
      read (socket, &nombre, sizeof (Nombre_sala));
      printf (" %s", nombre.nombre);
    }
    printf ("\n\n");
  }


/*
 * Envia todos los usuarios / salas pertenecientes al servidor.
 * Inicialmente no sabe si envia salas o usuarios ya que esa
 * decision fue tomada por la funcion servidor_recibe_mensaje.
 *
 * @param socket Socket a traves del cual el servidor se comunicara con el cliente.
 * @param t Tabla de hash que contiene los usuarios / salas.
 *
 */

  void servidor_envia_lista (int socket, HASHTABLES * t) {
    Cant_usu_sala cant;
    cant.cantidad = t->tam;
    write (socket, &cant, sizeof (Cant_usu_sala));
```

```
    NODOPILA * aux;
    Nombre_sala nsala;
    int i, j;

    /* para cada lista de sala se envian los usuarios / salas que contiene */
    for (i = 0; i < TAMHASH; i++)
    {
      if (t->tabla[i]->tam > 0)
      {
        aux = t->tabla[i]->inic->sig;
        for (j = t->tabla[i]->tam; j > 0; j--)
        {
          strcpy (nsala.nombre, aux->name);
          write (socket, &nsala, sizeof(Nombre_sala));
          aux = aux->sig;
        }
      }
    }
  }
```

```
/**
 *
 ********************************************************************************************************************
 ***************
 *
 *
 *
 *
 *      88  88  88            ad88888ba                              88
 88              *
 *      88  ""  88           d8"     "8b                             88                    ,d
 88              *
 *      88      88           Y8,                                     88                    88
 88              *
 *      88  88  88,dPPYba,   `Y8aaaaa,     ,adPPYba,    ,adPPYba,  88   ,d8   ,adPPYba,  MM88MMM  ,adPPYba,
 88,dPPYba,      *
 *      88  88  88P'    "8a    `""""8b,  a8"     "8a  a8"     ""  88 ,a8"   a8P_____88    88     I8[    ""
 88P'    "8a     *
 *      88  88  88       d8          `8b  8b       d8  8b          8888[     8PP"""""""    88      `"Y8ba,
 88       88     *
 *      88  88  88b,   ,a8"  Y8a     a8P  "8a,   ,a8"  "8a,   ,aa  88`"Yba,  "8b,   ,aa    88,    aa    ]8I  888
 88       88     *
 *      88  88  8Y"Ybbd8"'    "Y88888P"    `"YbbdP"'    `"Ybbd8"'  88   `Y8a  `"Ybbd8"'    "Y888  `"YbbdP"'  888
 88       88     *
 *
 *
 *
 *
 ********************************************************************************************************************
 ******************
 ******-> A U T O R E S <-*******
 *                            *
 * Johanna Chan        08-10218 *
 * Carlos Rodriguez    06-40189 *
 *                            *
 ******************************
 **/

#ifndef LIBSOCKETS
#define LIBSOCKETS

#include "hashtable.h"

/*
 * Estructura que almacena un entero
 * que representa un identificador de
 * alguna instruccion.
 *
 */

  typedef struct t_cabecera {
    int id;
  }Cabecera;


/*
 * Estructura que contiene la informacion
 * asociada a un mensaje, como el nombre
 * del cliente que lo envia, la sala a la
 * que pertenece y el contenido del mensaje.
 *
 */

  typedef struct t_manda_mensaje {
    char mensaje [1024];
    char nombre [200];
    char sala [200];
  }Manda_mensaje;
```

```c
/*
 * Estructura que contiene una cadena de
 * caracteres asociada al nombre de una sala
 * o usuario.
 *
 */

  typedef struct t_sala {
    char nombre [200];
  }Nombre_sala;


/*
 * Estructura que contiene un entero
 * que representa la cantidad de usuarios /
 * salas pertenecientes a un servidor.
 *
 */

  typedef struct t_cantidad_usu_sala {
    int cantidad;
  }Cant_usu_sala;


/*
 * Lista que enumera el identificador de
 * una instruccion envia por un cliente
 * a un servidor.
 *
 */

  typedef enum identificadores {
    Id_sal,
    Id_usu,
    Id_men,
    Id_sus,
    Id_des,
    Id_cre,
    Id_eli,
    Id_fue
  }Identificadores;


/*
 * Envia el nombre de un cliente a traves de un socket.
 *
 * @param socket Socket a traves del cual se enviara el nombre.
 * @param nombre Nombre del cliente.
 *
 */

  extern void enviar_mi_nombre (int socket, char * nombre);

/*
 * Envia una instruccion al servidor.
 * La accion a ejecutar depende
 * de la instruccion que se haya ingresado.
 *
 * @param socket Socket a traves del cual se enviara la instruccion al servidor.
 * @param id Identificador del tipo de instruccion.
 * @param msj Mensaje que complementa la instruccion.
 * @param tam Tamanho del mensaje.
 */

  extern void cliente_escribe_mensaje (int socket, int id, char * msj, int tam);

/*
 * Recibe una instruccion de un cliente y decide que operacion debe realizar
 * en funcion de la instruccion recibida.
 *
```

```
 * @param socket Socket a traves del cual se comunica el servidor con el cliente.
 * @param tU Tabla de hash de los usuarios.
 * @param name Nombre del usuario que envia la instruccion.
 * @param tS Tabla de hash de las salas.
 *
 * @return TRUE si logra realizar su funcion. FALSE en caso contrario.
 *
 */

  extern int servidor_recibe_mensaje (int socket, HASHTABLES * tU, char * name, HASHTABLES * tS);

/*
 * Recibe la lista de salas o usuarios segun sea el caso.
 * Se utiliza luego de que el cliente ejecuta la instruccion sal o usu.
 *
 * @param id Identificador de la instruccion utilizada. 0 para sal, 1 para usu.
 * @param socket Socket a traves del cual el cliente se comunica con el servidor.
 *
 */

  extern void cliente_recibe_lista (int id, int socket);

/*
 * Envia todos los usuarios / salas pertenecientes al servidor.
 * Inicialmente no sabe si envia salas o usuarios ya que esa
 * decision fue tomada por la funcion servidor_recibe_mensaje.
 *
 * @param socket Socket a traves del cual el servidor se comunicara con el cliente.
 * @param t Tabla de hash que contiene los usuarios / salas.
 *
 */

  extern void servidor_envia_lista (int socket, HASHTABLES * t);

/*
 * Envia un mensaje a todos los usuarios que pertenezcan a alguna
 * sala a la que pertenece el usuario que envia el mensaje.
 *
 * @param m Estructura con informacion necesaria acerca del mensaje.
 * @param t_usuarios Tabla de hash que contiene a los usuarios.
 * @param t_salas Tabla de hash que contiene a las salas.
 * @param name Nombre del cliente que envia el mensaje.
 *
 */

  extern void enviar_mensaje (Manda_mensaje m, HASHTABLES * t_usuarios, HASHTABLES * t_salas, char * name);

#endif
```

```
#############################################################################################
#                                                                                           #
#                                                                                           #
#      88b           d88              88                        ad88  88  88                #
#      888b         d888              88                       d8"    ""  88                #
#      88`8b       d8'88              88                       88         88                #
#      88 `8b     d8' 88  ,adPPYYba,  88   ,d8   ,adPPYba,  MM88MMM  88  88   ,adPPYba,     #
#      88  `8b   d8'  88  ""     `Y8  88 ,a8"   a8P_____88    88     88  88  a8P_____88     #
#      88   `8b d8'   88  ,adPPPPP88  8888[     8PP"""""""    88     88  88  8PP"""""""     #
#      88    `888'    88  88,    ,88  88`"Yba,  "8b,   ,aa    88     88  88  "8b,   ,aa     #
#      88     `8'     88  `"8bbdP"Y8  88   `Y8a  `"Ybbd8"'    88     88  88   `"Ybbd8"'     #
#                                                                                           #
#                                                                                           #
#############################################################################################
#                                                                                           #
# CONTENIDO: Makefile para aplicacion "main"                                                #
#                                                                                           #
#############################################################################################
#######-> A U T O R E S <-######
#                             #
# Johanna Chan        08-10218 #
# Carlos Rodriguez    06-40189 #
#                             #
###############################

CC      = gcc  -Wall -ggdb
OBJS    = funciones.o hashtable.o libSockets.o cchat.o
OBJS1   = funciones.o hashtable.o libSockets.o schat.o
HILO    = -pthread

all:            cchat schat

cchat:  $(OBJS)
                $(CC) $(OBJS) -o cchat $(HILO)

schat:  $(OBJS1)
                $(CC) $(OBJS1) -o sschat $(HILO)

cchat.o:        cchat.c funciones.h
                $(CC) -c cchat.c $(HILO)

schat.o:        schat.c funciones.h
                $(CC) -c schat.c $(HILO)

funciones.o:    funciones.c funciones.h
                $(CC) -c funciones.c $(HILO)

libSockets.o:   libSockets.c libSockets.h hashtable.h
                $(CC) -c libSockets.c $(HILO)

hashtable.o:    hashtable.c hashtable.h
                $(CC) -c hashtable.c $(HILO)

clean:
                rm -f ./*.o ./*.c~ ./*.h~ cchat sschat Makefile~
```