INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

REDES DE COMPUTADORAS

PROYECTO FINAL

"Creación de un chat en tiempo real con sockets en lenguaje C"

GRUPO: 2CV15

EQUIPO: El Siuuu Team

INTEGRANTES:

Fischer Salazar César Eduardo

López García José Eduardo

Meza Vargas Brandon David

PROFESOR: Fabian Gaspar Medina

FECHA: 20/12/21

índice

Introducción

A lo largo de la existencia de las redes, desde sus inicios era bastante importante conseguir que pudiera haber interacción a distancia entre dos personas que se encontraban bastante distanciadas una de otra, ya que antes era bastante complicado que se diera una comunicación en directo, cartas, faxs, telegramas, tardaban mucho en poder llegar a su destinatario, al igual que su respuesta.

Con el paso de los años, y de la mano con la evolución de la tecnología, ha sido posible para el ser humano poder contar con diferentes dispositivos y elementos que pueden conformar cualquier topología de red (anillo, estrella, malla, etc.) y que se cumpliera con dicho propósito.

Y es claro, se sabía dentro de esta área de debía existir alguna manera en las que dichas conversaciones fueran más fluidas, más rápidas, en tiempo real, alrededor del mundo, en donde se creara un ambiente agradable en el que se sintiera una buena convivencia y más conexiones de gente desde diferentes partes.

Hoy por hoy, podemos contar en nuestros ordenadores y celulares con sistemas de chat en línea que realizan esa interacción que se ha ido dando año con año entre diferentes plataformas, ha evolucionado y parece que irá teniendo un desarrollo y una evolución más allá de lo que tenemos al alcance de nuestras manos.

Llegados a este punto, con lo que hemos aprendido a lo largo de la materia, con prácticas realizadas en el sistema operativo Linux, y una que otra simulación, surgió un objetivo de poder tratar de recrear un sistema de chat parecido a uno con los que podemos contar actualmente; el cual se buscaba que tuviera una función bidireccional en el que se permitiera la interacción de usuarios de distintas partes, constando de un cliente y un servidor, donde sea posible verse reflejado en ambos puntos la cantidad de usuarios conectados, además de que contaran con un ID con el que se identificaran fácilmente al momento del envío de mensajes, e informar de su desconexión, todo esto a través de la programación en C, con apoyo del manual de Linux y el concepto de sockets crudos vistos en clase.

Desarrollo

Diagrama de flujo
Programa del cliente

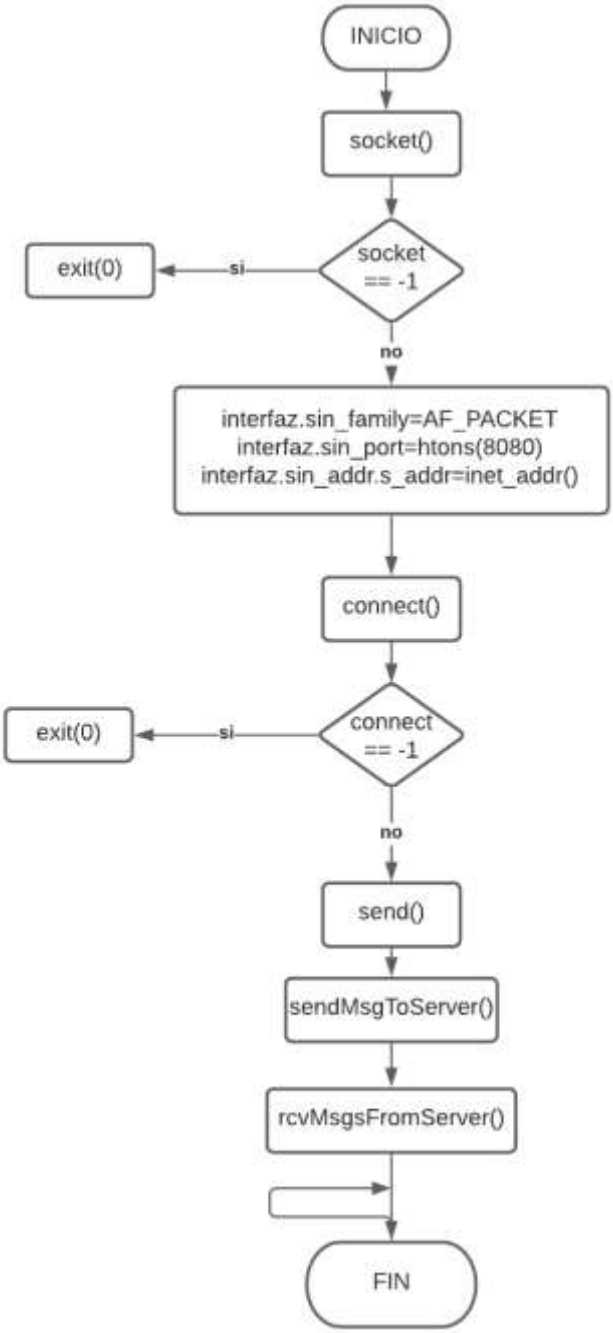

Imagen 1. main del programa cliente.

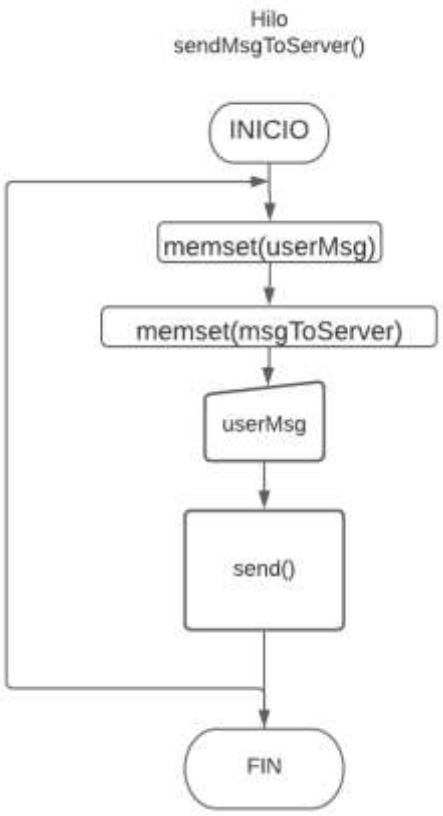

Imagen 2. Hilo que manda mensaje al servidor

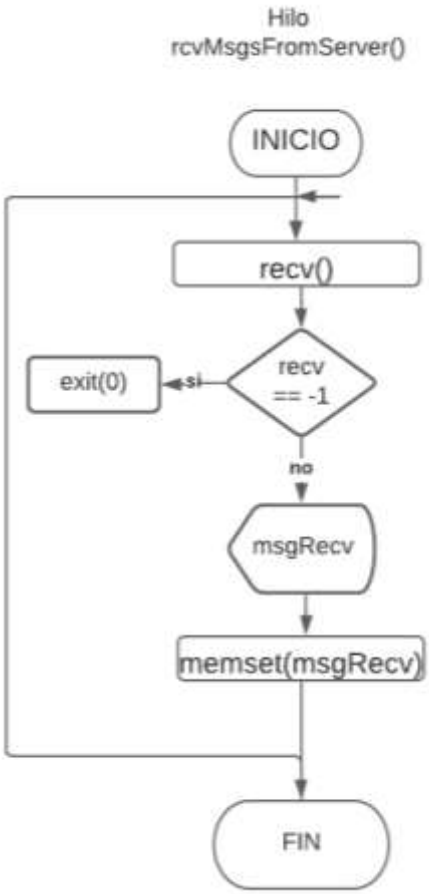

Imagen 3. Hilo que recibe mensajes del servidor.

Programa del servidor

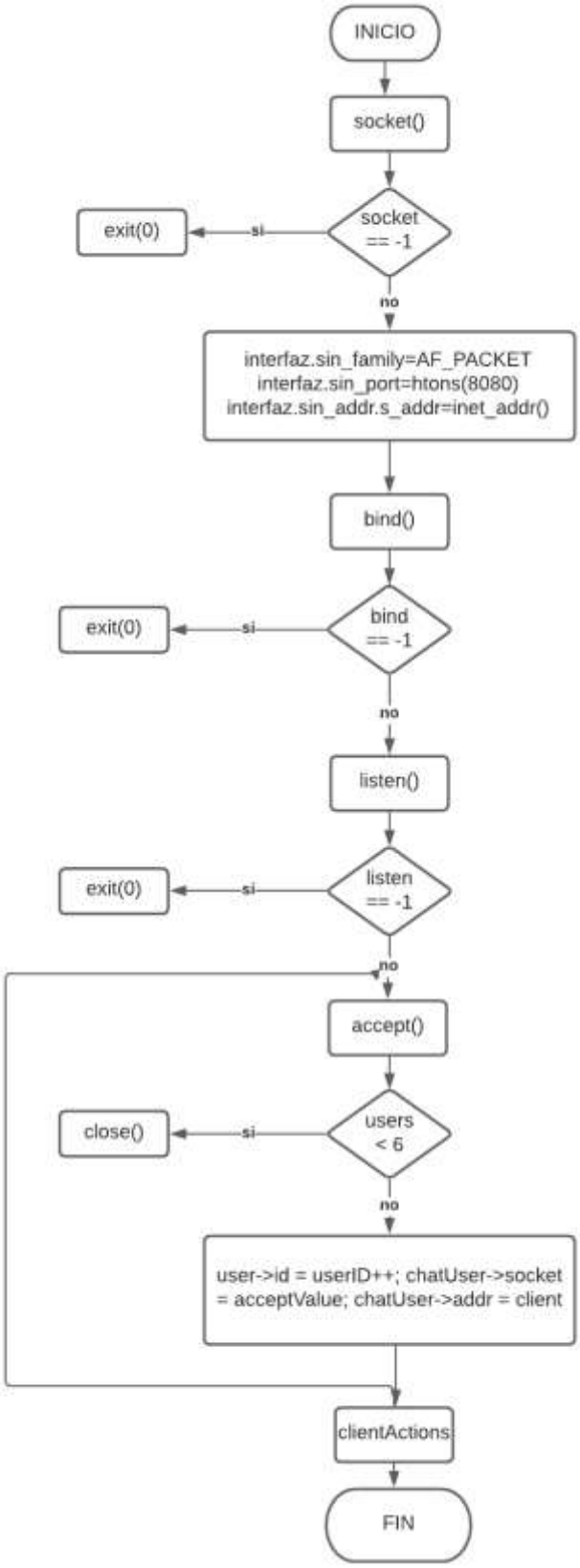

Imagen 4. main del servidor

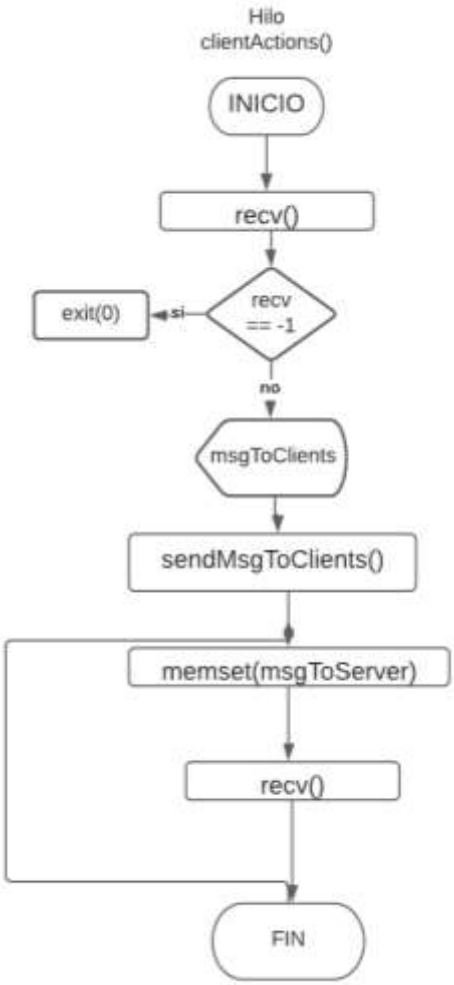

Imagen 5. Hilo que gestiona las acciones del cliente.

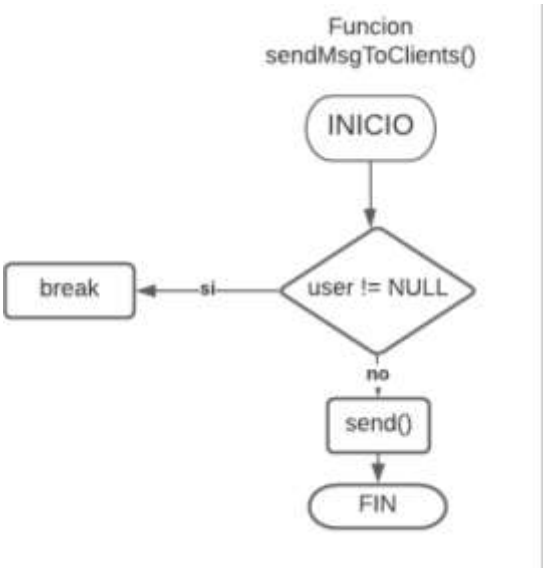

Imagen 6. Función que manda mensajes a los clientes conectados.

Sockets

Un socket es un tipo de software que actúa como un punto final, que funciona estableciendo un enlace de comunicación de red bidireccional entre el extremo del servidor y el programa receptor del cliente. Estos sockets se realizan y movilizan junto con llamadas de función, y es capaz de simplificar el funcionamiento de un programa porque los programadores sólo tienen que preocuparse de manipular las funciones del socket para transportar los mensajes a través de la red correctamente.

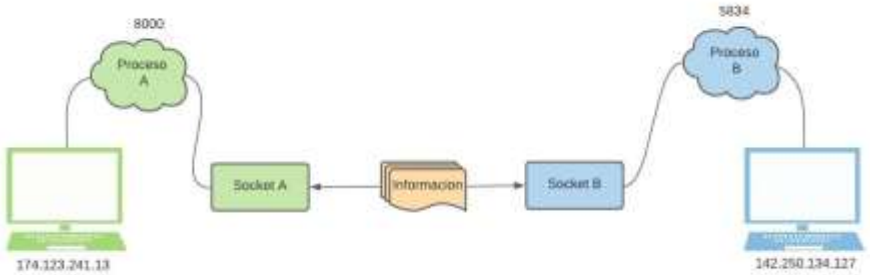

Imagen 7. Funcionamiento de sockets.

Para un modelo cliente-servidor orientado a la conexión, el socket en el proceso del servidor espera la petición de un cliente, aquí, el servidor necesita establecer una dirección que los clientes puedan usar para encontrar y conectarse al servidor. Cuando se establece una conexión con éxito, el servidor esperará a que los clientes soliciten un servicio. El intercambio de datos cliente-servidor tendrá lugar si el cliente se conecta al servidor a través del socket, entonces, el servidor responderá a la solicitud del cliente y enviará una respuesta.

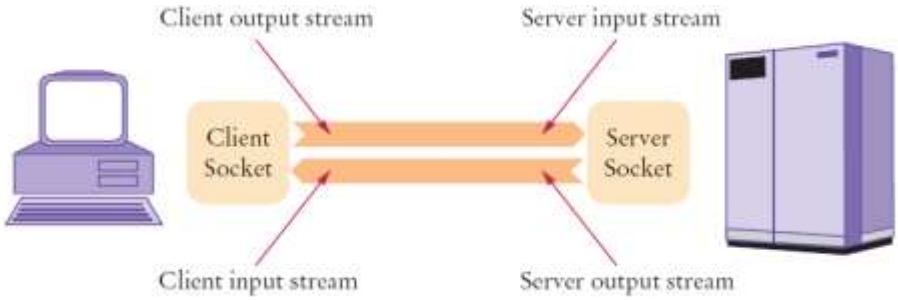

Imagen 8. Sockets de cliente y servidor

Sockets en c

Dentro del lenguaje C, la palabra socket designa un concepto abstracto por el cual dos programas, que posiblemente se encuentren situados en computadoras distintas, pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada.

Un socket es, un conector o enchufe con el que podremos conectarnos a ordenadores remotos o permitir que éstos se conecten al nuestro a través de la red. En realidad, un socket no es más que un descriptor de fichero un tanto especial. Dentro de lo que es UNIX todo es un fichero, así que, para enviar y recibir datos por la red, sólo tendremos que escribir y leer en un fichero un poco especial.

Funciones

I. Función socket()

Es una llamada al sistema, esta función devuelve un descriptor de archivo, al igual que la función open() al crear o abrir un archivo. Durante la llamada se reservan los recursos necesarios para el punto de comunicación, pero no se especifica nada con respecto a la dirección asociada al mismo.

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

II. Función bind()

Llamada al sistema que permite asignar una dirección a un socket existente. El sistema no atenderá a las conexiones de clientes, simplemente registra que cuando empiece a recibirlas le avisará a la aplicación. En esta llamada se debe indicar el número de servicio sobre el que se quiere atender.

```
int bind (int sockfd, const struct sockaddr *myaddr, int addrlen);
```

III. Función listen()

Avisar al sistema operativo que comience a atender la conexión de red. El sistema registrará la conexión de cualquier cliente para transferirla a la aplicación cuando lo solicite. Si llegan conexiones de clientes más rápido de lo que la aplicación es capaz de atender, el sistema almacena en una cola las mismas y se podrán ir obteniendo luego.

```
int listen (int s, int backlog)
```

IV. Función accept()

Pedir y aceptar las conexiones de clientes al sistema operativo. El sistema operativo entregará el siguiente cliente de la cola. Si no hay clientes, se quedará bloqueada hasta que algún cliente se conecte.

```
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```

V. Funciones de envío y recibo de datos

un socket es un descriptor de fichero, así que en teoría sería posible escribir con write() y leer con read(), pero hay funciones mucho más cómodas para hacer esto. Dependiendo si

el socket que utilicemos es de tipo socket de flujo o socket de datagramas emplearemos unas funciones u otras:

- Para sockets de flujo: send() y recv().
- Para sockets de datagramas: sendto() y recvfrom().

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

int sendto (int s, const void *msg, size_t len, int flags, const struct socka
ddr *to, socklen_t tolen);

int recv (int s, void *buf, size_t len, int flags);

int recvfrom (int s, void *buf, size_t len, int flags, struct sockaddr *from,
socklen_t *fromlen);
```

VI. Función close()

Esta función cierra el descriptor de fichero del socket, liberando el socket y denegando cualquier envío o recepción a través de este. Para un mejor control de este, puede utilizarse la función shutdown().

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

Programa realizado

Programa del cliente

A continuación, se presenta el código correspondiente al cliente, se encuentra comentada línea por línea, de esta forma no será necesario explicar más.

```
//************************************************************************
        //*       LIBRERIAS INCLUIDAS       *//
//************************************************************************

#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio_ext.h>
#include <pthread.h>

//************************************************************************
        //*       Estrucutra para argumentos de hilos       *//
//************************************************************************

struct args{
    char name[50];
    int socket;
};

//************************************************************************
        //*       Declaración de funciones       *//
//************************************************************************
void sendMSgsToServer(void *args);
void rcvMsgsFromServer(void *args);
```

```c
/*
    *Funciones para colores
*/
void red(){
    printf("\033[1;31m");
}
void reset(){
    printf("\033[0m");
}
void yellow(){
    printf("\033[1;33m");
}
void blue(){
    printf("\033[1;34m");
}
void green(){
    printf("\033[1;32m");
}
void white(){
    printf("\033[1;37m");
}
void black(){
    printf("\033[1;30m");
}
void purple(){
    printf("\033[1;35m");
}
void cyan(){
    printf("\033[1;36m");
}
//*************************************************************************
        //*       Variables globales       *//
//*************************************************************************
```

```c
int main(){
    int tcpSocket = 0;                  //*Variable para socket
    struct sockaddr_in server;          //*Estrucutra del servidor
    char userName[50];                  //*Variable para nombre de usuario
    int con = 0;                        //*Variable para la conexion al servidor
    pthread_t clientThread, serverThread; //*Variable para los hilos del cliente y servidor
    cyan();
    printf("\t\t\t*****Bienvenido al Siuuu Team chat!*****");
    printf("\n");
    printf("\n");
    green();
    printf("Ingresa tu nombre para conectarte al chat: ");
    reset();
    fgets(userName,50,stdin);                   //*Leemos el nombre del usuario
    userName[strlen(userName)-1]='\0';          //*QUitamos el salto de linea al final de la cadena

    /*
        *La siguiente función es para abrir un socket, en este proyecto estamos usando un scoket
        *tcp (SOCK_STREAM) ya que este nos permite una comunicación bidireccional y es lo que necesitamos
        *para el chat.
        *Para sus métodos y opcioens podemos consultar los manuales 7 ip y socket
    */
    tcpSocket = socket(AF_INET, SOCK_STREAM, 0);

    //*Si obtenemos -1 al abrir el socket arrojamos un error, en caso contrario seguimos con el programa
    if(tcpSocket == -1){
        red();
        perror("\nError al abrir el socket");
        exit(0);
```

```c
}else{
    server.sin_family = AF_INET;                    //*Siempre le asignamos AF_INET que es la familia de direcc
    server.sin_port = htons(8080);                  //*Es el puerto con los bytes en orden de red
    server.sin_addr.s_addr = inet_addr("10.0.2.15");    //*Dirección IP del host

    /*
        *Establecemos una conexion en un socket con la siguiente funcion connect.
        *Como nuestro socket es tipo SOCK_STREAM, esta funcion intenta hacer una conexion
        *a otro socket, siendo en este caso nuestro otro socket el servidor (remota).
        *El primer parametro es nuestro socket, el segundo es el otro socket a conectarnos
        *el ultimo parametro es la longitud.
        *Mas detalles en man connect
    */
    con = connect(tcpSocket, (struct sockaddr *)&server, sizeof(server));

    //*Si obtenemos -1 al hacer la conexion arrojamos un error, en caso contrario seguimos
    if(con == -1){
        red();
        perror("\nError al conectarse al chat :(");
        exit(0);
    }else{
        /*
            *Usamos send para mandar mensajes a otro socket, al estar usando connect, procedemos a usar send
            *ya que con send nos asegura el envio del mensaje solo si estamos
            *El primer parametro es el socket, el segundo el mensaje, el tercero la longitud del mensaje
            *y el ultimo el parametro flags
```

```
            *Mas detalles en el manual man send
        */
        send(tcpSocket, userName, 50, 0);               //*Mandamos el mensaje con el usuario que se conecto
        cyan();
        printf("\t\t\t*****Te has conectado %s!*****",userName);
        printf("\n");
        printf("\n");
        //*Establecemos los argumentos que tendrán los hilos
        struct args *thread_args = (struct args *)malloc(sizeof(struct args));
        strcpy(thread_args -> name, userName);
           thread_args -> socket = tcpSocket;
        /*
            *Creamos un nuevo hilo para el envio de mensajes del cliente al servidor
            *Para más detalles de hilos revisar el manual man pthreads
            *Si no devuelve 0 es que hubo error al crear el hilo
        */
        if(pthread_create(&clientThread, NULL, (void *)sendMSgsToServer, (void *)thread_args) != 0){
            red();
            perror("\nError al crear hilo de cliente");
            exit(0);
        }
        /*
            *Creamos el hilo sel servidor en donde nos encargaremos de estar recibiendo
               *los mensajes de otros usuarios o del servidor
            *Si no devuelve 0 es que hubo error al crear el hilo
        */
        if(pthread_create(&serverThread, NULL, (void *)rcvMsgsFromServer, (void *)thread_args) != 0){
            red();
            perror("\nError al crear hilo de cliente");
            exit(0);
        }

        while(1){
            //*For para que no se termine el programa hasta que el usuario salga del chat
        }
    }
}
```

```
    close(tcpSocket);       //*Cerramos el socket
    reset();
    return 0;               //*Salimso del programa
}


/**
 * @brief Hilo que se encarga de mandar mensajes al servidor
 * @param args Mandamos el nombre del usuario y el socket al hilo
 * @returns No retorna nada
*/
void sendMSgsToServer(void *args){
    char userMsg[2048];             //*Variable para el mensaje a enviar (la cantidad máxima es de 2048)
    char msgToServer[2048];         //*Mensaje final que se enviará al servidor

    //*Ciclo infinito para hacer el chat hasta que el usuario finalice el chat

    while(1){
        memset(userMsg, 0, 2048);           //*Limpiamos los mensajes
        memset(msgToServer, 0, 2048);       //*Limpiamos el mensaje al servidor
        green();
        printf("~ ");
        reset();
        fgets(userMsg, 2048, stdin);        //*Leemos el mensaje a enviar al chat
        userMsg[strlen(userMsg)-1]='\0';    //*Quitamos el salto de linea al final de la cadena


        //*Con sprintf damos formato al mensaje que se enviará y se mostrara en el servidor y los demas chats
        sprintf(msgToServer, "\033[1;32m[%s]: \033[0m%s\n", ((struct args*)args)->name, userMsg);

        //*Mandamos el mensaje al servidor
        send(((struct args*)args)->socket, msgToServer, strlen(msgToServer),0);

    }

    pthread_detach(pthread_self());     //*Quitamos el hilo
```

```
}

/**
 * @brief Hilo que se encarga de recibir las mensajes enviados del servidor
 * @param args argumentos del hilo, se pasa el socket
 * @returns No retorna nada
 */
void rcvMsgsFromServer(void *args){
    char msgRecv[2048];                 //*Mensaje recibido
    //*Ciclo infinito para recibir mensajes
    while(1){
        /*
            *Con la función recv recibimos un mensaje desde un socket en estado conectado,
            *es la misma función que hemos estado usando (recvfrom) solo que con el parámetro desde en NULL
            *El primer parametro es el socket, el segundo el mensaje a recibir, el tercero la longitud de este mensaje
            *y el ultimo son las banderas
        */
        if(recv(((struct args*)args)->socket, msgRecv, 2048, 0) == -1){ //*Si retorna -1 ocurrió un error
            red();
            printf("\nHa ocurrido un error o se ha alcanzado el limite de usarios\n");
            reset();
            exit(0);
            break;
        }else{
            printf("%s",msgRecv);       //*Imprimimos el mensaje
            green();
            printf("~ ");
            reset();
            fflush(stdout);

        }
        memset(msgRecv, 0, sizeof(msgRecv));    //*Limpiamos el buffer del mensaje recibido
    }

    pthread_detach(pthread_self());     //*Quitamos el hilo
}
```

Imagen 9. Programa del cliente.

Programa del servidor

A continuación, se presenta el código correspondiente al servidor, la cual se encuentra comentada línea por línea, de esta forma no será necesario explicar más.

```c
//*************************************************************************
        //*       LIBRERIAS INCLUIDAS       *//
//*************************************************************************

#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio_ext.h>
#include <pthread.h>
#define MAXUSERS 5

//*************************************************************************
        //*       Estructura para usuario       *//
//*************************************************************************
typedef struct user{
    char name[50];              //*Nombre del usuario máximo 50 carácteres
    int id;                     //*Id para identificar al usuario
    int socket;                 //*Socket que identifica al que se conecta el
    struct sockaddr_in addr;   //*Dirección del usuario
}User;
//*       Estrucutra para argumentos de hilos       *//
struct args{
    User *chatUser;
    int socket;
};


//*************************************************************************
        //*       Declaración de funciones       *//
//*************************************************************************
void sendMsgToClients(char *msg, int id);
void *clientActions(void *args);
```

```c
/*
    *Funciones para colores
*/
void red(){
    printf("\033[1;31m");
}
void reset(){
    printf("\033[0m");
}
void yellow(){
    printf("\033[1;33m");
}
void blue(){
    printf("\033[1;34m");
}
void green(){
    printf("\033[1;32m");
}
void white(){
    printf("\033[1;37m");
}
void black(){
    printf("\033[1;30m");
}
void purple(){
    printf("\033[1;35m");
}
void cyan(){
    printf("\033[1;36m");
}
//*************************************************************************
        //*       Variables globales       *//
//*************************************************************************
User *users[MAXUSERS];       //*Arreglo global que guardará los usuarios
int userID = 0;                 //*Variable para los ID
```

```c
int totalUsers = 0;                 //*Variable que guarda el total de usuarios conectados



int main(){
    int tcpSocket = 0;                  //*Variable para socket
    struct sockaddr_in client;          //*Estrucutra del client
    struct sockaddr_in server;          //*Estrucutra del servidor
    char userName[50];                  //*Variable para nombre de usuario
    int con = 0;                        //*Variable para la conexion al servidor
    pthread_t clientThread;               //*Variable para los hilos del cliente y servidor
    int bindValue;                      //*Variable para hacer el bind
    int listenValue;                    //*Variable para listen
    int acceptValue;                    //*Variable para accept

    printf("\t\t\t*****Servidor Siuuu Team chat!*****");
    printf("\n");
    printf("\n");

    tcpSocket = socket(AF_INET,SOCK_STREAM,0);

    if(tcpSocket == -1){
        perror("\nError al abrir el socket");
        exit(0);
    }else{
        server.sin_family = AF_INET;                    //*Siempre le asignamos AF_INET que es la familia de direcc
        server.sin_port = htons(8080);                  //*Es el puerto con los bytes en orden de red
        server.sin_addr.s_addr = inet_addr("10.0.2.15");    //*Dirección IP del host

        /*
            *Con bind enlazamos un nombre a un socket, como parametros esta el socket, la dirección
            *en este caso el servidor y el tamaño de la dirección
        */
        bindValue = bind(tcpSocket, (struct sockaddr *)&server,sizeof(server));

        if(bindValue == -1){
            perror("\nError al hacer bind :(");
```

```c
            exit(0);
        }else{
            /*
                *Listen espera conexiones en un socket.
                *El primer parametro es el socket y el segundo el backlog, este es el limite de la cola para
                *las conexiones, en este caso 5
            */
            listenValue = listen(tcpSocket,5);

            //*Si listen falla se regresa un error
            if(listenValue == -1){
                perror("\nError al escuchar la conexion :(");
                exit(0);
            }else{
                cyan();
                printf("\t\t\t*****Chat Room iniciado con exito*****");
                printf("\n");
                printf("\n");
                while(1){

                    /*
                        *La función accept acepta una conexion de parte del cliente y se mantiene en estado de espera
                        *para que un cliente se conecte, al conectarse un cliente con connect() la funcion accept
                        *devuelve un nuevo identificador de socket
                    */
                    socklen_t clis= sizeof(client);
                    acceptValue = accept(tcpSocket, (struct sockaddr *)&client, &clis);
                    if(totalUsers == MAXUSERS){
                        red();
                        printf("Se ha alzanzado el maximo de usuarios\n");
                        close(acceptValue); //*Cerramos la conexion del usuario actual
                        reset();
                    }else{
                        //*Creamos el usuario con sus datos
                        User *chatUser = (User*)malloc(sizeof(User));       //*Establecemos memoria para el usuario actu
                        chatUser->id = userID++;                            //*Le asignamos un id
                        chatUser->socket = acceptValue;                     //*Le asignamos un identificador de socket
```

```
                    chatUser->addr = client;                            //*Le asignamos su dirección
                    int i = 0;
                    //*Agregamos el usuario creado al arreglo de usuarios en el chat
                    for(i = 0;i<MAXUSERS;i++)
                        if(users[i] == NULL){   //*Si no hay ningun usuario lo guardamos
                            users[i] = chatUser;
                            break;              //*Salimos del for
                        }

                    //*Establecemos los argumentos que tendrán los hilos
                    struct args *thread_args = (struct args *)malloc(sizeof(struct args));
                    thread_args -> chatUser = chatUser;
                    thread_args -> socket = tcpSocket;
                    /*
                        *Creamos un nuevo hilo para el cliente
                        *Para más detalles de hilos revisar el manual man pthreads
                        *Si no devuelve 0 es que hubo error al crear el hilo
                    */
                    if(pthread_create(&clientThread, NULL, (void *)clientActions, (void *)thread_args) != 0){
                        perror("\nError al crear hilo");
                        exit(0);
                    }

                }

                reset();
            }
            reset();
        }
    }
    reset();
    close(tcpSocket);
    return 0;
}
```

```
/**
 * @brief Hilo que gestiona las acciones del cliente
 * @param args Argumentos del hilo, pasamos el cliente actual
 * @returns NULL para indicar que termina
 */
void *clientActions(void *args){
    totalUsers++;               //*Incrementamos en uno la cantidad de usuarios conectados
    char name[50];              //*Nombre del usuario
    char msgToClients[2048];   //*Mensaje desde el cliente
    User* user = ((struct args*)args)-> chatUser;
    int recvValue;              //*Variable para el valor de recv
    int i;
    int f = 0;
    memset(msgToClients, 0, 2048); //resetea el buffer de mensajes

    if(recv( user->socket, name, 50, 0) == -1 ){
        perror("\nError al recibir datos");
        exit(0);
    }else{
        strcpy(user->name, name);
        cyan();
        sprintf(msgToClients, "\033[1;36mSe ha conectado %s al chat!, usuarios conectados: %d\n", name,totalUsers);
        printf("%s",msgToClients);
        sendMsgToClients(msgToClients, user->id);
    }

    while(1){
        memset(msgToClients, 0, 2048); //resetea el buffer de mensajes

        recvValue =recv( user->socket, msgToClients, 2048, 0);
        if(recvValue == -1 ){
            perror("\nError al recibir datos");
            break;
        }else if(recvValue > 0){
```

```
            sendMsgToClients(msgToClients, user->id);
            printf("%s",msgToClients);
        }else{
            red();
            totalUsers--;
            sprintf(msgToClients, "\033[1;31mSe desconecto %s, usuarios conectados: %d\n",name, totalUsers);
            printf("%s",msgToClients);
            sendMsgToClients(msgToClients, user->id);
            reset();
            break;
        }
    }


    //*Cuando se desconecta el usuario cerramos su socket y lo quitamos del arreglo
    close(user->id);
    for(i = 0;i<MAXUSERS;i++)
        if(users[i] != NULL){   //*Si no hay ningun usuario lo guardamos
            users[i] = NULL;
            break;              //*Salimos del for
        }
    free(user);     //*LIberamos memoria del usuario
    pthread_detach(pthread_self());     //*Quitamos el hilo
    reset();
    return NULL;
}
```

```
/**
 * @brief FUnción que se encarga de mandar el mensaje a los clientes
 * @param msg Mensaje que se enviará a los clientes
 * @param id Id del usuario que manda el mensaje
 * @returns No retorna nada
 */
void sendMsgToClients(char *msg, int id){
    int i;
    for(i = 0;i<MAXUSERS;i++)
        if(users[i] != NULL)
            if(users[i]->id != id)
                if(send(users[i]->socket,msg, strlen(msg), 0) == -1){
                    perror("\nError al enviar :(");
                    exit(0);
                }

}
```

Pruebas de funcionamiento

Primeramente, tenemos que correr el servidor.

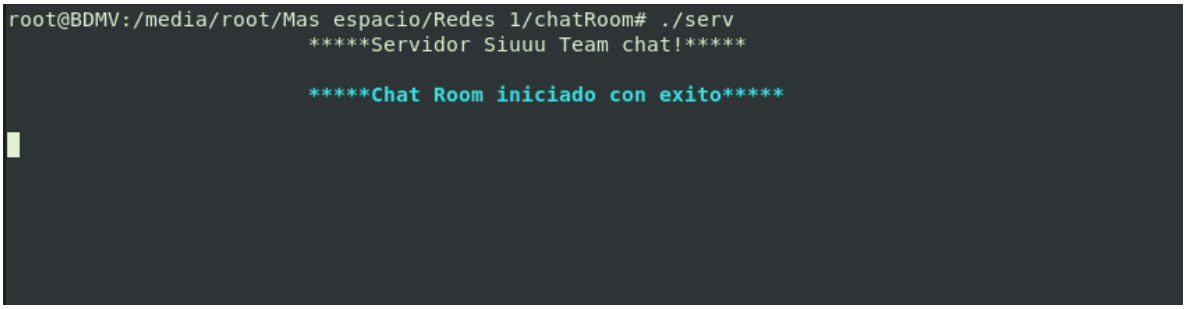

Imagen 10. Corriendo el servidor.

Posteriormente ya podemos ejecutar el cliente un máximo de 5 veces, pues son los usuarios permitidos.

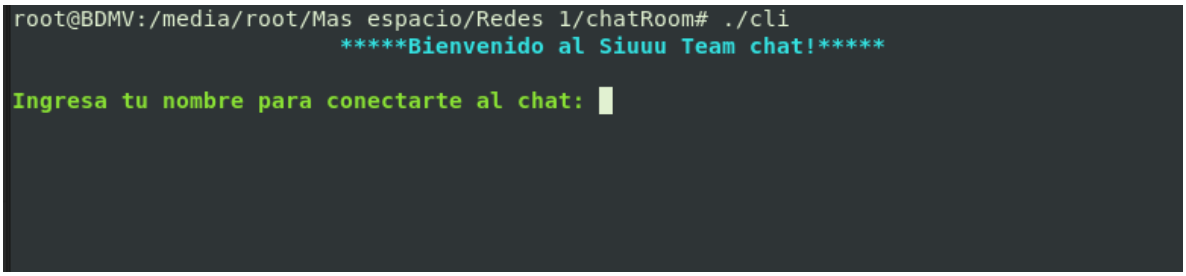

Imagen 11. Corriendo el cliente.

Como podemos ver el cliente nos pide un nombre de usuario, al ingresarlo y si estamos dentro del rango de usuarios permitidos ya podremos comenzar a chatear.

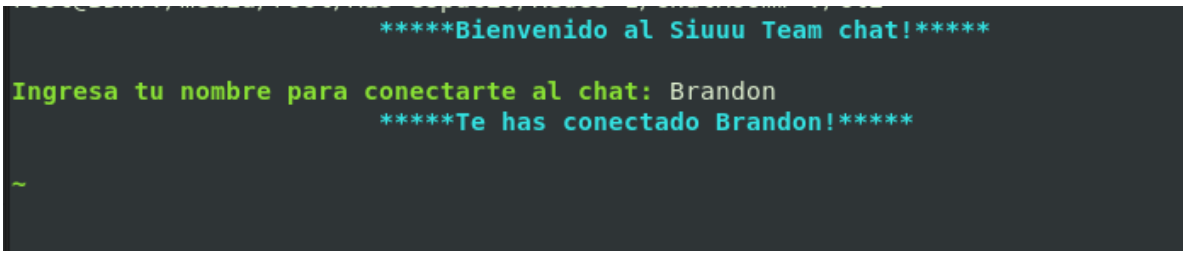

Imagen 12. Conectando al chat.

Podemos ver que el servidor va mostrando las interacciones que los usuarios hacen.

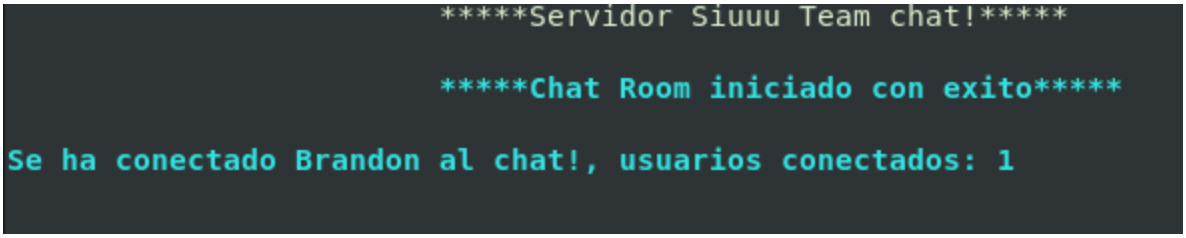

Imagen 13. Servidor.

Cuando otro usuario se conecta se le avisa a los demás usuarios, de igual forma el servidor registra estas acciones como se puede ver en las siguientes imágenes.

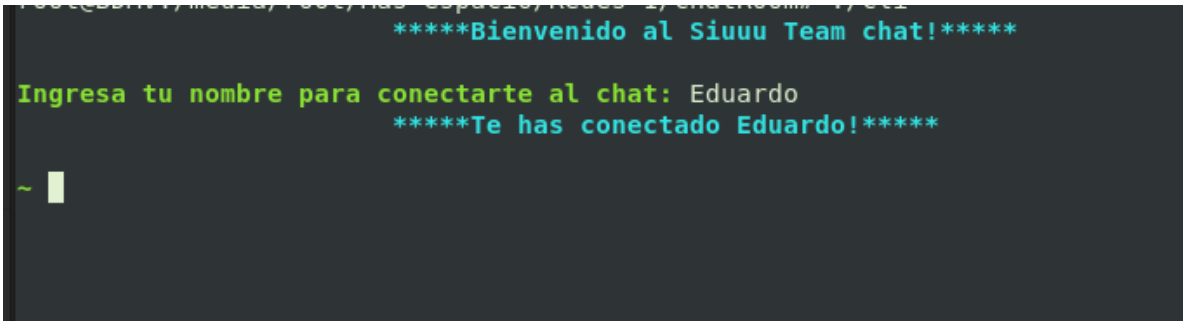

Imagen 14. Conexión de nuevo usuario.

Imagen 15. Aviso a otros usuarios.

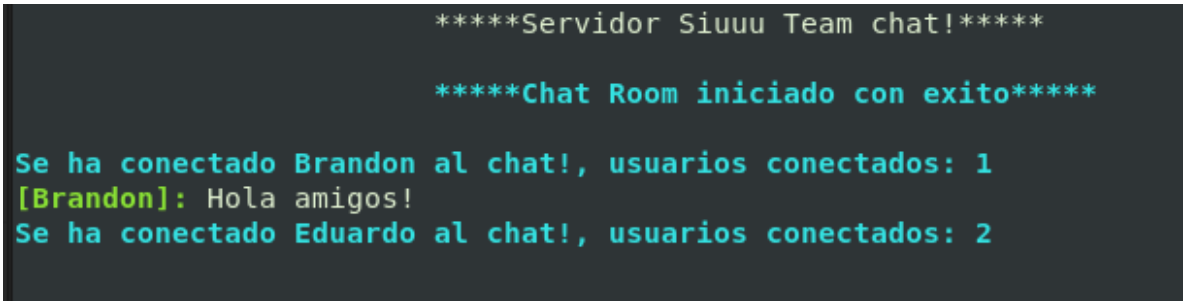

Imagen 16. Servidor y sus acciones.

De esta forma podemos empezar a chatear.

Imagen 17. Vista del usuario Brandon.

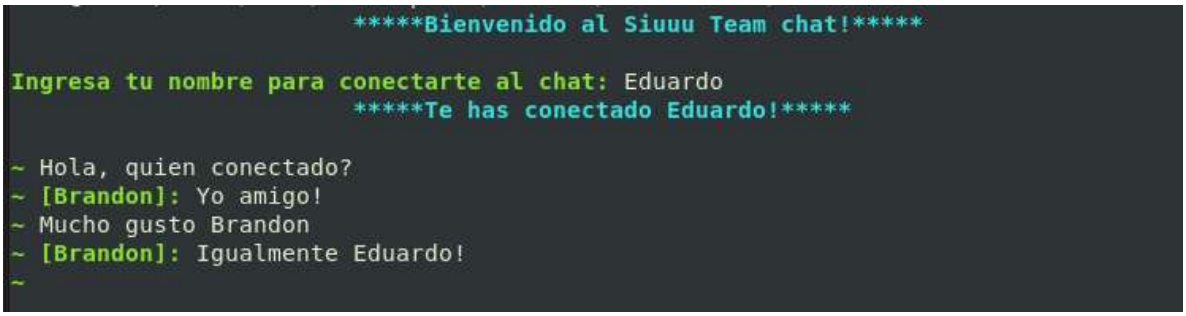

Imagen 18. Vista del usuario Eduardo.

Cuando llegamos al limite soportado de usuarios no se podrá conectar ese usuario y el servidor arrojará un mensaje como podemos ver en las siguientes capturas.

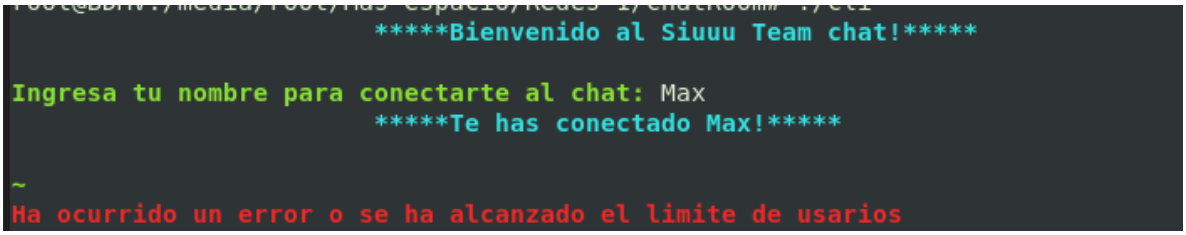

Imagen 19. Error en el usuario al alcanzar usuarios máximos.

Imagen 20. Mensaje del servidor al alcanzar los usuarios máximos.

Finalmente, se mostrará a los usuarios y en el servidor cuando un usuario se salga del chat o finalice su hilo de ejecución, es decir, cierre el programa.

```
                    *****Bienvenido al Siuuu Team chat!*****

Ingresa tu nombre para conectarte al chat: Brandon
                    *****Te has conectado Brandon!*****

~ Hola amigos!
~ Se ha conectado Eduardo al chat!, usuarios conectados: 2
~ [Eduardo]: Hola, quien conectado?
~ Yo amigo!
~ [Eduardo]: Mucho gusto Brandon
~ Igualmente Eduardo!
~ Se ha conectado Fabian al chat!, usuarios conectados: 3
~ [Fabian]: hola
~ Se ha conectado Lizeth al chat!, usuarios conectados: 4
~ [Lizeth]: Hola
~ Se ha conectado Jonathan al chat!, usuarios conectados: 5
~ [Jonathan]: Hola
~ Adios
~ ^C
root@BDMV:/media/root/Mas espacio/Redes 1/chatRoom#
```

Imagen 21. Salida del usuario Brandon.

```
                    *****Bienvenido al Siuuu Team chat!*****

Ingresa tu nombre para conectarte al chat: Eduardo
                    *****Te has conectado Eduardo!*****

~ Hola, quien conectado?
~ [Brandon]: Yo amigo!
~ Mucho gusto Brandon
~ [Brandon]: Igualmente Eduardo!
~ Se ha conectado Fabian al chat!, usuarios conectados: 3
~ [Fabian]: hola
~ Se ha conectado Lizeth al chat!, usuarios conectados: 4
~ [Lizeth]: Hola
~ Se ha conectado Jonathan al chat!, usuarios conectados: 5
~ [Jonathan]: Hola
~ [Brandon]: Adios
~ Se desconecto Brandon, usuarios conectados: 4
~
```

Imagen 22. Aviso de desconexión de un usuario.

```
root@BDMV:/media/root/Mas espacio/Redes 1/chatRoom# ./serv
                        *****Servidor Siuuu Team chat!*****

                        *****Chat Room iniciado con exito*****

Se ha conectado Brandon al chat!, usuarios conectados: 1
[Brandon]: Hola amigos!
Se ha conectado Eduardo al chat!, usuarios conectados: 2
[Eduardo]: Hola, quien conectado?
[Brandon]: Yo amigo!
[Eduardo]: Mucho gusto Brandon
[Brandon]: Igualmente Eduardo!
Se ha conectado Fabian al chat!, usuarios conectados: 3
[Fabian]: hola
Se ha conectado Lizeth al chat!, usuarios conectados: 4
[Lizeth]: Hola
Se ha conectado Jonathan al chat!, usuarios conectados: 5
[Jonathan]: Hola
Se ha alzanzado el maximo de usuarios
[Brandon]: Adios
Se desconecto Brandon, usuarios conectados: 4
```

Imagen 23. Mensajes del servidor.

De esta forma, si se desea, se pueden unir más usuarios.

```
                        *****Bienvenido al Siuuu Team chat!*****

Ingresa tu nombre para conectarte al chat: Brandon2
                        *****Te has conectado Brandon2!*****

~ HOla de nuevo
~
```

Imagen 24. Conexión de nuevo usuario.

```
                        *****Bienvenido al Siuuu Team chat!*****

Ingresa tu nombre para conectarte al chat: Eduardo
                        *****Te has conectado Eduardo!*****

~ Hola, quien conectado?
~ [Brandon]: Yo amigo!
~ Mucho gusto Brandon
~ [Brandon]: Igualmente Eduardo!
~ Se ha conectado Fabian al chat!, usuarios conectados: 3
~ [Fabian]: hola
~ Se ha conectado Lizeth al chat!, usuarios conectados: 4
~ [Lizeth]: Hola
~ Se ha conectado Jonathan al chat!, usuarios conectados: 5
~ [Jonathan]: Hola
~ [Brandon]: Adios
~ Se desconecto Brandon, usuarios conectados: 4
~ Se ha conectado Brandon2 al chat!, usuarios conectados: 5
~ [Brandon2]: HOla de nuevo
~
```

Imagen 25. Aviso a otros usuarios.

```
*****Servidor Siuuu Team Chat!*****

*****Chat Room iniciado con exito*****

Se ha conectado Brandon al chat!, usuarios conectados: 1
[Brandon]: Hola amigos!
Se ha conectado Eduardo al chat!, usuarios conectados: 2
[Eduardo]: Hola, quien conectado?
[Brandon]: Yo amigo!
[Eduardo]: Mucho gusto Brandon
[Brandon]: Igualmente Eduardo!
Se ha conectado Fabian al chat!, usuarios conectados: 3
[Fabian]: hola
Se ha conectado Lizeth al chat!, usuarios conectados: 4
[Lizeth]: Hola
Se ha conectado Jonathan al chat!, usuarios conectados: 5
[Jonathan]: Hola
Se ha alzanzado el maximo de usuarios
[Brandon]: Adios
Se desconecto Brandon, usuarios conectados: 4
Se ha conectado Brandon2 al chat!, usuarios conectados: 5
[Brandon2]: HOla de nuevo
```

Imagen 26. mensajes del servidor

Instrucciones de ejecución

Es importante ejecutar primeramente el servidor, en la siguiente imagen se muestra cómo hacerlo:

```
gcc servidor.c -o serv -lpthread
./serv
```

Imagen 27. Compilación y ejecución del servidor.

Siendo la primera línea la compilación y la segunda la ejecución.

Una vez compilado y ejecutado el servidor podemos compilar y ejecutar el cliente de la siguiente forma:

```
gcc cliente.c -o cli -lpthread
./cli
```

Imagen 28. Compilación y ejecución del cliente.

Siendo la primera línea la compilación y la segunda la ejecución.

Es importante que no se finalice el servidor antes de finalizar los procesos del cliente, pues podría ocurrir un error en los clientes al cerrarse la conexión con el servidor.

Conclusiones

Fischer Salazar César Eduardo

Durante el transcurso del curso se nos ha proporcionado información que nos servirían de cimientos que culminarían en la realización de este proyecto, dado que en prácticas anteriores hemos estado realizando la implementación de un socket crudo mediante su programación en lenguaje C así como el uso de manuales que nos brinda el sistema operativo de LINUX en cual se ha utilizado para observar el funcionamiento de estos.

El empleo practico de un socket y sus diferentes configuraciones me permitido observar las diferentes aplicaciones que estos pueden tener en ámbitos relacionados a las redes de computadora, pues la teoría puede llegar a abrumar un poco ya que es necesario tener identificados los manuales requeridos, por lo ya mencionado fue necesario investigar en fuentes secundarias sobre manuales dentro de LINUX que nos permitieran abrir un socket para establecer la conexión para llegar a un funcionamiento correcto.

Puedo concluir nuevamente que la realización de estos programas me ha parecido bastante interesante ya que estos me han permitido entender el cómo la implementación de un socket nos puede permitir obtener información sobre nuestra computadora, el capturar una trama y hasta el cómo poder tener una conexión bidireccional en vivo.

López García José Eduardo

A lo largo del desarrollo de las prácticas relacionadas con esta materia, hemos aterrizado en los entendimientos claros acerca del manual de Linux, el uso de los sockets dentro de la programación en este sistema operativo, y diferentes comandos que tienen relación con estos elementos; con la teoría quedó bien entendido, y con las prácticas se ha visualizado su funcionamiento.

El empleo de los sockets, no solamente para el proyecto sino también para otros sistemas existentes relacionados con las redes, tiene una vital importancia dentro de esta rama, ya que ha sido posible comprender el funcionamiento de un chat en vivo, ya sea unidireccional o bidireccional. Claro, fue necesario buscar más allá de los recursos (las anteriores prácticas) con los que contábamos, para que su desarrollo fuera óptimo y pudiera tener un funcionamiento adecuado, dado que el empleo del lenguaje C para este tipo de sistemas es muy robusto y extenso, era un poco predecible que hubiera pequeños problemas dentro del desarrollo, pero se lograron solucionar.

En general, fue bastante satisfactorio poder realizar una sala de chat como este, así se pudo recalcar y comprobar para qué pueden ser empleados los sockets en la programación en C y con la asistencia del manual de Linux, y así poder pensar en las diferentes posibilidades que esto puede generar en su área; dentro de lo que cabe, el haber desarrollado el proyecto nos ha dado una apertura bastante amplia acerca del funcionamiento de un sistema de chats en vivo, como lo hemos estado utilizando en años recientes.

Meza Vargas Brandon David

Los sockets, como bien hemos revisado a lo largo del semestre mediante las prácticas realizadas son muy importantes en las redes, con estos podemos crear desde un simple chat unidireccional o uno bidireccional como el realizado en este proyecto final. En el desarrollo del presente proyecto fue necesario investigar un poco más sobre los sockets y

su uso en lenguaje c, lenguaje bajo el cual fue desarrollado este proyecto, llegando así a la conclusión que los sockets utilizados para este proyecto fue la mejor decisión ya que estos permitieron de forma correcta el chat bidireccional entre clientes conectados a un servidor el cual gestionaba las operaciones que los usuarios conectados a él hacían.

Es increíble lo que podemos hacer con los sockets con lenguaje c, es cierto que hay lenguajes que facilitan mucho el uso de sockets y no es complicado hacer lo realizado en este proyecto, pero hacerlo en lenguaje c trabajando con sockets como se hace en este lenguaje nos permitió obtener mucho conocimiento y experiencia en la programación de estos, siempre es importante saber como funcionan las cosas desde cero.

Es destacable mencionar que durante el desarrollo del proyecto se presentaron varias problemáticas, las cuales fueron en su mayoría resueltas con manuales proporcionados por Linux, tales como socket, 7 ip, y los manuales de las funciones usadas en el proyecto.

Un proyecto donde se vio reflejado lo visto a lo largo de las prácticas aterrizándolo a un entorno práctica y entretenido como lo fue este chat room.

Referencias

- Speed Check. (Sin fecha). *Socket*. https://www.speedcheck.org/es/wiki/socket/
- Paszniuk, R. (2013). *Sockets en C (Parte I) – Linux*. https://www.programacion.com.py/noticias/sockets-en-c-parte-i-linux