

**Proyecto de sistemas operativos**

**Chatroom**

**Reynaldo Vega A01114523**

**Martes 27 de octubre de 2020**

**Scripts**

**client.c**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<string.h>

#include<signal.h>

#include<pthread.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#define LENGTH 2000

// Variable globales

int sockfd = 0;

char nombre[32];

volatile sig\_atomic\_t bandera = 0;

void iniciarChat() {

    printf("%s", "> ");

    fflush(stdout);

}

void almacenarMensajeONombre (char\* array, int length) {

    for (int i = 0; i < length; i++) {

        if (array[i] == '\n') {

            array[i] = '\0';

            break;

        }

    }

}

void catchCtrlCyExit(int signal) {

    bandera = 1;

    printf("\n");

}

void manejadorDeEnvioMsjs() {

    char mensaje[LENGTH] = {};

    char buffer[LENGTH + 32] = {};

    while(1) {

        iniciarChat();

        fgets(mensaje, LENGTH, stdin);

        almacenarMensajeONombre(mensaje, LENGTH);

        if(strlen(mensaje) > 250) {

            printf("> Ha excedido el limite de 250 caracteres\n");

        } else if(strcmp(mensaje, "exit") == 0 || strcmp(mensaje, "bye") == 0) {

            break;

        } else {

            sprintf(buffer, "%s: %s\n", nombre, mensaje);

            send(sockfd, buffer, strlen(buffer), 0);

        }

        bzero(mensaje, LENGTH);

        bzero(buffer, LENGTH + 32);

    }

    catchCtrlCyExit(2);

}

void manejadorReciboMsjs() {

    char mensaje[LENGTH] = {};

    while (1) {

        int receive = recv(sockfd, mensaje, LENGTH, 0);

        if (receive > 0) {

            printf("%s", mensaje);

            iniciarChat();

        } else if (receive == 0) {

            break;

        }

        memset(mensaje, 0, sizeof(mensaje));

    }

}

int main(int argc, char \*\*argv){

    char \*ip = argv[1];

    int port = atoi(argv[2]);

    signal(SIGINT, catchCtrlCyExit);

    strcat(argv[3],"\n");

    strcpy(nombre,argv[3]);

    almacenarMensajeONombre(nombre, strlen(nombre));

    if (strlen(nombre) > 32 || strlen(nombre) < 2){

        printf("ERROR: ingresar nombre de 2 a 32 caracteres\n");

        return EXIT\_FAILURE;

    }

    struct sockaddr\_in server\_addr;

    // Configuracion de sockets

    sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_addr.s\_addr = inet\_addr(ip);

    server\_addr.sin\_port = htons(port);

    // Establecer conexion con el servidor

    int err = connect(sockfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

    if (err == -1) {

        printf("Error al conectar\n");

        return EXIT\_FAILURE;

    }

    // Enviar nombre

    send(sockfd, nombre, 32, 0);

    printf("Bienvenido %s\n", nombre);

    pthread\_t hiloEnviarMsjs;

    if(pthread\_create(&hiloEnviarMsjs, NULL, (void \*) manejadorDeEnvioMsjs, NULL) != 0) {

        printf("ERROR\n");

        return EXIT\_FAILURE;

    }

    pthread\_t hiloRecibirMsjs;

    if(pthread\_create(&hiloRecibirMsjs, NULL, (void \*) manejadorReciboMsjs, NULL) != 0) {

        printf("ERROR\n");

        return EXIT\_FAILURE;

    }

    while (1) {

        if(bandera) {

            break;

        }

    }

    close(sockfd);

    return EXIT\_SUCCESS;

}

**server.c**

#include<string.h>

#include<sys/socket.h>

#include<stdio.h>

#include<sys/types.h>

#include<signal.h>

#include<stdlib.h>

#include<unistd.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<errno.h>

#include<pthread.h>

#define TAMANO\_BUFFER 2048

#define CLIENTES\_TOPE 5

static int uid = 10;

static \_Atomic unsigned int contadorClientes = 0;

// Estructura del cliente

typedef struct{

    struct sockaddr\_in direccion;

    int sockfd;

    int uid;

    char nombre[32];

} clientsStructure;

clientsStructure \*clientes[CLIENTES\_TOPE];

pthread\_mutex\_t clientes\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

void enviarMensajeServer(char \*mensaje) {

    pthread\_mutex\_lock(&clientes\_mutex);

    for(int i = 0; i < CLIENTES\_TOPE; i++) {

        if(clientes[i]) {

            write(clientes[i]->sockfd, mensaje, strlen(mensaje));

        }

    }

    pthread\_mutex\_unlock(&clientes\_mutex);

}

void catchCtrlC(int signal) {

    char \*mensajeByeGlobal = "Bye desde el servidor\n";

    enviarMensajeServer(mensajeByeGlobal);

    printf("\n");

    exit(EXIT\_SUCCESS);

}

void almacenarMsjONombre(char\* array, int length) {

    for (int i = 0; i < length; i++) {

        if (array[i] == '\n') {

            array[i] = '\0';

            break;

        }

    }

}

// Añadir clientes a la cola

void anadirCola(clientsStructure \*cl){

    pthread\_mutex\_lock(&clientes\_mutex);

    for(int i = 0; i < CLIENTES\_TOPE; i++){

        if(!clientes[i]){

            clientes[i] = cl;

            break;

        }

    }

    pthread\_mutex\_unlock(&clientes\_mutex);

}

// Quitar clientes de la cola

void removerCola(int uid){

    pthread\_mutex\_lock(&clientes\_mutex);

    for(int i = 0; i < CLIENTES\_TOPE; i++){

        if(clientes[i]){

            if(clientes[i]->uid == uid){

                clientes[i] = NULL;

                break;

            }

        }

    }

    pthread\_mutex\_unlock(&clientes\_mutex);

}

// Enviar mensaje a todos menos al que lo envió

void enviarMensaje(char \*mensaje, int uid){

    pthread\_mutex\_lock(&clientes\_mutex);

    for(int i = 0; i < CLIENTES\_TOPE; i++){

        if(clientes[i]){

            if(clientes[i]->uid != uid){

                if(write(clientes[i]->sockfd, mensaje, strlen(mensaje)) < 0){

                    perror("ERROR");

                    break;

                }

            }

        }

    }

    pthread\_mutex\_unlock(&clientes\_mutex);

}

// Manejar las comunicaciones del cliente

void \*manejadorCliente(void \*arg){

    char buff\_out[TAMANO\_BUFFER];

    char nombre[32];

    int banderaSalida = 0;

    contadorClientes++;

    clientsStructure \*cli = (clientsStructure \*)arg;

    // Manejar nombre

    if(recv(cli->sockfd, nombre, 32, 0) <= 0 || strlen(nombre) <  2 || strlen(nombre) >= 32-1){

        printf("No se ingreso nombre o excedio el limite.\n");

        banderaSalida = 1;

    } else{

        strcpy(cli->nombre, nombre);

        printf("> ");

        sprintf(buff\_out, "%s conectado\n", cli->nombre);

        printf("%s", buff\_out);

        enviarMensaje(buff\_out, cli->uid);

    }

    bzero(buff\_out, TAMANO\_BUFFER);

    while(1){

        if (banderaSalida) {

            break;

        }

        int receive = recv(cli->sockfd, buff\_out, TAMANO\_BUFFER, 0);

        if (receive > 0){

            if(strlen(buff\_out) > 0){

                enviarMensaje(buff\_out, cli->uid);

                almacenarMsjONombre(buff\_out, strlen(buff\_out));

                printf("> ");

                printf("%s\n", buff\_out, cli->nombre);

            }

        } else if (receive == 0 || strcmp(buff\_out, "bye") == 0 || strcmp(buff\_out, "exit") == 0){

            printf("> ");

            sprintf(buff\_out, "%s desconectado\n", cli->nombre);

            printf("%s", buff\_out);

            enviarMensaje(buff\_out, cli->uid);

            banderaSalida = 1;

        } else {

            banderaSalida = 1;

        }

        bzero(buff\_out, TAMANO\_BUFFER);

    }

    // Borrar cliente de la cola y mejorar el rendimiento del hilo

    close(cli->sockfd);

    removerCola(cli->uid);

    free(cli);

    contadorClientes--;

    pthread\_detach(pthread\_self());

    return NULL;

}

int main(int argc, char \*\*argv){

    char \*ip = "127.0.0.1";

    int port = atoi(argv[1]);

    int option = 1;

    int listenfd = 0, connfd = 0;

    struct sockaddr\_in serv\_addr;

    struct sockaddr\_in cli\_addr;

    pthread\_t tid;

    // Configuración del socket

    listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

    serv\_addr.sin\_family = AF\_INET;

    serv\_addr.sin\_addr.s\_addr = inet\_addr(ip);

    serv\_addr.sin\_port = htons(port);

    // Catch Ctr-C

    signal(SIGINT, catchCtrlC);

    if(setsockopt(listenfd, SOL\_SOCKET,(SO\_REUSEPORT | SO\_REUSEADDR),(char\*)&option,sizeof(option)) < 0){

        perror("ERROR");

        return EXIT\_FAILURE;

    }

    // Verificar que se puede hacer bind

    if(bind(listenfd, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) {

        perror("ERROR");

        return EXIT\_FAILURE;

    }

    // Manejar listen en caso de fallo

    if (listen(listenfd, 10) < 0) {

        perror("ERROR");

        return EXIT\_FAILURE;

    }

    printf("Servidor inicializado\n");

    while(1){

        socklen\_t clilen = sizeof(cli\_addr);

        connfd = accept(listenfd, (struct sockaddr\*)&cli\_addr, &clilen);

        // Checar si el numero de clientes se alcanzó

        if((contadorClientes + 1) == CLIENTES\_TOPE){

            printf("Tope de clientes alcanzado");

            printf(":%d\n", cli\_addr.sin\_port);

            close(connfd);

            continue;

        }

        // Configuración del cliente

        clientsStructure \*cli = (clientsStructure \*)malloc(sizeof(clientsStructure));

        cli->direccion = cli\_addr;

        cli->sockfd = connfd;

        cli->uid = uid++;

        // Añadir cliente a la cola y hacer fork del hilo

        anadirCola(cli);

        pthread\_create(&tid, NULL, &manejadorCliente, (void\*)cli);

        // Reducir el uso del CPU

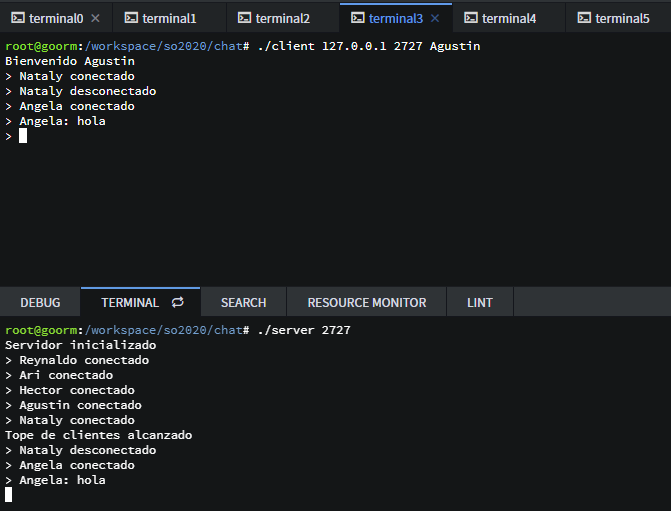
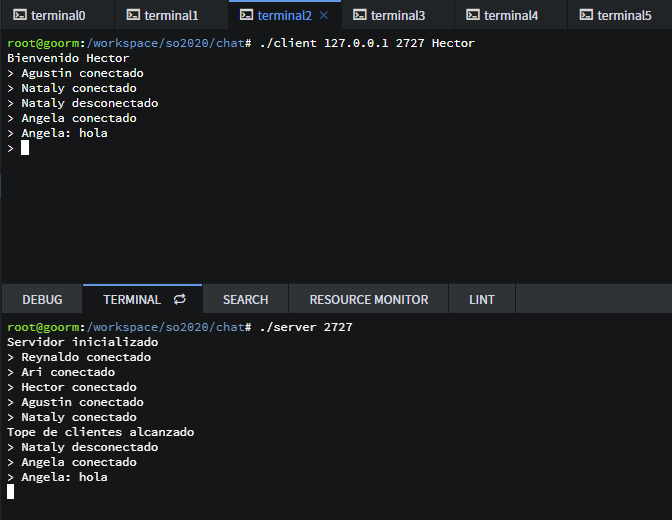
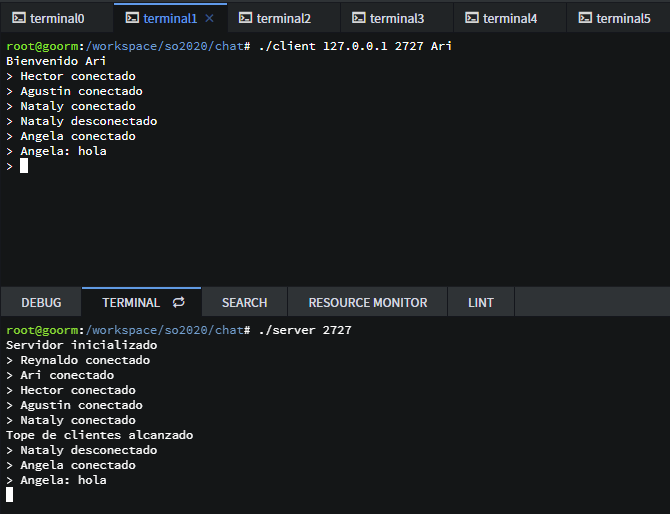
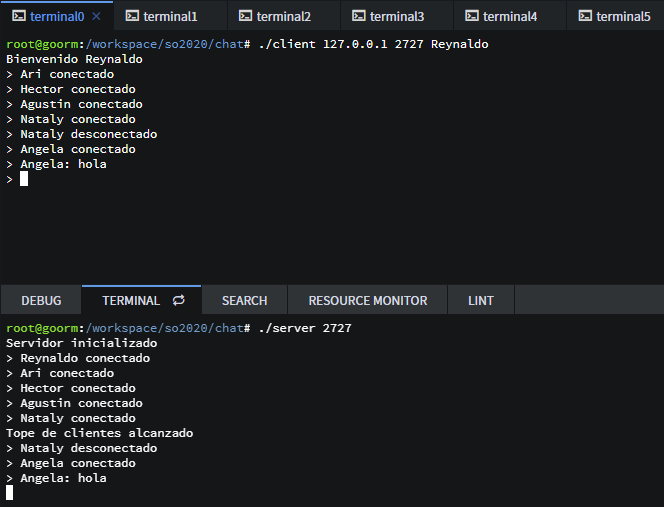
        sleep(1);

    }

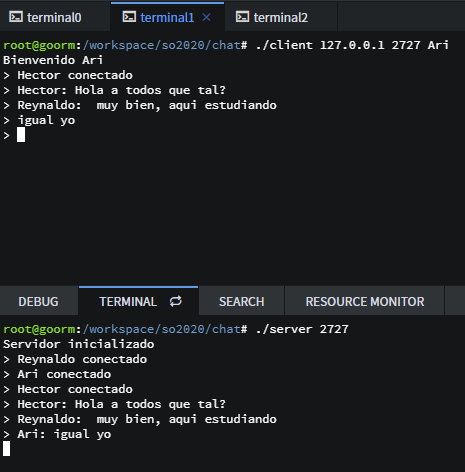
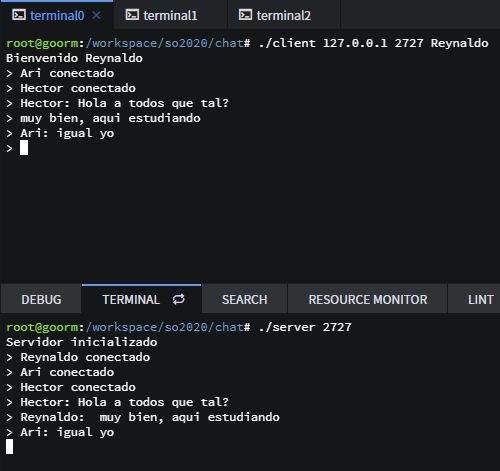
    return EXIT\_SUCCESS;

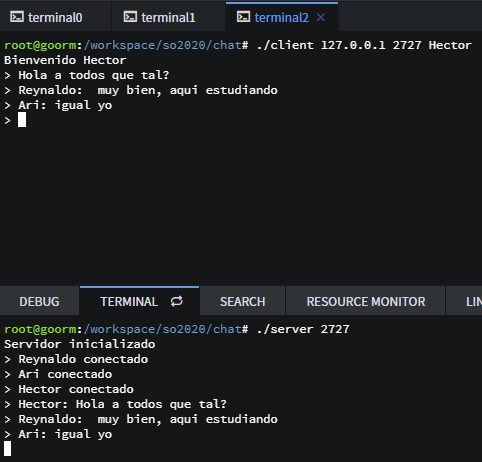
}

**Se muestra que los clientes se conectan y se desconectan**



**Clientes pueden mandar y recibir mensajes y saber quién los envió. Al igual que el servidor recibe los mensajes y los muestra**

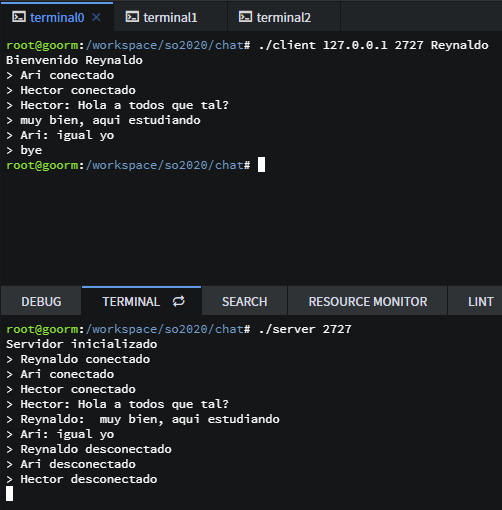


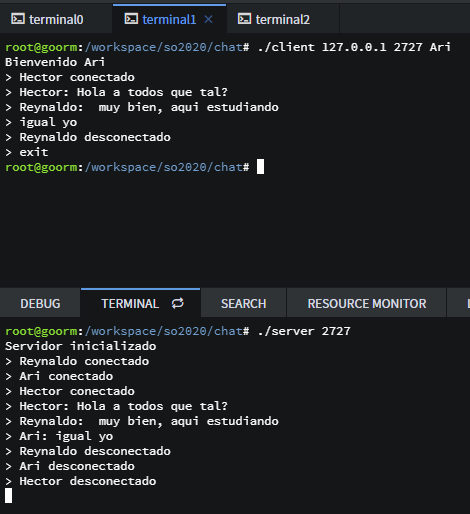


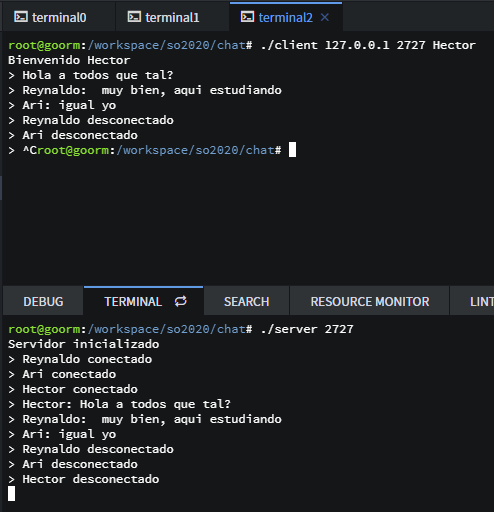
**Si el cliente excede el límite de 250 caracteres en un mensaje se manda mensaje de que se excedió y no se le manda nada a los demás clientes**



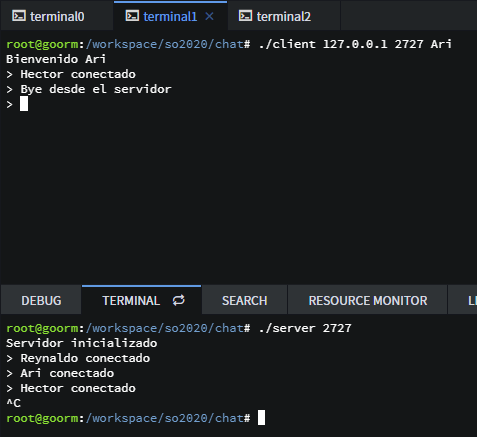
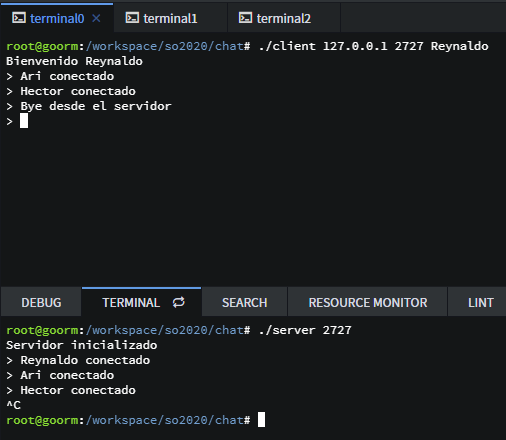


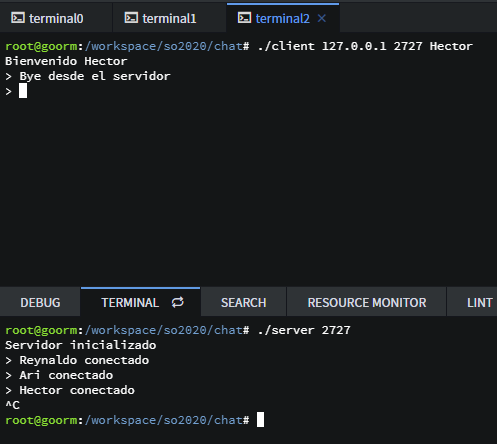
**Cliente al escribir bye o exit o presionar Ctrl-C puede salir de la conversación**



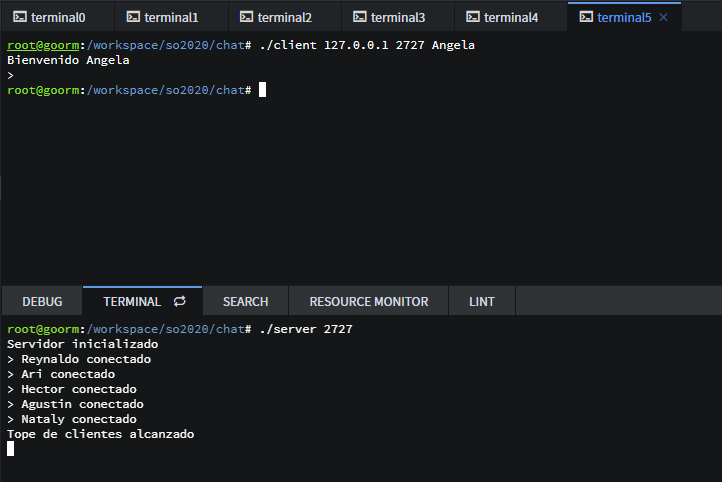


**Si el servidor se cierra con Ctrl-C se manda “bye desde el servidor” a todos los clientes**





**Servidor solo acepta máximo 5 clientes**



**Si se sale alguien puede entrar otra persona**