//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// File: servidor.c

// Author: Javier Vivas Gomez

// Date: Abril de 2020

// Coms: Ejemplo de servidor mediante sockets

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <pthread.h>

#include <sys/types.h>

#include <signal.h>

//Definimos 3 clientes como maximo

#define MAX\_CLIENTS 3

//Definimos 1024 bytes como maximo

#define BUFFER\_SZ 1024

static \_Atomic unsigned int cli\_count = 0;

static int uid = 10;

/\* Estructura de direcciones propia del servidor \*/

typedef struct{

//Direccion

struct sockaddr\_in address;

//Descriptor del socket

int sockfd;

//ID del struct

int uid;

//Puerto del cliente

int puerto;

} client\_est;

//Declaracion de vector de estructuras.

client\_est \*clients[MAX\_CLIENTS];

//Inicializa el mutex, para garantizar la exclusion mutua durante el proceso de envio/recepcion

pthread\_mutex\_t mutex\_cl = PTHREAD\_MUTEX\_INITIALIZER;

/\* Funcion que detecta el "enter" en la cadena y le pone fin a la cadena \*/

void fin\_cadena (char\* cadena, int length) {

int i;

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

if (cadena[i] == '\n') {

cadena[i] = '\0';

break;

}

}

}

/\* Eliminar clientes de la cola de clientes\*/

void eliminarCola(int uid){

//Le damos el control del mutex, garantizamos exclusion mutua

pthread\_mutex\_lock(&mutex\_cl);

for(int i=0; i < MAX\_CLIENTS; ++i){

if(clients[i]){

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

clients[i] = NULL;

//Nos evita seguir recorreriendo inserviblemente el vector

break;

}

}

}

//Le quitamos el control del mutex, liberamos la exclusion mutua

pthread\_mutex\_unlock(&mutex\_cl);

}

/\* Añadimos clientes a la cola de clientes\*/

void AnyadirCola(client\_est \*cl){

//Le damos el control del mutex, garantizamos exclusion mutua

pthread\_mutex\_lock(&mutex\_cl);

for(int i=0; i < MAX\_CLIENTS; ++i){

if(!clients[i]){

clients[i] = cl;

//Nos evita seguir recorreriendo inserviblemente el vector

break;

}

}

//Le quitamos el control del mutex, liberamos la exclusion mutua

pthread\_mutex\_unlock(&mutex\_cl);

}

/\* Envio de mensajes a todos los clientes \*/

void msg\_env(char \*s, int uid){

pthread\_mutex\_lock(&mutex\_cl);

//

for(int i=0; i<MAX\_CLIENTS; ++i){

if(clients[i]){

if(write(clients[i]->sockfd, s, strlen(s)) < 0){

perror("ERROR: write to descriptor failed");

break;

}

}

}

pthread\_mutex\_unlock(&mutex\_cl);

}

/\* Handle all communication with the client \*/

void \*gestionaCliente(void \*arg){

//Declaracion del buffer de almacenamiento

char buffer\_salida[BUFFER\_SZ];

int flag\_salida = 0;

//aumentamos el contador de clientes atendiendose.

cli\_count++;

client\_est \*cli = (client\_est \*)arg;

int puertoAbierto = cli->puerto;

char cadena[BUFFER\_SZ];

sprintf(cadena, "%d", puertoAbierto);

fin\_cadena(cadena, strlen(cadena));

while(1){

//Interrumpirá bucle while cuando sea flag\_salida=1

if (flag\_salida) {

break;

}

//Recibe mensaje del cliente y lo almacena en la variable buffer

int recibido = recv(cli->sockfd, buffer\_salida, BUFFER\_SZ, 0);

if (recibido > 0){

if(strlen(buffer\_salida) > 0){

sprintf(cadena, "%d", puertoAbierto);

strcat(cadena, " ");

strcat(cadena,buffer\_salida);

msg\_env(cadena, cli->uid);

fin\_cadena(cadena, strlen(cadena));

fin\_cadena(buffer\_salida, strlen(buffer\_salida));

printf("%s\n",cadena);

bzero(cadena, BUFFER\_SZ);

}

}

else if (recibido == 0 || strcmp(buffer\_salida, "exit") == 0){

printf("%s", buffer\_salida);

msg\_env(buffer\_salida, cli->uid);

flag\_salida = 1;

}

else {

printf("Fallo: -1\n");

flag\_salida = 1;

}

//Restaura el buffer a 0 para los siguientes mensajes

bzero(buffer\_salida, BUFFER\_SZ);

}

//Se cierra el socket del cliente

close(cli->sockfd);

//Se elimina cliente de la cola

eliminarCola(cli->uid);

//Liberamos el espacio asignado por malloc()

free(cli);

//Decrementamos el contador de clientes

cli\_count--;

//pthread\_detach(pthread\_self());

return NULL;

}

int main(int argc, char \*\*argv){

//Direccion de escucha del servidor.

char \*ip = "10.10.1.1";

int port = 2018;

int option = 1;

int escucha\_cliente = 0;

int conectar\_cliente = 0;

//Declaracion de la estructura de direcciones del server

struct sockaddr\_in s\_addr;

//Declaracion de la estructura de direcciones del cliente

struct sockaddr\_in c\_addr;

//Identificador del hilo del cliente

pthread\_t tid;

/\* Configuracion de opciones del socket \*/

//Asignamos IPv4(AF\_INET) y SOCK\_STREAM(tcp/ip)

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

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

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

//Convertimos los bytes de Host a Bytes de red

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

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

perror("Fallo: setsockopt failed");

return EXIT\_FAILURE;

}

/\* Bind \*/

if(bind(escucha\_cliente, (struct sockaddr\*)&s\_addr, sizeof(s\_addr)) < 0) {

perror("Fallo: Bind socket fallido");

return EXIT\_FAILURE;

}

/\* Escucha del servidor \*/

if (listen(escucha\_cliente, 10) < 0) {

perror("Fallo: Escucha del socket fallida");

return EXIT\_FAILURE;

}

while(1){

//Variable para almacenar el tamaño de la estructura de direcciones

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

//Aceptacion del cliente

conectar\_cliente = accept(escucha\_cliente, (struct sockaddr\*)&c\_addr, &clilen);

/\* Las opciones del cliente \*/

client\_est \*cli = (client\_est \*)malloc(sizeof(client\_est));

cli->address = c\_addr;

cli->sockfd = conectar\_cliente;

cli->uid = uid++;

cli->puerto = c\_addr.sin\_port;

//Añadimos el cliente a la cola

AnyadirCola(cli);

//Creamos hilo del cliente

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

}

return EXIT\_SUCCESS;

}