#include <string>

#include <stdio.h>

#include <iostream>

#include <stdlib.h> /\* atoi \*/

#include <fstream>

using namespace std;

#pragma comment(lib,"ws2\_32.lib")

#include <WinSock2.h>

#include "programa.h"

#define BUFFSIZE 1

int obtenerFuncion(char mensaje[100]){

int funtion=0;

string msj(mensaje);

string s1("(M)");

string s2("(T)");

string s3("(F)");

if((msj.compare(0, 2, s1, 0, 2)) == 0)funtion=1;

if((msj.compare(0, 2, s2, 0, 2)) == 0)funtion=2;

if((msj.compare(0, 2, s3, 0, 2)) == 0)funtion=3;

return funtion;

};

void receiveMessage(char mensaje[100]){

std::string newMensaje(mensaje);

newMensaje = newMensaje.substr(4);

std::cout << "Se recibio el mensaje: " << newMensaje << "\n";

};

void sendData(SOCKET socket, std::string cadena){

char cad2[100];

int n=0, b;

char mensaje[100];

char sendbuffer[100]; // envia de a 100 bits

std::string cad=cadena.substr(4);

strcpy(cad2, cad.c\_str());

FILE \*fp = fopen(cad2, "rb");

if(fp == NULL){

perror("File");

std::cout << "Error no se encontro el archivo. Espere a que le envien un mensaje.\n";

char msg[100]="error";

sendMessage(socket,msg);

}else{

strcpy(mensaje, cadena.c\_str());

send(socket, mensaje, sizeof(mensaje), 0);

std::cout << "Se empezara a mandar un archivo.\n";

int tam=0;

std::ifstream is; // usa lib <fstream> para manejo de archivo

is.open (cad2, std::ios::binary );

is.seekg (0, std::ios::end);

tam = is.tellg();

is.seekg (0, std::ios::beg);

send(socket, reinterpret\_cast<char\*>(&tam), sizeof tam , 0);

while( (b = fread(sendbuffer, 1, sizeof(sendbuffer), fp))>0 ){

send(socket, sendbuffer, b, 0);

}

std::cout << "Se envio el archivo.\n";

sleepTime(10);

}

}

void recvData(SOCKET socket, std::string mensaje){

char cad1[100];

std::string cad=mensaje.substr(0,3);

std::string nameFile=mensaje.substr(4);

strcpy(cad1, cad.c\_str());

char nombreArchivo[200];

char buff[1025];

int b,tam=0,tam\_ar;

strcpy(nombreArchivo, nameFile.c\_str());

std::cout << "Se recibio comando de transferencia de archivo." << "\n";

FILE\* fp = fopen(nombreArchivo, "wb");

if(fp != NULL){

std::cout << "El archivo se guardara dentro de: /" << nombreArchivo << "\n";

recv(socket, reinterpret\_cast<char\*>(&tam\_ar), sizeof tam\_ar, 0);

while(tam<tam\_ar){

b = recv(socket, buff, 1024,0);

fwrite(buff, 1, b, fp);

tam+=b;

}

if(tam<tam\_ar){

std::cout << "El archivo NO se recibio completamente" << "\n";

}

fclose(fp);

std::cout << "Archivo recibido." << "\n";

} else {

perror("File");

}

}

bool closer(SOCKET conexion){

std::cout << "Se proceder� a cerrar la aplicaci�n\n" ;

// Env�a mensaje

send(conexion, "(F)", sizeof("(F)"), 0);

// Finalizo

closesocket(conexion);

WSACleanup();

sleepTime(90);

return true;

}

int iniciarPrograma(char name[]){

SetConsoleTitleA(name);

SOCKADDR\_IN DIRECCION;

SOCKET conexion = INVALID\_SOCKET;

WSAData wsaData;

WORD dll = MAKEWORD(2, 1);

int puerto;

int iResult;

std::string IP;

// Initialize Winsock

iResult = WSAStartup(dll, &wsaData);

if (iResult != 0) {

std::cout << "WSAStartup failed with error: %d\n" << iResult;

return 1;

}

// Initialize Winsock

int sizeofaddr = sizeof(DIRECCION);

IP = pedirIP();

puerto = pedirPuerto();

DIRECCION.sin\_addr.s\_addr = inet\_addr(IP.c\_str());

DIRECCION.sin\_port = htons(puerto);

DIRECCION.sin\_family = AF\_INET;

// Create a SOCKET for connecting to server

// Socket: Crea un descriptor de socket. t

conexion = socket(AF\_INET, SOCK\_STREAM, NULL);

if (conexion == INVALID\_SOCKET) {

std::cout << "socket failed with error: %ld\n" << WSAGetLastError();

WSACleanup();

return 1;

}

// Connect to server.

// Connect: Inicia conexi�n con conector remoto

iResult = connect(conexion, (SOCKADDR\*)&DIRECCION, sizeofaddr);

if (iResult == SOCKET\_ERROR) {

// Close: Cierra socket

closesocket(conexion);

WSACleanup();

return 1;

}

std::cout << "solicitud de conexi�n aceptada" << "\n";

sleepTime(10);

std::cin.get();

int quit=false;

while (!quit)

{

std::string cadena;

char mensaje[100];

int opct = 0;

while(opct<1 || opct>3){

cadena = pedirCadena(" Cliente: ");

strcpy(mensaje, cadena.c\_str());

opct = obtenerFuncion(mensaje);

}

switch (opct){

case 1: // Funci�n M (Mensaje):

send(conexion, mensaje, sizeof(mensaje), 0);

break;

case 2: // Funci�n T (Transferencia de archivo):

sendData(conexion,mensaje);

break;

case 3: // Funci�n F (Fin):

quit = closer(conexion);

break;

default:

break;

}

if(!quit){

char mensaje2[100];

// Recv, Read: Recibe mensaje

recv(conexion, mensaje2, sizeof(mensaje2), 0);

opct = obtenerFuncion(mensaje2);

switch (opct){

case 1: // Funci�n M (Mensaje):

receiveMessage(mensaje2);

break;

case 2: // Funci�n T (Transferencia de archivo):

recvData(conexion,mensaje2);

break;

case 3: // Funci�n F (Fin):

quit = closer(conexion);

break;

default:

break;

}

}

}

// cleanup

WSACleanup();

};

std::string pedirCadena(std::string text){

std::string cadena;

std::cout << text;

std::getline(std::cin, cadena);

return cadena;

};

int pedirPuerto(){

int puerto = 0;

std::cout << "Require Port: ";

std::cin >> puerto;

return puerto;

};

std::string pedirIP(){

std::string IP ;

std::cout << "Require IP: ";

std::cin >> IP;

return IP;

};

void sleepTime(int time){

int i = 0;

while(i<time) {

i++;

Sleep(60);

}

};

void finalizar(){

WSACleanup();

};

#ifndef PROGRAMA\_H\_INCLUDED

#define PROGRAMA\_H\_INCLUDED

std::string pedirCadena(std::string text);

int pedirPuerto();

int obtenerFuncion(char mensaje[100]);

std::string pedirIP();

void sleepTime(int time);

void receiveMessage(char mensaje2[100]);

int iniciarPrograma(char name[]);

void finalizar();

#endif // PROGRAMA\_H\_INCLUDED

#include <iostream>

#include "programa.h"

int main()

{

iniciarPrograma("CLIENTE\_1");

finalizar();

return 0;

}

#include <string>

#include <stdio.h>

#include <iostream>

#include <stdlib.h> /\* atoi \*/

#include <fstream>

using namespace std;

#pragma comment(lib,"ws2\_32.lib")

#include <WinSock2.h>

#include "programa.h"

#define BUFFSIZE 1

int obtenerFuncion(char mensaje[100]){

int funtion=0;

string msj(mensaje);

string s1("(M)");

string s2("(T)");

string s3("(F)");

if((msj.compare(0, 2, s1, 0, 2)) == 0)funtion=1;

if((msj.compare(0, 2, s2, 0, 2)) == 0)funtion=2;

if((msj.compare(0, 2, s3, 0, 2)) == 0)funtion=3;

return funtion;

};

void sendMessage(SOCKET socket, char mensaje[100]){

send(socket, mensaje, sizeof(mensaje), 0);

}

void receiveMessage(char mensaje[100]){

std::string newMensaje(mensaje);

newMensaje=newMensaje.substr(4);

std::cout << "Se recibio el mensaje: " << newMensaje << "\n";

};

void sendData(SOCKET socket, std::string cadena){

char cad2[100];

int n=0, b;

char mensaje[100];

char sendbuffer[100]; // envia de a 100 bits

std::string cad=cadena.substr(4);

strcpy(cad2, cad.c\_str());

FILE \*fp = fopen(cad2, "rb");

if(fp == NULL){

perror("File");

std::cout << "Error no se encontro el archivo. Espere a que le envien un mensaje.\n";

char msg[100]="error";

sendMessage(socket,msg);

}else{

strcpy(mensaje, cadena.c\_str());

send(socket, mensaje, sizeof(mensaje), 0);

std::cout << "Se empezara a mandar un archivo.\n";

int tam=0;

std::ifstream is; // usa lib <fstream> para manejo de archivo

is.open (cad2, std::ios::binary );

is.seekg (0, std::ios::end);

tam = is.tellg();

is.seekg (0, std::ios::beg);

send(socket, reinterpret\_cast<char\*>(&tam), sizeof tam , 0);

while( (b = fread(sendbuffer, 1, sizeof(sendbuffer), fp))>0 ){

send(socket, sendbuffer, b, 0);

}

std::cout << "Se envio el archivo.\n";

sleepTime(10);

}

}

void recvData(SOCKET socket, std::string mensaje){

char cad1[100];

std::string cad=mensaje.substr(0,3);

std::string nameFile=mensaje.substr(4);

strcpy(cad1, cad.c\_str());

char nombreArchivo[200];

char buff[1025];

int b,tam=0,tam\_ar;

strcpy(nombreArchivo, nameFile.c\_str());

std::cout << "Se recibio comando de transferencia de archivo." << "\n";

FILE\* fp = fopen(nombreArchivo, "wb");

if(fp != NULL){

std::cout << "El archivo se guardara dentro de: /" << nombreArchivo << "\n";

recv(socket, reinterpret\_cast<char\*>(&tam\_ar), sizeof tam\_ar, 0);

while(tam<tam\_ar){

b = recv(socket, buff, 1024,0);

fwrite(buff, 1, b, fp);

tam+=b;

}

if(tam<tam\_ar){

std::cout << "El archivo NO se recibio completamente" << "\n";

}

fclose(fp);

std::cout << "Archivo recibido." << "\n";

} else {

perror("File");

}

}

bool closer(SOCKET conexion){

std::cout << "Se proceder� a cerrar la aplicaci�n\n" ;

// Env�a mensaje

sendMessage(conexion, "(F)");

// Finalizo

closesocket(conexion);

WSACleanup();

sleepTime(90);

return true;

}

int iniciarPrograma(char name[]){

SetConsoleTitleA(name);

WSAData wsaData;

WORD dll = MAKEWORD(2, 1);

int puerto;

// Initialize Winsock

if(WSAStartup(dll, &wsaData) != 0)

{

MessageBoxA(0, "ERROR, EJECUTE ESTE PROGRAMA COMO ADMINISTRADOR", "usuario", MB\_OK);

exit(0);

}

puerto = pedirPuerto(); // se pide el puerto en el que correra el servidor

// Resolve the server address and port

SOCKADDR\_IN direccion;

int tamano = sizeof(direccion);

direccion.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // ip en donde correra el servidor

direccion.sin\_port = htons(puerto); // se le pasa el puerto para hacer la direccion

direccion.sin\_family = AF\_INET;

// Create a SOCKET for connecting to server

SOCKET ESCUCHAR = INVALID\_SOCKET;

ESCUCHAR = socket(AF\_INET, SOCK\_STREAM, NULL);

if (ESCUCHAR == INVALID\_SOCKET) { //veo que carge bien el socket

std::cout << "socket failed with error: %ld\n" << WSAGetLastError();

WSACleanup();

return 1;

}

// bind el socket y ver si no hay error

int iResult;

iResult = bind(ESCUCHAR, (SOCKADDR\*)&direccion, sizeof(direccion));

if (iResult == SOCKET\_ERROR) {

std::cout << "bind failed with error: %d\n" << WSAGetLastError();

closesocket(ESCUCHAR);

WSACleanup();

return 1;

}

iResult = listen(ESCUCHAR, SOMAXCONN);

if (iResult == SOCKET\_ERROR) {

std::cout << "listen failed with error: %d\n" << WSAGetLastError();

closesocket(ESCUCHAR);

WSACleanup();

return 1;

}

// poner en accept para esperar conexion

SOCKET NUEVA\_CONEXION = INVALID\_SOCKET;

system("cls");

std::cout << "El servidor se inicio correctamente y esta esperando solicitudes en el puerto "<< puerto << "\n";

NUEVA\_CONEXION = accept(ESCUCHAR, (SOCKADDR\*)&direccion, &tamano);

if (NUEVA\_CONEXION == INVALID\_SOCKET) {

std::cout << "accept failed with error: %d\n" << WSAGetLastError();

closesocket(ESCUCHAR);

WSACleanup();

return 1;

}

// cuando se conecta

closesocket(ESCUCHAR);

std::cout << "se recibio una solicitud de conexion " << "\n";

sleepTime(10);

std::cin.get();

int quit=false;

while (!quit)

{

std::string cadena;

char mensaje[100];

char mensaje2[100];

int opct=0;

// Recv, Read: Recibe mensaje

recv(NUEVA\_CONEXION, mensaje2, sizeof(mensaje2), 0);

opct = obtenerFuncion(mensaje2);

switch (opct){

case 1: // Funci�n M (Mensaje):

receiveMessage(mensaje2);

break;

case 2: // Funci�n T (Transferencia de archivo):

recvData(NUEVA\_CONEXION,mensaje2);

break;

case 3: // Funci�n F (Fin):

quit = closer(NUEVA\_CONEXION);

break;

default:

break;

}

opct=0;

if(!quit){

while(opct<1 || opct>3){

cadena = pedirCadena(" Server: ");

strcpy(mensaje, cadena.c\_str());

opct = obtenerFuncion(mensaje);

}

switch (opct){

case 1: // Funci�n M (Mensaje):

send(NUEVA\_CONEXION, mensaje, sizeof(mensaje), 0);

break;

case 2: // Funci�n T (Transferencia de archivo):

sendData(NUEVA\_CONEXION,mensaje);

break;

case 3: // Funci�n F (Fin):

quit = closer(NUEVA\_CONEXION);

break;

default:

break;

}

}

}

// cleanup

closesocket(NUEVA\_CONEXION);

WSACleanup();

}

int pedirPuerto(){

int puerto = 0;

std::cout << "Initialize Port: ";

std::cin >> puerto;

return puerto;

}

std::string pedirCadena(std::string text){

std::string cadena;

std::cout << text;

std::getline(std::cin,cadena);

return cadena;

}

void sleepTime(int time) {

int i = 0;

while(i<time) {

i++;

Sleep(60);

}

}

void finaliar(){

WSACleanup();

}

#ifndef PROGRAMA\_H\_INCLUDED

#define PROGRAMA\_H\_INCLUDED

int iniciarPrograma(char name[]);

void finaliar();

int pedirPuerto();

void sleepTime(int time);

std::string pedirCadena(std::string text);

#endif // PROGRAMA\_H\_INCLUDED

#include <string>

#include <iostream>

using namespace std;

#include "programa.h"

int main()

{

iniciarPrograma("SERVIDOR");

finaliar();

return 0;

}

Manejo de comandos:

La aplicación cuenta con tres comandos:

* (M): envía un mensaje al otro usuario. Ej: (M) ¿Hola como estas?
* (T): transfiere un archivo. Ej: (T) C:\nombrearchivo.extencion (sin comillas).
* (F): cierra la comunicación entre los usuarios. Ej: (F).

Durante el envío de un mensaje lo primero que se hace es determinar que comando ingreso el usuario. Para esto una vez que se recibe la cadena que se ingresó por pantalla, se cortan los primeros 4 caracteres con la función substr() y se los guardan en otro string recientemente creado. A través de la función strcpy() se transforma el string en una cadena para poder usar el comando strcmp() y poder comparar esos primeros 4 caracteres con las funciones pre-designadas. Para poder determinar que comando se ingresó se usa un bucle switch() que determina que comando ingreso el usuario. De acuerdo al comando se realiza la acción correspondiente, pero si se ingresó una función inexistente si le pide al usuario que lo vuelva a intentar.

Para recibir un mensaje se utiliza el mismo método para determinar de qué comando se trata y así poder realizar la acción correcta, ya sea solo imprimir un mensaje, cerrar la aplicación o en su defecto guardar un archivo.

Envío y recepción de mensajes:

Para poder enviar un mensaje primero se crea un string en donde se guardará la cadena que ingrese el usuario. Luego se llamará a la función pedirCadena() la cual se encarga de mostrarle un mensaje al usuario y recibir la cadena que ingrese que luego devolverá para determinar qué tipo de comando ingresó. Si es un comando de mensaje entonces se lo enviará utilizando el comando send() que provee la librería winsock.

Para recibir un mensaje es similar al envío. Primero se comprueba a que comando pertenece y si es un mensaje lo recibe y lo muestra por pantalla.

Transferencia de archivos:

Para la transferencia de archivos, primero se le avisa al usuario que inicio un envío de algún archivo, y se abre el archivo para ver si existe, en caso de que no existe se le avisa al usuario el error. Si no hay error al abrir el archivo primero se crea una variable entera en la cual se guardara el tamaño total del archivo y se la enviará al usuario receptor para que lo pueda recibir correctamente y sin errores. Luego dentro de un bucle while se comenzará a leer y se enviará de a solo 100 bites hasta que se pasen todos los datos. Se le avisará al usuario que el archivo se envió de forma correcta y quedará listo para poder recibir un mensaje.

A la hora de recibir un archivo, se le avisa al usuario que esta por recibir uno, luego con el nombre y la ruta que se pasó desde el otro usuario se crea el archivo y se controla que no haya error, si lo hay se emitirá un mensaje. Si el archivo se pude crear de forma correcta el usuario pasará a estar en “recibiendo” hasta que le llegue el tamaño total. Una vez tenga el tamaño se armará un bucle while que se cortará una vez se llegue al tamaño total del archivo. Dentro del bucle se toman los datos que van llegando en el buffer y los va escribiendo en el archivo con la función fwrite(). Si el elemento no se pudo transferir completamente se emitirá un mensaje, caso contrario se le indicará al usuario que lo recibió sin errores.

Manejo de Socket:

Para poder realizar las llamadas de sockets, desde el servidor, junto con un cliente, nuestro programa siendo primitivo, el código del servidor se puede compilar y ejecutar en cualquier maquina Windows conectada a una red local, también se puede compilar y ejecutar el código del cliente en cualquier otra máquina que posee dicho Sistema operativo.

Primero explicó el código del servidor. Requerirá un PUERTO como 8080, este número es elegido al azar, todo número desde 1024 al 65535 funcionara siempre y cuando otro proceso no lo esté utilizando. Esto permite al socket inicializarse y verifica que no tenga errores. Continúa con verificar si la IP cargada por código fuente es correcta llamando la función “bind”. si fue correcto, continua y escucha la petición y lista el socket para anunciar que el servidor está dispuesto a aceptar llamadas entrantes e indicar al sistema que almacene la llamada de un cliente, una vez realizado correctamente todo, el servidor esta iniciada y espera una conexión del cliente en dicho puerto.

Desde el cliente se debe iniciar y cargar la IP y un número de Puerto, este se obtiene desde el código fuente del servidor, y configuraciones cargadas de dicha aplicación. A continuación, inicializa un socket para establecer la conexión TCP con el servidor mediante la función “connect”. Si el servidor está activo y ejecutándose en la máquina especificada y enlazada al Puerto, y si está vacía su cola de “listen” se realiza la conexión. El cliente puede comenzar enviando comando por consola al servidor. Esto lo lleva a un bucle esperando peticiones por teclado hasta que la variable “quit” este en true y termina la conexión cerrando estos sockets.

Elección del Sistema operativo:

Se realizó pruebas tanto en Windows como en Linux, logrando mejores expectativas en Windows con las compatibilidades y acceso de nuestros equipos, se eligió el S.O debido por su interface cómoda y compatibilidad con el lenguaje de programación en C++.

Organización del bucle del Programa:

La aplicación se organiza de la siguiente manera, el siguiente diagrama de flujo demuestra de cómo se maneja las funciones en nuestra aplicación del servidor:

La aplicación comienzo esperando un comando de cliente, este lo recibe y lo lee , si es >>(M) va dirigido a la función “receiveMessage()” , si es >>(T) va a la función de recibir datos “recvData()”, y si es >>(F) finaliza y cierra el bucle caso contrario continua, los siguientes pasos son, pedir por consola un comando y realiza si es >>(M) enviar mensaje por la función “send()”, si es >>(T) a la función de enviar datos “sendData()”, y si es (F) finaliza la aplicación y cierra el socket.

Con respecto a Cliente se maneja de igual manera solamente que esta invertido, solicita primero un comando y luego espera respuesta del servidor.

