Завдання 26, клієнт-сервер.

Опис роботи

Сервер

Створюємо структуру даних , в якій зберігається повідомлення (масив символів), та індекс клієнта, який відправив повідомлення

#define BUF\_SIZE 1024

#define Clients 10 // Підключення до 10 клієнтів

using namespace std;

struct MsgData

{

char mess [BUF\_SIZE];

int fromClient;

};

Cтворюємо масив клієнтів та ініціалізуємо його нулями

MsgData MsgArr [Clients];

SOCKET Sockets[Clients] = {0,0,0,0,0,0,0,0,0,0};

Cтворюємо функцію, яка перевіряє підключення, форматує текст, відправляє його клієнту, та закриває з’єднання.

Перевірка підключення (якщо який-небудь з сокетів повертає значення INVALID\_SOCKET, то ми закриваємо підключення

while (1){

if (Sockets[idx] == INVALID\_SOCKET){

break;

}

//інші перевірки

} closesocket(Sockets[idx]);

Sockets[idx] = 0;

cout <<"Socket with index "<<idx<<" closed."<<endl;

return;

Перевірка успішності отримання повідомленя та пошук індексів з повідомлення

int result = recv(Sockets[idx],buffer,BUF\_SIZE,0);

if (result > 0){

string msg(buffer);

// format: "fromId=XX\_toId=XX\_msg:message"

if (msg.find("fromId=") == 0 && msg.find("toId=") == 10){

int fromId = atoi(msg.substr(7,2).c\_str());

int toId = atoi(msg.substr(15,2).c\_str());

Запис повідомлення для клієнта з індексом toId

if (fromId < Clients){

int start\_pos = msg.find\_first\_of(":")+1;

string message = msg.substr(start\_pos,msg.length()-start\_pos);

string storedMesssge (MsgArr[toId-1].mess);

if (storedMesssge.length()){

storedMesssge+='\n';

message = storedMesssge + message;

}

strcpy(MsgArr[toId-1].mess, message.c\_str());

MsgArr[toId-1].fromClient = fromId;

char outbuff [1024] = "\0";

Перевірка повідомлення для клієнта, його відправка та очистка переданих повідомлень

if (MsgArr[fromId-1].fromClient > 0){

sprintf(outbuff, "fromId=%02d\_msg:%s", MsgArr[fromId-1].fromClient, MsgArr[fromId-1].mess);

}

send (Sockets[idx],outbuff,sizeof(outbuff),0);

MsgArr[fromId-1].fromClient = 0;

MsgArr[fromId-1].mess[0] = '\0';

В функції main відбувається ініціалізація бібліотеки Windows Sockets, задається адреса, порт та протокол, прив’язується сокет, відбувається його прослуховування та відпрацювання приєднань/ від’єднань

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

{

int index = 0;

WSAData wsaData;

WORD DllVersion = MAKEWORD (2, 1);

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

{

cout<<"Error downloading lib"<<endl;

exit (1);

}

SOCKADDR\_IN addr;

int sizeofaddr = sizeof(addr);

addr.sin\_addr.s\_addr = INADDR\_ANY;

addr.sin\_port = htons (1111);

addr.sin\_family = AF\_INET;

SOCKET sListen = socket (AF\_INET, SOCK\_STREAM,IPPROTO\_TCP);

bind (sListen, (SOCKADDR\*)&addr, sizeof (addr));

listen (sListen, SOMAXCONN);

if (listen (sListen, SOMAXCONN) == SOCKET\_ERROR)

{

cout <<"Listening failed with error: %d/n",WSAGetLastError();

WSACleanup();

return 1;

}

Відпрацювання підключень та відключень:

while (1){

while (Sockets[index]!=0){

(index < Clients -1)?index++:index=0;

}

Sockets[index] = accept (sListen, (SOCKADDR\*)&addr,&sizeofaddr);

if (Sockets[index] == INVALID\_SOCKET){

cout <<"Connection Error!"<<endl;

closesocket(Sockets[index]);

WSACleanup();

Sockets[index] = 0;

continue;

}

cout <<"New client successfully connected"<<endl;

Створення нового потоку для обробки даних (задля того, щоб одночасно можна було приєднувати/від’єднувати клієнтів та обробляти іх запроси)

CreateThread(NULL,0, (LPTHREAD\_START\_ROUTINE)ClientHandler, (LPVOID) index ,0,NULL);

}

Клієнт

Створюємо структуру даних , в якій зберігається повідомлення (масив символів), та індекс клієнта, який відправив повідомлення

struct MsgData

{

char mess [BUF\_SIZE];

int toClient;

int fromClient;

};

MsgData data;

SOCKET connection;

Створюємо функцію connectToServer, в якій ми ініціалізуємо бібліотеку WindowsSockets, , задаємо адресу, порт та протокол, намагаємось під’єднатись до сервера

bool connectToServer (void){

WSAData wsaData;

WORD DllVersion = MAKEWORD (2,2);

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

{

cout << "Error downloading lib" << endl;

exit (1);

}

SOCKADDR\_IN addr;

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

addr.sin\_port = htons (1111);

addr.sin\_family = AF\_INET;

connection = socket (AF\_INET, SOCK\_STREAM,IPPROTO\_TCP);

if (connect(connection, (SOCKADDR\*)&addr, sizeof(addr))!=0){

cout<<"Error!Failed to connect to server"<<endl;

return false;

} else {

cout<<"Successfuly connected"<<endl;

return true;

}

}

Для зручності створюємо окрему функцію для від’єднання від сервера

void disconnectFromServer (void){

closesocket(connection);

WSACleanup();

}

В функції main ми вводимо ID клієнта, потім перевіряємо чи вдалось під’єднатись до сервера, якщо так, то даємо ввести повідомлення (спочатку слово до пробілу, потім те, що залишилось після пробілу, склеюємо текст) Якщо текст – це слово exit, клієнт завершує свою роботу (перевірка за допомогою strcmp). Потім вводимо індекс клієнта, якому хочемо відправити повідомлення. Наступним кроком є перевірка з’єднання, форматування повідомлення, пакування його в буфер, та відправка цього буфера.

if (connectToServer()){

while (true){

string line, text;

cout<<"Enter your message: ";

cin>>text; // Читаємо перше слово до пробілу

getline(cin, line); // Читаємо частину строки після пробілу

line = text+line; // Додавання строк

cout<<endl;

strcpy(data.mess,line.c\_str());

if (strcmp(data.mess,"exit") == 0){

break;

}

cout<<"Send to (index): ";

cin>>data.toClient;

cout<<endl;

string str(data.mess);

if (connection == INVALID\_SOCKET)

break;

char outbuf[BUF\_SIZE];

sprintf(outbuf, "fromId=%02d\_toId=%02d\_msg:%s", data.fromClient, data.toClient, data.mess);

send(connection,outbuf,sizeof(outbuf),0);

Останнім кроком є отримання повідомлення від сервера, його обробка та виведення

char inbuf[BUF\_SIZE];

int result = recv(connection,inbuf,BUF\_SIZE,0);

if (result > 0){

string msg(inbuf);

if (msg.find("fromId=") == 0){

int fromId = atoi(msg.substr(7,2).c\_str());

int start\_pos = msg.find\_first\_of(":")+1;

string context = msg.substr(start\_pos,msg.length()-start\_pos);

cout <<"New message from [" << fromId << "]: " << context << endl;

}