#include <iostream>

#include <winsock2.h>

#include <Windows.h>

#include <string>

#include <vector>

#pragma warning(disable: 4996)

#pragma comment (lib, "Ws2\_32.lib")

using namespace std;

CRITICAL\_SECTION cs;

vector <pair<string, SOCKET>> sockStore(0);

const short serverPort = 1234; // Порт сервера

enum typeOfMessage {

CONNECT,

DISCONNECT,

PRIVATE,

ALL

};

struct Client {

char nick[1024];

typeOfMessage current;

char privateNick[1024];

char message[1024];

};

void initializeLibrary() {

WSADATA wsaData;//содержит информацию о реализации сокетов Windows

int errorCode = WSAStartup(MAKEWORD(2, 2), &wsaData); // MAKEWORD(2, 2) == 0x0202 - версия 2.2

if (errorCode != 0) {

cout << "Error initialization" << endl;

exit(1);

}

}

SOCKET createSocket() {

SOCKET sock = socket(AF\_INET, SOCK\_STREAM, 0); // Создает TCP сокет

if (sock == INVALID\_SOCKET) {

cout << "Error socket creation" << endl;

exit(1);

}

return sock;

}

void bindSocket(SOCKET sock) {

sockaddr\_in address;

address.sin\_family = AF\_INET;

address.sin\_port = serverPort;

address.sin\_addr.s\_addr = 0;

int errorCode = bind(sock, (sockaddr\*)&address, sizeof(address));

if (errorCode != 0) {

cout << "Error bind" << endl;

exit(1);

}

}

SOCKET acceptConnection(SOCKET listeningSock) {

sockaddr\_in clientAddress;

int clientAddressSize = sizeof(clientAddress);

SOCKET result = accept(listeningSock, (sockaddr\*)&clientAddress, &clientAddressSize);

// Делаем что то с адресом клиента clientAddress

// Если не нужно, можно просто сделать accept(listeningSock, NULL, NULL)

HOSTENT\* hst;

hst = gethostbyaddr((char\*)&clientAddress.sin\_addr, 4, AF\_INET);

cout << "New connection: ";

cout << ((hst) ? hst->h\_name : "Unknown host") << "/" << inet\_ntoa(clientAddress.sin\_addr) << "/" << ntohs(clientAddress.sin\_port) << '\n';

//inet\_ntoa преобразует сетевой адрес Интернета (Iv4) в строку ASCII

//ntohs преобразует u\_short из сетевого порядка байтов TCP / IP в порядок байтов хоста

return result;

}

Client clearClient(Client example) {

example.nick[0] = '\n';

example.current = DISCONNECT;

example.privateNick[0] = '\n';

example.message[0] = '\n';

return example;

}

Client readRequest(SOCKET sock) {

Client buffer;

int bytes = recv(sock, (char\*)&buffer, sizeof(buffer), 0);

if (bytes == 0) { // Если клиент отсоединился

buffer = clearClient(buffer);

return buffer;

}

else if (bytes < 0) {

cout << "ERROR" << endl;

buffer = clearClient(buffer);

return buffer;

}

else return buffer;

}

bool alreadyExist(string nick) {

for (int i = 0; i < sockStore.size(); i++) {

if (sockStore[i].first == nick)

return true;

}

return false;

}

// Вернет true, если клиент отсоединился

bool writeResponse(SOCKET sock, string str) {

int bytes = send(sock, str.c\_str(), str.length() + 1, 0);//c\_str - возвращает символ const char\* , указывающий на строку с завершением null

if (bytes < 0) {

cout << "ERROR" << endl;

return true;

}

else return bytes == 0;

}

DWORD WINAPI ThreadWork(LPVOID lpParameter) {

SOCKET clientSock = \*(SOCKET\*)lpParameter;

while (true) {

Client request = readRequest(clientSock);

EnterCriticalSection(&cs);

if (request.current == CONNECT) {

if (alreadyExist(string(request.nick))) {

writeResponse(clientSock, "Nick has already been taken");

closesocket(clientSock);

LeaveCriticalSection(&cs);

break;

}

else {

string list\_of\_members = "";

if (!sockStore.empty()) {

list\_of\_members += "\nOnline:\n";

for (pair<string, SOCKET> example : sockStore) {

list\_of\_members += example.first;

list\_of\_members += "\n";

}

}

writeResponse(clientSock, "Welcome to the chat. Commands: '-leave'(to leave) '-private'(private massege).\n" + list\_of\_members);

sockStore.push\_back(make\_pair(string(request.nick), clientSock));

}

}

else if (request.current == DISCONNECT) {

closesocket(clientSock);

sockStore.erase(std::find(sockStore.begin(), sockStore.end(), make\_pair(string(request.nick), clientSock)));

for (int i = 0; i < sockStore.size(); i++) {

writeResponse(sockStore[i].second, string(request.nick) + " leave the chat.\n");

}

LeaveCriticalSection(&cs);

break;

}

else if (request.current == ALL) {

for (int i = 0; i < sockStore.size(); i++) {

if (clientSock != sockStore[i].second)

writeResponse(sockStore[i].second, string(request.message));

}

}

else if (request.current == PRIVATE) {

for (int i = 0; i < sockStore.size(); i++) {

if (sockStore[i].first == string(request.privateNick))

writeResponse(sockStore[i].second, "PRIVATE\_MESSAGE\_FROM " + string(request.message));

}

}

LeaveCriticalSection(&cs);

}

return 0;

}

int main() {

setlocale(LC\_ALL, 0);

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

initializeLibrary();

InitializeCriticalSection(&cs);

SOCKET listeningSock = createSocket();

bindSocket(listeningSock);

listen(listeningSock, SOMAXCONN);

cout << "Listening..." << endl;

while (true) {

SOCKET clientSock = acceptConnection(listeningSock);

CreateThread(NULL, 0, &ThreadWork, &clientSock, 0, NULL);

}

}