**МІНІСТЕРСТВО НАУКИ ТА ОСВІТИ УКРАЇНИ**

**ОДЕСЬКА НАЦІОНАЛЬНА АКАДЕМІЯ ЗВ'ЯЗКУ ІМ. О. С. ПОПОВА**

**ЗВІТ**

**до лабораторного завдання №3**

**з дисципліни «Мережеве програмування»**

Виконав

студент групи ІПЗ-3.02

Писарев С. Д.

Перевірив

Викладач Вороний С. М.

Задание

Усовершенстврвать серверную часть программы, разработанную в лабораторной работе 3 так, чтобы работа с сокетами была реализована по модели select.

**Вариант №7**

Клиентская программа оправляет на сервер текстовый файл из N строк (путь к файлу задается через пользовательский интерфейс клиента). Сервер подсчитывает общее количество слов длиной не менее пяти символов в строках файла. Это количество возвращает клиенту.

Листинг

Server.cpp

#include <iostream>

#include <fstream>

#include <thread>

#include <WinSock2.h>

#include <ws2tcpip.h>

#include "Server.h"

#include "Helpers.h"

#pragma comment(lib, "Ws2\_32.lib")

using namespace std;

constexpr auto DEFAULT\_BUFLEN = 512;

const string filename = "fileToProcess.txt";

typedef struct \_SOCKET\_INFORMATION {

    CHAR Buffer[DEFAULT\_BUFLEN];

    WSABUF DataBuf;

    SOCKET Socket;

    OVERLAPPED Overlapped;

    DWORD BytesSEND;

    DWORD BytesRECV;

} SOCKET\_INFORMATION, \* LPSOCKET\_INFORMATION;

DWORD TotalSockets = 0;

LPSOCKET\_INFORMATION SocketArray[FD\_SETSIZE];

SOCKET ListenSocket = INVALID\_SOCKET;

SOCKET ClientSocket = INVALID\_SOCKET;

constexpr auto TIMEOUT\_SEC = 150;

constexpr auto TIMEOUT\_MSEC = 0;

timeval timeout = { TIMEOUT\_SEC, TIMEOUT\_MSEC };

void OutputClientState(SocketData\* socketData, bool isConnected)

{

    int foreground\_color, background\_color;

    string state;

    if (isConnected)

    {

        foreground\_color = FOREGROUND\_GREEN;

        background\_color = BACKGROUND\_GREEN;

        state = "connected";

    }

    else

    {

        foreground\_color = FOREGROUND\_RED;

        background\_color = BACKGROUND\_RED;

        state = "disconnected";

    }

    HANDLE hConsoleOutput = GetStdHandle(STD\_OUTPUT\_HANDLE);

    SetConsoleTextAttribute(hConsoleOutput, foreground\_color);

    cout << "The client with an IP address ";

    SetConsoleTextAttribute(hConsoleOutput, background\_color);

    cout << socketData->ip;

    SetConsoleTextAttribute(hConsoleOutput, foreground\_color);

    cout << " was " << state << " via " << socketData->port << " port." << endl;

    SetConsoleTextAttribute(hConsoleOutput, FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_RED);

}

BOOL \_\_stdcall CtrlHandler(DWORD fdwCtrlType)

{

    CorrectClosing(ListenSocket);

    return FALSE;

}

void ClientSession(SocketData socketData, fd\_set readset)

{

    OutputClientState(&socketData, true);

    ClientSocket = socketData.socket;

    FD\_ZERO(&readset);

    FD\_SET(ClientSocket, &readset);

    if (select(1, &readset, NULL, NULL, &timeout) <= 0)

    {

        cout << "Error with SELECT function" << endl;

        return;

    }

    if (!FD\_ISSET(ClientSocket, &readset))

    {

        printf("Error. FD\_ISSET is false");

        return;

    }

    // ��������� ������ �� �������, ������ ��

    char recvbuf[DEFAULT\_BUFLEN];

    string fileData;

    int iResult,

        fileSize = 0,

        recvbuflen = DEFAULT\_BUFLEN;

    // Receive until the peer shuts down the connection.

    while (iResult = recv(ClientSocket, recvbuf, recvbuflen, 0))

    {

        if (iResult > 0)

        {

            string actualData(recvbuf, recvbuf + iResult);

            if (fileSize == 0)

            {

                fileSize = stoi(actualData);

                continue;

            }

            fileData.append(actualData);

            fileSize -= iResult;

            if (fileSize == 0)

            {

                // A file receiving is complete.

                // Writes file contents in binary to a new file.

                fstream myfile(filename, ios::binary | ios::out | ios::trunc);

                myfile.write(fileData.c\_str(), fileData.length());

                myfile.close();

                myfile.open(filename, ios::in);

                string word;

                int numOfWordsGTEQFiveLetters = 0;

                // Reads the file word by word.

                while (myfile >> word)

                {

                    if (word.length() >= 5)

                    {

                        numOfWordsGTEQFiveLetters++;

                    }

                }

                string valueToSend = to\_string(numOfWordsGTEQFiveLetters);

                send(ClientSocket, valueToSend.c\_str(), valueToSend.length(), 0);

                myfile.close();

                fileData.clear();

            }

        }

        else

        {

            OutputClientState(&socketData, false);

            shutdown(ClientSocket, SD\_BOTH);

            closesocket(ClientSocket);

            return;

        }

    }

}

Server::Server(string& serverPort)

{

    WSADATA wsaData;

    int iResult;

    // Initialize Winsock

    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

    if (iResult != 0) {

        printf("WSAStartup failed: %d\n", iResult);

        return;

    }

    SOCKET ClientSocket = INVALID\_SOCKET;

    struct addrinfo\* result = NULL, \* ptr = NULL, hints;

    ZeroMemory(&hints, sizeof(hints));

    hints.ai\_family = AF\_INET;

    hints.ai\_socktype = SOCK\_STREAM;

    hints.ai\_protocol = IPPROTO\_TCP;

    hints.ai\_flags = AI\_PASSIVE;

    // Resolve the local address and port to be used by the server

    iResult = getaddrinfo(NULL, serverPort.c\_str(), &hints, &result);

    if (iResult != 0) {

        printf("getaddrinfo failed: %d\n", iResult);

        WSACleanup();

        return;

    }

    // Create a SOCKET for the server to listen for client connections

    ListenSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

    if (ListenSocket == INVALID\_SOCKET) {

        printf("Error at socket(): %ld\n", WSAGetLastError());

        freeaddrinfo(result);

        WSACleanup();

        return;

    }

    // Change the socket mode on the listening socket from blocking to

    // non-block so the application will not block waiting for requests

    u\_long NonBlock = 1;

    if (ioctlsocket(ListenSocket, FIONBIO, &NonBlock) == SOCKET\_ERROR)

    {

        printf("ioctlsocket() failed with error %d\n", WSAGetLastError());

        return;

    }

    // Setup the TCP listening socket

    // Without a double colon behind the bind function it won't work. It means that it will use the bind func from the global namespace.

    iResult = ::bind(ListenSocket, result->ai\_addr, (int)result->ai\_addrlen);

    if (iResult == SOCKET\_ERROR) {

        printf("bind failed with error: %d\n", WSAGetLastError());

        freeaddrinfo(result);

        closesocket(ListenSocket);

        WSACleanup();

        return;

    }

    freeaddrinfo(result);

    // Not necessary.

    SetConsoleCtrlHandler(CtrlHandler, true);

    if (listen(ListenSocket, SOMAXCONN) == SOCKET\_ERROR) {

        printf("Listen failed with error: %ld\n", WSAGetLastError());

        closesocket(ListenSocket);

        WSACleanup();

        return;

    }

    while (1)

    {

        fd\_set readset;

        FD\_ZERO(&readset);

        FD\_SET(ListenSocket, &readset);

        int SelectTiming = select(1, &readset, NULL, NULL, &timeout);

        switch (SelectTiming)

        {

        case 0:

            // Timed out, do whatever you want to handle this situation

            printf("\nServer: Timeout lor while waiting you retard client!...\n");

            return;

        case -1:

            // Error occurred, more tweaking here and the recvTimeOutTCP()...

            printf("\nServer: Some error encountered with code number : % ld\n", WSAGetLastError());

            return;

        default:

            printf("Select is OK\n");

            break;

        }

        struct sockaddr socketInfo = {};

        socklen\_t sockLen = sizeof(socketInfo);

        if (FD\_ISSET(ListenSocket, &readset))

        {

            cout << "FD\_ISSET(listener, &readset) succesfull" << endl;

            // accept

            ClientSocket = accept(ListenSocket, &socketInfo, &sockLen);

            if (ClientSocket < 0)

            {

                cout << "Error with ACCEPT function" << endl;

            }

            else {

                cout << "ACCEPT succesfull" << endl;

            }

            u\_long ul1 = 1;

            int nRet1;

            nRet1 = ioctlsocket(ClientSocket, FIONBIO, &ul1);

        }

        // Gets an IP address.

        char ip[INET\_ADDRSTRLEN];

        getIP(socketInfo, ip);

        string sip(ip);

        SocketData socketData;

        socketData.socket = ClientSocket;

        socketData.ip = sip;

        socketData.port = serverPort;

        ClientSession(socketData, readset);

    }

}

ClientServer.cpp

#undef UNICODE

#define WIN32\_LEAN\_AND\_MEAN

#include <iostream>

#include <stdio.h>

#include <algorithm>

#include <iterator>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <boost/program\_options.hpp>

#include "Client.h"

#include "Server.h"

#include "ClientServer.h"

#include "Helpers.h"

#pragma comment(lib, "Ws2\_32.lib")

#pragma comment (lib, "Mswsock.lib")

#pragma comment (lib, "AdvApi32.lib")

using namespace std;

namespace po = boost::program\_options;

int main(int argc, char\* argv[])

{

    string PORT, IP\_ADDRESS;

    try {

        po::options\_description desc("Options");

        desc.add\_options()

            ("help,h", "Produce a help message.")

            ("server,S", "The program will start as a server.")

            ("client,C", "The program will start as a client.")

            ("ip,I", po::value< string >(&IP\_ADDRESS), "Provides an IP address.")

            ("port,P", po::value< string >(&PORT), "Provides a port.")

            ;

        po::variables\_map vm;

        po::store(po::parse\_command\_line(argc, argv, desc), vm);

        po::notify(vm);

        if (vm.count("help")) {

            cout << desc << "\n";

            return 0;

        }

        cout << "IP: " << IP\_ADDRESS << endl

             << "Port: " << PORT << endl << endl;

        //SetConsoleCtrlHandler(CtrlHandler, true);

        if (vm.count("server")) {

            cout << "The program was started as a server." << "\n";

            new Server(PORT);

        }

        else if (vm.count("client"))

        {

            cout << "The program was started as a client." << "\n";

            new Client(PORT, IP\_ADDRESS);

        }

    }

    catch (std::exception& e) {

        cerr << "error: " << e.what() << "\n";

        return 1;

    }

    catch (...) {

        cerr << "Exception of unknown type!\n";

    }

    return 0;

}

Client.cpp

#undef UNICODE

#define WIN32\_LEAN\_AND\_MEAN

#include <iostream>

#include <fstream>

#include <WinSock2.h>

#include <ws2tcpip.h>

#include <thread>

#include "Client.h"

#include "Helpers.h"

#pragma comment(lib, "Ws2\_32.lib")

#pragma comment (lib, "Mswsock.lib")

#pragma comment (lib, "AdvApi32.lib")

using namespace std;

constexpr auto DEFAULT\_BUFLEN = 512;

Client::Client(string& PORT, string& IP\_ADDRESS)

{

    WSADATA wsaData;

    int iResult;

    SOCKET ConnectSocket = INVALID\_SOCKET;

    struct addrinfo\* result = NULL,

                   \* ptr = NULL,

                     hints;

    // Initializes Winsock.

    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

    if (iResult != 0)

    {

        printf("WSAStartup failed with error: %d\n", iResult);

        return;

    }

    ZeroMemory(&hints, sizeof(hints));

    hints.ai\_family = AF\_UNSPEC;

    hints.ai\_socktype = SOCK\_STREAM;

    hints.ai\_protocol = IPPROTO\_TCP;

    // Resolves a server address and port.

    iResult = getaddrinfo(IP\_ADDRESS.c\_str(), PORT.c\_str(), &hints, &result);

    if (iResult != 0)

    {

        printf("getaddrinfo failed with error: %d\n", iResult);

        WSACleanup();

        return;

    }

    // Attempt to connect to an address until one succeeds

    for (ptr = result; ptr != NULL; ptr = ptr->ai\_next)

    {

        // Creates a SOCKET for connecting to a server.

        ConnectSocket = socket(ptr->ai\_family,

                               ptr->ai\_socktype,

                               ptr->ai\_protocol);

        if (ConnectSocket == INVALID\_SOCKET)

        {

            printf("socket failed with error: %ld\n", WSAGetLastError());

            WSACleanup();

            return;

        }

        // Connects to a server.

        iResult = connect(ConnectSocket, ptr->ai\_addr, (int)ptr->ai\_addrlen);

        if (iResult == SOCKET\_ERROR)

        {

            closesocket(ConnectSocket);

            ConnectSocket = INVALID\_SOCKET;

            continue;

        }

        break;

    }

    freeaddrinfo(result);

    if (ConnectSocket == INVALID\_SOCKET)

    {

        printf("Unable to connect to server!\n");

        WSACleanup();

        return;

    }

    string input;

    char recvbuf[DEFAULT\_BUFLEN];

    int recvbuflen = DEFAULT\_BUFLEN;

    while (true)

    {

        string path;

        cout << "Type the path to a file." << endl;

        cin >> path;

        // Reads contents of the file.

        ifstream in(path);

        string contents((istreambuf\_iterator<char>(in)),

            istreambuf\_iterator<char>());

        // Sends the size of the file contents and then the contents itself.

        string contentsSize = to\_string(contents.length());

        send(ConnectSocket, contentsSize.c\_str(), contentsSize.length(), 0);

        iResult = send(ConnectSocket, contents.c\_str(), contents.length(), 0);

        if (iResult == SOCKET\_ERROR)

        {

            SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_RED);

            printf("The server was shutdown.");

            CorrectClosing(ConnectSocket);

            return;

        }

        else

        {

            // Receives the answer from a server.

            iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);

            string actualData(recvbuf, recvbuf + iResult);

            // Nicely displays the answer.

            SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_BLUE);

            cout << "The num of words 5 letters or more in the file: ";

            SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), BACKGROUND\_BLUE);

            cout << actualData << endl << endl;

            SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE);

        }

    }

}

Helpers.cpp

#include <vector>

#include <iterator>

#include <string>

#include <ws2tcpip.h>

using namespace std;

struct SocketData

{

    SOCKET socket;

    string ip;

    string port;

};

// A helper function to simplify the main part.

template<class T>

ostream& operator<<(ostream& os, const vector<T>& v)

{

    copy(v.begin(), v.end(), ostream\_iterator<T>(os, " "));

    return os;

}

void CorrectClosing(SOCKET socket)

{

    // Sets the console attributes to default values.

    SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE);

    closesocket(socket);

    // Should be called at the end of the program and be paired with WSAStartup.

    WSACleanup();

}

void getIP(sockaddr socketInfo, char\* ip)

{

    sockaddr\_in\* sin = reinterpret\_cast<sockaddr\_in\*>(&socketInfo);

    inet\_ntop(AF\_INET, &sin->sin\_addr, ip, INET\_ADDRSTRLEN);

}

Скриншоты выполнения программы



