БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Международный Институт Дистанционного Образования

Кафедра “Информационные системы и технологии”

Контрольная работа по дисциплине «ООП»

Тема работы:

**«Компьютерные сети»**

Выполнил:   
студент 3 курса, гр. 41703120  
Реут Владислав Леонидович  
№ зачетной книжки 417031208

Проверил: Русак Л.В.

Минск 2023

**ВВЕДЕНИЕ**

Компьютерная сеть – это взаимосвязанные вычислительные устройства, которые могут обмениваться данными и совместно использовать ресурсы. Эти сетевые устройства используют систему правил, называемых коммуникационными протоколами, для передачи информации посредством физических или беспроводных технологий.

Узлы и ссылки являются опорными точками компьютерных сетях. Сетевым узлом может быть оборудование передачи данных (data communication equipment, DCE), такое как модем, концентратор или коммутатор, или терминальное оборудование обработки данных (data terminal equipment, DTE), такое как два или более компьютеров и принтеров. Канал относится к среде передачи, соединяющей два узла. Связи могут быть физическими, такими как кабели или оптические волокна, или свободным пространством, используемым беспроводными сетями.  
  
В работающей компьютерной сети узлы следуют набору правил или протоколов, которые определяют, как отправлять и получать электронные данные по ссылкам. Архитектура компьютерной сети определяет конструкцию этих физических и логических компонентов. Он предоставляет спецификации для физических компонентов сети, функциональной организации, протоколов и процедур.

Компьютерные сети были впервые созданы в конце 1950-х годов для использования в вооруженных силах и обороне. Первоначально они использовались для передачи данных по телефонным линиям и имели ограниченное коммерческое и научное применение. С появлением интернет-технологий компьютерная сеть стала незаменимой для предприятий.

Современные сетевые решения обеспечивают больше, чем возможность подключения. Сегодня они имеют решающее значение для цифровой трансформации и успеха бизнеса. Базовые сетевые возможности стали более программируемыми, автоматизированными и безопасными.

Базовая физическая сетевая инфраструктура может быть логически разделена для создания нескольких оверлейных сетей. В оверлейной компьютерной сети узлы виртуально связаны, и данные могут передаваться между ними по нескольким физическим путям. Например, многие корпоративные сети накладываются на Интернет.

Современные сетевые сервисы соединяют физически распределенные компьютерные сети. Эти сервисы могут оптимизировать сетевые функции за счет автоматизации и мониторинга для создания одной крупномасштабной высокопроизводительной сети. Сетевые услуги можно увеличивать или уменьшать в зависимости от спроса.

Многие компьютерные сети программно-определяемы. Трафик можно направлять и контролировать централизованно с помощью цифрового интерфейса. Эти компьютерные сети поддерживают управление виртуальным трафиком.

Все сетевые решения поставляются со встроенными функциями безопасности, такими как шифрование и контроль доступа. Сторонние решения, такие как антивирусное ПО, брандмауэры и антивредоносные программы, могут быть интегрированы, чтобы сделать сеть более безопасной.

**Задание №1**

1. Отображение MAC-адреса компьютера (можно воспользоваться функцией netbios).

2. Отображение всех рабочих групп, компьютеров в сети и их ресурсов (папок, открытых для общего доступа, принтеров).

Листинг программы(включает в себя 2 задачи):

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <winsock2.h>

#include <iptypes.h>

#include <iphlpapi.h>

#include <windows.h>

#include <iostream>

#include <stdio.h>

#include <winnetwk.h>

#pragma comment(lib, "mpr.lib")

#pragma comment(lib, "netapi32.lib")

void DisplayStruct(int i, LPNETRESOURCE lpnrLocal);

typedef struct \_ASTAT\_

{

ADAPTER\_STATUS adapt;

NAME\_BUFFER NameBuff[30];

}ASTAT, \* PASTAT;

ASTAT Adapter;

void GetMacAddress()

{

NCB Ncb;

UCHAR uRetCode;

LANA\_ENUM lenum;

memset(&Ncb, 0, sizeof(Ncb));

Ncb.ncb\_command = NCBENUM;

Ncb.ncb\_buffer = (UCHAR\*)&lenum;

Ncb.ncb\_length = sizeof(lenum);

uRetCode = Netbios(&Ncb);

for (int i = 0; i < lenum.length; i++) {

memset(&Ncb, 0, sizeof(Ncb));

Ncb.ncb\_command = NCBRESET;

Ncb.ncb\_lana\_num = lenum.lana[i];

uRetCode = Netbios(&Ncb);

memset(&Ncb, 0, sizeof(Ncb));

Ncb.ncb\_command = NCBASTAT;

Ncb.ncb\_lana\_num = lenum.lana[i];

strcpy((char\*)Ncb.ncb\_callname, "\* ");

Ncb.ncb\_buffer = (unsigned char\*)&Adapter;

Ncb.ncb\_length = sizeof(Adapter);

uRetCode = Netbios(&Ncb);

if (uRetCode == 0) {

printf("The Ethernet Number on LANA %d is: % 02x % 02x % 02x % 02x % 02x % 02x\n",

lenum.lana[i],

Adapter.adapt.adapter\_address[0],

Adapter.adapt.adapter\_address[1],

Adapter.adapt.adapter\_address[2],

Adapter.adapt.adapter\_address[3],

Adapter.adapt.adapter\_address[4],

Adapter.adapt.adapter\_address[5]);

}

}

}

BOOL WINAPI EnumerateFunc(LPNETRESOURCE lpnr)

{

DWORD dwResult, dwResultEnum;

HANDLE hEnum;

DWORD cbBuffer = 16384;

DWORD cEntries = -1;

LPNETRESOURCE lpnrLocal;

DWORD i;

dwResult = WNetOpenEnum(RESOURCE\_GLOBALNET,

RESOURCETYPE\_ANY,

0,

lpnr,

&hEnum);

if (dwResult != NO\_ERROR)

{

printf("WnetOpenEnum failed with error %d\n", dwResult);

return FALSE;

}

lpnrLocal = (LPNETRESOURCE)GlobalAlloc(GPTR,

cbBuffer);

if (lpnrLocal == NULL) {

printf("WnetOpenEnum failed with error %d\n", dwResult);

return FALSE;

}

do

{

ZeroMemory(lpnrLocal, cbBuffer);

dwResultEnum = WNetEnumResource(hEnum,

&cEntries,

lpnrLocal,

&cbBuffer);

if (dwResultEnum == NO\_ERROR)

{

for (i = 0; i < cEntries; i++)

{

DisplayStruct(i, &lpnrLocal[i]);

if (RESOURCEUSAGE\_CONTAINER == (lpnrLocal[i].dwUsage

& RESOURCEUSAGE\_CONTAINER))

if (!EnumerateFunc(&lpnrLocal[i]))

printf("EnumerateFunc returned FALSE\n");

}

}

else if (dwResultEnum != ERROR\_NO\_MORE\_ITEMS)

{

printf("WNetEnumResource failed with error %d\n", dwResultEnum);

break;

}

} while (dwResultEnum != ERROR\_NO\_MORE\_ITEMS);

GlobalFree((HGLOBAL)lpnrLocal);

dwResult = WNetCloseEnum(hEnum);

if (dwResult != NO\_ERROR)

{

printf("WNetCloseEnum failed with error %d\n", dwResult);

return FALSE;

}

return TRUE;

}

void DisplayStruct(int i, LPNETRESOURCE lpnrLocal)

{

printf("NETRESOURCE[%d] Scope: ", i);

switch (lpnrLocal->dwScope) {

case (RESOURCE\_CONNECTED):

printf("connected\n");

break;

case (RESOURCE\_GLOBALNET):

printf("all resources\n");

break;

case (RESOURCE\_REMEMBERED):

printf("remembered\n");

break;

default:

printf("unknown scope %d\n", lpnrLocal->dwScope);

break;

}

printf("NETRESOURCE[%d] Type: ", i);

switch (lpnrLocal->dwType) {

case (RESOURCETYPE\_ANY):

printf("any\n");

break;

case (RESOURCETYPE\_DISK):

printf("disk\n");

break;

case (RESOURCETYPE\_PRINT):

printf("print\n");

break;

default:

printf("unknown type %d\n", lpnrLocal->dwType);

break;

}

printf("NETRESOURCE[%d] DisplayType: ", i);

switch (lpnrLocal->dwDisplayType) {

case (RESOURCEDISPLAYTYPE\_GENERIC):

printf("generic\n");

break;

case (RESOURCEDISPLAYTYPE\_DOMAIN):

printf("domain\n");

break;

case (RESOURCEDISPLAYTYPE\_SERVER):

printf("server\n");

break;

case (RESOURCEDISPLAYTYPE\_SHARE):

printf("share\n");

break;

case (RESOURCEDISPLAYTYPE\_FILE):

printf("file\n");

break;

case (RESOURCEDISPLAYTYPE\_GROUP):

printf("group\n");

break;

case (RESOURCEDISPLAYTYPE\_NETWORK):

printf("network\n");

break;

default:

printf("unknown display type %d\n", lpnrLocal->dwDisplayType);

break;

}

printf("NETRESOURCE[%d] Usage: 0x%x = ", i, lpnrLocal->dwUsage);

if (lpnrLocal->dwUsage & RESOURCEUSAGE\_CONNECTABLE)

printf("connectable ");

if (lpnrLocal->dwUsage & RESOURCEUSAGE\_CONTAINER)

printf("container ");

printf("\n");

printf("NETRESOURCE[%d] Localname: %S\n", i, lpnrLocal->lpLocalName);

printf("NETRESOURCE[%d] Remotename: %S\n", i, lpnrLocal->lpRemoteName);

printf("NETRESOURCE[%d] Comment: %S\n", i, lpnrLocal->lpComment);

printf("NETRESOURCE[%d] Provider: %S\n", i, lpnrLocal->lpProvider);

printf("\n");

}

int main()

{

printf("Getting the MAC Address: \n");

GetMacAddress();

printf("\nEnumerating Network Resources: \n");

LPNETRESOURCE lpnr = NULL;

if (EnumerateFunc(lpnr) == FALSE) {

printf("Call to EnumerateFunc failed\n");

return 1;

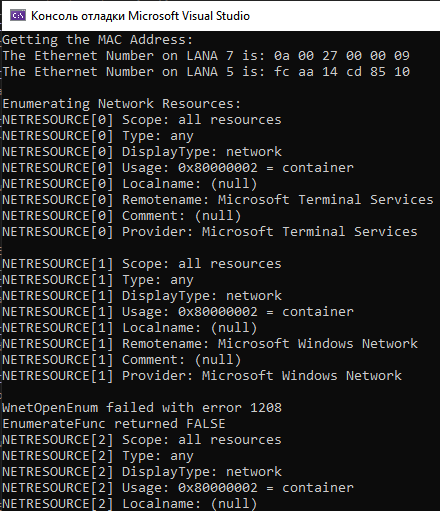
}

else

return 0;

}

Резудьтат:



**Задание № 2**

**Вариант 1**: Программа представляет простейший «чат» (от англ. chat), позволяющий обмениваться текстовыми сообщениями между двумя компьютерами.

Листинг программы с использованием протокола TCP:

**Server:**

**Header:**

#ifndef TCP\_SERVER\_H

#define **TCP\_SERVER\_H**

#include <QTcpServer>

#include <QTcpSocket>

#include <QVector>

#include <QDataStream>

#include <QTime>

*class* **TCP\_Server** : *public* QTcpServer

{

Q\_OBJECT

*public*:

**TCP\_Server**();

QTcpSocket \***socket**;

*private*:

QVector<QTcpSocket\*> **Sockets**;

QByteArray **Data**;

quint16 **nextBlockSize**;

void **SendToClient**(QString **str**);

*public* slots:

void ***incomingConnection***(qintptr **socketDescriptor**) *override*;

void **slotReadyRead**();

};

#endif *//* *TCP\_SERVER\_H*

***CPP:***

#include "tcp\_server.h"

#include <QDebug>

TCP\_Server::**TCP\_Server**()

{

*if* (*this*->listen(QHostAddress::*Any*, 11111))

{

qDebug() << "start";

}

*else*

{

qDebug() << "error";

}

nextBlockSize = 0;

}

void TCP\_Server::***incomingConnection***(qintptr **socketDescriptor**)

{

socket = *new* QTcpSocket;

socket->*setSocketDescriptor*(socketDescriptor);

connect(socket, &QTcpSocket::readyRead, *this*, &TCP\_Server::slotReadyRead);

connect(socket, &QTcpSocket::disconnected, socket, &QTcpSocket::deleteLater);

Sockets.push\_back(*socket*);

qDebug() << "client connected" << socketDescriptor;

}

void TCP\_Server::**slotReadyRead**()

{

socket = (QTcpSocket\*)sender();

QDataStream **in**(*socket*);

in.setVersion(QDataStream::*Qt\_6\_4*);

*if* (in.status() == QDataStream::*Ok*)

{

qDebug() << "read...";

*for* (;;)

{

*if* (nextBlockSize == 0)

{

qDebug() << "nextBlockSize == 0";

*if* (socket->*bytesAvailable*() < 2)

{

qDebug() << "Data < 2, break.";

*break*;

}

in >> nextBlockSize;

qDebug() << "nextBlockSize = " << nextBlockSize;

}

*if* (socket->*bytesAvailable*() < nextBlockSize)

{

qDebug() << "Do not full, break.";

*break*;

}

QString **str**;

QTime **time**;

in >> time >> str;

nextBlockSize = 0;

qDebug() << str;

SendToClient(str);

*break*;

}

}

*else*

{

qDebug() << "DataStream error";

}

}

void TCP\_Server::**SendToClient**(QString **str**)

{

Data.clear();

QDataStream **out**(&*Data*, QIODevice::*WriteOnly*);

out.setVersion(QDataStream::*Qt\_6\_4*);

out << quint16(0) << QTime::currentTime() << str;

out.device()->*seek*(0);

out << quint16(Data.size() - *sizeof*(quint16));

*for* (int **i** = 0; i < Sockets.size(); ++i)

{

Sockets[i]->write(Data);

}

}

**Main:**

#include <QCoreApplication>

#include "tcp\_server.h"

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

{

QCoreApplication **a**(*argc*, *argv*);

TCP\_Server **tcp\_s**;

*return* a.exec();

}

**Client:**

**Header:**

#ifndef MAINWINDOW\_H

#define **MAINWINDOW\_H**

#include <QMainWindow>

#include <QTcpSocket>

#include <QDataStream>

#include <QTime>

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \***parent** = *nullptr*);

~**MainWindow**();

*private* slots:

void **on\_pb\_connect\_clicked**();

void **on\_pb\_send\_clicked**();

void **on\_lineEdit\_returnPressed**();

*private*:

Ui::MainWindow \***ui**;

QTcpSocket \***socket**;

QByteArray **Data**;

void **SendToServer**(QString **str**);

quint16 **nextBlockSize**;

*public* slots:

void **slotReadyRead**();

};

#endif *//* *MAINWINDOW\_H*

***CPP:***

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \***parent**)

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

socket = *new* QTcpSocket(*this*);

connect(socket, &QTcpSocket::readyRead, *this*, &MainWindow::slotReadyRead);

connect(socket, &QTcpSocket::disconnected, socket, &QTcpSocket::deleteLater);

nextBlockSize = 0;

}

MainWindow::~**MainWindow**()

{

*delete* ui;

}

void MainWindow::**on\_pb\_connect\_clicked**()

{

socket->*connectToHost*("127.0.0.1", 11111);

}

void MainWindow::**SendToServer**(QString **str**)

{

Data.clear();

QDataStream **out**(&*Data*, QIODevice::*WriteOnly*);

out.setVersion(QDataStream::*Qt\_6\_4*);

out << quint16(0) << QTime::currentTime() << str;

out.device()->*seek*(0);

out << quint16(Data.size() - *sizeof*(quint16));

socket->write(Data);

ui->lineEdit->clear();

}

void MainWindow::**slotReadyRead**()

{

QDataStream **in**(*socket*);

in.setVersion(QDataStream::*Qt\_6\_4*);

*if* (in.status() == QDataStream::*Ok*)

{

*//* *QString* *str;*

*//* *in* *>>* *str;*

*//* *ui->textBrowser->append(str);*

*for* (;;)

{

*if* (nextBlockSize == 0)

{

*if* (socket->*bytesAvailable*() < 2)

{

*break*;

}

in >> nextBlockSize;

}

*if* (socket->*bytesAvailable*() < nextBlockSize)

{

*break*;

}

QString **str**;

QTime **time**;

in >> time >> str;

nextBlockSize = 0;

ui->textBrowser->append(time.toString() + " " + str);

}

}

*else*

{

ui->textBrowser->append("read error");

}

}

void MainWindow::**on\_pb\_send\_clicked**()

{

SendToServer(ui->lineEdit->text());

}

void MainWindow::**on\_lineEdit\_returnPressed**()

{

SendToServer(ui->lineEdit->text());

}

**Main:**

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication **a**(*argc*, *argv*);

MainWindow **w**;

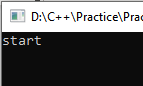
w.show();

*return* a.exec();

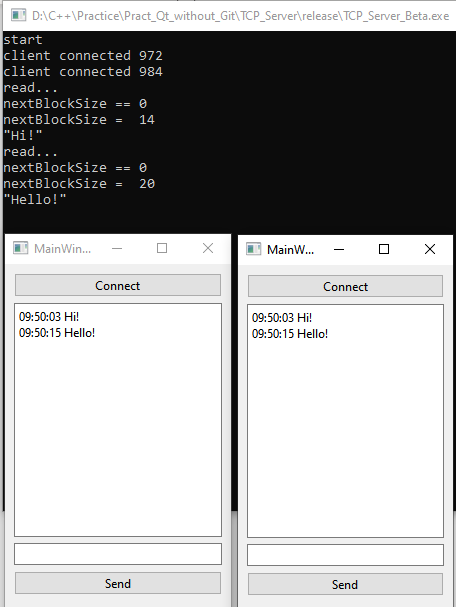
}

Результат:

Запуск сервера:



Клиенты и сервер:



Листинг программы с использованием протокола UDP:

**Header:**

#ifndef UDP\_CLIENT\_H

#define **UDP\_CLIENT\_H**

#include <QMainWindow>

#include <QUdpSocket>

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **UDP\_Client**; }

QT\_END\_NAMESPACE

*class* **UDP\_Client** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**UDP\_Client**(QWidget \***parent** = *nullptr*);

~**UDP\_Client**();

*public* slots:

void **reading\_data**();

*private* slots:

void **on\_pushButton\_clicked**();

*private*:

Ui::UDP\_Client \***ui**;

QUdpSocket \***socket**;

};

#endif *//* *UDP\_CLIENT\_H*

***CPP:***

#include "udp\_client.h"

#include "ui\_udp\_client.h"

UDP\_Client::**UDP\_Client**(QWidget \***parent**)

: QMainWindow(*parent*)

, ui(*new* Ui::UDP\_Client)

{

ui->setupUi(*this*);

socket = *new* QUdpSocket(*this*);

socket->bind(QHostAddress::*LocalHost*, 11111);

connect (socket, SIGNAL(readyRead()),*this*, SLOT(reading\_data()));

}

UDP\_Client::~**UDP\_Client**()

{

*delete* ui;

}

void UDP\_Client::**reading\_data**()

{

QHostAddress **sender**;

quint16 **sender\_port**;

*while* (socket->hasPendingDatagrams())

{

QByteArray **datagram**;

datagram.resize(socket->pendingDatagramSize());

socket->readDatagram(datagram.data(),datagram.size(),&*sender*,&*sender\_port*);

ui->textEdit->append("<font color=red>" + QString(datagram) + "</font");

qDebug() << datagram.data() << "IP: " + sender.toString() << "Port: " + QString("%1").arg(sender\_port);

}

}

void UDP\_Client::**on\_pushButton\_clicked**()

{

socket->writeDatagram(ui->lineEdit->text().toUtf8(), QHostAddress::*LocalHost*, 11112);

ui->textEdit->append("You: <font color=cyan>" + ui->lineEdit->text() + "</font");

ui->lineEdit->clear();

}

**Main:**

#include "udp\_client.h"

#include <QApplication>

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

{

QApplication **a**(*argc*, *argv*);

UDP\_Client **w**;

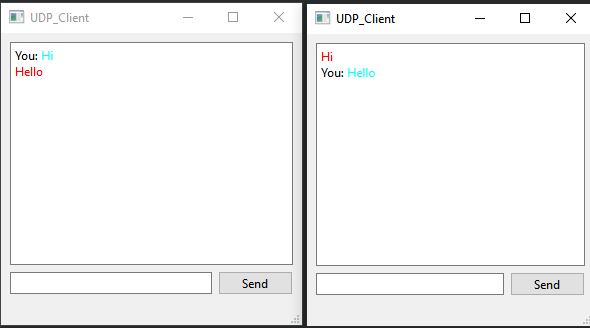
w.show();

*return* a.exec();

}

Код 2го клиента отличается номером порта.

Результат:



**Задание № 3**

**Вариант 5**: Протокол Finger.

Листинг программы:

**Server:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <Windows.h>

#include <conio.h>

#include <fstream>

#include <string.h>

using namespace std;

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

#define FINGER\_PORT 79 // Порт, который слушает сервер

// макрос для печати количества активных

// пользователей

#define PRINTNUSERS if (nclients)\

printf("%d users\n",nclients);\

else printf("No users\n");

// прототип функции, обслуживающий

// подключившихся пользователей

DWORD WINAPI WorkWithClient(LPVOID client\_socket);

// глобальная переменная – количество

// активных пользователей

int nclients = 0;

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

{

char buff[1024]; // Буфер для различных нужд

printf("Starting up server...\n");

// Шаг 1 - Инициализация Библиотеки Сокетов

if (WSAStartup(0x0202, (WSADATA\*)&buff[0]))

{

// Ошибка!

printf("Error during WSAStartup %d\n",

WSAGetLastError());

return -1;

}

// Шаг 2 - создание сокета

SOCKET mysocket;

// AF\_INET - сокет Интернета

// SOCK\_STREAM - потоковый сокет (с установкой соединения)

// 0 - по умолчанию выбирается TCP протокол

if ((mysocket = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

// Ошибка!

printf("Error during socket creation: %d\n", WSAGetLastError());

WSACleanup();

// Деиницилизация библиотеки Winsock

return -1;

}

// Шаг 3 связывание сокета с локальным адресом

sockaddr\_in local\_addr;

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

local\_addr.sin\_port = htons(FINGER\_PORT);

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

// сервер принимает подключения

// на все IP-адреса

// вызываем bind для связывания

if (bind(mysocket, (sockaddr\*)&local\_addr, sizeof(local\_addr)))

{

// Ошибка

printf("Binding error: %d\n", WSAGetLastError());

closesocket(mysocket); // закрываем сокет!

WSACleanup();

return -1;

}

// Шаг 4 ожидание подключений

// размер очереди – 0x100

if (listen(mysocket, 20))

{

// Ошибка

printf("Error during listen: %d\n", WSAGetLastError());

closesocket(mysocket);

WSACleanup();

return -1;

}

printf("Server started successfully. Waiting for clients...\n");

// Шаг 5 извлекаем сообщение из очереди

SOCKET client\_socket[20]; // сокет для клиента

sockaddr\_in client\_addr; // адрес клиента

// (заполняется системой)

// функции accept необходимо передать размер

// структуры

int client\_addr\_size = sizeof(client\_addr);

// цикл извлечения запросов на подключение из

// очереди

/\* accept - держит управление и не даёт циклу вращаться

(то есть не даёт потоку- нити выполняться вообще)

пока не поступит очередной запрос на соединение\*/

while ((client\_socket[nclients] = accept(mysocket, (sockaddr\*)&client\_addr, &client\_addr\_size)))

{

nclients++; // увеличиваем счетчик

// подключившихся клиентов

// пытаемся получить имя хоста

HOSTENT\* hst;

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

// вывод сведений о клиенте

printf("+%s [%s] User connected!\n", (hst) ? hst->h\_name : "", inet\_ntoa(client\_addr.sin\_addr));

PRINTNUSERS

// Вызов нового потока для обслужвания клиента

// Да, для этого рекомендуется использовать

// \_beginthreadex но, поскольку никаких вызов

// функций стандартной Си библиотеки поток не

// делает, можно обойтись и CreateThread

DWORD thID;

CreateThread(NULL, NULL, WorkWithClient, client\_socket, NULL, &thID);

/\* запускаем функции прослушивания чужих сообщений

нам нужен именно отдельный поток, потому что recv()

забирает управление и не позволит послать клиенту

сообщение пока от этого самого ("родного" для данной

нити сервера) клиента что-нибудь не придёт \*/

//CreateThread(NULL,NULL, CheckCommonBuffer, &client\_socket,NULL,&thID);

}

return 0;

}

// Эта функция создается в отдельном потоке и

// обсуживает очередного подключившегося клиента

// независимо от остальных

DWORD WINAPI WorkWithClient(LPVOID sockets)

{

SOCKET my\_sock = ((SOCKET\*)sockets)[nclients - 1];

char buff[1024] = "123124";

#define CRLF "\r\n"

#define HELLO\_STR "Connection established. \n You can use one of the following queries:\n\

1) Send an '\\all' message to recieve information about all users\n\

2) Send a message in format ( [User name | User login] ) to recieve information about specific user\r\n"

#define LOGIN\_STR "User login: "

#define NAME\_STR "User name: "

#define GROUP\_STR "User group: "

#define PHONE\_STR "User phone: "

#define USERINFO\_STR "User information file:\r\n"

#define NOUSERINFO\_STR "User don't have an information file\r\n"

#define USERNOTFOUND\_STR "There is no such user\r\n"

#define ECHO "foobar"

// отправляем клиенту приветствие

send(my\_sock, HELLO\_STR, sizeof(HELLO\_STR), 0);

//Принимаем запрос от клиента и отправляем соответствующую информацию в ответ

int bytes\_recv;

if ((bytes\_recv = recv(my\_sock, buff, sizeof(buff), 0)) != SOCKET\_ERROR) {

//Открыть файл с информацией о пользователях

ifstream users;

users.open("USERS.txt", ios::in);

if (!users.is\_open())

{

cout << "\nFile is not open\n";

}

cout << "Socket: " << my\_sock << " Message: " << buff << endl; // печатает сообщения всех клиентов в консоли сервера

//Если Запрос вида \r\n, то по стандарту возвращается информация о всех пользователях

if (strcmp(CRLF, buff) == 0)

{

while (users.getline(buff, sizeof(buff)))

{

strcat(buff, CRLF);

send(my\_sock, buff, strlen(buff) + 1, 0);

recv(my\_sock, buff, sizeof(buff), 0);

}

}

else //Любой другой запрос расценивается как запрос о конкретном пользователе (по логину или по имени)

{

bool userFound = false;

string tmp, userLogin, userName;

char userInfo[256], tmpStr[128];

//От запроса откидываются последние два символа, которые всегда \r\n по требованию стандарта

buff[strlen(buff) - 2] = '\0';

//Обходим весь файл с пользователями

while (users.getline(userInfo, sizeof(userInfo)))

{

userLogin = strtok(userInfo, " \_");

userName = strtok(NULL, " \_");

//Если запрос совпал с именем либо с логином пользователя, то отправляем информацию о нём

if (strcmp(buff, userLogin.c\_str()) == 0 || strcmp(buff, userName.c\_str()) == 0)

{

//Устанавливаем флаг что есть совпадение

userFound = true;

//Отправляем строку с логином в удобочитаемом виде

tmp = LOGIN\_STR + userLogin + "\r\n";

strcpy(tmpStr, tmp.c\_str());

send(my\_sock, tmpStr, strlen(tmpStr) + 1, 0);

recv(my\_sock, buff, sizeof(buff), 0);

//Отправляем строку с именем и фамилией в удобочитаемом виде

tmp = NAME\_STR + userName + " " + strtok(NULL, " ") + "\r\n";

strcpy(tmpStr, tmp.c\_str());

send(my\_sock, tmpStr, strlen(tmpStr) + 1, 0);

recv(my\_sock, buff, sizeof(buff), 0);

//Отправляем строку с номером группы в удобочитаемом виде

//тут костыльно, потому что конкатенация плюсовых строк почему-то ругается,

//если в выражении только сишные строки, поэтому тут ещё и tmp="" засунуто

tmp = "";

tmp = tmp + GROUP\_STR + strtok(NULL, " ") + "\r\n";

strcpy(tmpStr, tmp.c\_str());

send(my\_sock, tmpStr, strlen(tmpStr) + 1, 0);

recv(my\_sock, buff, sizeof(buff), 0);

//Отправляем строку с номером. Тут такой же костыль

tmp = "";

tmp = tmp + PHONE\_STR + strtok(NULL, " ") + "\r\n";

strcpy(tmpStr, tmp.c\_str());

send(my\_sock, tmpStr, strlen(tmpStr) + 1, 0);

recv(my\_sock, buff, sizeof(buff), 0);

//У каждого пользователя может быть свой информационный файл.

//Называться он должен логин\_пользователя.txt

//Проверяем существует ли он и если да, то перекидываем построчно клиенту

//если нет, то отправляем соответствующее сообщение

ifstream userInfoFile;

userInfoFile.open(userLogin + ".txt", ios::in);

if (userInfoFile.is\_open())

{

send(my\_sock, USERINFO\_STR, sizeof(USERINFO\_STR), 0);

recv(my\_sock, buff, sizeof(buff), 0);

while (userInfoFile.getline(buff, sizeof(buff)))

{

strcat(buff, CRLF);

send(my\_sock, buff, strlen(buff) + 1, 0);

recv(my\_sock, buff, sizeof(buff), 0);

}

}

else

{

send(my\_sock, NOUSERINFO\_STR, sizeof(NOUSERINFO\_STR), 0);

recv(my\_sock, buff, sizeof(buff), 0);

}

}

}

//Если совпадений не найдено, то отправляем соответствующее сообщение

if (!userFound)

{

send(my\_sock, USERNOTFOUND\_STR, sizeof(USERNOTFOUND\_STR), 0);

recv(my\_sock, buff, sizeof(buff), 0);

}

}

}

nclients--; // уменьшаем счетчик активных клиентов

printf("-disconnecting\n"); PRINTNUSERS

// закрываем сокет

closesocket(my\_sock);

return 0;

}

**Client:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <Windows.h>

#include <conio.h>

#include <string.h>

#define PORT 79

#define SERVERADDR "127.0.0.1"

#define CRLF "\r\n"

#define ALL "\\all"

#define ECHO "foobar"

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

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

{

char buff[1024];

//printf("FINGER TCP CLIENT\n");

// Шаг 1 - инициализация библиотеки Winsock

if (WSAStartup(0x202, (WSADATA\*)&buff[0]))

{

printf("WSAStart error %d\n", WSAGetLastError());

return -1;

}

// Шаг 2 - создание сокета

SOCKET my\_sock;

my\_sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (my\_sock < 0)

{

printf("Socket() error %d\n", WSAGetLastError());

WSACleanup();

return -1;

}

// Шаг 3 - установка соединения

// заполнение структуры sockaddr\_in

// указание адреса и порта сервера

sockaddr\_in dest\_addr;

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

dest\_addr.sin\_port = htons(PORT);

dest\_addr.sin\_addr.s\_addr = inet\_addr(SERVERADDR);

HOSTENT\* hst;

// адрес сервера получен – пытаемся установить соединение

if (connect(my\_sock, (sockaddr\*)&dest\_addr, sizeof(dest\_addr)))

{

printf("Connect error %d\n", WSAGetLastError());

WSACleanup();

return -1;

}

recv(my\_sock, buff, sizeof(buff), 0);

printf(buff);

scanf("%s", buff);

if (strcmp(buff, ALL) == 0)

{

send(my\_sock, CRLF, sizeof(CRLF), 0);

}

else

{

strcat(buff, CRLF);

send(my\_sock, buff, strlen(buff), 0);

}

while (recv(my\_sock, buff, sizeof(buff), 0))

{

printf(buff);

send(my\_sock, ECHO, sizeof(ECHO), 0);

}

printf("Server closed the connection. Press any key to exit.\n", WSAGetLastError());

system("pause");

closesocket(my\_sock);

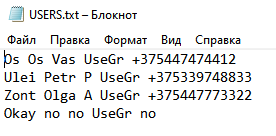
WSACleanup();

return 1;

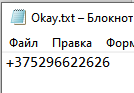
}

Для работы нам необходимы 2 текстовых файла:

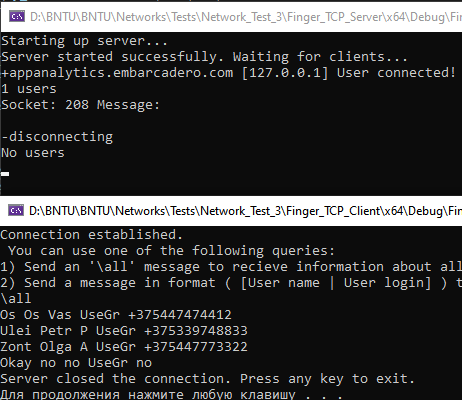
USERS.txt:

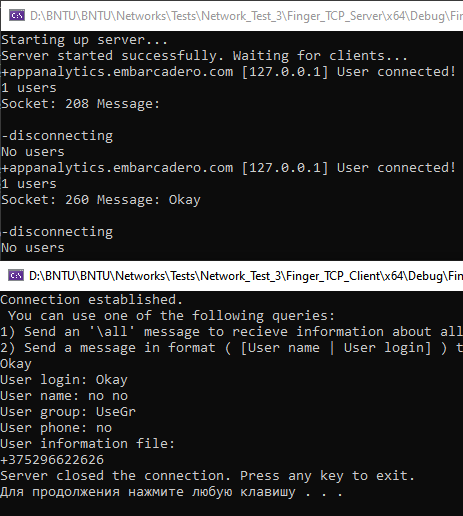


И например, Okay.txt:



Результаты:





**Задание №4**

Вариант 1 Написать программу, реализующую функции HTTP-сервера версии 1.0. В обязательном порядке должны поддерживаться следующие виды запросов: GET, POST, HEAD, а так же наиболее распространенные коды ответов. Сервер конфигурируется на определенный каталог, где расположены html-файлы и другие подкаталоги. В главном окне сервера расположено поле типа memo, в котором отображается весь протокол общения HTTP клиента с HTTP сервером, например: GET /index.html HTTP/1.О Connection: Keep-Alive User-Agent: Mozilla/4.05 (WinNT; 1) 32 Host: www.ora.com Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, \*/\* HTTP/1.0 200 Document follows Date: Fri, 20 Jan 1998 08:17:58 GMT Server: Apache/1.2.6 Last-modified: Mon, 20 Jun 1997 21:53:08 GMT Content-type: text/html Content-length: 2482 .....

Тестирование и подача программы-сервера производится при помощи Web-броузера.

Листинг программы:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <netdb.h>

#include <time.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <unistd.h>

// http://localhost:8000

#define SERVER\_PORT "8000"

#define MAX\_CONNECTION 1000

typedef enum

{

eHTTP\_UNKNOWN = 0

,eHTTP\_CONNECT

,eHTTP\_DELETE

,eHTTP\_GET

,eHTTP\_HEAD

,eHTTP\_OPTIONS

,eHTTP\_PATCH

,eHTTP\_POST

,eHTTP\_PUT

,eHTTP\_TRACE

}eHTTPMethod;

typedef struct

{

eHTTPMethod type;

char path[255];

}sHTTPHeader;

void \*get\_client\_addr(struct sockaddr \*);

int create\_socket(const char \*);

void http\_request(int);

void parse\_http\_request(const char\*, sHTTPHeader \*);

void send\_message(int, const char\*);

void send\_404(int);

int main()

{

int sock;

sock = create\_socket(SERVER\_PORT);

if(sock < 0)

{

fprintf(stderr, "error create socket\n");

return -1;

}

printf("server created!\n");

struct sockaddr\_storage client\_addr;

int client\_d;

//char client\_ip

while(1)

{

socklen\_t s\_size = sizeof(client\_addr);

client\_d = accept(sock, (struct sockaddr\*)&client\_addr, &s\_size);

if(client\_d == -1)

{

fprintf(stderr, "error accept\n");

return -1;

}

char ip[INET6\_ADDRSTRLEN];

inet\_ntop(client\_addr.ss\_family, get\_client\_addr((struct sockaddr \*)&client\_addr), ip, sizeof ip);

printf("server: got connection from %s\n", ip);

// read

http\_request(client\_d);

close(client\_d);

}

return 0;

}

void \*get\_client\_addr(struct sockaddr \*sa)

{

if (sa->sa\_family == AF\_INET)

{

return &(((struct sockaddr\_in\*)sa)->sin\_addr);

}

return &(((struct sockaddr\_in6\*)sa)->sin6\_addr);

}

int create\_socket(const char \*apstrPort)

{

struct addrinfo hints;

struct addrinfo \*servinfo;

struct addrinfo \*p;

memset(&hints, 0, sizeof(hints));

// IPv4 or IPv6

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_flags = AI\_PASSIVE;

int r = getaddrinfo(NULL, apstrPort, &hints, &servinfo);

if( r != 0)

{

fprintf(stderr, "error getaddrinfo()\n");

return -1;

}

int sock;

int yes = 1;

for(p = servinfo; p != NULL; p = p->ai\_next)

{

sock = socket(p->ai\_family, p->ai\_socktype, p->ai\_protocol);

if(sock == -1)

continue;

if (setsockopt(sock, SOL\_SOCKET, SO\_REUSEADDR, &yes, sizeof(int)) == -1)

{

fprintf(stderr, "error setsockopt\n");

close(sock);

freeaddrinfo(servinfo); // all done with this structure

return -2;

}

if(bind(sock, p->ai\_addr, p->ai\_addrlen) == -1)

{

close(sock);

continue;

}

break;

}

freeaddrinfo(servinfo); // all done with this structure

if(p == NULL)

{

fprintf(stderr, "failed to find address\n");

return -3;

}

if(listen(sock, MAX\_CONNECTION) == -1)

{

fprintf(stderr, "error listen\n");

return -4;

}

return sock;

}

void http\_request(int aSock)

{

const int request\_buffer\_size = 65536;

char request[request\_buffer\_size];

int bytes\_recvd = recv(aSock, request, request\_buffer\_size - 1, 0);

if (bytes\_recvd < 0)

{

fprintf(stderr, "error recv\n");

return;

}

request[bytes\_recvd] = '\0';

printf("request:\n%s\n",request);

sHTTPHeader req;

parse\_http\_request(request, &req);

if(req.type == eHTTP\_GET)

{

send\_message(aSock, "sensor 1: 10<br> sensor 2: 20<br><a href=\"http://cppprosto.blogspot.com/2017/09/blog-post\_23.html\">external</a><br><a href=\"internal\">internal</a>");

}

else

{

send\_404(aSock);

}

}

void parse\_http\_request(const char \*apstrRequest, sHTTPHeader \*apHeader)

{

int type\_length = 0;

char type[255] = {0};

int index = 0;

apHeader->type = eHTTP\_UNKNOWN;

sscanf(&apstrRequest[index], "%s", type);

type\_length = strlen(type);

if(type\_length == 3)

{

if(type[0] == 'G' && type[1] == 'E' && type[2] == 'T')

apHeader->type = eHTTP\_GET;

index += type\_length + 1;

sscanf(&apstrRequest[index], "%s", apHeader->path);

}

}

void send\_message(int aSock, const char \*apstrMessage)

{

char buffer[65536] = { 0 };

strcat(buffer, "HTTP/1.1 200 OK\n\n");

strcat(buffer, "<h1>");

strcat(buffer, apstrMessage);

strcat(buffer, "</h1>");

int len = strlen(buffer);

send(aSock, buffer, len, 0);

}

void send\_404(int aSock)

{

const char \*buffer = "HTTP/1.1 404 \n\n";

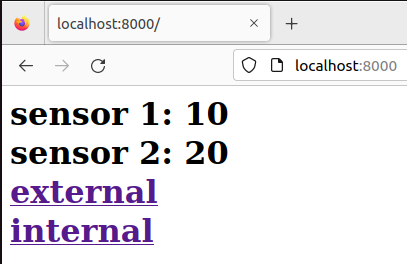
int len = strlen(buffer);

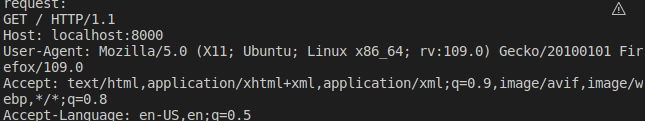
send(aSock, buffer, len, 0);

}

Результат:







|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |