МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»
(БГТУ им. В.Г.Шухова)

Лабораторная работа №8
дисциплина «Сети ЭВМ и телекомуникации»
по теме «Программирование протокола HTTP»

Выполнил: студент группы ВТ-31 Макаров Д.С.
Проверил: Федотов Е.А.

Белгород 2020

Лабораторная работа №8

«Программирование протокола HTTP»

Цель работы:изучить протокол HTTP и составить программу согласно заданию.

Вариант 6

Содержание отчета

1. Краткие теоретические сведения.
2. Основные функции API, использованные в данной работе.
3. Разработка программы. Блок-схемы программы.
4. Анализ функционирования разработанных программ.
5. Выводы.
6. Тексты программ. Скриншоты программ.

Ход работы

1. **Краткие теоретические сведения.**

HTTP (Hyper Text Transfer Protocol – протокол передачи гипертекста) – протокол прикладного уровня стека протоколов TCP/IP, предназначенный для передачи данных по сети с использованием транспортного протокола TCP. Текущая версия протокола HTTP v1.1, его спецификация приводится в документе RFC 2616 Протокол HTTP может использоваться также в качестве «транспорта» для других протоколов прикладного уровня, таких как SOAP или XML-RPC. Основой HTTP является технология «клиент-сервер». HTTP-клиенты отсылают HTTP-запросы, которые содержат метод, обозначающий потребность клиента. Также такие запросы содержат универсальный идентификатор ресурса, указывающий на желаемый ресурс. Обычно такими ресурсами являются хранящиеся на сервере файлы. По умолчанию HTTP-запросы передаются на порт 80 HTTP-сервер отсылает коды состояния, сообщая, успешно ли выполнен HTTP-запрос или же нет.

Унифицированный идентификатор ресурса представляет собой сочетание унифицированного указателя ресурса (Uniform Resource Locator, URL) и унифицированного имени ресурса (Uniform Resource Name, URN).

1. **Разработка программы.**

   В ходе работы было разработано консольное приложение HTTP сервер на языке C.

   Реализованы методы со стороны сервера:

- GET (только текстовые файлы).
- HEAD.

2. **Анализ функционирования разработанных программ.**

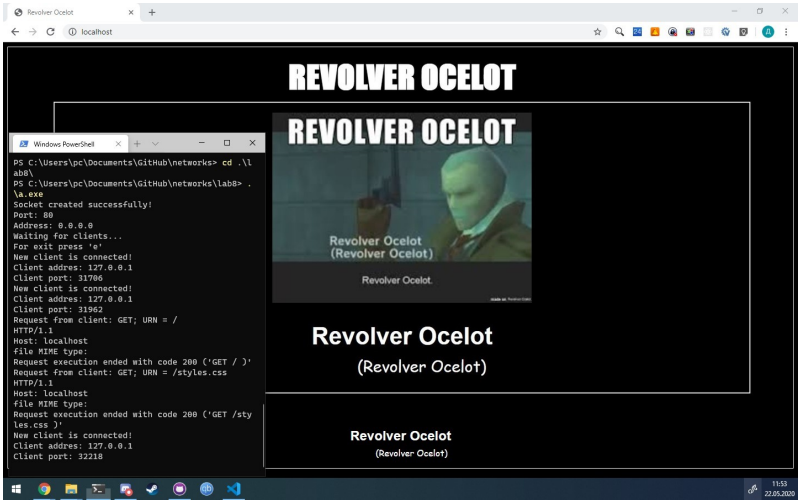

Рис. 1: Пример работы программы

3. **Выводы.**

В данной лабораторной работе была реализован простейший HTTP сервер, написанный с использованием библиотека Winsock. Библиотека Winsock позволяет быстро и удобно писать сетевые приложения для ОС Windows.

1. **Тексты программ. Скриншоты программ.**

Тексты программ см. в приложении.

7. **Контрольные вопросы**

1. Как расшифровывается аббревиатура HTTP?
   HTTP - HyperText Transfer Protocol — «протокол передачи гипертекста»
2. Какой уровень занимает протокол в стеке TCP/IP?
   прикладной
3. На какой технологии построен протокол HTTP?
   клиент-сервер
4. Какие преимущества протокола HTTP?
   Низкая сложность расширения протокола путем добавления новых заголовков, несовместимые серверы и клиенты будут их просто игнорировать.

5. Какие недостатки протокола HTTP?

   Отсутствие возможности получить весь список файлов на сервере как в FTP, избыточность.

6. Какие методы существуют в протоколе HTTP?

- GET
- POST
- PUT
- DELETE
- HEAD
- TRACE
- OPTIONS

7. Какие нововведения содержит версия HTTP 1.1?

- TCP соединение остается открытым после ответа на клиентский запрос, позволяя отправлять несколько запросов за 1 соединение
- Клиент обязан посылать информацию о имени хоста к которому он обращается.

8. Какова структура протокола HTTP? Охарактеризуйте каждый элемент

   - Стартовая строка - тип сообщения
   - Заголовки - фрагмент описывающий тело сообщения и прочие сведения
   -
   - Тело - данные сообщения

9. Какие существуют классы кодов состояния?

- 1xx - информационные состояния
- 2xx - состояние успеха
- 3xx - состояние перенаправления
- 4xx - состояние ошибок клиента
- 5xx - состояние ошибок сервера

10. Какие существуют группы заголовков HTTP?

- Основные заголовки
- Заголовки запроса
- Заголовки ответа
- Заголовки сущности

11. Что такое cookie-файлы? Для чего они используются?

Используются для поддержки неанонимного доступа в HTTP.

12. Что такое HTTP referrer? Для чего он используется?

Поле используется для возобновления прерванной сессии.

Приложение

Содержимое файла mailclient.cpp

```
#include "mail_client.h"

MailClient::MailClient(){
    connect(&loginButton,SIGNAL(pressed()),this,SLOT(loginButtonPush()));
    connect(&updateButton,SIGNAL(pressed()),this,SLOT(updateButtonPush()));
    connect(&deleteButton,SIGNAL(pressed()),this,SLOT(delButtonPush()));

    ↪  connect(&mailList,SIGNAL(itemClicked(QListWidgetItem)),this,SLOT(listItemClicked(QListWidgetIte
    generalLayout.addLayout(&configLayout);
        configLayout.addLayout(&formLayout);
            serverAdressLabel.setText("Адрес почтового сервера: ");
            adressLabel.setText("Адрес электронной почты: ");
            passwordLabel.setText("Пароль: ");

            formLayout.addRow(&serverAdressLabel,&serverAdressLine);
            formLayout.addRow(&adressLabel,&adressLine);
            formLayout.addRow(&passwordLabel,&passwordLine);
        loginButton.setText("Подключиться");
        configLayout.addWidget(&loginButton);
    generalLayout.addLayout(&emailListLayout);
        updateButton.setText("Обновить список");
        emailListLayout.addWidget(&mailList);
        emailListLayout.addWidget(&updateButton);
    generalLayout.addLayout(&emailContentLayout);
        deleteButton.setText("Удалить сообщение");
        emailContentLayout.addWidget(&emailContent);
        emailContentLayout.addWidget(&deleteButton);
    this->setLayout(&generalLayout);
};


void MailClient::loginButtonPush(){
    QTextStream log_stream(&log);
    QString request;
    QString response;
    if (this->connected){
        request = "QUIT\r\n";
        socket.write(request.toLocal8Bit());
        socket.waitForBytesWritten();
        qDebug() << "[Client]: " << request;
        log_stream << "[Client]: " << request;
        socket.waitForReadyRead();
        response = socket.readAll();
        qDebug() << "[Server]: " << response;
        log_stream << "[Server]: " << response;
        socket.close();
        loginButton.setText("Подключение");
        this->connected = false;
        QMessageBox msgBox;
        msgBox.setText("Лог сессии");
        msgBox.setDetailedText(log);msgBox.setStandardButtons(QMessageBox::Ok);
        msgBox.setDefaultButton(QMessageBox::Ok);
        msgBox.exec();
    }else{
        qDebug() << "Connecting...";
        socket.connectToHost(serverAdressLine.text(), 110);
```

```
        if(socket.waitForConnected(5000)){
            qDebug() << "Connected!";
            socket.waitForReadyRead();
            response = socket.readAll();
            qDebug() << "[Server]: " << response;
            log_stream << "[Server]: " << response;

                request = "USER " + adressLine.text() + "\r\n";
                socket.write(request.toLocal8Bit());
                socket.waitForBytesWritten();
                qDebug() << "[Client]: " << request;
                log_stream << "[Client]: " << request;

                socket.waitForReadyRead();
                response = socket.readAll();
                qDebug() << "[Server]: " << response;
                log_stream << "[Server]: " << response;

                request = "PASS " + passwordLine.text() + "\r\n";
                socket.write(request.toLocal8Bit());
                socket.waitForBytesWritten();
                qDebug() << "[Client]: " << request;
                log_stream << "[Client]: " << request;

                socket.waitForReadyRead();
                response = socket.readAll();
                qDebug() << "[Server]: " << response;
                log_stream << "[Server]: " << response;

                loginButton.setText("Отключиться");
                this->connected = true;

            }else{
                qDebug() << "Not connected!";
            }
        }
};
void MailClient::updateButtonPush(){
    mailList.clear();
    if (!connected){
        return;
    }
    QTextStream log_stream(&log);
    QString request;
    QString response;

    request = "LIST\r\n";
    socket.write(request.toLocal8Bit());
    socket.waitForBytesWritten();
    qDebug() << "[Client]: " << request;
    log_stream << "[Client]: " << request;

    socket.waitForReadyRead();
//    qDebug() << "Ready to receive " << socket.bytesAvailable() << " bytes\n";
    response = socket.readAll();
    qDebug() << "[Server]: " << response;
    log_stream << "[Server]: " << response;

    int messages = response.split(" ")[1].toInt();
    QStringList items;
```

```
    while (items.length() < messages){
        socket.waitForReadyRead();
        response = socket.readAll();
        qDebug() << "[Server]: " << response;
        log_stream << "[Server]: " << response;
        items.append(response.split("\r\n"));
    }
    for (auto item: items){
        if (item != "." && item != ""){
            mailList.addItem(item + " байт");
        }
    }
    response = socket.readAll();
    qDebug() << "[Server]: " << response;
    log_stream << "[Server]: " << response;
};
void MailClient::delButtonPush(){
    QTextStream log_stream(&log);
    QString request;
    QString response;
    int index = curr_msg;
    request = "DELE " + QString::number(index) + "\r\n";
    socket.write(request.toLocal8Bit());
    socket.waitForBytesWritten();
    qDebug() << "[Client]: " << request;
    log_stream << "[Client]: " << request;

    socket.waitForReadyRead();
//    qDebug() << "Ready to receive " << socket.bytesAvailable() << " bytes\n";
    response = socket.readAll();
    qDebug() << "[Server]: " << response;
    log_stream << "[Server]: " << response;
    curr_msg = 0;
};
void MailClient::listItemClicked(QListWidgetItem *item){
    QTextStream log_stream(&log);
    QString request;
    QString response;
    int index = item->text().split(" ")[0].toInt();
    curr_msg = index;
    request = "RETR " + QString::number(index) + "\r\n";
    socket.write(request.toLocal8Bit());
    socket.waitForBytesWritten();
    qDebug() << "[Client]: " << request;
    log_stream << "[Client]: " << request;

    socket.waitForReadyRead();
    response = socket.readAll();
    qDebug() << "[Server]: " << response;
    log_stream << "[Server]: " << response;

    QString message;
    QTextStream message_stream(&message);
    int read = 0;
    int all = response.split(" ")[1].toInt();
    while (socket.waitForReadyRead(1000)){
        read += socket.bytesAvailable();
        response = socket.readAll();
        qDebug() << read << "/" << all;
        message_stream << response;
```

```
    }

    emailContent.setText(message);
};
```

Содержимое файла mailclient.h

```
#pragma once
#include <QtWidgets>

#include <QFileDialog>
#include <QDebug>
#include <QMessageBox>
#include <QTcpSocket>

class MailClient : public QWidget{
    Q_OBJECT
    public:
        MailClient();
    private:
        QString log;
        bool connected;
        QTcpSocket socket;
        int curr_msg = 0;

        QHBoxLayout generalLayout;
            QVBoxLayout configLayout;
                QFormLayout formLayout;
                    QLabel passwordLabel,adressLabel,serverAdressLabel;
                    QLineEdit passwordLine,adressLine,serverAdressLine;
                QPushButton loginButton;
            QVBoxLayout emailListLayout;
                QListWidget mailList;
                QPushButton updateButton;
            QVBoxLayout emailContentLayout;
                QTextEdit emailContent;
                QPushButton deleteButton;
    public slots:
        void loginButtonPush();
        void updateButtonPush();
        void delButtonPush();
        void listItemClicked(QListWidgetItem *item);
};
```

Содержимое файла main.cpp

```
#include <QApplication>
#include "mail_client.h"
int main(int argc, char *argv[]){
    QApplication app(argc,argv);
    MailClient client;
    client.show();
    return app.exec();
}
```