МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

Отчет   
по лабораторной работе №3  
на тему: « Взаимодействие прикладных программ с помощью

протокола передачи данных FTP»  
по дисциплине «Компьютерные сети»

Выполнили: Марочкин М.А. Шифр: 170584   
 Шорин В.Д. Шифр: 171406  
ИПАИТ  
Направление: 09.03.04 «Программная инженерия»  
Группа: 71-ПГ  
Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Отметка о зачете:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

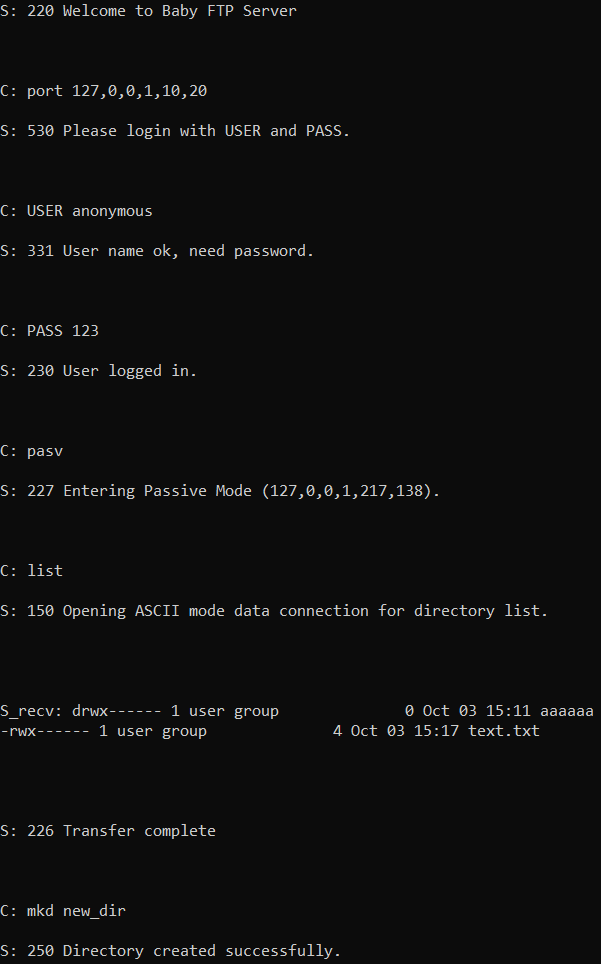
Дата «\_\_\_\_» \_\_\_\_\_\_ 2018 г.

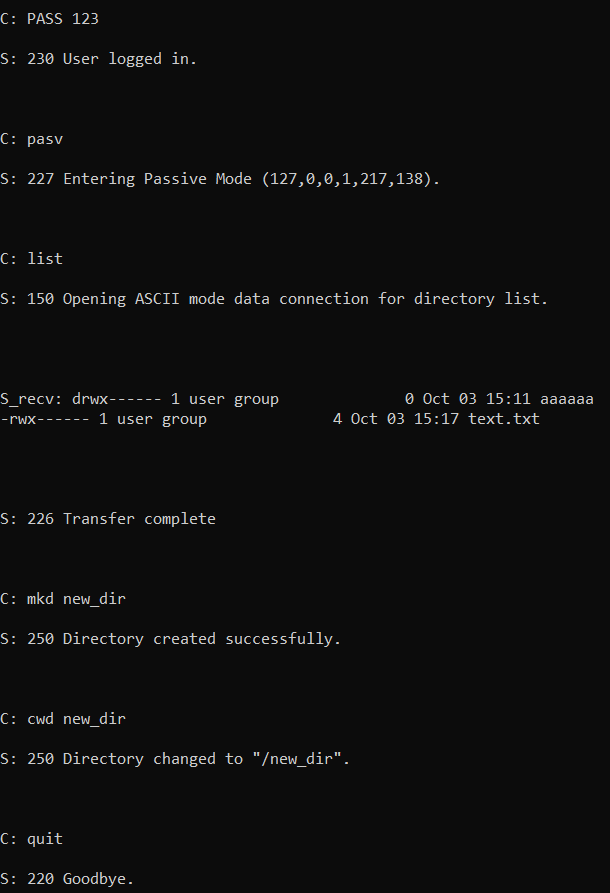
Орел, 2018 г.

**Цель работы:**

Изучение принципов организации взаимодействия прикладных программ с помощью протокола передачи данных FTP и приобретение практических навыков создания клиентских приложений, использующих протокол FTP.

**Выполнение работы**

****

****

****

**Контрольные вопросы:**

**1. Сколько логических соединений необходимо для работы протокола**

**FTP? Каким образом они используются?**

Команды FTP определяют параметры канала передачи данных и самого процесса передачи. Они также определяют и характер работы с удаленной и локальной файловыми системами. Сессия управления инициализирует канал передачи данных. При организации канала передачи данных последовательность действий другая, отличная от организации канала управления. В этом случае сервер инициирует обмен данными в соответствии с согласованными в сессии управления параметрами. Канал данных устанавливается для того же узла, что и канал управления, через который ведется настройка канала данных. Канал данных может быть использован как для приема, так и для передачи данных.

**2. Какие стадии можно выделить в FTP-сессии?**

Алгоритм работы протокола FTP состоит в следующем:

1. Сервер FTP использует в качестве управляющего соединение на TCP порт 21, который всегда находится в состоянии ожидания соединения со стороны пользователя FTP.

2. После того как устанавливается управляющее соединение модуля «Интерпретатор протокола пользователя» с модулем сервера – «Интерпретатор протокола сервера», пользователь (клиент) может отправлять на сервер команды. FTP-команды определяют параметры соединения передачи данных: роль участников соединения (активный или пассивный), порт соединения (как для модуля «Программа передачи данных пользователя», так и для модуля «Программа передачи данных сервера»), тип передачи, тип передаваемых данных, структуру данных и управляющие директивы, обозначающие действия, которые пользователь хочет совершить (например, сохранить, считать, добавить или удалить данные, или файл и другие).

3. После того как согласованы все параметры канала передачи данных, один из участников соединения (пассивный модуль) становится в режим ожидания открытия соединения на определенный порт. После этого другая сторона (активный модуль) открывает соединение на указанный порт и начинается обмен данными.

4. После окончания передачи данных соединение между «Программой передачи данных сервера» и «Программой передачи данных пользователя» закрывается, но управляющее соединение «Интерпретатора протокола сервера» и «Интерпретатора протокола пользователя» остается открытым. Пользователь, не закрывая сессии FTP, может еще раз открыть канал передачи данных.

**3. В чем заключается особенность работы протокола FTP в активном режиме?**

Установление соединения передачи данных при активном режиме работы происходит следующим образом:

1. Клиент создает сокет на выбранном им порте Р и активизирует его (переводит в режим ожидания соединения).

2. Клиент направляет серверу по управляющему соединению команду PORT (описание приведено ниже), в которой указывает свой IP-адрес и выбранный для передачи данных порт Р.

3. В случае успешного получения и обработки команды PORT сервер по управляющему соединению отправляет клиенту положительный отклик. Затем сервер пытается соединиться с портом Р клиента со своего локального порта данных А (может быть выбран сервером произвольно).

4. Клиент обнаруживает событие соединения на порт Р и начинает процесс передачи данных с использованием управляющего соединения для отправки команд FTP-сервиса и соединения передачи данных для получения либо отправки данных.

**4. В чем заключается особенность работы протокола FTP в пассивном режиме?**

Установление соединения передачи данных при пассивном режиме работы происходит следующим образом:

1. Клиент отправляет по управляющему соединению команду PASV, указывающую серверу о намерении клиента работать в пассивном режиме.

2. В случае успешного получения и обработки команды PASV сервер создает сокет на выбранном им порте Р, активизирует его (переводит его в режим ожидания соединения) и отправляет клиенту по управляющему соединению отклик, в котором указываются параметры соединения (IP-адрес сервера порт Р).

3. Клиент, получив положительный отклик с параметрами соединения, создает на произвольном порте А сокет и активизирует его, то есть пытается соединиться с портом Р сервера.

4. В случае успешного установления соединения клиент начинает процесс передачи данных с использованием управляющего соединения для отправки команд FTP-сервиса и соединения передачи данных для получения либо отправки данных.

**5. Какая сторона соединения отвечает за закрытие канала передачи данных? В каких случаях это происходит?**

Как правило, сервер FTP ответственен за открытие и закрытие канала передачи данных. Сервер FTP должен самостоятельно закрыть канал передачи данных в следующих случаях:

1. Сервер закончил передачу данных в формате, который требует закрытия соединения.

2. Сервер получил от пользователя команду «прервать соединение».

3. Пользователь изменил параметры порта передачи данных.

4. Было закрыто управляющее соединение.

5. Возникли ошибки, при которых невозможно возобновить передачу

данных.

**6. Опишите структуру команды и ответа протокола FTP.**

Все команды протокола FTP отправляются «Интерпретатором протокола пользователя» в текстовом виде – по одной команде в строке. Каждая строка команды – идентификатор и аргументы – заканчиваются символами CRLF. Имя команды отделяется от аргумента символом пробела. Обработчик команд возвращает код обработки каждой команды, состоящий из трех цифр. Коды обработки составляют определенную иерархическую структуру и, как правило, определенная команда может возвратить только определенный набор кодов. За кодом обработки команды следует символ пробела и текст пояснения.

**7. На какие группы можно разбить команды протокола FTP? Каково назначение команд каждой из групп?**

Команды протокола FTP, которыми обмениваются «Интерпретатор протокола сервера» и «Интерпретатор протокола пользователя», можно разделить на три группы.

1. Команды управления доступом к системе обеспечивают авторизацию пользователя в системе, выход из нее и настройку некоторых текущих параметров соединения.

2. Команды управления потоком данных устанавливают параметры передачи данных. Все параметры, описываемые этими командами, имеют значение по умолчанию, поэтому команды управления потоком используются только тогда, когда необходимо изменить значение параметров передачи, используемых по умолчанию. Команды управления потоком могут подаваться в любом порядке, но все они должны предшествовать командам FTP-сервиса.

3. Команды FTP-сервиса определяют действия, которые необходимо произвести с указанными файлами. Как правило, аргументом команд этой группы является имя файла.

**Код:**

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Threading;using System.Threading.Tasks;using System.Net;using System.Net.Sockets;

namespace FTP\_Client{

class Program {

public static Socket PD\_Socket;

public static Thread recv = new Thread(Receiver);

public static Thread send = new Thread(Sender);

public static string file\_path = "";

static void Sender() {

if (file\_path != "") { PD\_Socket.SendFile(file\_path); file\_path = ""; }

}

static void Receiver() {

int b\_cnt; byte[] bytes = new byte[4086]; byte[] cmd = { };

string answer; string command;

b\_cnt = PD\_Socket.Receive(bytes);

answer = Encoding.UTF8.GetString(bytes, 0, b\_cnt);

if(answer != "") { Console.WriteLine("\nS\_recv: {0}\n\n", answer); }

}

static void Disconnect\_PD\_Socket() {

if(recv.ThreadState == ThreadState.Running) { recv.Abort(); }

if (send.ThreadState == ThreadState.Running) { send.Abort(); }

PD\_Socket.Close();

}

static void PassiveDataConnection(string str) {

byte[] cmd = new byte[1024]; byte[] ans = { }; int bytesRec;

int bytesSent; string command; string answer = "";

string[] parts = str.Split(new char[] { ',' });

parts[parts.Length - 1] = parts[parts.Length - 1].Substring(0, parts[parts.Length - 1].Length - 4);

int port = Int32.Parse(parts[parts.Length - 2]) \* 256 + Int32.Parse(parts[parts.Length - 1]);

IPAddress PD\_IpAddr = IPAddress.Parse("192.168.43.88");

IPEndPoint PD\_EndPoint = new IPEndPoint(PD\_IpAddr, port);

PD\_Socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

PD\_Socket.Connect(PD\_EndPoint);

Console.WriteLine("Data connection successfull!\n");

recv = new Thread(Receiver);

recv.Priority = ThreadPriority.Highest;

recv.Start();

}

static void ParseStor(string str) {

string[] parts = str.Split(new char[] {' '});

Console.WriteLine(parts[1]);

file\_path = parts[1];

send = new Thread(Sender);

send.Priority = ThreadPriority.Highest;

send.Start();

}

static void Main(string[] args) {

try { byte[] bytes = new byte[1024]; byte[] msg = { };

int bytesRec; int bytesSent; string message; string answer = "";

int port = 21;

IPAddress ipAddr = IPAddress.Parse("192.168.43.88");

IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, port);

Socketsocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

socket.Connect(ipEndPoint);

bytesRec = socket.Receive(bytes);

Console.WriteLine("\nS: {0}\n\n", Encoding.UTF8.GetString(bytes, 0, bytesRec));

message = "USER anonymous\r\n";

msg = Encoding.UTF8.GetBytes(message);

bytesSent = socket.Send(msg);

bytesRec = socket.Receive(bytes);

answer = Encoding.UTF8.GetString(bytes, 0, bytesRec);

Console.WriteLine("\nS: {0}\n\n", answer);

message = "Pass test\r\n";

msg = Encoding.UTF8.GetBytes(message);

bytesSent = socket.Send(msg);

bytesRec = socket.Receive(bytes);

answer = Encoding.UTF8.GetString(bytes, 0, bytesRec);

Console.WriteLine("\nS: {0}\n\n", answer);

while (socket.Connected) {

Console.Write("C: ");

message = Console.ReadLine();

if (message.Contains("stor")) { ParseStor(message); }

message += "\r\n";

msg = Encoding.UTF8.GetBytes(message);

bytesSent = socket.Send(msg);

bytesRec = socket.Receive(bytes);

answer = Encoding.UTF8.GetString(bytes, 0, bytesRec);

Console.WriteLine("\nS: {0}\n\n", answer);

if (answer.Contains("227")) { PassiveDataConnection(answer); }

if (answer.Contains("226")) { Disconnect\_PD\_Socket(); }

}

}

catch (Exception ex) { Console.WriteLine(ex.Message); }

Console.Read();

}

}

}