Санкт-Петербургский Политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчёт о лабораторной работе

Дисциплина: Сети ЭВМ и телекоммуникации

Тема: Сервер протокола POP3

Выполнил студент гр. 43501/3 _____ Никитенко А.П.
(подпись)

Руководитель _____ Вылегжанина К.Д.
(подпись)

“___” _____2016 г.

Санкт-Петербург

2016

1. Индивидуальное задание

Разработать приложение для операционных систем семейства Windows или Linux, обеспечивающее функции сервера протокола POP3.

2. Цель работы

Изучить протокол POP3 прикладного уровня стека TCP/IP.

3. Описания протокола FTP

POP3 —стандартный интернет-протокол прикладного уровня, используемый клиентами электронной почты для получения почты с удаленного сервера по TCP/IP-соединению. POP поддерживает простые требования «загрузи-и-удали» для доступа к удаленным почтовым ящикам. Хотя большая часть POP-клиентов предоставляет возможность оставить почту на сервере после загрузки, использующие POP клиенты обычно соединяются, извлекают все письма, сохраняют их на пользовательском компьютере как новые сообщения, удаляют их с сервера, после чего разъединяются.

POP3-сервер прослушивает общеизвестный порт 110. Доступные сообщения клиента фиксируются при открытии почтового ящика POP-сессией и определяются количеством сообщений для сессии, или, по желанию, с помощью уникального идентификатора, присваиваемого сообщению POP-сервером. Этот уникальный идентификатор является постоянным и уникальным для почтового ящика и позволяет клиенту получить доступ к одному и тому же сообщению в разных POP-сессиях. Почта извлекается и помечается для удаления с помощью номера сообщения. При выходе клиента из сессии помеченные сообщения удаляются из почтового ящика.

4. POP3-сервер

4.1. Описание приложения

Разработанный POP3-сервер реализует следующие функции:

1) Хранение идентификационной и аутентификационной информации нескольких пользователей

2) Хранение почтовых папок входящих сообщений нескольких пользователей

3) Обработка подключения клиента

4) Выдача состояния ящика (количество новых писем, их суммарная длина)

5) Выдача списка всех писем сервера с длиной в байтах

6) Выдача содержимого указанного письма

8) Удаление всех помеченных писем при разрыве соединения

9) Протоколирование соединения сервера с клиентом

Реализованные команды протокола:

USER – получение от клиента идентификационной информации пользователя

PASS – получение от клиента пароля пользователя

STAT – отправка клиенту состояния почтового ящика

LIST – отправка клиенту списка сообщения почтового ящика

RETR – отправка клиенту сообщения

UIDL – выдача уникального идентификатора сообщения

QUIT – удаление всех помеченных сообщений и завершение сеанса

4.2. Демонстрация работы приложения

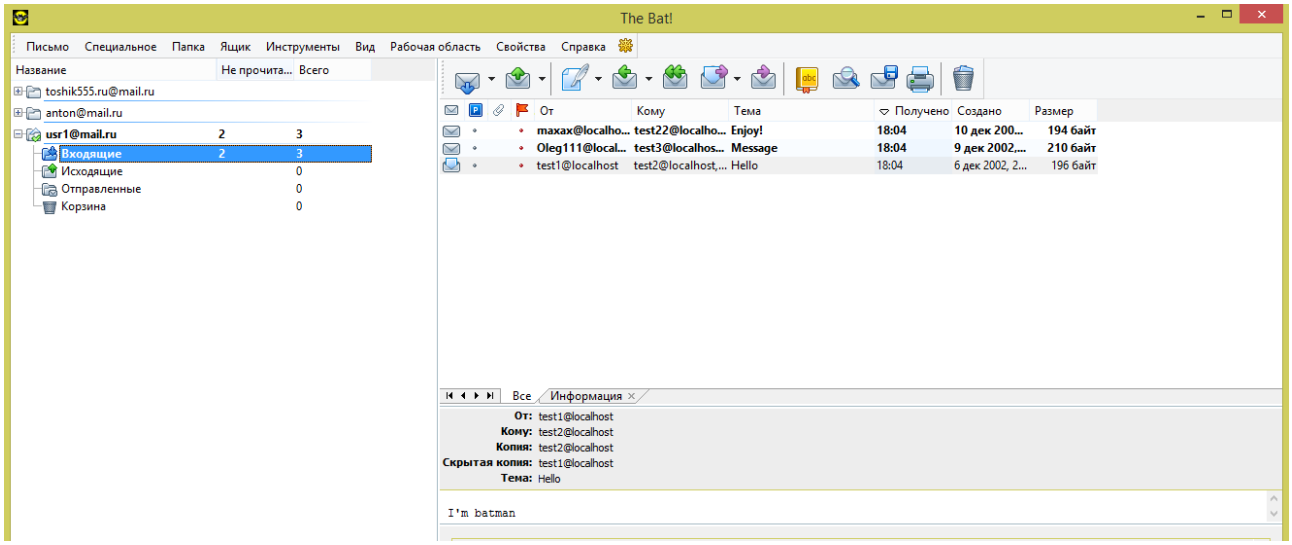
Для демонстрации работы приложения с помощью клиента The Bat был создан почтовый ящик "usr1@mail.ru". Аутентификационная информация пользователей лежит в файле "login.txt", а письма для каждого пользователя в папке INBOX, в соответствующей директории. Перед началом работы программы на сервер были помещены 3 новых письма для пользователя usr1@mail.ru.

Проверим теперь работу приложения.

```
run:
Waiting for a client...
```

Включили сервер, теперь он ждет подключения пользователей.

С помощью клиента The Bat попробуем получить с сервера новые письма.



Как видно в окне программы-клиента, пользователь получил 3 новых письма.

В консоли выполнения сервера можно посмотреть лог обмена данными между клиентом и сервером.

```
run:
Waiting for a client...
Client with ip /127.0.0.1 and port 56893 was connected
text from client with port 56893:::  CAPA
text from client with port 56893:::  USER usr1@mail.ru
text from client with port 56893:::  PASS usr1
text from client with port 56893:::  STAT
+OK 3 582text from client with port 56893:::  LIST
+OK 3 messages (582 octets)

1 190
2 204
3 188
text from client with port 56893:::  UIDL
+OK 3 messages (582 octets)
1 1
2 2
3 3
.

text from client with port 56893:::  RETR 1
text from client with port 56893:::  RETR 2
text from client with port 56893:::  RETR 3
text from client with port 56893:::  QUIT
Close client with port 56893
```

Здесь видно, что сервер принял подключение клиента, попросил его аутентификационные данные, сверил их со своими и передал клиента информацию о состоянии почтового ящика. Затем клиент просит у сервера список сообщения почтового ящика и, получив информацию обо всех сообщениях в ящике, просит выдать их уникальные идентификаторы, а затем с помощью команды RETR поочередно просит отправить сообщения, указывая их идентификатор в качестве аргумента. Получив письма, клиент отключается от сервера.

4.3. Архитектура приложения

В состав POP3-сервера входят следующие классы:

1) MultServer – содержит точку входа в приложение, позволяет разорвать соединение на стороне сервера;
2) Acception – поток для приема соединений с сервером;
3) ClientThread реализует взаимодействие сервера с клиентом.

Выводы

В ходе выполнения работы были изучены особенности протокола POP3, а также получен навык реализации клиента прикладного протокола на языке программирования Java.

Протокол POP3 имеет ряд достоинств и недостатков.

Достоинства:

- простота настройки (так как у POP маленькая функциональность, то и настраивать ничего не нужно)
- Минимальное время соединения с сервером, а поэтому минимальная загрузка канала
- минимальное использование серверных ресурсов, за счет того, что почта один раз перекачивается на клиентский ПК и далее вся работа осуществляется без участия сервера.

Недостатки:

- Главный недостаток POP3 - он умеет синхронизировать только папку "Входящие" только в направлении от сервера к рабочей станции.
  При использовании POP3 содержимое папки "Отправленные" и прочих папок видно только на том компьютере, на котором они созданы. Почта из папки "Входящие" после скачивания чаще всего удаляется с сервера, соответственно все ваша корреспонденция хранится только на локальном компьютере, что может привести к потере всех писем в случае поломки ПК или ОС.
- Невозможность управления почтой прямо на сервере. Сообщения в этом протоколе загружаются с сервера все сразу, после чего, как правило, удаляются с него - фактически пользователю не дают выбрать, какие сообщения он хочет скачать, а какие - удалить прямо с сервера, а также нет возможности создания папок для хранения писем на сервере - все письма скачиваются кучей в папку "Входящие"

Приложение

Acception.java

```
package multserver;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Acception extends Thread{
ServerSocket servers = null;
Socket fromclient = null;
   public Acception(ServerSocket serverSoket ) {
      servers = serverSoket;
   }

   @Override
   public void run() {
      System.out.println("Waiting for a client...");
      try {
         while (true)
         {
            fromclient = servers.accept();
            System.out.println("Client with ip " + fromclient.getInetAddress() + " and port " +
fromclient.getPort() + " was connected");
            ClientThread newClient = new ClientThread(fromclient);
            newClient.start();
         }

      } catch (IOException ex) {
         System.out.println("Program close. Accept");
         System.exit(-2);
      }

      try {
         fromclient.close();
         servers.close();
      } catch (IOException ex) {
         System.out.println("Can't close socket");
         System.exit(-3);
      }
   }
}
```

ClientThread.java

```
package multserver;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.StringTokenizer;
import java.io.File;
import java.util.Random;


public class ClientThread extends Thread {
Socket  clientSocket;
BufferedReader in = null;
PrintWriter out = null;
static String CAPA = "CAPA";
static String USER = "USER";
static String PASS = "PASS";
static String STAT = "STAT";
static String LIST = "LIST";
static String UIDL = "UIDL";
static String RETR = "RETR";
static String QUIT = "QUIT";
static String TOP  = "TOP";
static String NOOP = "NOOP";

static String CRLF = "\r\n";
static String EOM = ".\r\n";

static String password = "";
static String user_inbox = "";
static String path_to_box = "INBOX/";
boolean authorization = true, transaction = false, update = false;

   public ClientThread(Socket clientsocket) {
      this.clientSocket = clientsocket;
   }
   public void CAPA_available_cmds()
   {
      out.println("+OK CAPA list follows" + CRLF);
      out.println(USER + CRLF );
//       out.print(TOP + CRLF );
      out.println(STAT + CRLF );
      out.println(LIST + CRLF );
      out.println(QUIT + CRLF );
      out.println(EOM);
   }
   public void USER_parse_login(String cmd_from_client)
   {
      String line = "";
      FileReader myFile = null;
      BufferedReader buff = null;
      StringTokenizer st = new StringTokenizer(cmd_from_client," ");
      st.nextToken();
      user_inbox = st.nextToken();
```

```
    try {
       myFile = new FileReader("login.txt");
    } catch (FileNotFoundException ex) {
       System.out.println("Can't find the login file");
       System.exit(-1);
    }
    buff = new BufferedReader(myFile);
    while(true)
    {
       try {
          line = buff.readLine();
       } catch (IOException ex) {
          System.out.println("Can't read the login file");
          System.exit(-2);
       }

       if (line == null) out.println("-ERR never heard of mailbox name");

       StringTokenizer st1 = new StringTokenizer(line,"USERPA: \n");
       while(st1.hasMoreTokens())
       {
           if (user_inbox.equals(st1.nextToken()))
           {
              password = st1.nextToken();
              out.println("+OK name is a valid mailbox" );
              return;
           }
          else st1.nextToken();
       }
   }


}
public void PASS_password_control(String cmd_from_client)
{
    StringTokenizer st = new StringTokenizer(cmd_from_client," ");
    st.nextToken();
    String client_pass = st.nextToken();
    if (client_pass.equals(password))
    {
       out.println("+OK maildrop locked and ready");
       authorization = false;
       transaction = true;
       path_to_box += user_inbox + "/";
    }
    else out.print("-ERR invalid password");
}
public void STAT_discovery_dir()
{
    File path = new File(path_to_box);
    File[] list = path.listFiles();
```

```
        int size = 0;
        for (int i = 0 ; i < list.length; i++)
        {
           size += list[i].length();
        }
        System.out.print("+OK " + list.length + " " + size);
        out.println("+OK " + list.length + " " + size);
        //out.print("+OK 1 100");
    }
    public void LIST_size_list()
    {

        File path = new File(path_to_box);
        File[] list = path.listFiles();

        // количество байт всех сообщений
        int size = 0;
        for (int i = 0 ; i < list.length; i++)
        {
           size += list[i].length();
        }
        out.println("+OK " + list.length + " messages (" + size + " octets)" + CRLF);
        System.out.println("+OK " + list.length + " messages (" + size + " octets)" + CRLF);
        for (int i = 1 ; i <= list.length ; i++)
        {
           out.println(i + " " + list[i-1].length() + CRLF);
           System.out.print(i + " " + list[i-1].length() + CRLF);
        }
        out.println(EOM);


//        out.println("+OK 1 message (100 octets)\r\n");
//                        out.println("1 100\r\n");
//                        out.println(".\r\n");
    }
    public void UIDL_uniq_number()
    {
        File path = new File(path_to_box);
        File[] list = path.listFiles();
        Random rand = new Random();

        // количество байт всех сообщений
        int size = 0;
        for (int i = 0 ; i < list.length; i++)
        {
           size += list[i].length();
        }
        out.print("+OK " + list.length + " messages (" + size + " octets)" + CRLF);
        System.out.print("+OK " + list.length + " messages (" + size + " octets)" + CRLF);
        for (int i = 1 ; i <= list.length ; i++)
        {
           out.println(i + " " + i + CRLF);
```

```
            //out.print(i + " " + rand.nextInt(10000) + CRLF);
            System.out.print(i + " " + i + CRLF);
        }
        out.println(EOM);
        System.out.println(EOM);



//                          out.println("+OK 1 messages (100 octets)\r\n");
//                  out.println("1 1\r\n");
//                  out.println(".\r\n");
    }
    public void RETR_transmit_message(String cmd_from_client)
    {
        out.println("+OK message follows\r\n");

        FileReader myFile = null;
        BufferedReader buff = null;

        File path = new File(path_to_box);
        File[] list = path.listFiles();

        String line = "";

        // парсим строку, выбирая номер сообщения для скачивания
        StringTokenizer st = new StringTokenizer(cmd_from_client," ");
        st.nextToken();
        int number_of_message =  Integer.parseInt(st.nextToken());

        // открываем файл сообщения
        try {
            myFile = new FileReader(path_to_box + list[number_of_message - 1].getName());

        } catch (FileNotFoundException ex) {
            System.out.print("Can't find chosen message file");
            System.exit(-3);
        }

        buff = new BufferedReader(myFile);

        while (true)
        {
            try {
                line = buff.readLine();
            } catch (IOException ex) {
                System.out.print("Can't read chosen message file");
                System.exit(-4);
            }
            if (line == null) break;
            if (!line.equals("")) out.println(line);
            else                  out.println(CRLF);
        }
```

```
    out.println(EOM);
}
public void QUIT_end_connection()
{
    out.println("+OK");
    path_to_box = "INBOX/";
    authorization = true;
    transaction = false;
}
public void TOP(String cmd_from_client)
{
    FileReader myFile = null;
    BufferedReader buff = null;

    File path = new File(path_to_box);
    File[] list = path.listFiles();

    String line = "";

    // парсим строку, выбирая номер сообщения
    StringTokenizer st = new StringTokenizer(cmd_from_client," ");
    st.nextToken();
    int number_of_message =  Integer.parseInt(st.nextToken());

    // открываем файл сообщения
    try {
        myFile = new FileReader(path_to_box + list[number_of_message - 1].getName());

    } catch (FileNotFoundException ex) {
        System.out.println("Can't find chosen message file");
        System.exit(-3);
    }

    buff = new BufferedReader(myFile);

    while (true)
    {
        try {
            line = buff.readLine();
        } catch (IOException ex) {
            System.out.println("Can't read chosen message file");
            System.exit(-4);
        }
        if (line == null) break;
        if (!line.equals("")) out.println(line + CRLF);
        else                  break;
    }
    out.println(EOM);

}
@Override
public void run() {
```

```
String input, output;
FileReader myFile = null;
BufferedReader buff = null;

 try {
    in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
    out = new PrintWriter(clientSocket.getOutputStream(), true);
} catch (IOException ex) {
    System.out.println("Can't read socket");
    return;
}
try {
    // приветственное сообщение
    out.println("+OK POP3 server ready");

    // TEMP
    int i = 0;

    while(true)
    {
        input = in.readLine();
        System.out.println("text from client with port " + clientSocket.getPort() + "::: " + input);
        StringTokenizer st = new StringTokenizer(input, " ");
        String cmd = st.nextToken();
        if (authorization)
        {
            if      (cmd.equals(CAPA))  CAPA_available_cmds();
            else if (cmd.equals(USER))  USER_parse_login(input);
            else if (cmd.equals(PASS))  PASS_password_control(input);
        }
        else if (transaction)
        {
            if      (cmd.equals(STAT))  STAT_discovery_dir();
            else if (cmd.equals(LIST))  LIST_size_list();
            else if (cmd.equals(UIDL))  UIDL_uniq_number();
            else if (cmd.equals(RETR))  RETR_transmit_message(input);
            else if (cmd.equals(QUIT))
            {
                QUIT_end_connection();
                System.out.println("Close client with port " + clientSocket.getPort());
                break;
            }
            else if (cmd.equals(TOP)) TOP(input);
            else if (cmd.equals(NOOP))    out.println("+OK");
        }


    }
    out.close();
    in.close();
    clientSocket.close();
} catch (IOException ex) {
```

```
            System.out.println("Problem with streams");
            System.exit(-100);
        }

    }
}
```

MultServer.java

```
package multserver;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;



public class MultServer {
 public static ServerSocket servers = null;

    public static void exit() throws IOException
    {
        System.out.println("Exit subprogram is running");
        servers.close();
    }


    public static void main(String[] args) throws IOException {

        BufferedReader in = null;
        PrintWriter out = null;
        String input, output;


        // создаем серверный сокет
        try {
            servers = new ServerSocket(110);
        } catch (IOException ex) {
            System.out.println("Couldn't listen to port 4444");
            System.exit(-1);
        }

        // Создаем потоки принятия клиентов
        Acception acc = new Acception(servers);
        acc.start();

        BufferedReader inu = null;
        inu = new BufferedReader(new InputStreamReader(System.in));

        String fuser;

        while(true)
```

```
        {
            fuser = inu.readLine();
            if (fuser.equalsIgnoreCase("exit"))
            {
                exit();
                break;
            }

        }
    }
}
```