

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

Отчёт по лабораторной работе

Дисциплина: Сети и телекоммуникации

Тема: Сервер протокола POP3

Выполнил студент гр. 43501/3

(подпись)

В.Е. Бушин

Руководитель

(подпись)

К.Д. Вылегжанина

“ _ ” _____ 2016 г.

Санкт-Петербург

2016

Задание

Разработать приложение для операционных систем семейства Windows или Linux, обеспечивающее функции сервера протокола POP3. Основные возможности. Приложение должно реализовывать 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
- 7) Пометка письма для последующего удаления
- 8) Удаление всех помеченных писем при разрыве соединения
- 9) Протоколирование соединения сервера с клиентом

Поддерживаемые команды. Разработанное приложение должно реализовывать следующие команды протокола POP3:

- 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
- DELE – пометка сообщения на удаление
- RSET – сброс всех пометок на удаление сообщений
- QUIT – удаление всех помеченных сообщений и завершение сеанса

Настройки приложения. Разработанное приложение должно обеспечивать настройку следующих параметров:

- настройку номера порта сервера (по умолчанию - 110)
- создание, редактирование и удаление пользователей почтового сервера
- настройку аутентификационной информации для пользователей почтового сервера

Выполнение задания:

1. Реализация приложения

Протокол POP3 описан в RFC 1939. Сервер был реализован на языке C++, использовалась среда разработки Microsoft Visual Studio 2013. Были созданы собственные классы для работы POP3-сервера: класс письма – Letter, класс пользователя – User, класс почтового ящика – Mailbox, класс обработчика почты – MailHandler.

Класс Letter описан в заголовочном файле Letter.h и он содержит все необходимые поля письма (from, to, subject, data) и также метку, которая будет показывать, нужно ли удалить это письмо или нет при отсоединении от сервера. Данный класс содержит методы, которые возвращают значения всех переменных класса, и методы, которые меняют переменные класса.

Класс Mailbox описан в заголовочном файле Mailbox.h и он содержит в качестве переменной класса вектор из писем, а в качестве методов – функции добавления в почтовый ящик и удаления из него письма, функции, которые возвращают конкретное письмо, количество писем, размер всех писем, размер конкретного письма и устанавливают метку в конкретном письме.

Класс User описан в заголовочном файле User.h и он содержит в качестве переменных класса строки с именем и паролем пользователя, почтовый ящик пользователя, метку, которая показывает, находится ли данный пользователь online или нет. В методах этого класса находятся различные функции, которые обращаются к почтовому ящику, чтобы добавить в него письмо, удалить, узнать количество писем и т.д., также функции, которые возвращают имя пользователя и пароль.

Класс MailHandler описан в заголовочном файле MailHandler.h и он содержит в качестве переменных класса вектор из пользователей и вектор из писем. Этот класс содержит различные функции для работы с клиентом, и из функции main вызывается функция, создания нового потока из этого класса и дальше идёт работа с клиентом.

Также используется заголовочный файл Responses.h, в котором определены некоторые ответы сервера клиенту или части ответов сервера клиенту, и заголовочный файл util.h и файл с исходным кодом util.cpp, где определены вспомогательные функции, которые часто используются.

2. Тестирование приложения

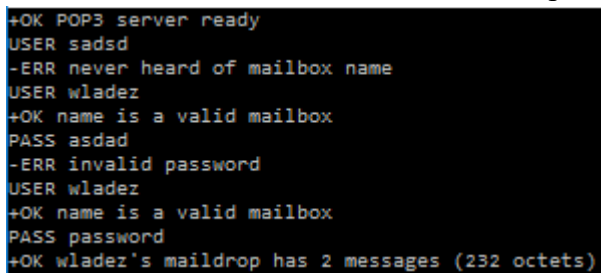
Для тестирования приложения использовался TCP-клиент Telnet, так как протокол POP3 работает на 110 порту протокола TCP было очень удобно проверять и отлаживать работу различных функций сервера по мере его реализации. В процессе тестирования были проверены все основные возможности реализованного приложения. Подключение происходит следующим образом (рис. 1).



```
Microsoft Telnet> open 127.0.0.1 110
```

Рис. 1. Подключение к серверу.

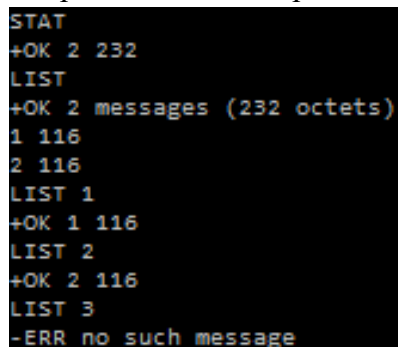
Процесс аутентификации с возможными ошибками показан на рис. 2.



```
+OK POP3 server ready
USER sadsd
-ERR never heard of mailbox name
USER wladez
+OK name is a valid mailbox
PASS asdad
-ERR invalid password
USER wladez
+OK name is a valid mailbox
PASS password
+OK wladez's maildrop has 2 messages (232 octets)
```

Рис. 2. Процесс аутентификации.

Работа информационных команд протокола POP3 продемонстрирована на рис. 3.



```
STAT
+OK 2 232
LIST
+OK 2 messages (232 octets)
1 116
2 116
LIST 1
+OK 1 116
LIST 2
+OK 2 116
LIST 3
-ERR no such message
```

Рис. 3. Информационные команды протокола POP3.

Передача писем с помощью команды RETR продемонстрирована на рис. 4.

```
RETR 1
+OK 116 octets
From: lera
To: wladez
Subject: Just for fun
Data: Hi! How are you?
.
RETR 2
+OK 116 octets
From: azat
To: wladez
Subject: Study
Data: You need to get zachot for seti!
.
RETR 3
-ERR no such message
```

Рис. 4. Передача писем.

Удаление писем и их восстановление с помощью команды RSET показано на рис. 5.

```
DELE 1
+OK message 1 deleted
DELE 1
-ERR message 1 already deleted
RETR 1
-ERR message 1 was deleted
DELE 2
+OK message 2 deleted
RETR 2
-ERR message 2 was deleted
RSET
+OK maildrop has 2 messages
RETR 1
+OK 116 octets
From: lera
To: wladez
Subject: Just for fun
Data: Hi! How are you?
.
RETR 2
+OK 116 octets
From: azat
To: wladez
Subject: Study
Data: You need to get zachot for seti!
.
```

Рис. 5. Удаление писем и их восстановление.

Завершение сеанса и удаление всех сообщений при выходе продемонстрировано на рис. 6.

```
DELE 1
+OK message 1 deleted
DELE 2
+OK message 2 deleted
QUIT
+OK dewey POP3 server signing off (maildrop empty)

Подключение к узлу утеряно.
Нажмите любую клавишу...
```

Рис. 6. Завершение сеанса.

Вывод

Был изучен протокол POP3 и разработан простейший POP3-сервер, обеспечивающий основные функции протокола POP3. Эти функции были реализованы с использованием созданных классов для работы с письмами, почтовыми ящиками и пользователями. Реализованный сервер был протестирован соответственно исходному заданию. Тесты показали, что сервер работает корректно и все его требуемые функции реализованы.

Приложения

Листинги

Заголовочные файлы

Листинг заголовочного файла Letter.h:

```
#pragma once
#include <string>
using namespace std;

class Letter
{
public:
    Letter();
    ~Letter();

    void addFrom(string& fr){
        from = fr;
    }

    void addSubject(string& sub){
        subject = sub;
    }

    void addData(string& dat){
        data += dat;
    }

    void addTo(string& t){
        to = t;
    }

    void setMarker(bool m){
        marker = m;
        string t = data;
    }

    string getFrom(){
        return from;
    }

    string getData(){
        return data;
    }

    string getTo(){
        return to;
    }

    string getSubject(){
        return subject;
    }

    bool getMarker(){
        return marker;
    }

private:
    string from;
    string to;
    string subject;
    string data;
    bool marker; //метка письма, изначально должна быть FALSE
};
```

Листинг заголовочного файла Mailbox.h:

```
#pragma once
#include <vector>
#include <string>
#include "headers/Letter.h"
class Mailbox
{
public:
    Mailbox();
    ~Mailbox();
    void addLetterToMailbox(const Letter& let){
        letters.push_back(let);
    }

    void eraseLetterFromMailbox(const int& num){
        letters.erase(letters.begin() + num);
    }

    //возвращает кол-во писем
    int returnCountOfLetters(){
        return letters.size();
    }

    //возвращает размер всех писем
    int returnSizeOfAllLetters(){
        int res = 0;
        for (int i = 0; i < letters.size();i++){
            res += sizeof(letters.at(i+1));
        }
        return res;
    }

    //возвращает размер письма
    int returnSizeOfLetter(int i){
        return sizeof(letters.at(i+1));
    }

    //возвращает письмо
    Letter returnLetter(int i){
        Letter l = letters.at(i-1);
        return l;
    }

    void setFlag(int i, bool b){
        letters.at(i - 1).setMarker(b);
    }

private:
    vector<Letter>letters;
};
```

Листинг заголовочного файла User.h:

```
#pragma once
#include <string>
#include "headers/Mailbox.h"
using namespace std;
class User
{
public:
    User();
    User(const string& name, const string& pass);
    ~User();
    string getName(){
        return userName;
```

```

    }

    string getPass(){
        return password;
    }

    void addLetter(const Letter& l){
        box.addLetterToMailbox(l);
    }

    void eraseLetter(int num){
        box.eraseLetterFromMailbox(num);
    }

    //возвращает количество писем в ящике у пользователя
    int lettersCount(){
        return box.returnCountOfLetters();
    }

    //возвращает общий размер всех писем в ящике
    int overallSize(){
        return box.returnSizeOfAllLetters();
    }

    //возвращает размер конкретного письма
    int letterSize(int i){
        return box.returnSizeOfLetter(i);
    }

    //возвращает письмо
    Letter getLetter(int i){
        Letter l = Letter();
        l = box.returnLetter(i);
        return l;
    }

    void deleteLetter(int i){
        box.setFlag(i,true);
    }

    void recoverLetter(int i){
        box.setFlag(i,false);
    }

    void setOnline(){
        isOnline = true;
    }

    void setOffline(){
        isOnline = false;
    }

private:
    string userName;
    string password;
    Mailbox box;
    bool isOnline;
};

```

Листинг заголовочного файла MailHandler.h:

```

#ifndef POP3SERVER_MAILHANDLER_H
#define POP3SERVER_MAILHANDLER_H
#include "util.h"
#include "User.h"

```

```

#include "Responses.h"
#include <string>
#include <thread>
#include <vector>
#include <map>
using namespace std;

class MailHandler
{
public:
    MailHandler();
    DWORD WINAPI clientHandler(LPVOID param);
    thread createThread(LPVOID param);
    int checkUser(const string &name);
    int authentication(SOCKET client_socket, int& status, User& curUser);
    string connectedUser(User& curUser);
    void status(SOCKET client_socket, User& curUser);
    void getList(SOCKET client_socket, string req, User& curUser);
    void getMessage(SOCKET client_socket, string req, User& curUser);
    void deleteMessage(SOCKET client_socket, string req, User& curUser);
    void reset(SOCKET client_socket, User& curUser);
    void disconnect(SOCKET client_socket, int& status, User& curUser);

private:
    vector<User> users;
    vector<Letter> let;
};

#endif

```

Листинг заголовочного файла util.h:

```

#ifndef SERVER_UTIL_H
#define SERVER_UTIL_H
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

// Need to link with Ws2_32.lib, Mswsock.lib, and Advapi32.lib
#pragma comment (lib, "Ws2_32.lib")
#pragma comment (lib, "Mswsock.lib")
#pragma comment (lib, "AdvApi32.lib")
#define MAX_STR_LEN 4096
#define SIZE_OF_BUF 4096
#define numCl 2

int recvn(SOCKET fd, char *bp, size_t len);
int sendn(SOCKET s, char* buf, int lenbuf, int flags);
int recvLine(SOCKET sock, char* buffer, int buffSize);
int sendLine(int sock, const char* str);
#endif /*SERVER_UTIL_H*/

```

Листинг заголовочного файла Responses.h:

```

#ifndef SERVER_RESPONSES_H
#define SERVER_RESPONSES_H

//512 - максимальная длина каждого элемента
const char responses[][512] = {
    { "-ERR no such message\r\n" }, //0 - ошибка, нет такого сообщения

```



```

{ "+OK " },//1 - часть ответа от сервера
{ "+OK POP3 server ready\r\n" },//2 сервер готов к работе с клиентом
{ "+OK name is a valid mailbox\r\n" },//3 положительный ответ на команду USER
{ "-ERR never heard of mailbox name\r\n" },//4 отрицательный ответ на команду USER
{ "+OK %s's maildrop has %d messages (%d octets)\n" },//5 доступ к почтовому
серверу открыт - ответ на команду PASS
{ "-ERR invalid password\r\n" },//6 неправильный пароль - ответ на команду PASS
{ "-ERR unable to lock maildrop\n" },//7 нет доступа к почтовому ящику - ответ на
команду PASS
{ "+OK %d %d\n" },//8 ответ на команду STAT
{ "+OK %s\n" },//9 ответ на команду LIST
{ "+OK %d octets\n%s\n" },//10 ответ на команду RETR
{ "+OK message " },//11 ответ на команду DELE
{ "+OK\n" },//12 ответ на команду NOOP
{ "+OK maildrop has " },//13 ответ на команду RSET
{ "+OK\n%s\n" },//14 ответ на команду TOP
{ "+OK %d %s\n" },//15 ответ на команду UIDL
{ "+OK dewey POP3 server signing off\n" },//16 ответ на команду QUIT в состоянии
авторизации
{ "'s maildrop has " },//17
{ " messages (" },//18
{ "+OK dewey POP3 server signing off (" },//19 ответ на команду QUIT при
отсоединении пользователя
{ "+OK dewey POP3 server signing off (maildrop empty)" },//20 ответ на команду
QUIT при отсоединении пользователя при пустом ящике
{ "+OK Capability list follows\r\n" }//21 CAPA
};

#endif /*SERVER_RESPONSES_H*/

```

Файлы с исходным кодом

Листинг основного модуля main.cpp:

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "headers/util.h"
#include "headers/MailHandler.h"
using namespace std;

HANDLE mutex;
HANDLE allhandlers[numCl];
int numClients = 0;

int main(void) {
    MailHandler mh;
    //используется для инициализации библиотеки сокетов
    char buf[SIZE_OF_BUF]; //буфер приема и передачи сообщения
    int readbytes; //число прочитанных байт
    WSADATA WSStartData; //Инициализация WinSock и проверка его запуска
    if (WSAStartup(MAKEWORD(2, 0), &WSStartData) != 0) {
        printf("WSAStartup failed with error: %d\n", GetLastError());
        return 100;
    } //создание сокета
    SOCKET server_socket; //по умолчанию используется протокол tcp
    printf("Server is started.\nTry to create socket -----");
    if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
        printf("error with creation socket. GetLastError= %d\n", GetLastError());
        return 1000;
    }
    printf("CHECK\n"); //Привязывание сокета конкретному IP и номеру порта
    struct sockaddr_in sin;
    sin.sin_addr.s_addr = INADDR_ANY; // используем все интерфейсы
    sin.sin_port = htons(110); // используем порт протокола POP3
    sin.sin_family = AF_INET; printf("Try to bind socket -----");
    if (::bind(server_socket, (struct sockaddr *)&sin, sizeof(sin)) == SOCKET_ERROR) {
        printf("error with bind socket. GetLastError= %d\n", GetLastError());
        return 1001;
    }
    printf("CHECK\n"); //делаем сокет прослушиваемым
    printf("Try to set socket listening -----");
    if (listen(server_socket, 5) != 0) {
        printf("error with listen socket. GetLastError= %d\n", GetLastError());
        return 1002;
    }
    printf("CHECK\n");
    printf("Server starts listening\n"); //Ждем клиента. Создаем пустую структуру,
    //которая будет содержать параметры сокета,
    //иницирующего соединение
    struct sockaddr_in from;
    int fromlen = sizeof(from); // начинаем "слушать" входящие запросы на подключение
    int ind = 0;
    //на каждое подключение к серверному сокету
    while (SOCKET client_socket = accept(server_socket, (struct sockaddr*)&from,
    &fromlen)){
        //создаём новый поток, в котором работает clientHandler() из MailHandler,
        //detach отправляет его выполняться независимо от главного потока
        mh.createThread((LPVOID)client_socket).detach();
        cout << "Client connect to server\n" <<endl;
    }
    closesocket(server_socket);
    return 0;
}
```

Листинг файла Letter.cpp:

```
#include "headers/Letter.h"

//пустой конструктор
Letter::Letter()
{
    from = "";
    to = "";
    subject = "";
    data = "";
    marker = false;
}

Letter::~Letter()
{
}
```

Листинг файла Mailbox.cpp:

```
#include "headers/Mailbox.h"

Mailbox::Mailbox()
{
}

Mailbox::~Mailbox()
{
}
```

Листинг файла User.cpp:

```
#include "headers/User.h"

//пустой конструктор с инициализацией
User::User() : userName(""), password(""), isOnline(false), box(Mailbox()) { }

User::User(const string &name, const string &pass) : userName(name), password(pass),
isOnline(false), box(Mailbox()) { }

User::~User()
{
}
```

Листинг файла MailHandler.cpp:

```
#include "headers/MailHandler.h"
#include <fstream>

MailHandler::MailHandler()
{
    // в конструкторе добавляются такие юзера в users, users - private поле в
заголовочном
    users.push_back(User("wladex", "password"));
    users.push_back(User("azat", "12345"));
    users.push_back(User("lera", "1q2w3e"));
    // FOR TESTS!
    //и нулевому юзеру добавляются два тестовых письма
    Letter letter = Letter();
    letter.addFrom((string)"lera");
    letter.addSubject((string)"Just for fun");
    letter.addTo((string)"wladex");
    letter.addData((string)"Hi! How are you?");
    users[0].addLetter(letter);
    Letter letter1 = Letter();
```

```

letter1.addFrom((string)"azat");
letter1.addSubject((string)"Study");
letter1.addTo((string)"wladez");
letter1.addData((string)"You need to get zachot for seti!");
users[0].addLetter(letter1);
}

DWORD WINAPI MailHandler::clientHandler(LPVOID param){
    SOCKET client_socket = (SOCKET)param;
    if (client_socket == INVALID_SOCKET) {
        printf("error with accept socket. GetLastError= %d\n", GetLastError());
        return 1003;
    }
    char buf[SIZE_OF_BUF] = "/0";
    //пересылаем код ответа номер 2 (ready) через клиентский сокет
    //POP3-коды ответа отпределены в Responses.h как двумерный массив
    sendLine(client_socket, responses[2]);
    //значение устанавливается в answer(), по смыслу номер состояния после принятия
запроса
    int status = 1;
    int aut;
    User curUs = User();
    do {
        aut = authentication(client_socket, status, curUs); //процесс
аутентификации клиента
        if (status == 0) {
            break; //Проверка на выход
        }
    } while (aut < 0);
    while (recvLine(client_socket, buf, SIZE_OF_BUF)>0){
        //принимаем сообщения размером buf из клиентского сокета, записываем в buf
        //int len = recvLine(client_socket, buf, SIZE_OF_BUF);
        //if (len > 2){
            //если что-то пришло, отвечаем
            string request=string(buf);
            request.erase(request.size() - 1);
            if (request.find("STAT") != string::npos){
                MailHandler::status(client_socket, curUs);
            }
            else if (request.find("LIST") != string::npos){
                getList(client_socket, request, curUs);
            }
            else if (request.find("RETR") != string::npos){
                getMessage(client_socket, request, curUs);
            }
            else if (request.find("DELE") != string::npos){
                deleteMessage(client_socket, request, curUs);
            }
            else if (request.find("RSET") != string::npos){
                reset(client_socket, curUs);
            }
            else if (request.find("QUIT") != string::npos){
                disconnect(client_socket, status, curUs);
            }
        }
        //}
        if (status == 0) {
            break; //Проверка на выход
        }
    }
    closesocket(client_socket);
}

int MailHandler::authentication(SOCKET client_socket, int& status, User& curUser){
    char buf[SIZE_OF_BUF] = "/0";

```

```

int len = recvLine(client_socket, buf, SIZE_OF_BUF);
cout << buf << endl;
int res=-1;
if (len>0){
    string request = string(buf);
    string pass;
    request.erase(request.size() - 1);
    cout << "request = " << request << endl;
    if (request.find("USER") != string::npos){
        request.erase(request.begin(), request.begin() + 5);
        int check = checkUser(request);
        if (check == -1){
            sendLine(client_socket, responses[4]); //ответ о том, что нет
такого пользователя
            res = -1;
        }
        else{
            sendLine(client_socket, responses[3]); //ответ о том, что есть
такой пользователь
            len = recvLine(client_socket, buf, SIZE_OF_BUF);
            if (len>0){
                pass = string(buf);
                pass.erase(pass.size() - 1);
                if (pass.find("PASS") != string::npos){
                    pass.erase(pass.begin(), pass.begin() + 5);
                    if (users[check].getPass() == pass){
                        curUser = users[check];
                        curUser.setOnline();
                        string answer = connectedUser(curUser);
                        sendLine(client_socket,
answer.c_str()); //правильный пароль
                        res = 1;
                    }
                    else {
                        sendLine(client_socket,
responses[6]); //неправильный пароль
                        res = -1;
                    }
                }
                else if (pass.find("QUIT") != string::npos){
                    status = 0;
                    sendLine(client_socket, responses[16]);
                }
            }
        }
    }
    else if (request.find("QUIT") != string::npos){
        status = 0;
        sendLine(client_socket, responses[16]);
    }
    else if (request.find("CAPA") != string::npos){
        string ans;
        ans.append("+OK Capability list follows\r\nUSER\r\n.");
        sendLine(client_socket, ans.c_str());
    }
}
return res;
}

string MailHandler::connectedUser(User& curUser){
    int cnt = 0, cap = 0;
    cnt = curUser.lettersCount();
    cap = curUser.overallSize();
    string answer;

```

```

        answer.append(responses[1] + curUser.getName() + responses[17] + to_string(cnt) +
responses[18] + to_string(cap) + " octets)\r\n");
        return answer;
    }

    void MailHandler::status(SOCKET client_socket, User& curUser){
        int cnt = 0, cap = 0;
        cnt = curUser.lettersCount();
        cap = curUser.overallSize();
        string answer;
        answer.append(responses[1]+to_string(cnt)+" "+to_string(cap)+"\r\n");
        sendLine(client_socket, answer.c_str());
    }

    void MailHandler::getList(SOCKET client_socket, string req, User& curUser){
        string request = req;
        string answer;
        int cnt = 0, cap = 0;
        cnt = curUser.lettersCount();
        cap = curUser.overallSize();
        if (request.length() < 5){
            answer.append(responses[1] + to_string(cnt) + " messages (" +
to_string(cap) + " octets)\r\n");
            for (int i = 1; i <= cnt; i++){
                answer.append(to_string(i) + " " + to_string(curUser.letterSize(i)) +
"\r\n");
            }
        }
        else {
            request.erase(request.begin(), request.begin() + 5);
            int num = stoi(request);
            if (num <= cnt && num>0) answer.append(responses[1] + to_string(num) + " "
+ to_string(curUser.letterSize(num)) + "\r\n");
            else answer.append(responses[0]);
        }
        sendLine(client_socket, answer.c_str());
    }

    void MailHandler::getMessage(SOCKET client_socket, string req, User& curUser){
        string request = req;
        string answer;
        int cnt = 0, cap = 0, num=0;
        cnt = curUser.lettersCount();
        if (req.size() > 5){
            string temp = request.substr(5, 1);
            try{
                num = stoi(temp, nullptr);
            }
            catch (invalid_argument){
                cout << "Error! Invalid argument!" << endl;
                answer.append("-ERR Invalid argument!");
            }
            if (num <= cnt && num > 0) {
                Letter let = Letter();
                let = curUser.getLetter(num);
                if (let.getMarker() == false){
                    cout << "number is " << num << endl;
                    answer.append(responses[1] +
to_string(curUser.letterSize(num)) + " octets\r\n");
                    answer.append("From: " + let.getFrom());
                    answer.append("\r\nTo: " + let.getTo());
                    answer.append("\r\nSubject: " + let.getSubject());
                    answer.append("\r\nData: " + let.getData() + "\r\n.");
                }
            }
        }
    }

```

```

        else answer.append("-ERR message " + to_string(num) + " was
deleted\r\n");
    }
    else answer.append(responses[0]);
}
else answer.append(responses[0]);
sendLine(client_socket, answer.c_str());
}

void MailHandler::deleteMessage(SOCKET client_socket, string req, User& curUser){
    string request = req;
    string answer;
    int cnt = 0, num = 0;
    cnt = curUser.lettersCount();
    string temp = request.substr(5, 1);
    try{
        num = stoi(temp, nullptr);
    }
    catch (invalid_argument){
        cout << "Error! Invalid argument!" << endl;
        answer.append("-ERR Invalid argument!");
    }
    if (num <= cnt && num > 0){
        if (curUser.getLetter(num).getMarker() == false){
            curUser.deleteLetter(num);
            answer.append(responses[11] + to_string(num) + " deleted\r\n");
        }
        else answer.append("-ERR message " + to_string(num) + " already
deleted\r\n");
    }
    else answer.append(responses[0]);
    sendLine(client_socket, answer.c_str());
}

void MailHandler::reset(SOCKET client_socket, User& curUser){
    string answer;
    int cnt = 0;
    cnt = curUser.lettersCount();
    for (int i = 1; i <= cnt; i++){
        if (curUser.getLetter(i).getMarker() == true){
            //curUser.getLetter(i).setMarker(false);
            curUser.recoverLetter(i);
        }
    }
    answer.append(responses[13] + to_string(cnt) + " messages\r\n");
    sendLine(client_socket, answer.c_str());
}

void MailHandler::disconnect(SOCKET client_socket, int& status, User& curUser){
    string answer;
    int cnt = 0;
    cnt = curUser.lettersCount();
    for (int i = cnt; i > 0; i--){
        if (curUser.getLetter(i).getMarker() == true){
            curUser.eraseLetter(i-1);
        }
    }
    int fin = curUser.lettersCount();
    if (fin == 0) answer.append(responses[20]);
    else answer.append(responses[19] + to_string(fin) + " messages left");
    curUser.setOffline();
    status = 0;
    sendLine(client_socket, answer.c_str());
}

```

```

//возвращает индекс первого юзера в массиве, имя которого (возвращаемое getName())
совпало с аргументом, если таких нет, то -1
int MailHandler::checkUser(const string &name){
    for (int i = 0; i < users.size(); ++i){
        if (users[i].getName() == name){
            return i;
        }
    }
    return -1;
}
}
thread MailHandler::createThread(LPVOID param){
    //при использовании из main.cpp param - client_socket отсюда
    return thread(&MailHandler::clientHandler, this, param);
}

```

Листинг файла util.cpp:

```

#include "headers/util.h"

int recvn(SOCKET fd, char *bp, size_t len) {
    return recv(fd, bp, len, MSG_WAITALL);
}

int sendn(SOCKET s, char* buf, int lenbuf, int flags) {
    int bytesSended = 0; //
    int n; //
    while (bytesSended < lenbuf) {
        n = send(s, buf + bytesSended, lenbuf - bytesSended, flags);
        if (n < 0) {
            cout << ("Error with send in sendn\n");
            break;
        }
        bytesSended += n;
    }
    return (n == -1 ? -1 : bytesSended);
}

int recvLine(SOCKET sock, char* buffer, int buffSize) { //функция приема сообщения
    char* buff = buffer; //указатель на начало внешнего буфера
    char* currPosPointer; //указатель для работы со временным буфером
    int count = 0; //число прочитанных символов (без удаления из буфера сокета)
    char tempBuf[100]; //временный буфер для приема
    char currChar; //текущий анализируемый символ (ищем разделитель)
    int tmpcount = 0;
    while (--buffSize > 0){
        if (--count <= 0) {
            recvn(sock, tempBuf, tmpcount);
            count = recv(sock, tempBuf, sizeof (tempBuf), MSG_PEEK);
            if (count <= 0) { return count; }
            currPosPointer = tempBuf;
            tmpcount = count;
        }
        currChar = *currPosPointer++;
        *buffer++ = currChar;
        if (currChar == '\n') {
            *(buffer - 1) = '\0';
            recvn(sock, tempBuf, tmpcount - count + 1);
            return buffer - buff - 1;
        }
    }
    return -1;
}

int sendLine(int sock, const char* str) {

```



```
char tempBuf[MAX_STR_LEN];
strcpy(tempBuf, str);
if (tempBuf[strlen(tempBuf) - 1] != '\n')
    strcat(tempBuf, "\n");
return sendn(sock, tempBuf, strlen(tempBuf), 0);
}
```