

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

Отчет по лабораторным работам

Дисциплина: Сети и телекоммуникации

Тема: Программирование клиент/серверных приложений прикладного протокола

Выполнил
студент гр. 43501/3

_____ Зобков Д. А.
(подпись)

Преподаватель

_____ Зозуля А. В.
(подпись)

“__” _____ 2017 г.

Санкт-Петербург
2017 г.

СОДЕРЖАНИЕ

1	Цель работы	3
2	Программа работы	3
3	Индивидуальное задание	3
3.1	Прокси POP3	3
3.1.1	Анализ протокола	4
3.1.2	Архитектура приложения	5
3.1.3	Тестирование	5
3.2	FTP клиент (пассивный режим работы)	8
3.2.1	Анализ протокола	9
3.2.2	Архитектура приложения	11
3.2.3	Тестирование	11
4	Вывод	13
	Приложение А Исходный код прокси POP3	14
	Приложение Б Исходный код FTP-клиента с пассивным режимом работы	20

1 Цель работы

Ознакомление с заданными протоколами по RFC и их реализация.

2 Программа работы

1. Ознакомление со протоколом (по RFC), определение набора необходимых команд для реализации;
2. Создание базовой версии клиентского приложения/прокси, тестирование и отладка;
3. Разработка GUI для клиентского приложения;
4. Изучение средств взаимодействия GUI с потоками;
5. Создание клиентского приложения с GUI, тестирование и отладка.

3 Индивидуальное задание

3.1 Прокси POP3

Задание: разработать сервер-посредник для протокола POP3, функционирующий под управлением операционных систем Windows или Linux.

Основные возможности приложения:

1. Подключение нескольких почтовых клиентов;
2. Разбор аутентификационных данных и получение имени POP3-сервера;
3. Трансляция команд от клиента к серверу и обратно;
4. Протоколирование обмена между почтовым клиентом и сервером.

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

- USER — получение от клиента идентификационных данных пользователя вместе с адресом реального сервера;
- PASS — получение от клиента пароля пользователя;
- STAT — отправка клиенту состояния почтового ящика;
- LIST — отправка клиенту списка сообщения почтового ящика;
- RETR — отправка клиенту текста сообщения;
- DELE — пометка сообщения на удаление;
- TOP — отправка клиенту первых нескольких строк сообщения;
- UIDL — выдача уникального идентификатора сообщения;
- RSET — сброс всех пометок на удаление сообщений;
- QUIT — удаление всех помеченных сообщений и завершение сеанса.

Методика тестирования. Для тестирования приложения используется стандартный клиент электронной почты, имеющийся в лаборатории (Mozilla Thunderbird,

The Bat, MS Outlook Express). Параметры учетной записи настраиваются на сокет, используемый разработанным приложением.

В качестве почтового сервера можно использовать лабораторные POP3-серверы или бесплатные почтовые службы (например, www.mail.ru, www.yandex.ru и т. п.). Спроектированный прокси-сервер проверяется на выполнение набора основных функций протокола POP3.

3.1.1 Анализ протокола

Основные команды протокола приведены в табл. 3.1. Необходимые команды и их функции описаны в техническом задании (см. разд. 3.1 на предыдущей странице).

Таблица 3.1. Команды протокола POP3

Имя	Аргументы	Ограничения	Возможные ответы
USER	[имя]	—	+OK name is a valid mailbox -ERR never heard of mailbox name
PASS	[пароль]	Работает после успешной передачи имени почтового ящика	+OK maildrop locked and ready -ERR invalid password -ERR unable to lock maildrop
DELE	[сообщение]	Доступна после успешной аутентификации	+OK message deleted -ERR no such message
LIST	[сообщение]	Доступна после успешной аутентификации	+OK scan listing follows -ERR no such message
RETR	[сообщение]	Доступна после успешной аутентификации	+OK message follows -ERR no such message
RSET	—	Доступна после успешной аутентификации	+OK
STAT	—	Доступна после успешной аутентификации	+OK [кол-во писем] [размер в октетах]
TOP	[сообщение] [количество строк]	Доступна после успешной аутентификации	+OK n octets -ERR no such message
QUIT	—	—	+OK Bye

Прокси-сервер — сервер-посредник, позволяющий клиентам совершать косвенные запросы к сетевым службам. Сначала клиент подключается к прокси-

серверу и запрашивает какой-либо ресурс (в данном случае — электронные письма), расположенный на другом сервере. Затем прокси-сервер подключается к данному серверу и получает ресурс у него. В некоторых случаях запрос клиента или ответ сервера может быть модифицирован прокси-сервером. Прокси-сервер позволяет защитить клиента от некоторых сетевых атак и помогает сохранить его анонимность.

3.1.2 Архитектура приложения

Реализованное приложение состоит из трех классов: *Main*, *MainThread* и *ProxyThread*.

Класс *Main* создает поток класса *MainThread*. *MainThread* реализует мультипоточный сервер с возможностью отображать список подключенных клиентов, принудительного отключения клиентов и отключения самого прокси-сервера, а также создание для каждого клиента потока класса *ProxyThread* для выполнения основного функционала прокси-сервера.

Класс *ProxyThread* получает из команды USER email пользователя, из которого извлекается целевой домен, к которому прокси-сервер подключается по SSL-соединению. После этого прокси-сервер переходит в штатный режим: пересылает команды от клиента к серверу и ответы сервера клиенту с протоколированием обмена.

3.1.3 Тестирование

Тестирование прокси-сервера проводилось с помощью клиента электронной почты Mozilla Thunderbird и адресов электронной почты доменов yandex.ru, gmail.com и mail.ru.

Пример настройки Thunderbird для работы с прокси-сервером представлен на рис. 3.1.

	Server hostname	Port	SSL	Authentication
Incoming: POP3	127.0.0.1	12555	None	Normal password
Outgoing: SMTP	smtp.mail.ru	465	SSL/TLS	Normal password

Username: Incoming: zobkov.d@mail.ru Outgoing: zobkov.d@mail.ru

Рис. 3.1. Настройка Thunderbird для работы с прокси-сервером

Протокол подключения к почтовому серверу через прокси представлен в стандартном потоке вывода (листинг 3.1).

```
1 New connection: 127.0.0.1:55005
2 127.0.0.1:55005 client -> AUTH
  -ERR What?
4 127.0.0.1:55005 client -> CAPA
  +OK Capability list follows
6 USER
  .
8 127.0.0.1:55005 client -> USER zobkov.d@mail.ru
  Connected to pop.mail.ru:995
10 pop.mail.ru:995 server -> +OK

12 127.0.0.1:55005 client -> PASS *****
  pop.mail.ru:995 server -> +OK Welcome!
14
  127.0.0.1:55005 client -> QUIT
16 pop.mail.ru:995 server -> +OK POP3 server at mail.ru signing off

18 Disconnected: 127.0.0.1:55005 (Socket closed)
  Socket closed.
```

Листинг 3.1. Результат подключения к серверу

Попытка получения всех писем с почтового ящика представлена в листинге 3.2 (для краткости приводятся только первые 5 сообщений).

```
1 New connection: 127.0.0.1:55180
2 127.0.0.1:55180 client -> AUTH
  -ERR What?
4 127.0.0.1:55180 client -> CAPA
  +OK Capability list follows
6 USER
  .
8 127.0.0.1:55180 client -> USER zobkov.d@mail.ru
  Connected to pop.mail.ru:995
10 pop.mail.ru:995 server -> +OK

12 127.0.0.1:55180 client -> PASS *****
  pop.mail.ru:995 server -> +OK Welcome!
14
  127.0.0.1:55180 client -> STAT
16 pop.mail.ru:995 server -> +OK 67 4069642

18 127.0.0.1:55180 client -> LIST
  pop.mail.ru:995 server -> +OK 67 messages (4069642 octets)
20 1 9811
  2 8537
22 3 8537
  4 47758
24 5 10388
  .
26
  127.0.0.1:55180 client -> UIDL
```

```

28 pop.mail.ru:995 server -> +OK 67 messages (4069642 octets)
   1 145392614473
30  2 1456503272707
   3 1456505765206
32  4 1464150824129
   5 1465206601280
34  .

36 127.0.0.1:55180 client -> RETR 1
pop.mail.ru:995 server -> +OK 9811 octets <...>
38 127.0.0.1:55180 client -> RETR 2
pop.mail.ru:995 server -> +OK 8537 octets <...>
40 127.0.0.1:55180 client -> RETR 3
pop.mail.ru:995 server -> +OK 8537 octets <...>
42 127.0.0.1:55180 client -> RETR 4
pop.mail.ru:995 server -> +OK 47758 octets <...>
44 127.0.0.1:55180 client -> RETR 5
pop.mail.ru:995 server -> +OK 10388 octets <...>
46 127.0.0.1:55180 client -> QUIT
pop.mail.ru:995 server -> +OK POP3 server at mail.ru signing off

48 Disconnected: 127.0.0.1:55180 (Socket closed)
50 Socket closed.

```

Листинг 3.2. Получение всех сообщений (вывод сокращен до 5 сообщений)

Пример полученного сообщения представлен на рис. 3.2.

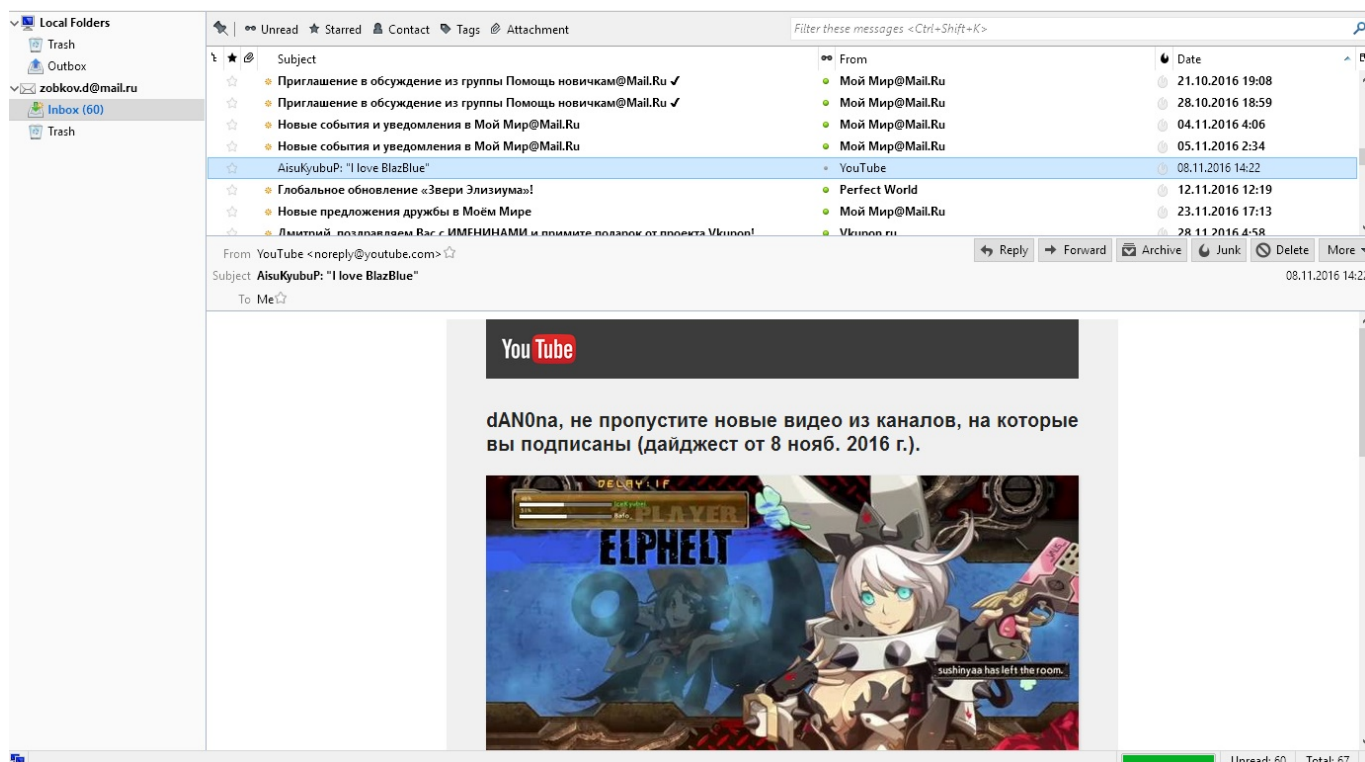


Рис. 3.2. Пример полученного сообщения






Попытка удаления сообщения приведена в листинге 3.3 на следующей странице и на рис. 3.3 на следующей странице.

```

1 127.0.0.1:55712 client -> DELE 37
2 pop.mail.ru:995 server -> +OK message 37 deleted

```

Листинг 3.3. Удаление сообщения

<input type="checkbox"/>		Мой Мир@mail.ru	Новые предложения дружбы в Моём Мире	Виктория Филиппова и еще 1 человек хочет подружиться с вами Виктория Филиппова Г	23.11.16
<input type="checkbox"/>		Perfect World	Глобальное обновление «Звери Элизимуа»!	Если письмо отображается некорректно, перейдите по этой ссылке Уважаемый пользо	12.11.16
<input type="checkbox"/>		YouTube	AisuKyubuP: "I love BlazBlue"	dAN0na, не пропустите новые видео из каналов, на которые вы подписаны (дайджест от 8 нояб. 2016	08.11.16
<input type="checkbox"/>		Мой Мир@mail.ru	Новые события и уведомления в Мой Мир@Mail.Ru	Возможно, вы знакомы Еще Алексей Говорухин 42 года Подружиться Scorp23	05.11.16
<input type="checkbox"/>		Мой Мир@mail.ru	Новые события и уведомления в Мой Мир@Mail.Ru	Возможно, вы знакомы Еще Алексей Говорухин 42 года Подружиться Scorp23	04.11.16





<input type="checkbox"/>		Мой Мир@Mail.Ru	Новые предложения дружбы в Моём Мире	Виктория Филиппова и еще 1 человек хочет подружиться с вами Виктория Филиппова Г	23.11.16
<input type="checkbox"/>		Perfect World	Глобальное обновление «Звери Элизимуа»!	Если письмо отображается некорректно, перейдите по этой ссылке Уважаемый пользо	12.11.16
<input type="checkbox"/>		Мой Мир@Mail.Ru	Новые события и уведомления в Мой Мир@Mail.Ru	Возможно, вы знакомы Еще Алексей Говорухин 42 года Подружиться Scorp23	05.11.16
<input type="checkbox"/>		Мой Мир@Mail.Ru	Новые события и уведомления в Мой Мир@Mail.Ru	Возможно, вы знакомы Еще Алексей Говорухин 42 года Подружиться Scorp23	04.11.16

Рис. 3.3. Удаление сообщения на сервере (до и после)

Аналогичным образом были протестированы оставшиеся команды, требуемые по заданию. Также была проверена возможность подключения нескольких клиентов к разным серверам и их параллельная работа и продемонстрирована преподавателю.

3.2 FTP клиент (пассивный режим работы)

Задание: Разработать приложение для операционных систем семейства Windows или Linux, обеспечивающее функции клиента протокола FTP.

Основные возможности: Приложение должно реализовывать следующие функции:

1. Подключение к указанному серверу;
2. Получение списка файлов в каталоге;
3. Навигация по системе каталогов;
4. Копирование файла на сервер;
5. Копирование файла с сервера;
6. Создание и удаление каталогов;
7. Удаление файлов;
8. Обеспечение работы со ссылками на файлы и каталоги;
9. Протоколирование соединения сервера с клиентом.

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

- USER — передача серверу идентификационной информации пользователя;
- PASS — передача серверу пароля пользователя;

- CWD — смена текущего каталога сервера;
- MKD — создание каталога;
- RMD — удаление каталога;
- DELE — удаление файла на сервере;
- PASV — переключение сервера в пассивный режим;
- LIST — получение списка файлов в текущем каталоге сервера в расширенном формате;
- NLST — получение списка файлов в текущем каталоге сервера в сокращенном формате;
- RETR — получение файла с сервера;
- STOR — посылка файла на сервер;
- QUIT — завершение сеанса.

Настройки приложения. Разработанное приложение должно обеспечивать настройку следующих параметров:

1. IP-адрес или доменное имя файлового сервера;
2. Имя пользователя;
3. Пароль пользователя.

Методика тестирования. Для тестирования приложения следует использовать файловые серверы, имеющиеся в лаборатории, а также открытые файловые серверы, имеющиеся в сети Internet (ftp://ftp.funet.fi, ftp://ftp.relcom.ru и т.п.).

Для разработанных приложений проверяется возможности подключения к серверу, копирования файла на сервер, копирования файла с сервера, удаления файла на сервере, переименовании файла на сервере, создания каталога, удаления каталога.

3.2.1 Анализ протокола

Команды протокола FTP, реализованные в клиенте, приведены на табл. 3.2 на следующей странице.

Таблица 3.2. Команды протокола FTP

Имя	Аргументы	Описание
USER	[имя]	Имя пользователя
PASS	[пароль]	Пароль пользователя
PWD	—	Вывод текущей директории
CWD	[директория]	Смена директории
MKD	[имя]	Создание новой директории
RMD	[директория]	Удаление директории
DELE	[имя файла]	Удаление файла
PASV	—	Перевод сервера в пассивный режим и получение IP и порта для получения данных
LIST	— или [директория]	Список файлов и директорий в стиле UNIX-команды <code>ls -l</code>
NLST	[сообщение]	Аналогично LIST, но вывод в сокращенном формате — только имена файлов/директорий
RETR	[имя файла]	Скачать указанный файл
STOR	[имя файла]	Загрузить файл на сервер
SYST	—	Возвращает тип системы сервера
TYPE	[тип]	Выбор режима передачи
SIZE	[имя файла]	Возвращает размер файла
QUIT	—	Отключение

Сервер отвечает на команды клиента трехзначными кодами.

Первая цифра отвечает за один из трёх исходов: успех, отказ или указание на ошибку либо неполный ответ.

- 2xx — Успешный ответ;
- 4xx/5xx — Команда не может быть выполнена;
- 1xx/3xx — Ошибка или неполный ответ.

Вторая цифра определяет тип ошибки:

- x0z — Синтаксическая;
- x1z — Информация. Соответствует информационному сообщению;
- x2z — Соединения. Сообщение относится к управляющему соединению либо к соединению данных;
- x3z — Соответствует сообщениям об аутентификации пользователя и его правах;
- x4z — Не определено;
- x5z — Файловая система. Соответствует сообщению о состоянии файловой системы.

Третья цифра окончательно специфицирует ошибку.

3.2.2 Архитектура приложения

Реализованное приложение состоит из четырех классов: *Gui*, *FTPClient*, *FTPFile* и *FTPResponse*.

Объект класса *FTPFile* содержит информацию о имени, размере и типе (файл, директория или символическая ссылка) файла, полученного с помощью команды LIST.

Объект класса *FTPResponse* содержит информацию о трехзначном коде ответа сервера и тексте сообщения, а также о наличии в ответе сервера разделителя „–“ вместо пробела, сообщающего о наличии дополнительных строк ответа сервера с данным кодом.

Класс *Gui* реализует графический интерфейс приложения с использованием библиотеки Swing.

Класс *FTPClient* реализует основной функционал приложения: подключение к серверу с аутентификацией, обеспечение работы в пассивном режиме, выполнение команд и протоколирование команд и ответов сервера.

3.2.3 Тестирование

Внешний вид графического интерфейса представлен на рис. 3.4.

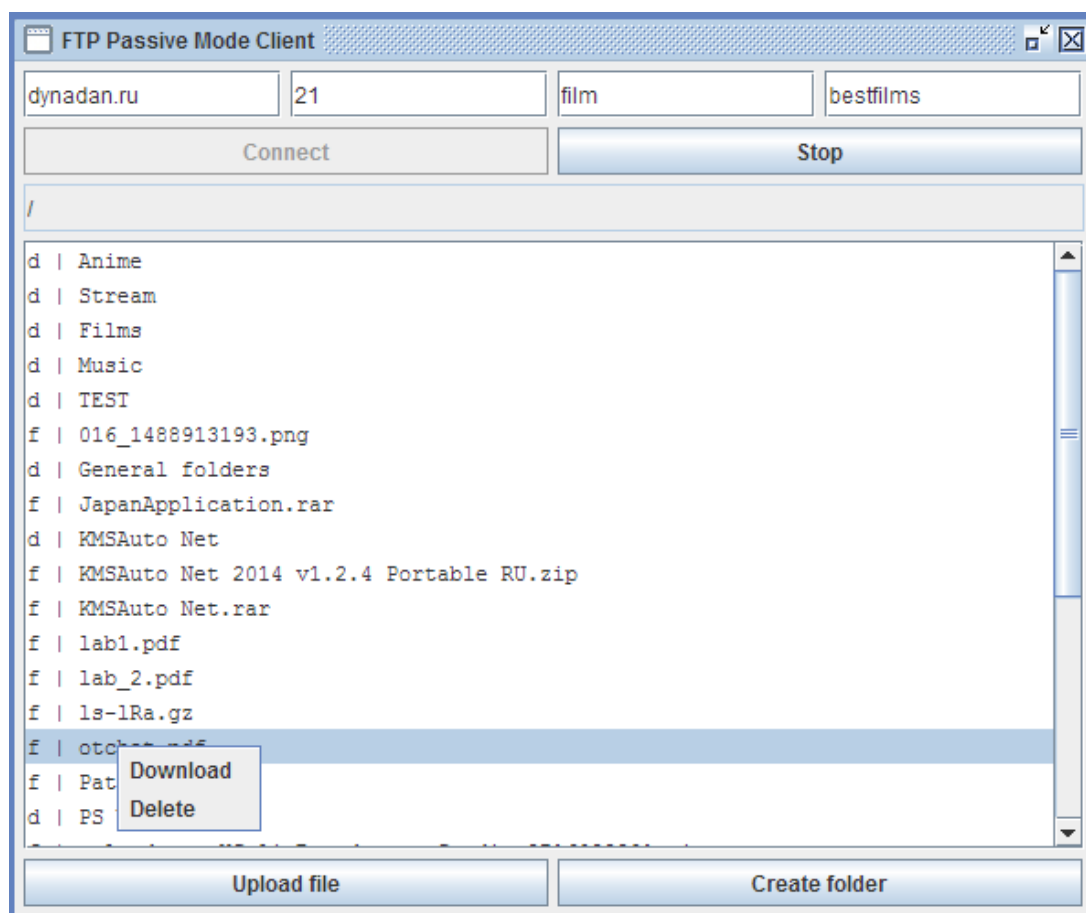


Рис. 3.4. Графический интерфейс

После подключения к FTP-серверу в стандартном потоке вывода отобразился процесс подключения (листинг 3.4).

```
1 <<< 220-FileZilla Server 0.9.55 beta
2 <<< 220-written by Tim Kosse (tim.kosse@filezilla-project.org)
  <<< 220 Please visit https://filezilla-project.org/
4 >>> USER film
  <<< 331 Password required for film
6 >>> PASS bestfilms
  <<< 230 Logged on
8 >>> SYST
  <<< 215 UNIX emulated by FileZilla
10 >>> PWD
  <<< 257 "/" is current directory.
12 >>> TYPE I
  <<< 200 Type set to I
14 >>> PASV
  <<< 227 Entering Passive Mode (31,134,157,210,59,85)
16 <<< 31.134.157.210:15189
  >>> LIST /
18 <<< 150 Opening data channel for directory listing of "/"
  <<< 226 Successfully transferred "/"
```

Листинг 3.4. Результат подключения к серверу

Попытка перехода на другую директорию отражена в листинге 3.5.

```
1 >>> CWD TEST
2 <<< 250 CWD successful. "/TEST" is current directory.
  >>> PWD
4 <<< 257 "/TEST" is current directory.
  >>> SYST
6 <<< 215 UNIX emulated by FileZilla
  >>> PWD
8 <<< 257 "/TEST" is current directory.
  >>> TYPE I
10 <<< 200 Type set to I
  >>> PASV
12 <<< 227 Entering Passive Mode (31,134,157,210,59,70)
  <<< 31.134.157.210:15174
14 >>> LIST /TEST
  <<< 150 Opening data channel for directory listing of "/TEST"
16 <<< 226 Successfully transferred "/TEST"
```

Листинг 3.5. Переход в папку тест

Аналогичным образом были протестированы все поддерживаемые команды (за исключением NLST и SIZE, т. к. они не используются графическим интерфейсом приложения, поэтому были протестированы при разработке). Также была проверена возможность параллельного скачивания и загрузки нескольких файлов большого размера и продемонстрирована преподавателю.

4 Вывод

В данных лабораторных работах были изучены средства создания клиентских/серверных сокетов TCP, средства записи/чтения из сокетов и средства многопоточной работы в Java. Были разработаны прокси POP3 и FTP-клиент с работой в пассивном режиме, для которого был создан графический интерфейс с использованием библиотеки Swing.

Разработанные приложения были протестированы в реальных условиях. Для тестирования POP3 прокси использовался почтовый клиент Mozilla Thunderbird и адреса электронной почты доменов yandex.ru, gmail.com и mail.ru. Для тестирования FTP-клиента использовались как FTP-сервер, созданный в локальной сети, что позволило протестировать действия, требующие прав на выполнение (создание/удаление папок, загрузка/удаление файлов), так и открытые файловые серверы в сети Интернет.

В настоящее время существует множество сторонних библиотек, упрощающих разработку данных протоколов до нескольких строк. Например, с использованием библиотеки Apache можно значительно облегчить разработку POP3 и FTP клиентов. Также можно воспользоваться библиотекой Javamail для упрощенной разработки POP3 и IMAP клиентов.

Разработка сетевых приложений при использовании языка Java значительно упрощается по сравнению с разработкой на языках C/C++. Это вызвано упрощенной работой с сокетами, большим количеством существующих библиотек, меньшим количеством отслеживаемых ошибок.

Приложение А Исходный код прокси POP3

```
1 public class Main {
2     public static void main(String[] args) {
3         MainThread st=new MainThread();
4         new Thread(st).start();
5     }
6 }
```

Листинг А.1. Main.java

```
1 import java.io.IOException;
2 import java.net.InetAddress;
3 import java.net.ServerSocket;
4 import java.net.Socket;
5 import java.util.Map;
6 import java.util.Scanner;
7 import java.util.concurrent.ConcurrentHashMap;
8
9 public class MainThread implements Runnable {
10     private int SERVER_PORT = 12555;
11     private String SERVER_IP = "127.0.0.1";
12
13     private ServerSocket serverSocket = null;
14     private boolean isStopped = false;
15
16     private ConcurrentHashMap<ProxyThread, String> mClients;
17
18     private Scanner s = new Scanner(System.in);
19
20     public void run() {
21         Runnable server = () -> {
22             while(!isStopped){
23                 String input = s.next();
24                 switch (input) {
25                     case "l":
26                         showClients();
27                         break;
28                     case "k":
29                         int id=Integer.parseInt(s.next());
30                         deleteClientById(id);
31                         break;
32                     case "q":
33                         stop();
34                         break;
35                 }
36             }
37         };
38         new Thread(server).start();
39         mClients = new ConcurrentHashMap<>();
40         openServerSocket();
41
42         while(!isStopped){
43             Socket clientSocket;
```

```

44         try {
45             clientSocket = this.serverSocket.accept();
46         } catch (IOException e) {
47             System.out.println("Can't accept socket.");
48             return;
49         }
50         addClient(clientSocket);
51     }
52 }

54 private void addClient(Socket clientSocket){
55     ProxyThread CT=new ProxyThread(clientSocket, this);
56     CT.start();
57     synchronized (mClients) {
58         mClients.put(CT, "");
59     }
60 }

62 private synchronized void deleteClientById(int id){
63     int i=1;
64     for (Map.Entry<ProxyThread, String> entry : mClients.entrySet()
65 ) {
66         ProxyThread key = entry.getKey();
67         if(i==id){
68             key.finish();
69             mClients.remove(key);
70             return;
71         }
72         i++;
73     }
74     System.out.println("Wrong ID");
75 }

76 public synchronized void removeClient(ProxyThread CT){
77     synchronized (mClients) {
78         mClients.remove(CT);
79     }
80 }

82 private synchronized void showClients(){
83     int i=1;
84     for (Map.Entry<ProxyThread, String> entry : mClients.entrySet()
85 ) {
86         ProxyThread key = entry.getKey();
87         System.out.println(i+"|"+key.getAddr());
88         i++;
89     }
90     if(mClients.size() == 0) System.out.println("Empty");
91 }

92 public synchronized void stop(){
93     for (Map.Entry<ProxyThread, String> entry : mClients.entrySet()
94 ) {
95         ProxyThread key = entry.getKey();

```

```

    key.finish();
    mClients.remove(key);
}
96
    this.isStopped = true;
    try {
98
        this.serverSocket.close();
        System.out.println("Server stopped.");
100
    } catch (IOException e) {
        System.out.println("Error closing server (" + e + ")");
102
    }
104
}

106
private void openServerSocket() {
    try {
108
        this.serverSocket = new ServerSocket(this.SERVER_PORT, 0,
        InetAddress.getBy_name(SERVER_IP));
110
    } catch (IOException e) {
        throw new RuntimeException("Cannot open port " +
        SERVER_PORT, e);
112
    }
    System.out.println("Server started.") ;
114
}
}

```

ЛИСТИНГ А.2. MainThread.java

```

1 import javax.net.ssl.SSLSocketFactory;
2 import java.io.*;
3 import java.net.Socket;
4 import java.net.SocketAddress;
5 import java.util.logging.Level;
6 import java.util.logging.Logger;

7
8 public class ProxyThread extends Thread {
9     private Socket clientSocket;
10    private Socket serverSocket;
11    private SocketAddress clientAddr;
12    private MainThread ST;

13
14    private String inputLine;

15
16    private static int SERVER_PORT=995;
17    private String SERVER_IP;

18
19    public ProxyThread(Socket clientSocket, MainThread ST) {
20        this.clientSocket = clientSocket;
21        this.ST=ST;
22        clientAddr = this.clientSocket.getRemoteSocketAddress();
23    }

24
25    public void run() {
26        try{
27            System.out.println("New connection: " + getAddr());
28

```



```

        PrintWriter outClient = new PrintWriter(clientSocket.
getOutputStream(), true);
30        BufferedReader inClient = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

32        SERVER_IP = getServerAddress(inClient, outClient);

34        SSLSocketFactory fac = (SSLSocketFactory) SSLSocketFactory.
getDefault();
        serverSocket = fac.createSocket(SERVER_IP, SERVER_PORT);
36

        PrintWriter outServer = new PrintWriter(serverSocket.
getOutputStream(), true);
38        BufferedReader inServer = new BufferedReader(new
InputStreamReader(serverSocket.getInputStream()));

40        Thread t = new Thread(() -> responseRedirect(inServer,
outClient));
        t.start();

42        // Get +OK respond from server and send USER
44        inServer.readLine();
        outServer.println(inputLine);

46        System.out.println("Connected to " + SERVER_IP + ":" +
SERVER_PORT);

48        requestRedirect(inClient, outServer);
50    } catch (IOException e) {
        ST.removeClient(this);
52        System.out.println("Disconnected: " + getAddr() + " (" + e.
getMessage() + ")");
        try {
54            if(serverSocket != null) serverSocket.close();
            clientSocket.close();
56            System.out.println("Socket closed.");
        } catch (IOException ex) {
58            Logger.getLogger(ProxyThread.class.getName()).log(Level
.SEVERE, null, ex);
        }
60    }
}

62    private void requestRedirect(BufferedReader in, PrintWriter outEcho
) throws IOException {
64        while ((inputLine = in.readLine()) != null) {
            outEcho.println(inputLine);
66            if(inputLine.contains("PASS")) inputLine = "PASS *****
";
            System.out.println(getAddr() + "\tclient -> " + inputLine);
68        }
        throw new IOException("Dropped from client");
70    }

```

```

72     private void responseRedirect(BufferedReader inEcho, PrintWriter
out) {
    StringBuilder response = new StringBuilder();
74     String line;
    try {
76         while((line = inEcho.readLine()) != null){
            response.append(line);
78             response.append("\r\n");

            if(inputLine.matches("(RETR|TOP|LIST|UIDL) (.*)") && !
80 line.equals(".")) continue;
            if(!inEcho.ready()){
82                 String stdoutString = response.toString();
                if(inputLine.matches("(RETR|TOP) (.*)"))
84                     stdoutString = stdoutString.split("\r\n", 2)[0]
+ " <...>";
                System.out.println(SERVER_IP + ":" + SERVER_PORT +
"\tserver -> " + stdoutString);
86                 out.println(response.toString());

                if(inputLine.contains("PASS") && response.toString
88 ().contains("-ERR")) finish();
                response.setLength(0);
90             }

            if(inputLine.equals("QUIT") && inputLine != null)
92 finish();
        }
94     } catch (IOException | NullPointerException e) {}
    }

96     private String getServerAddress(BufferedReader in, PrintWriter out)
throws IOException {
98         inputLine = "";
        String ip = null;
100         String proxyResponse = "+OK POP3";
        boolean work = true;

102         out.println(proxyResponse);

104         while(work && (inputLine = in.readLine()) != null){
            System.out.println(getAddr() + "\tclient -> " + inputLine);
106             if(!inputLine.contains("USER")){
                if(inputLine.contains("QUIT")) {
108                     proxyResponse = "+OK Bye";
                    System.out.println("proxy -> " + proxyResponse);
110                     out.println(proxyResponse);
                    finish();
                }else{
112                     if (inputLine.contains("AUTH")) proxyResponse = "-
ERR What?";
                    if (inputLine.contains("CAPA"))
114                         proxyResponse = "+OK Capability list follows\r\
nUSER\r\n.";

```

```

118         System.out.println(proxyResponse);
        out.println(proxyResponse);
        }
120     }else {
        String[] login = inputLine.trim().split("@");
122         try {
            ip = "pop." + login[1];
124         }catch (ArrayIndexOutOfBoundsException e){
            proxyResponse = "-ERR Wrong login, please use full
email address";
126             out.println(proxyResponse);
            finish();
128         }
        work = false;
130     }
    }
132
    return ip;
134 }

136 public void finish(){
    try{
138         if(serverSocket != null) serverSocket.close();
        clientSocket.close();
140     }catch(final IOException e){
        System.out.println("Can't close socket (" + e.getMessage()
+ ")");
142     }
    }
144
    public String getAddr(){
146         return clientAddr.toString().replace("/", "");
    }
148 }

```

Листинг A.3. ProxyThread.java

Приложение Б Исходный код FTP-клиента с пассивным режимом работы

```
1 import ftp.FTPClient;
2 import ftp.FTPFile;

4 import javax.swing.*;
import java.awt.*;
6 import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
8 import java.io.IOException;
import java.net.SocketException;
10 import java.util.regex.Matcher;
import java.util.regex.Pattern;

12
public class Gui extends JFrame {
14     private JButton stopButton, startButton, uploadButton, mkdirButton;
    private JTextField hostField, portField, loginField, passwordField,
    pathField;
16     private FTPClient ftp;
    private String dir;

18
    public Gui() {
20         super("FTP Passive Mode Client");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

22
        dir = "/";

24
        JPanel mainPanel = new JPanel();
26         mainPanel.setLayout(new BorderLayout(5, 5));
        mainPanel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5)
    );

28
        JPanel connectPanel = new JPanel();
30         connectPanel.setLayout(new GridLayout(3, 4, 5, 5));
        mainPanel.add(connectPanel, BorderLayout.NORTH);

32
        JPanel fieldPanel = new JPanel();
34         fieldPanel.setLayout(new GridLayout(1, 4, 5, 0));
        connectPanel.add(fieldPanel, BorderLayout.NORTH);

36
        hostField = new JTextField("ftp.neva.ru");
38 //         hostField = new JTextField("ftp.maths.tcd.ie");
        fieldPanel.add(hostField);
40         portField = new JTextField("21");
        fieldPanel.add(portField);
42         loginField = new JTextField("anonymous");
        fieldPanel.add(loginField);
44         passwordField = new JTextField("");
        fieldPanel.add(passwordField);

46
        JPanel buttonsPanel1 = new JPanel();
48         buttonsPanel1.setLayout(new GridLayout(1, 2, 5, 0));
        connectPanel.add(buttonsPanel1, BorderLayout.CENTER);
```

```

50     JPanel pathPanel = new JPanel();
51     pathPanel.setLayout(new GridLayout(1, 1, 5, 0));
52     connectPanel.add(pathPanel, BorderLayout.SOUTH);
53
54     pathField = new JTextField("/");
55     pathField.setEditable(false);
56     pathPanel.add(pathField);
57
58     JPanel buttonsPanel2 = new JPanel();
59     buttonsPanel2.setLayout(new GridLayout(1, 2, 5, 0));
60     mainPanel.add(buttonsPanel2, BorderLayout.SOUTH);
61
62     final DefaultListModel listModel = new DefaultListModel();
63
64     final JList listField = new JList(listModel);
65     listField.setFocusable(false);
66     listField.setEnabled(false);
67     listField.setSelectionMode(ListSelectionModel.SINGLE_SELECTION)
68 ;
69
70     listField.setFont(new Font("monospaced", Font.PLAIN, 12));
71     mainPanel.add(new JScrollPane(listField), BorderLayout.CENTER);
72
73     listField.addMouseListener(new MouseAdapter() {
74         public void mouseClicked(MouseEvent e) {
75             JList tempList = (JList) e.getSource();
76             if (SwingUtilities.isRightMouseButton(e)) {
77                 tempList.setSelectedIndex(tempList.locationToIndex(
78 e.getPoint()));
79                 String value = tempList.getSelectedValue().toString
80 ();
81                 String type = value.substring(0, 1);
82                 String name = getFileFolderName(value, type);
83
84                 JPopupMenu menu = new JPopupMenu();
85                 JMenuItem itemRemove = new JMenuItem("Delete");
86                 if (type.equals("f")) {
87                     JMenuItem itemDownload = new JMenuItem("
88 Download");
89                     itemDownload.addActionListener(e1 -> {
90                         JFileChooser downloadFolder = new
91 JFileChooser();
92                         downloadFolder.setCurrentDirectory(new java
93 .io.File("."));
94                         downloadFolder.setDialogTitle("Download
95 file");
96                         downloadFolder.setFileSelectionMode(
97 JFileChooser.DIRECTORIES_ONLY);
98                         downloadFolder.setAcceptAllFileFilterUsed(
99 false);
100
101                         if (downloadFolder.showDialog(null, "Select
102 folder") == JFileChooser.APPROVE_OPTION) {

```

```

        String path = downloadFolder.
getSelectedFile() + "\\\" + name;

94
        Thread t = new Thread(() -> {
96            try {
                ftp.getFile(name, path);
98            } catch (SocketException e2){
                System.out.println("File " +
name + " not fully downloaded!");
100            } catch (Exception e2) {
                e2.printStackTrace();
102            }
            });

104        t.start();

106    }
    });
108    menu.add(itemDownload);
    }
110    itemRemove.addActionListener(e12 -> {
        int confirm = JOptionPane.showConfirmDialog(
null, "Are you sure to delete " + name + "?", "Delete file/folder",
JOptionPane.OK_CANCEL_OPTION);
112        if(confirm == JOptionPane.OK_OPTION){
            if(type.equals("f")){
114                try {
                    ftp.deleteFile(name);
116                    fillList(listModel);
                } catch (Exception e1) {
                    e1.printStackTrace();
118                }
            }
120            else{
                try {
122                    ftp.deleteDir(name);
                    fillList(listModel);
124                } catch (IOException e1) {
                    e1.printStackTrace();
126                }
            }
128        }
    });
130    menu.add(itemRemove);
132    menu.show(tempList, e.getPoint().x, e.getPoint().y)
;

        } else {
134            try {
                String value = tempList.getSelectedValue().
toString();
136                String type = value.substring(0, 1);
                if (e.getClickCount() == 2 && !type.equals("f")
) {
138                    String folderName = getFileFolderName(value
, type);

```

```

140         try {
141             dir = ftp.cwd(folderName);
142             pathField.setText(dir);
143             fillList(listModel);
144         } catch (IOException e1) {
145             System.out.print("Connection failed!\n"
146 );
147         }
148     } catch (NullPointerException e1) {
149         System.out.print("Empty click on list\n");
150     }
151 }
152 ));
153
154 startButton = new JButton("Connect");
155 startButton.setFocusable(false);
156 startButton.addActionListener(e -> {
157
158     try {
159         ftp = new FTPClient(hostField.getText(), Integer.
160 parseInt(portField.getText()), loginField.getText(), passwordField.
161 getText());
162         ftp.connect();
163         fillList(listModel);
164         startButton.setEnabled(false);
165         stopButton.setEnabled(true);
166         uploadButton.setEnabled(true);
167         mkdirButton.setEnabled(true);
168     } catch (IOException | NullPointerException |
169 NumberFormatException e1) {
170         System.out.print("Connection failed!\n");
171     }
172     listField.setEnabled(true);
173 ));
174 buttonsPanel1.add(startButton);
175
176 stopButton = new JButton("Stop");
177 stopButton.setFocusable(false);
178 stopButton.addActionListener(e -> {
179     ftp.close();
180     stopButton.setEnabled(false);
181     startButton.setEnabled(true);
182     uploadButton.setEnabled(false);
183     mkdirButton.setEnabled(false);
184     listField.setEnabled(false);
185     listModel.removeAllElements();
186     dir = "/";
187     pathField.setText(dir);
188 });
189 stopButton.setEnabled(false);
190 buttonsPanel1.add(stopButton);

```

```

190     uploadButton = new JButton("Upload file");
191     uploadButton.setFocusable(false);
192     uploadButton.addActionListener(e -> {
193         JFileChooser fileopen = new JFileChooser();
194         int ret = fileopen.showDialog(null, "Upload file");
195         if(ret == JFileChooser.APPROVE_OPTION){
196             String filepath = fileopen.getSelectedFile().
getAbsolutePath();

198             Thread t = new Thread(() ->{
199                 try {
200                     ftp.uploadFile(filepath, dir);
201                     fillList(listModel);
202                 } catch (SocketException e1){
203                     System.out.println("File " + filepath + " not
fully uploaded!");
204                 } catch (Exception e1){
205                     e1.printStackTrace();
206                 }
207             });
208             t.start();
209         }
210     });
211     uploadButton.setEnabled(false);
212     buttonsPanel2.add(uploadButton);
213
214     mkdirButton = new JButton("Create folder");
215     mkdirButton.setFocusable(false);
216     mkdirButton.addActionListener(e -> {
217         try {
218             String name = JOptionPane.showInputDialog(null, "Enter
new folder name:", "Create folder", JOptionPane.QUESTION_MESSAGE);
219             if(name != null && !name.isEmpty()){
220                 ftp.mkdir(name);
221                 fillList(listModel);
222             }
223         } catch (IOException e1) {
224             e1.printStackTrace();
225         }
226     });
227     mkdirButton.setEnabled(false);
228     buttonsPanel2.add(mkdirButton);
229
230     getContentPane().add(mainPanel);
231     setPreferredSize(new Dimension(600, 500));
232     setResizable(false);
233     pack();
234     setLocationRelativeTo(null);
235     setVisible(true);
236 }

237
238 private String getFileFolderName(String value, String type) {

```



```

240     String folderName = "";
        Pattern regex;
242     if (type.equals("l"))
            regex = Pattern.compile("(?:.){4}(.)\\s+-->\\s+.+");
244     else regex = Pattern.compile("(?:.){4}(.)");
        Matcher matcher = regex.matcher(value);
246     if (matcher.find()) folderName = matcher.group(1);
        return folderName;
248 }

    private void fillList(DefaultListModel listModel) throws
IOException {
        listModel.removeAllElements();
252     FTPFile[] list = ftp.list(dir);
        if (!dir.equals("/")) listModel.addElement("d | ..");
254     for (FTPFile file : list) {
            if(file.getType().equals("") || file.getName().equals(""))
                continue;
256         String entry = file.getType() + " | " + file.getName();
        //         if(file.getType().equals("f")) entry += " (" + file.
        //             getSize() + " Bytes) ";
258         System.out.print(file.toString());
        //         System.out.print(entry);
260         listModel.addElement(entry);
        }
262     }

    public static void main(String args[]){
        JFrame.setDefaultLookAndFeelDecorated(true);
264         new Gui();
266     }
268 }

```

ЛИСТИНГ Б.1. Gui.java

```

1 package ftp;
2
3 import java.io.*;
4 import java.net.Socket;
5 import java.util.Map;
6 import java.util.concurrent.ConcurrentHashMap;
7
8 public class FTPClient {
9     private final String host;
10    private final int port;
11    private final String user;
12    private final String password;
13
14    private Socket socket;
15    private BufferedReader in;
16    private BufferedWriter out;
17
18    private ConcurrentHashMap<Socket, String> mSockets;

```

```

20     public enum FTPCommand {
        LIST, PASS, PASV, CWD, PWD, MKD, RMD, QUIT, RETR, SIZE, STOR,
        SYST, TYPE, USER, DELE, NLST
22     }

24     public FTPClient(String host, int port, String user, String
password) {
        assert host != null;

26
        this.host = host;
28         this.port = port == 0 ? 21 : port;
        this.user = user;
30         this.password = password;
    }

32
    public void connect() throws IOException {
        socket = new Socket(host, port);

36
        in = new BufferedReader(new InputStreamReader(socket.
getInputStream()));
        out = new BufferedWriter(new OutputStreamWriter(socket.
getOutputStream()));

38
        mSockets = new ConcurrentHashMap<>();

40
        login(in, out);
42    }

44    public synchronized void close() {
        try {
46            if (socket != null) {
                request(FTPCommand.QUIT, out);
48                longResponse(in);
                socket.close();

50
                for (Map.Entry<Socket, String> entry : mSockets.entrySet
()) {
52
                    Socket key = entry.getKey();
                    key.close();
54                    mSockets.remove(key);
                }
            }
56
        } catch (IOException cause) {
58            // Skip exceptions
        } finally {
60            socket = null;
        }

62    }

64    public long size(String filename) throws IOException {
        request(FTPCommand.SIZE, out, filename);
66        FTPResponse response = response(in);
        return response.getCode() == 213 ? Long.parseLong(response.
getText()) : -1;

```

```

68     }

70     public FTPFile[] list(String dir) throws IOException {
71         Socket socket = openChannel(in, out);
72         request(FTPCommand.LIST, out, dir == null ? "/" : dir);
73         response(in);

74         BufferedInputStream input = new BufferedInputStream(socket.
getInputStream());

76         byte[] buffer = new byte[1024];
77         StringBuilder buf = new StringBuilder();
78         while (input.read(buffer) > 0) {
79             buf.append(new String(buffer));
80         }
81         input.close();

82         response(in);

83         String[] lines = buf.toString().trim().split("\n");
84         FTPFile[] files = new FTPFile[lines.length];
85         for (int i = 0; i < lines.length; i++) {
86             files[i] = new FTPFile(lines[i]);
87         }

88         return files;
89     }

90

91     public String cwd(String dir) throws IOException {
92         request(FTPCommand.CWD, out, dir == null ? "/" : dir);

93         FTPResponse response;
94         do response = response(in);
95         while (response.isDelimiter() || response.getStatusCode() == 0);

96         request(FTPCommand.PWD, out);

97         String currentDir = response(in).getText();
98         currentDir = currentDir.substring(1, currentDir.lastIndexOf("\n"
99         ));

100         return currentDir;
101     }

102

103     public void uploadFile(String filepath, String dir) throws
104     Exception {

105         File firstLocalFile = new File(filepath);

106         String firstRemoteFile = dir + "/" + firstLocalFile.getName();
107         InputStream inputStream = new FileInputStream(firstLocalFile);

108         Socket newSocket = new Socket(host, port);

```

```

120     synchronized (mSockets) {
121         mSockets.put(newSocket, "");
122     }

124     BufferedReader newIn = new BufferedReader(new InputStreamReader
(newSocket.getInputStream()));
    BufferedWriter newOut = new BufferedWriter(new
OutputStreamWriter(newSocket.getOutputStream()));

126     Socket socket = getTransmissionSocket(newIn, newOut);

128     request(FTPCommand.STOR, newOut, firstRemoteFile);
    FTPResponse response = response(newIn);
130     if (!(response.getCode() == 150 || response.getCode() == 226))
{
132         return;
    }

134     BufferedInputStream input = new BufferedInputStream(inputStream
);
    BufferedOutputStream output = new BufferedOutputStream(socket.
getOutputStream());

138     byte[] buffer = new byte[4096];
    int bytesRead;
140     while ((bytesRead = input.read(buffer)) != -1) {
        output.write(buffer, 0, bytesRead);
142     }
    output.flush();
144     output.close();
    input.close();

146     response(newIn);

148     synchronized (mSockets) {
150         mSockets.remove(newSocket);
    }

152     newSocket.close();
154 }

156 public void getFile(String filename, String path) throws Exception
{

158     File firstLocalFile = new File(path);
    OutputStream outputStream = new FileOutputStream(firstLocalFile
);

160     Socket newSocket = new Socket(host, port);

162     synchronized (mSockets) {
164         mSockets.put(newSocket, "");
    }

```

```

166         BufferedReader newIn = new BufferedReader(new InputStreamReader
(newSocket.getInputStream()));
168         BufferedWriter newOut = new BufferedWriter(new
OutputStreamWriter(newSocket.getOutputStream()));

170         Socket socket = getTransmissionSocket(newIn, newOut);

172         request(FTPCommand.RETR, newOut, filename);
FTPResponse response = response(newIn);
174         if (!(response.getCode() == 150 || response.getCode() == 226))
{
            return;
176         }

178         BufferedInputStream input = new BufferedInputStream(socket.
getInputStream());

180         byte[] buffer = new byte[4096];
int bytesRead;
182         while ((bytesRead = input.read(buffer)) != -1) {
            outputStream.write(buffer, 0, bytesRead);
184         }
outputStream.close();
186         input.close();

188         response(newIn);

190         synchronized (mSockets) {
            mSockets.remove(newSocket);
192         }

194         newSocket.close();
    }

196     public void deleteFile(String filename) throws Exception {
198         request(FTPCommand.DELE, out, filename);
FTPResponse response = response(in);
200         if (!(response.getCode() == 150 || response.getCode() == 226))
{
            return;
202         }
    }

204     public FTPFile[] nlist(String dir) throws IOException {
206         Socket socket = openChannel(in, out);
request(FTPCommand.NLST, out, dir == null ? "/" : dir);
208         response(in);

210         BufferedInputStream input = new BufferedInputStream(socket.
getInputStream());

212         byte[] buffer = new byte[1024];
StringBuilder buf = new StringBuilder();

```

```

214     while (input.read(buffer) > 0) {
215         buf.append(new String(buffer));
216     }
217     input.close();
218
219     response(in);
220
221     String[] lines = buf.toString().trim().split("\r\n");
222     FTPFile[] files = new FTPFile[lines.length];
223     for (int i = 0; i < lines.length; i++) {
224         files[i] = new FTPFile(lines[i]);
225     }
226
227     return files;
228 }
229
230 public void makeDir(String name) throws IOException {
231     request(FTPCommand.MKD, out, name);
232     response(in);
233 }
234
235 public void deleteDir(String name) throws IOException {
236     request(FTPCommand.RMD, out, name);
237     response(in);
238 }
239
240 private void login(BufferedReader in, BufferedWriter out) throws
IOException {
241     longResponse(in);
242     request(FTPCommand.USER, out, user);
243     response(in);
244     request(FTPCommand.PASS, out, password);
245     longResponse(in);
246 }
247
248 private void request(FTPCommand command, BufferedWriter out, String
... args) throws IOException {
249     StringBuilder request = new StringBuilder(command.toString());
250     for (String arg : args) {
251         request.append(' ').append(arg);
252     }
253     log(request.toString(), false);
254
255     out.write(request.toString() + "\r\n");
256     out.flush();
257 }
258
259 private FTPResponse response(BufferedReader in) throws IOException
{
260     String str = in.readLine();
261     log(str, true);
262     return new FTPResponse(str);
263 }
264

```

```

266     private void longResponse(BufferedReader in) throws IOException {
        FTPResponse response;
        do response = response(in);
268     while (response.isDelimiter());
    }

270
    private void log(String message, boolean in) {
272     System.out.println((in ? "<<< " : ">>> ") + message);
    }

274
    private Socket openChannel(BufferedReader in, BufferedWriter out)
    throws IOException {
276     request(FTPCommand.SYST, out);
        response(in);
278     request(FTPCommand.PWD, out);
        response(in);
280     request(FTPCommand.TYPE, out, "I");
        response(in);
282
        request(FTPCommand.PASV, out);
284
        FTPResponse response = response(in);
286     if(response.getStatusCode() == 530) return null;

        String text = response.getText();
        text = text.substring(text.indexOf('(') + 1, text.indexOf(')'))
288
    ;
290     String[] buf = text.split(",");

        String ip = buf[0] + "." + buf[1] + "." + buf[2] + "." + buf
292     [3];
        int port = (Integer.parseInt(buf[4]) << 8) | Integer.parseInt(
        buf[5]);
294
        log(ip + ":" + port, true);
296
        return new Socket(host, port);
298     }

300     private Socket getTransmissionSocket(BufferedReader newIn,
        BufferedWriter newOut) throws IOException {
        login(newIn, newOut);
302     return openChannel(newIn, newOut);
    }
304 }

```

Листинг Б.2. FTPClient.java

```

1 package ftp;
2
import java.util.regex.Matcher;
4 import java.util.regex.Pattern;

6 public class FTPFile {

```

```

8      private String name = "";
      private String type = "";
      private long size;

10
      public FTPFile(String line) {
12          Pattern regex = Pattern.compile("(.)\\{9\\}(?:\\s+\\.+){3}\\s+(\\d+)"
          "\\s+\\w{3}\\s+\\d{1,2}\\s+[\\d:]{4,5}\\s+(.*)");
          Matcher matcher = regex.matcher(line);
14          if (matcher.find()) {
              this.type = matcher.group(1);
16              if(this.type.equals("-")) this.type = "f";
              this.size = Long.parseLong(matcher.group(2));
18              this.name = matcher.group(3);
          }
20      }

22      public String toString(){
          return "type = " + type + "\nsize = " + size + "\nname = " +
name + "\n\n";
24      }

26      public long getSize() {
          return size;
28      }

30      public String getName() {
          return name;
32      }

34      public String getType() {
          return type;
36      }
    }

```

Листинг Б.3. FTPFile.java

```

1 package ftp;
2
3 public class FTPResponse {
4     private final int code;
5     private final String text;
6     private final String delimiter;
7
8     public FTPResponse(String answer) {
9         int number;
10        int pos;
11        try {
12            String num = answer.trim().split("\\s|-)", 2)[0];
13            number = Integer.parseInt(num);
14            pos = String.valueOf(number).length();
15        } catch (IndexOutOfBoundsException | NumberFormatException e){
16            number = 0;
17            pos = 0;
18        }
19    }

```



```

    this.code = number;

    this.delimiter = Character.toString(answer.charAt(pos));

    this.text = answer.substring(pos + 1).trim();
}

public int getCode() {
    return code;
}

public String getText() {
    return text;
}

public boolean isDelimiter() {
    return delimiter.equals("-");
}
}

```

Листинг Б.4. FTPResponse.java