МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ

Факультет прикладной математики и информатики

Кафедра ТПИ

Дисциплина: «Сетевые информационные технологии»

Лабораторная работа №5

ПРОТОКОЛЫ ЭЛЕКТРОННОЙ ПОЧТЫ

Факультет: ФПМИ

Группа: ПМИМ-31

Студенты: Тарулин М.А., Холодова В.С.

Преподаватель: Кобылянский В.Г.

Дата выполнения:

Отметка о защите:

Новосибирск, 2024 г.

1. Цель работы

Целью работы является изучение основных принципов работы электронной почты и почтового протокола SMTP, а также разработка программы, реализующей этот протокол.

2. Задание

2.1. Ознакомиться с типовой сессией SMTP.

2.2. Запустить анализатор Wireshark и загрузить в него файл smtp_test2.pcapng. Отфильтровать из общего потока пакетов сеанс связи с SMTP-сервером и выполнить его анализ.

2.3. Получить логин и пароль для доступа к почтовому серверу. Разработать клиентское приложение для отправки текстовых сообщений по протоколу SMTP с учетом следующих требований:

- все команды и данные должен вводить пользователь (адреса получателя и отправителя, текст сообщения);
- подключение к почтовому серверу реализовать на основе сокетов;
- приложение должно формировать строки команд в соответствии с протоколом SMTP, выводить их на экран и отправлять на сервер. Ответы сервера также должны выводится на экран.
- весь процесс почтовой сессии должен сохраняться в файле журнала smtp_7.log.

2.4. С помощью разработанного приложения отправить сообщение бригаде №8 и с помощью Wireshark выполнить анализ переданных пакетов путем фильтрации исходящего трафика по IP-адресам источника и получателя, названию протокола (SMTP) или номеру порта SMTP-сервера. Сравнить результаты анализа с данными из файла журнала.

2.5. По захваченным данным построить диаграмму потоков.

2.6. Запустить анализатор Wireshark и загрузить в него файл pop3_test7.pcapng. Отфильтровать из общего потока пакетов сеанс связи с POP3-сервером и выполнить его анализ.

2.7. Разработать клиентское приложение для получения почтовых сообщений по протоколу POP3 с учетом требований из пункта 2.3 задания для SMTP-клиента.

2.8. С помощью разработанного приложения получить из почтового ящика сообщения, которые были направлены при выполнения пункта 2.4 задания и выполнить анализ трафика путем фильтрации по IP-адресам источника и получателя, названию протокола и номеру порта сервера. Сравнить результаты анализа с данными из файла журнала.

2.9. По захваченным данным построить диаграмму потоков.

2.10. С помощью почтового сервиса, установленного на локальном компьютере открыть почтовый ящик с сообщениями, которые были направлены при выполнении пункта 2.4 задания. Выполнить анализ служебных заголовков одного из писем, занести заголовки в отчет.

Отчет по работе должен содержать описание алгоритмов, исходные тексты программ, полную сессию обмена данными с SMTP- и POP3-серверами, а также анализ перехваченных пакетов SMTP и POP3.

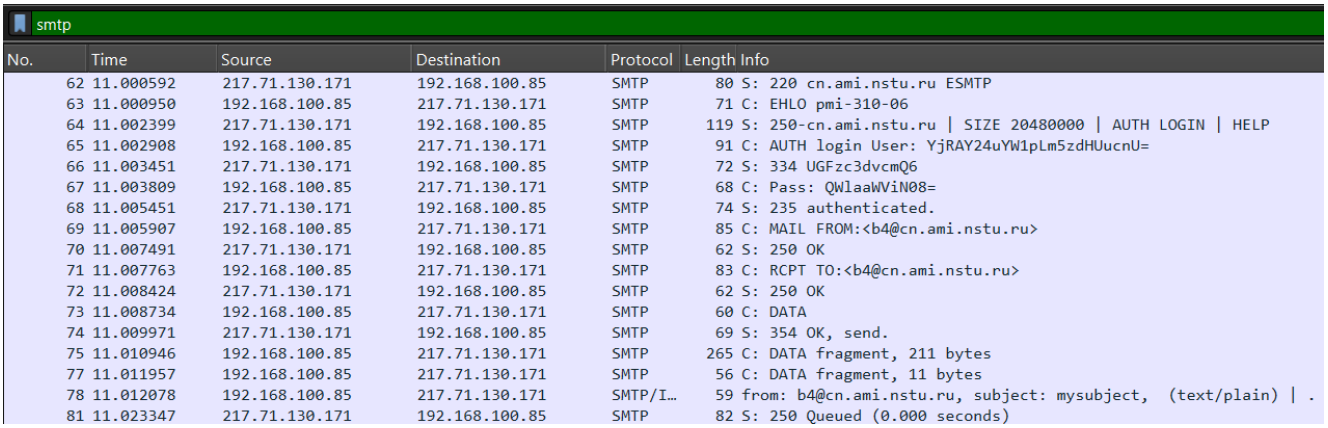
2.11. Разработать клиентское почтовое приложение, реализующие протоколы SMTP, POP3. Основу приложения должны составлять программы, разработанные в предыдущих пунктах задания и объединенные общим графическим интерфейсом.

Требования к программе:

- обеспечить выполнение следующих функций: просмотр содержимого почтового ящика: отправка, прием и удаление сообщений электронной почты;
- имена отправителей и получателей должны задаваться пользователем;
- сеансы отправки и получения сообщений должны выводится в отдельные текстовые поля формы.

3. Ход работы

3.1. Загрузим файл *smtp_test2.pcapng* в Wireshark, отфильтруем из общего потока сеанс связи с SMTP-сервером.



smtp

No.	Time	Source	Destination	Protocol	Length	Info
62	11.000592	217.71.130.171	192.168.100.85	SMTP	80	S: 220 cn.ami.nstu.ru ESMTP
63	11.000950	192.168.100.85	217.71.130.171	SMTP	71	C: EHLO pmi-310-06
64	11.002399	217.71.130.171	192.168.100.85	SMTP	119	S: 250-cn.ami.nstu.ru | SIZE 20480000 | AUTH LOGIN | HELP
65	11.002908	192.168.100.85	217.71.130.171	SMTP	91	C: AUTH login User: YjRAY24uYW1pLm5zdHUucnU=
66	11.003451	217.71.130.171	192.168.100.85	SMTP	72	S: 334 UGFzc3dvcmQ6
67	11.003809	192.168.100.85	217.71.130.171	SMTP	68	C: Pass: QWlaaWViN08=
68	11.005451	217.71.130.171	192.168.100.85	SMTP	74	S: 235 authenticated.
69	11.005907	192.168.100.85	217.71.130.171	SMTP	85	C: MAIL FROM:<b4@cn.ami.nstu.ru>
70	11.007491	217.71.130.171	192.168.100.85	SMTP	62	S: 250 OK
71	11.007763	192.168.100.85	217.71.130.171	SMTP	83	C: RCPT TO:<b4@cn.ami.nstu.ru>
72	11.008424	217.71.130.171	192.168.100.85	SMTP	62	S: 250 OK
73	11.008734	192.168.100.85	217.71.130.171	SMTP	60	C: DATA
74	11.009971	217.71.130.171	192.168.100.85	SMTP	69	S: 354 OK, send.
75	11.010946	192.168.100.85	217.71.130.171	SMTP	265	C: DATA fragment, 211 bytes
77	11.011957	192.168.100.85	217.71.130.171	SMTP	56	C: DATA fragment, 11 bytes
78	11.012078	192.168.100.85	217.71.130.171	SMTP/I…	59	from: b4@cn.ami.nstu.ru, subject: mysubject, (text/plain) | .
81	11.023347	217.71.130.171	192.168.100.85	SMTP	82	S: 250 Queued (0.000 seconds)

Рисунок 1 – Сеанс связи с SMTP-сервером.

Клиент подключается к почтовому серверу cn.ami.nstu.ru через стандартный SMTP-порт 25. Сервер отвечает кодом 220, подтверждая готовность к работе. Клиент посылает команду EHLO pmi-310-06, запрашивая поддержку расширенных возможностей, которые сервер подтверждает, перечисляя функции SIZE 20480000, AUTH LOGIN, HELP.

Затем клиент начинает процесс аутентификации командой AUTH login. Сервер требует ввода имени пользователя и пароля. После успешной проверки учетных данных сервер сообщает об этом кодом 235 authenticated.

Далее клиент отправляет письмо, указывая отправителя командой MAIL FROM:<b4@cn.ami.nstu.ru> и получателя командой RCPT TO:<b4@cn.ami.nstu.ru>. Сервер подтверждает правильность этих действий кодом 250 OK.

После этого начинается передача содержимого письма. Клиент использует команду DATA, а затем отправляет два фрагмента данных: первый длиной 211 байт и второй — 11 байт. Сервер принимает эти данные и объединяет их в одно сообщение с темой mysubject и текстовым содержанием.

В завершение сервер подтверждает прием письма кодом 250 Queued (0.000 seconds), что означает успешное принятие письма и постановку его в очередь для дальнейшей обработки и доставки.

3.2. Разработаем клиентское приложение для отправки текстовых сообщений по протоколу SMTP с возможность ввода всех команд, с реализацией подключения на основе сокетов, с сохранением всей почтовой сессии в файле журнала.

```
Добро пожаловать в SMTP клиент!

Примеры SMTP команд:
AUTH LOGIN
MAIL FROM:<sender@example.com>
RCPT TO:<recipient@example.com>
DATA
QUIT

Введите адрес SMTP-сервера: smtp.mail.ru
Введите порт: 587
<-- 220 smtp56.i.mail.ru ESMTP ready (Looking for Mail for your domain? Visit https://biz.mail.ru)

--> EHLO MarkPC
<-- 250-smtp56.i.mail.ru
250-SIZE 73400320
250-8BITMIME
250-PIPELINING
250-DSN
250-SMTPUTF8
250 STARTTLS
```

Рисунок 2 – Пример работы разработанного приложения.

```
Соединение установлено. Теперь вы можете вводить SMTP команды.
Для завершения работы введите 'QUIT'


Введите SMTP команду: AUTH LOGIN
--> AUTH LOGIN
<-- 334 VXNlcm5hbWU6


Введите SMTP команду: QUIT
--> QUIT
<-- 334 UGFzc3dvcmQ6


Соединение закрыто. Сессия сохранена в файле smtp_20241116_102510.log
```

Рисунок 3 – Пример работы разработанного приложения.

Приведем пример записи почтовой сессии:

2024-11-16 10:25:10 <-- 220 smtp56.i.mail.ru ESMTP ready (Looking for Mail for your domain? Visit https://biz.mail.ru)

2024-11-16 10:25:10 --> EHLO MarkPC
2024-11-16 10:25:10 <-- 250-smtp56.i.mail.ru

250-SIZE 73400320

250-8BITMIME

250-PIPELINING

250-DSN

250-SMTPUTF8

250 STARTTLS

2024-11-16 10:25:10 --> STARTTLS
2024-11-16 10:25:10 <-- 220 2.0.0 Start TLS

2024-11-16 10:25:10 --> EHLO MarkPC
2024-11-16 10:25:10 <-- 250-smtp56.i.mail.ru

250-SIZE 73400320

250-8BITMIME

250-PIPELINING

250-DSN

250-SMTPUTF8

250 AUTH PLAIN LOGIN XOAUTH2

2024-11-16 10:25:27 --> AUTH LOGIN
2024-11-16 10:25:27 <-- 334 VXNlcm5hbWU6

2024-11-16 10:26:31 --> QUIT
2024-11-16 10:26:31 <-- 334 UGFzc3dvcmQ6

3.3. С помощью разработанного приложения отправим сообщение бригаде №8 и с помощью Wireshark выполним анализ переданных пакетов путем фильтрации исходящего трафика по IP-адресам и по названию протокола SMTP.

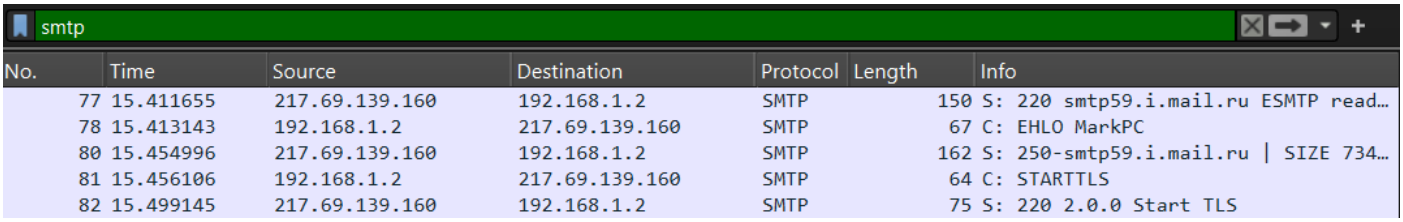


Рисунок 4 – Переданные пакеты по протоколу SMTP.

Приведем запись почтовой сессии, полученной с помощью записей в журнале разработанного приложения.

2024-11-16 10:48:33 <-- 220 smtp59.i.mail.ru ESMTP ready (Looking for Mail for your domain? Visit https://biz.mail.ru)

2024-11-16 10:48:33 --> EHLO MarkPC
2024-11-16 10:48:33 <-- 250-smtp59.i.mail.ru

250-SIZE 73400320

250-8BITMIME

250-PIPELINING

250-DSN

250-SMTPUTF8

250 STARTTLS

2024-11-16 10:48:33 --> STARTTLS
2024-11-16 10:48:33 <-- 220 2.0.0 Start TLS

2024-11-16 10:48:33 --> EHLO MarkPC
2024-11-16 10:48:33 <-- 250-smtp59.i.mail.ru

250-SIZE 73400320

250-8BITMIME

250-PIPELINING

250-DSN

250-SMTPUTF8

250 AUTH PLAIN LOGIN XOAUTH2

2024-11-16 10:48:39 --> AUTH LOGIN
2024-11-16 10:48:39 <-- 334 VXNlcm5hbWU6

2024-11-16 10:49:02 --> aWdvci5ub3Zpa292MTk4NkBtYWlsLnJ1
2024-11-16 10:49:02 <-- 334 UGFzc3dvcmQ6

2024-11-16 10:49:17 --> TGF0Q3ltVkJiYUE1ZW4yeWRmdU0NCg==
2024-11-16 10:49:17 <-- 235 Authentication succeeded

2024-11-16 10:49:49 --> MAIL FROM: <igor.novikov1986@mail.ru>
2024-11-16 10:49:50 <-- 250 OK

2024-11-16 10:50:06 --> RCPT TO: <igor.novikov1986@mail.ru>
2024-11-16 10:50:06 <-- 250 Accepted

2024-11-16 10:50:17 --> DATA
2024-11-16 10:50:17 <-- 354 Enter message, ending with "." on a line by itself

2024-11-16 10:50:43 --> Test message.

.
2024-11-16 10:50:43 <-- 250 OK id=1tC9pB-00000000F7I-2M0q

2024-11-16 10:50:54 --> QUIT
2024-11-16 10:50:54 <-- 221 exim-smtp-5c7f98b595-9bcx5 closing connection

В результате сравнения перехваченных пакетов и записей в журнале разработанного приложения видно, что протокол SMTP используется до подключения сеанса по защищенному соединению, дальше сеанс проходит по протоколу TLS.

3.4. По захваченным данным построим диаграмму потоков.

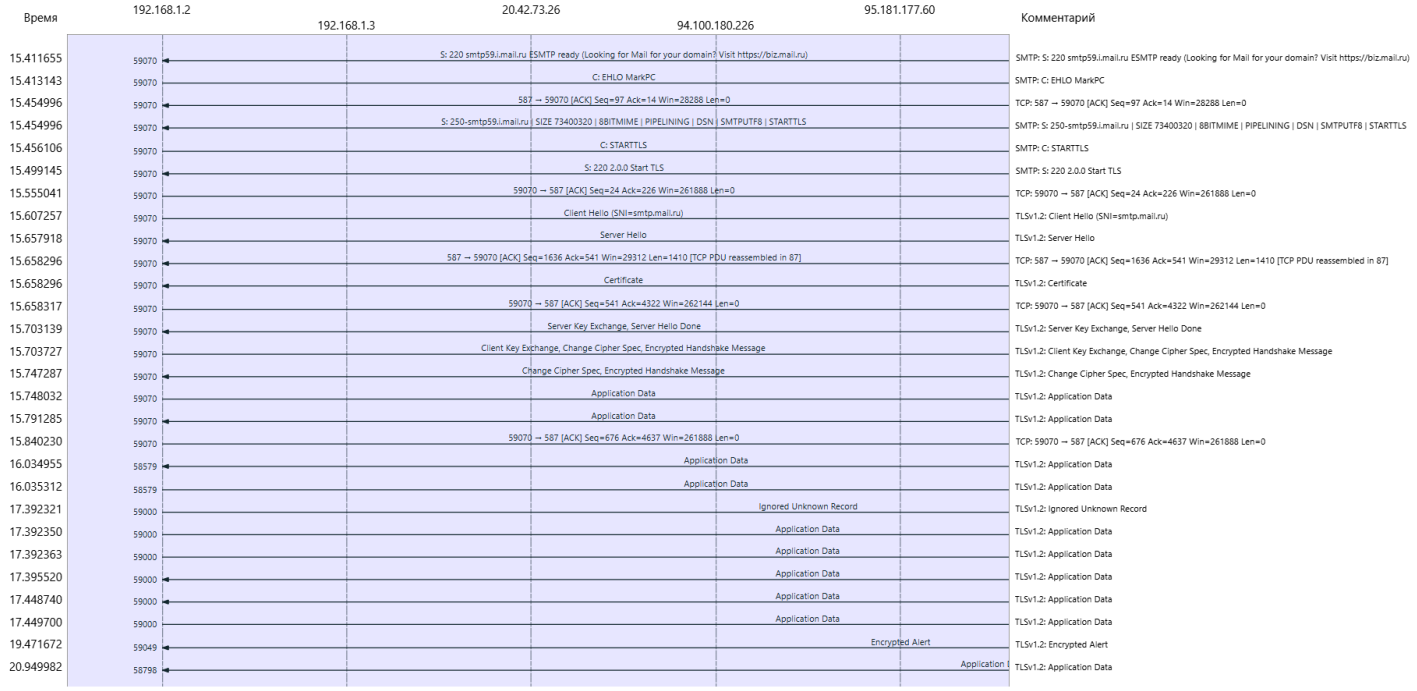
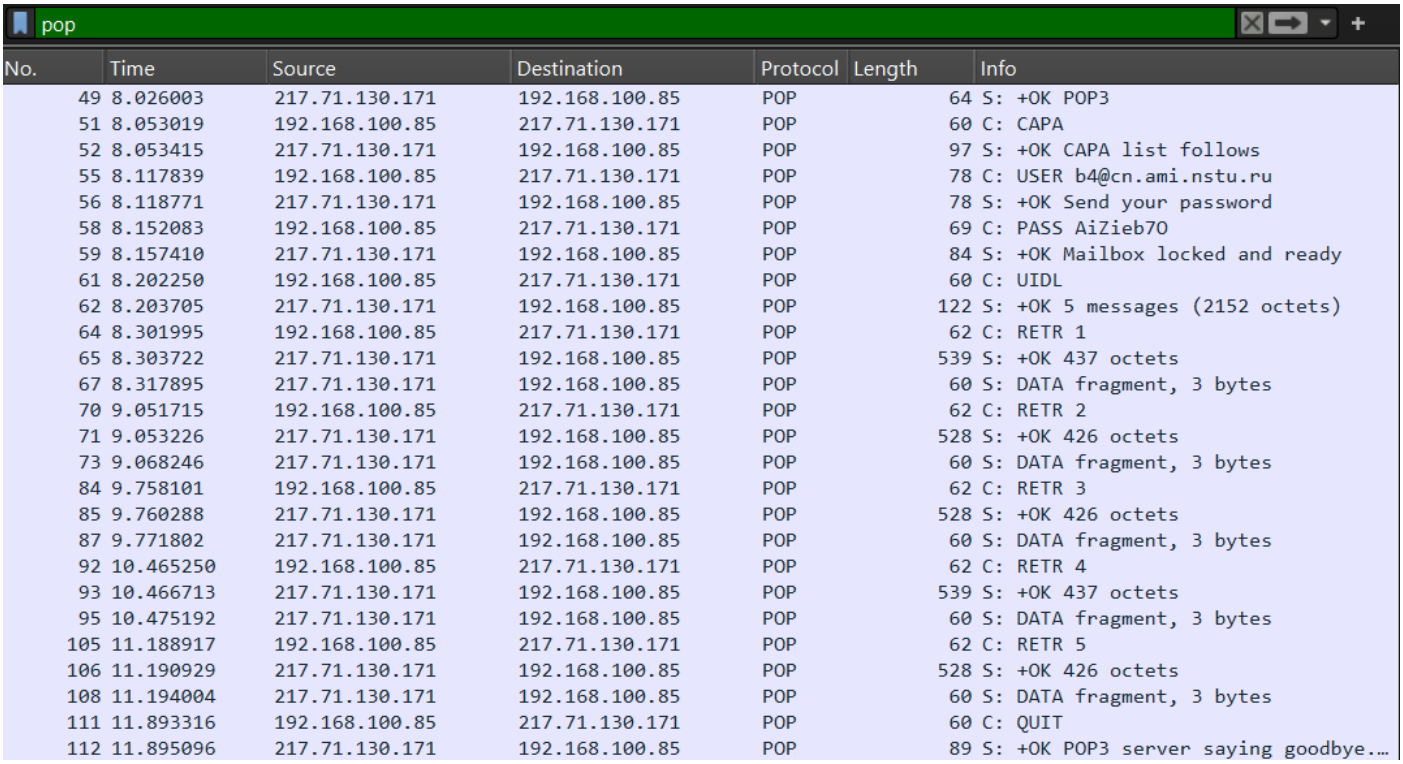


Рисунок 5 – Диаграмма перехваченного потока.

3.5. Загрузим в анализатор Wireshark файл pop3_test7.pcapng. Отфильтруем из общего потока пакетов сеанс связи с POP3-сервером.



pop

No.	Time	Source	Destination	Protocol	Length	Info
49	8.026003	217.71.130.171	192.168.100.85	POP	64	S: +OK POP3
51	8.053019	192.168.100.85	217.71.130.171	POP	60	C: CAPA
52	8.053415	217.71.130.171	192.168.100.85	POP	97	S: +OK CAPA list follows
55	8.117839	192.168.100.85	217.71.130.171	POP	78	C: USER b4@cn.ami.nstu.ru
56	8.118771	217.71.130.171	192.168.100.85	POP	78	S: +OK Send your password
58	8.152083	192.168.100.85	217.71.130.171	POP	69	C: PASS AiZieb7O
59	8.157410	217.71.130.171	192.168.100.85	POP	84	S: +OK Mailbox locked and ready
61	8.202250	192.168.100.85	217.71.130.171	POP	60	C: UIDL
62	8.203705	217.71.130.171	192.168.100.85	POP	122	S: +OK 5 messages (2152 octets)
64	8.301995	192.168.100.85	217.71.130.171	POP	62	C: RETR 1
65	8.303722	217.71.130.171	192.168.100.85	POP	539	S: +OK 437 octets
67	8.317895	217.71.130.171	192.168.100.85	POP	60	S: DATA fragment, 3 bytes
70	9.051715	192.168.100.85	217.71.130.171	POP	62	C: RETR 2
71	9.053226	217.71.130.171	192.168.100.85	POP	528	S: +OK 426 octets
73	9.068246	217.71.130.171	192.168.100.85	POP	60	S: DATA fragment, 3 bytes
84	9.758101	192.168.100.85	217.71.130.171	POP	62	C: RETR 3
85	9.760288	217.71.130.171	192.168.100.85	POP	528	S: +OK 426 octets
87	9.771802	217.71.130.171	192.168.100.85	POP	60	S: DATA fragment, 3 bytes
92	10.465250	192.168.100.85	217.71.130.171	POP	62	C: RETR 4
93	10.466713	217.71.130.171	192.168.100.85	POP	539	S: +OK 437 octets
95	10.475192	217.71.130.171	192.168.100.85	POP	60	S: DATA fragment, 3 bytes
105	11.188917	192.168.100.85	217.71.130.171	POP	62	C: RETR 5
106	11.190929	217.71.130.171	192.168.100.85	POP	528	S: +OK 426 octets
108	11.194004	217.71.130.171	192.168.100.85	POP	60	S: DATA fragment, 3 bytes
111	11.893316	192.168.100.85	217.71.130.171	POP	60	C: QUIT
112	11.895096	217.71.130.171	192.168.100.85	POP	89	S: +OK POP3 server saying goodbye.…

Рисунок 6 – Сеанс связи с POP3-сервером.

Клиент установил соединение с POP3-сервером и получил подтверждение готовности к работе. Затем он запросил возможности сервера с помощью команды CAPA, на что сервер ответил списком поддерживаемых функций.

После этого клиент передал имя пользователя и пароль, получив подтверждение успешной авторизации. Почтовый ящик был заблокирован для других клиентов и подготовлен к работе. Далее клиент запросил уникальные идентификаторы для каждого письма в ящике с помощью команды UIDL. Сервер предоставил информацию о пяти письмах и их размерах. Используя команду RETR, клиент последовательно запрашивал каждое из пяти писем. Сервер отвечал подтверждением и начинал передачу содержимого письма.

Пакеты с пометкой DATA fragment указывают на фрагменты данных, составляющих содержание писем. По завершении загрузки всех писем клиент завершил сессию командой QUIT, на которую сервер ответил подтверждением и сообщением о закрытии соединения.

3.6. Разработаем клиентское приложение для получения почтовых сообщений по протоколу POP3 с учетом требований из пункта 2.11.

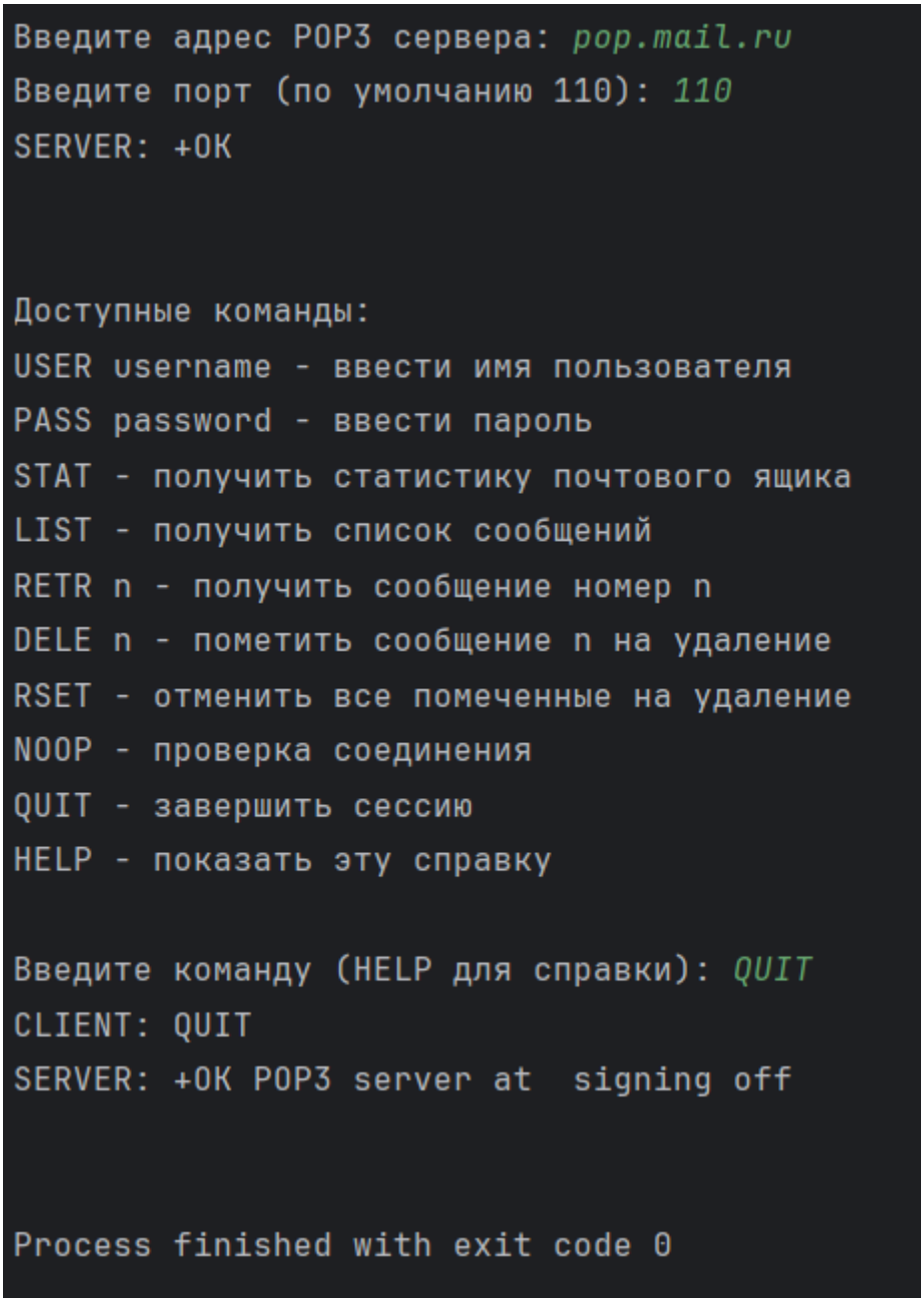


Рисунок 7 – Пример работы разработанной программы.

Приведем запись почтовой сессии, полученной с помощью записей в журнале разработанного приложения.

[2024-11-16 11:54:55] SERVER: +OK

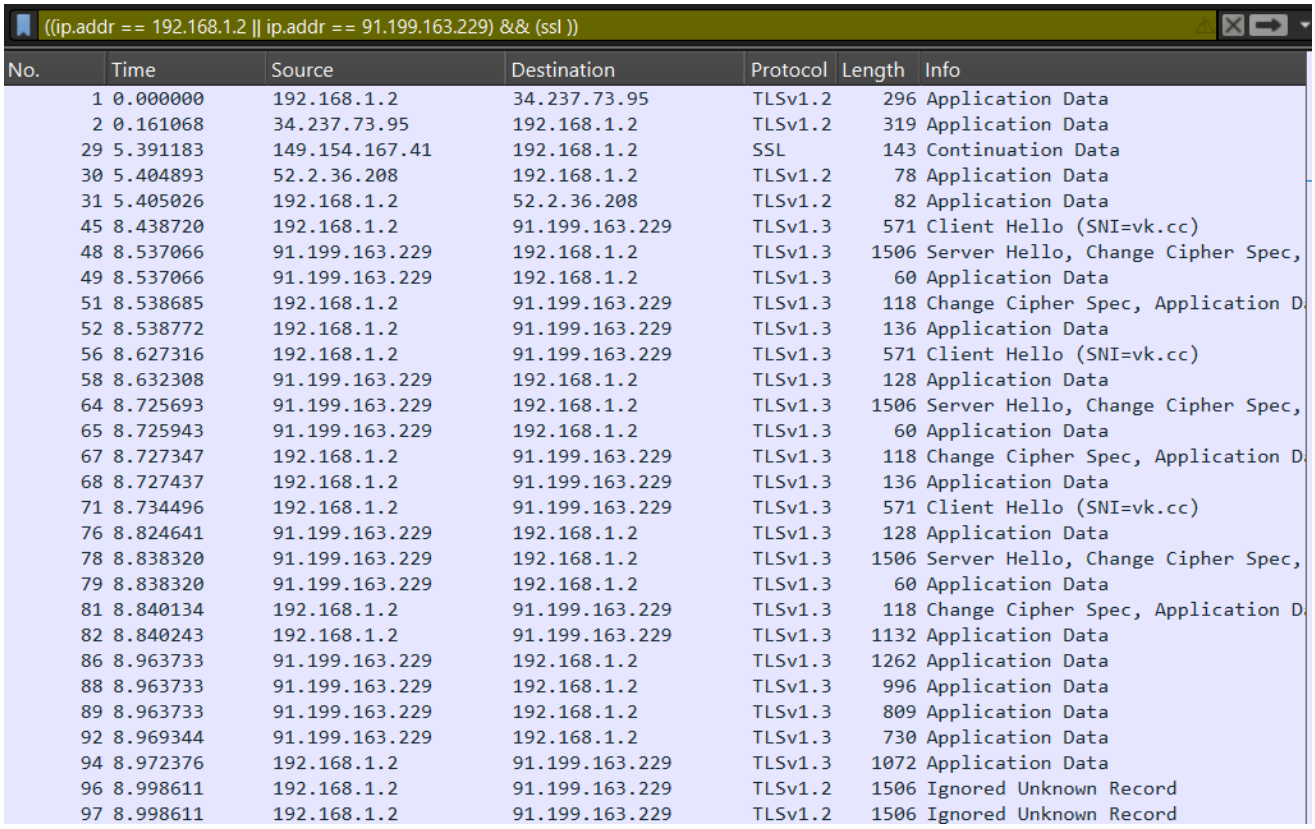
[2024-11-16 11:55:07] CLIENT: QUIT
[2024-11-16 11:55:07] SERVER: +OK POP3 server at signing off

3.7. С помощью разработанного приложения получим из почтового ящика сообщение, которое было направлено при выполнении пункта 3.3.

```
Введите команду (HELP для справки): RETR 3
CLIENT: RETR 3
SERVER: +OK 3972 octets

SERVER: Получено многострочное сообщение

=== Заголовки сообщения ===
From: igor.novikov1986@mail.ru
Date: Sat, 16 Nov 2024 06:50:43 +0300

=== Содержимое сообщения ===
Test message.
.


Введите команду (HELP для справки): QUIT
CLIENT: QUIT
SERVER: +OK no changes to commit, closing connection
```

Рисунок 8 – Текст полученного сообщения.

Отфильтруем захваченный трафик по IP-адресам клиента и сервера и протоколу ssl.



((ip.addr == 192.168.1.2 || ip.addr == 91.199.163.229) && (ssl))

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.2	34.237.73.95	TLSv1.2	296	Application Data
2	0.161068	34.237.73.95	192.168.1.2	TLSv1.2	319	Application Data
29	5.391183	149.154.167.41	192.168.1.2	SSL	143	Continuation Data
30	5.404893	52.2.36.208	192.168.1.2	TLSv1.2	78	Application Data
31	5.405026	192.168.1.2	52.2.36.208	TLSv1.2	82	Application Data
45	8.438720	192.168.1.2	91.199.163.229	TLSv1.3	571	Client Hello (SNI=vk.cc)
48	8.537066	91.199.163.229	192.168.1.2	TLSv1.3	1506	Server Hello, Change Cipher Spec,
49	8.537066	91.199.163.229	192.168.1.2	TLSv1.3	60	Application Data
51	8.538685	192.168.1.2	91.199.163.229	TLSv1.3	118	Change Cipher Spec, Application D
52	8.538772	192.168.1.2	91.199.163.229	TLSv1.3	136	Application Data
56	8.627316	192.168.1.2	91.199.163.229	TLSv1.3	571	Client Hello (SNI=vk.cc)
58	8.632308	91.199.163.229	192.168.1.2	TLSv1.3	128	Application Data
64	8.725693	91.199.163.229	192.168.1.2	TLSv1.3	1506	Server Hello, Change Cipher Spec,
65	8.725943	91.199.163.229	192.168.1.2	TLSv1.3	60	Application Data
67	8.727347	192.168.1.2	91.199.163.229	TLSv1.3	118	Change Cipher Spec, Application D
68	8.727437	192.168.1.2	91.199.163.229	TLSv1.3	136	Application Data
71	8.734496	192.168.1.2	91.199.163.229	TLSv1.3	571	Client Hello (SNI=vk.cc)
76	8.824641	91.199.163.229	192.168.1.2	TLSv1.3	128	Application Data
78	8.838320	91.199.163.229	192.168.1.2	TLSv1.3	1506	Server Hello, Change Cipher Spec,
79	8.838320	91.199.163.229	192.168.1.2	TLSv1.3	60	Application Data
81	8.840134	192.168.1.2	91.199.163.229	TLSv1.3	118	Change Cipher Spec, Application D
82	8.840243	192.168.1.2	91.199.163.229	TLSv1.3	1132	Application Data
86	8.963733	91.199.163.229	192.168.1.2	TLSv1.3	1262	Application Data
88	8.963733	91.199.163.229	192.168.1.2	TLSv1.3	996	Application Data
89	8.963733	91.199.163.229	192.168.1.2	TLSv1.3	809	Application Data
92	8.969344	91.199.163.229	192.168.1.2	TLSv1.3	730	Application Data
94	8.972376	192.168.1.2	91.199.163.229	TLSv1.3	1072	Application Data
96	8.998611	192.168.1.2	91.199.163.229	TLSv1.2	1506	Ignored Unknown Record
97	8.998611	192.168.1.2	91.199.163.229	TLSv1.2	1506	Ignored Unknown Record

Рисунок 9 – Отфильтрованный трафик.

Результат файла журнала разработанного приложения.

[2024-11-16 12:30:52] SERVER: +OK

[2024-11-16 12:31:06] CLIENT: USER igor.novikov1986@mail.ru
[2024-11-16 12:31:06] SERVER: +OK

[2024-11-16 12:31:20] CLIENT: PASS LatCymVBbaA5en2ydfuM
[2024-11-16 12:31:21] SERVER: +OK hello igor.novikov1986@mail.ru

[2024-11-16 12:31:28] CLIENT: RETR 3
[2024-11-16 12:31:28] SERVER: +OK 3972 octets

[2024-11-16 12:31:29] SERVER: Получено многострочное сообщение
[2024-11-16 12:31:36] CLIENT: QUIT
[2024-11-16 12:31:36] SERVER: +OK no changes to commit, closing connection

При перехвате сетевого трафика с использованием Wireshark из-за использования зашифрованного соединения с использованием ssl пакеты представлены в зашифрованном виде, в случае с журналом приложения команды сохраняются в открытом виде.

3.8. Построим диаграмму потоков по перехваченным данным.

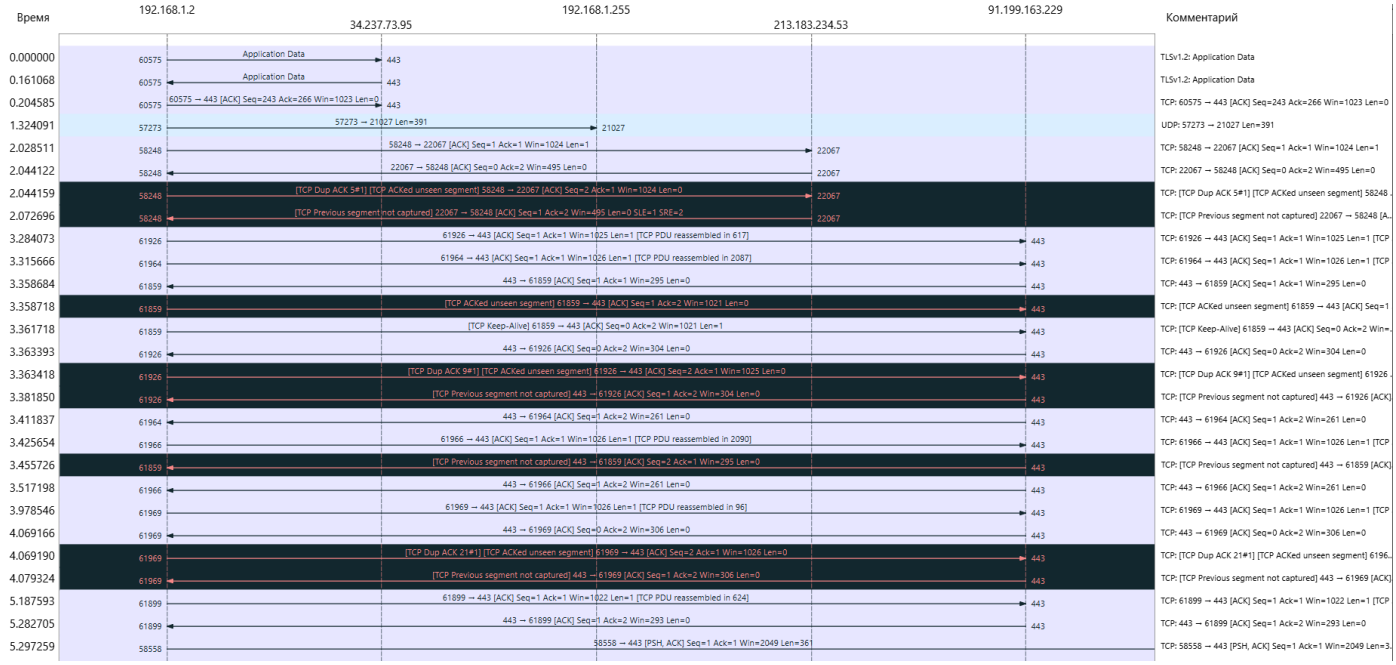


Рисунок 10 – Диаграмма потоков.

3.9. С помощью клиента mail.ru получим текст служебных заголовков письма, полученного при выполнении пункта 3.3.

Delivered-To: igor.novikov1986@mail.ru
DKIM-Signature: v=1; a=rsa-sha256; q=dns/txt; c=relaxed/relaxed; d=mail.ru;
s=mail4; h=Date:From:Message-Id:From:Sender:Reply-To:Subject:To:Cc:
MIME-Version:Content-Type:Content-Transfer-Encoding:Content-ID:
Content-Description:Resent-Date:Resent-From:Resent-Sender:Resent-To:Resent-Cc
:Resent-Message-ID:In-Reply-To:References:List-Id:List-Help:List-Unsubscribe:
List-Subscribe:List-Post:List-Owner:List-Archive:X-Cloud-Ids:
Disposition-Notification-To; bh=5qknsAm/oEch/AUUccNShJHNISJlxdm3J2HGyTY9wfA=;
t=1731729043; x=1731819043; b=B3BLmAcQ8ZAAjxZKNVF/kC+x3VcFoC3tL2pgUEfkrpYN8rT
m4LDmqdIJH+h7aFDHIXllf9aeabnamJAq2kj/b/cA4BFBMD1SBims+apnfUpqTii+XyAWKyOzdZHa
IU+ngJRKHJQ7jmDGwFJXhvHt10/sswSRUnx+ntyTkIPIC3IntLf1WeVyWSBjuS0Lw8nR57UJ4STOn
bKVId1qcMi+jQXYgNPIC6kQ4lJU7Gz4uqtUwBTdK3aNWocCwHl1+sMx2hExmxO7QXlH/gMyF8JPb2
ktVZv6D4kDMZxrA0BvAwaSyN9gfjydDLbBCQ0oGiJRC7Og0XhHJrkVGw580tvY0g==;
Return-path: <igor.novikov1986@mail.ru>
Received: by exim-smtp-5c7f98b595-9bcx5 with esmtpa (envelope-from <igor.novikov1986@mail.ru>)
id 1tC9pB-00000000F7I-2M0q
for igor.novikov1986@mail.ru; Sat, 16 Nov 2024 06:50:43 +0300
Message-Id: <E1tC9pB-00000000F7I-2M0q.igor-novikov1986-mail-ru@exim-smtp-5c7f98b595-9bcx5>
From: igor.novikov1986@mail.ru
Date: Sat, 16 Nov 2024 06:50:43 +0300
Authentication-Results: exim-smtp-5c7f98b595-9bcx5; auth=pass smtp.auth=igor.novikov1986@mail.ru
smtp.mailfrom=igor.novikov1986@mail.ru
X-Mailru-Src: smtp
X-4EC0790: 1
X-7564579A: 646B95376F6C166E
X-77F55803:
4F1203BC0FB41BD980BE60BCDF25A5691800B77A5852DE5626A57C8FE4E52768182A05F538085040AF4EE00D7539
23D83DE06ABAFEAF670543B3F693B97D9D0A8FC9E24F67D9D7D3522A44AD59F264DF
X-7FA49CB5:
FF5795518A3D127A4AD6D5ED66289B5259CC434672EE6371C2A783ECEC0211AD4AD6D5ED66289B524E70A05D12
97E1BBAC83A81C8FD4AD239742502CCDD46D0DB5C78E0E843E24DAAC83A81C8FD4AD23D82A6BABE6F325ACD27

06015D7A263A7C38EE81DF43BD05AC6CDE5D1141D2B1C975F05D08FF117F3FEBF82CC325AD445E2DFDCBDCF5A69
48B41B459C0287730CCACD7DF95DA8FC8BD5E8D9A59859A8B6A0EE70D6C4970CA7A471835C12D1D9774AD6D5ED
66289B5278DA827A17800CE7212612128AA291179FA2833FD35BB23D2EF20D2F80756B5F868A13BD56FB6657A47
1835C12D1D977725E5C173C3A84C3CAFEFF123806BC82117882F4460429728AD0CFFFB425014E868A13BD56FB665
7D81D268191BDAD3DC09775C1D3CA48CFF05F678DE7EF82F1BA3038C0950A5D36C8A9BA7A39EFB766D91E3A1F1
90DE8FDBA3038C0950A5D36D5E8D9A59859A8B6AF7C1DACA10CF2AB76E601842F6C81A1F004C906525384303E02
D724532EE2C3F43C7A68FF6260569E8FC8737B5C2249E5E764EB5D94DBD4E827F84554CEF50127C277FBC8AE2E8B
A83251EDC214901ED5E8D9A59859A8B600CCD0CEE79FE7A3089D37D7C0E48F6C5571747095F342E88FB05168BE4C
E3AF
X-87b9d050: 1
X-C1DE0DAB:
0D63561A33F958A5A000A8C12297383B5002B1117B3ED69601D4D0EE689BF02747A99E6294EE8661823CB91A9FED
034534781492E4B8EEAD0CC187E7B980E3E7C79554A2A72441328621D336A7BC284946AD531847A6065A535571D
14F44ED41
X-C8649E89:
1C3962B70DF3F0AD73CAD6646DEDE1915D665688C8F0839577DD89D51EBB7742D3581295AF09D3DF87807E08234
42EA2ED31085941D9CD0AF7F820E7B07EA4CF64B5CF36F8358D77076D183E61D5C5E02AEF62043A2292EBDCBAAF
4CAA6964A3EBD4E4314DA78065BC2BD7C9E12E88DE09C5D6C2DB065A97FCEF77066F19A09A7C79991AF1108DE53
83C93C5AD2DA449913E6812662D5F2A17CD894B68AE32D4C7882C87067C7812D641C46E99723820CC2E138FFB4A
CBED
X-D57D3AED:
3ZO7eAau8CL7WIMRKs4sN3D3tLDjz0dLbV79QFUyzQ2Ujvy7cMT6pYYqY16iZVKkSc3dCLJ7zSJH7+u4VD18S7Vl4ZUrpaV
fd2+vE6kuoey4m4VkSEu530nj6fImhcD4MUrOEAnl0W826KZ9Q+tr5ycPtXkTV4k65bRjmOUUP8cvGozZ33TWg5HZplvh
hXbhDGzqmQDTd6OAevLeAnq3Ra9uf7zvY2zzsIhlcp/Y7m53TZgf2aB4JOg4gkr2biojTepOeAKVzwnr8YxETnct4A==
X-F696D7D5: mNmChv+vf1D1jcri5e/lpl0GAF10tQoDGbZqfsEk4aJ7S+NbloiRr8SFaSSc9wbX
X-Mailru-Sender:
D343CA3F660FD3B9BE26B7A8EE5F7E3C4BBA9C45C3A08320BCF21BD12D43251C4575D139BC71989476F9B73EB83
D6B8E429B340BAA44668E345D137B0D0974DF8C22379A334CEFA02FF8E7307A4FDB1FADB96C49FFED27524B34AF
81CEAECC31D558B9A725E586C1FAAFEF51054327844D052C77F8525D5250EED3D69EA06B8EB4A721A3011E896F
X-Mras: Ok
X-Mailru-Intl-Transport: d,2a2f988

Письмо было отправлено 16 ноября 2024 года с адреса igor.novikov1986@mail.ru и доставлено на тот же адрес. Путь письма проходил через систему Mail.Ru с использованием стандартных механизмов аутентификации и защиты, таких как DKIM и SMTP AUTH. Проверки подтвердили подлинность отправителя и источника письма. Почта была обработана и передана через внутренние сервисы Mail.Ru.

3.10. Разработаем клиентское почтовое приложение, реализующие протоколы SMTP и POP3.

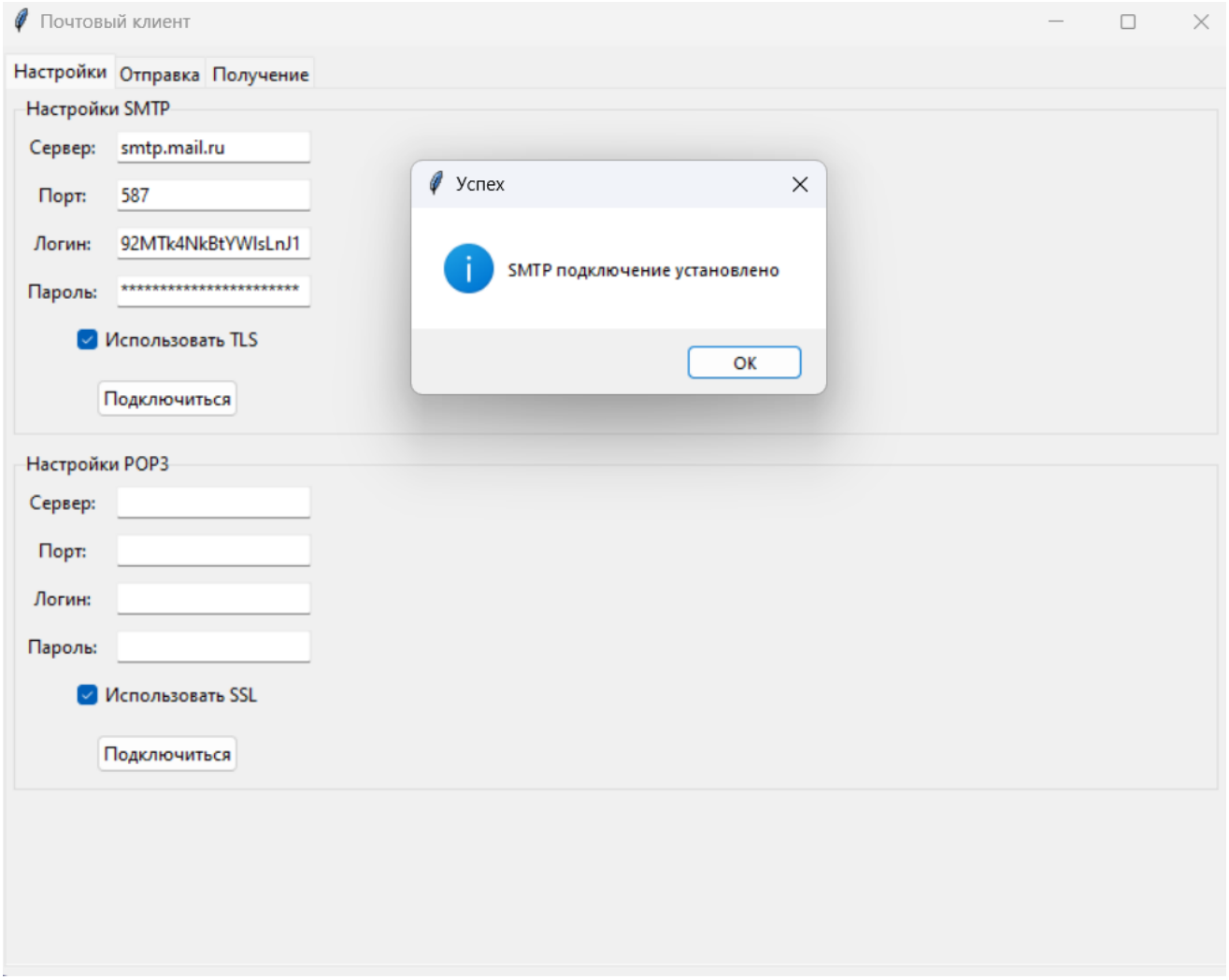


Рисунок 11 – Пример работы разработанного приложения.

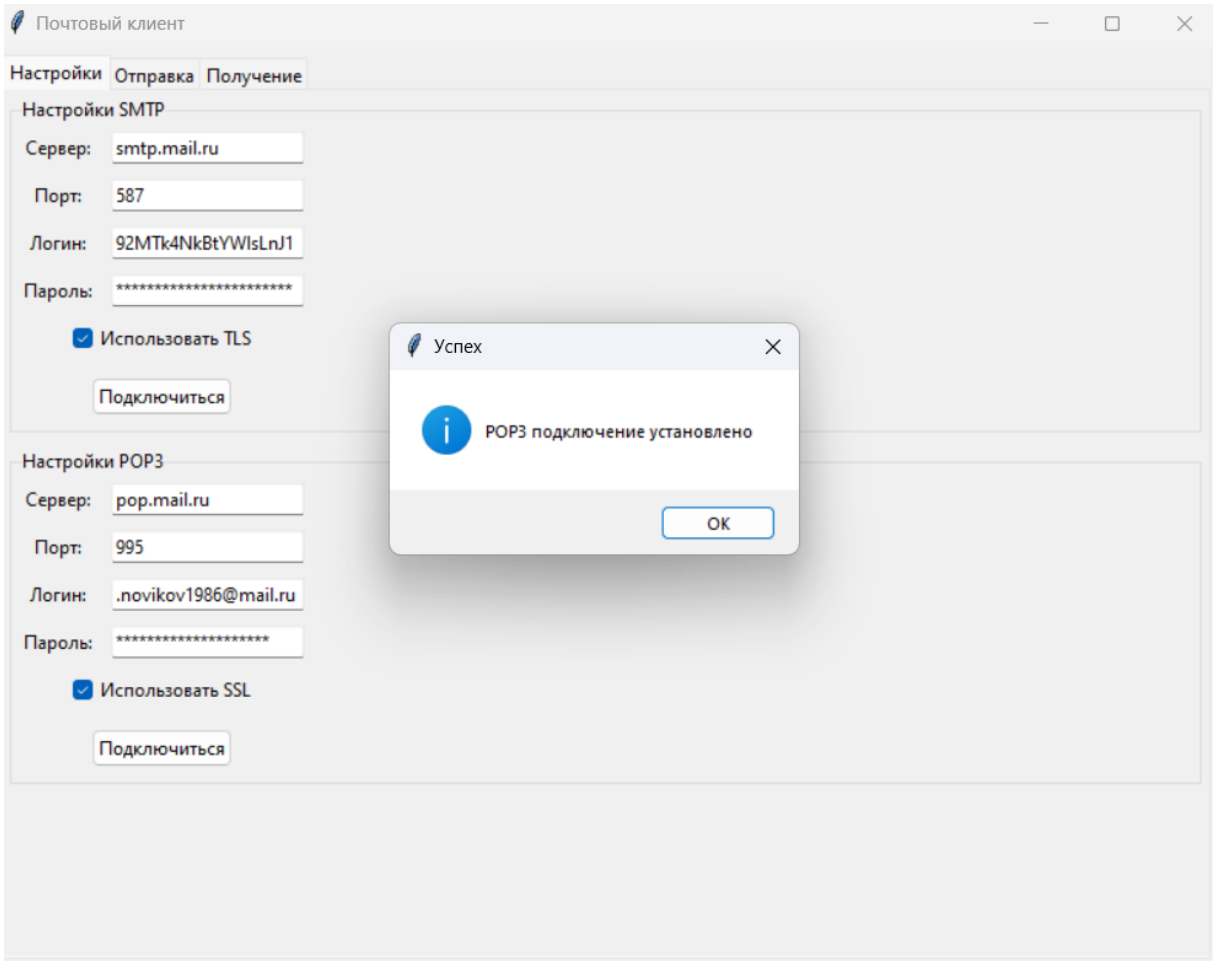


Рисунок 12 – Пример работы разработанного приложения.

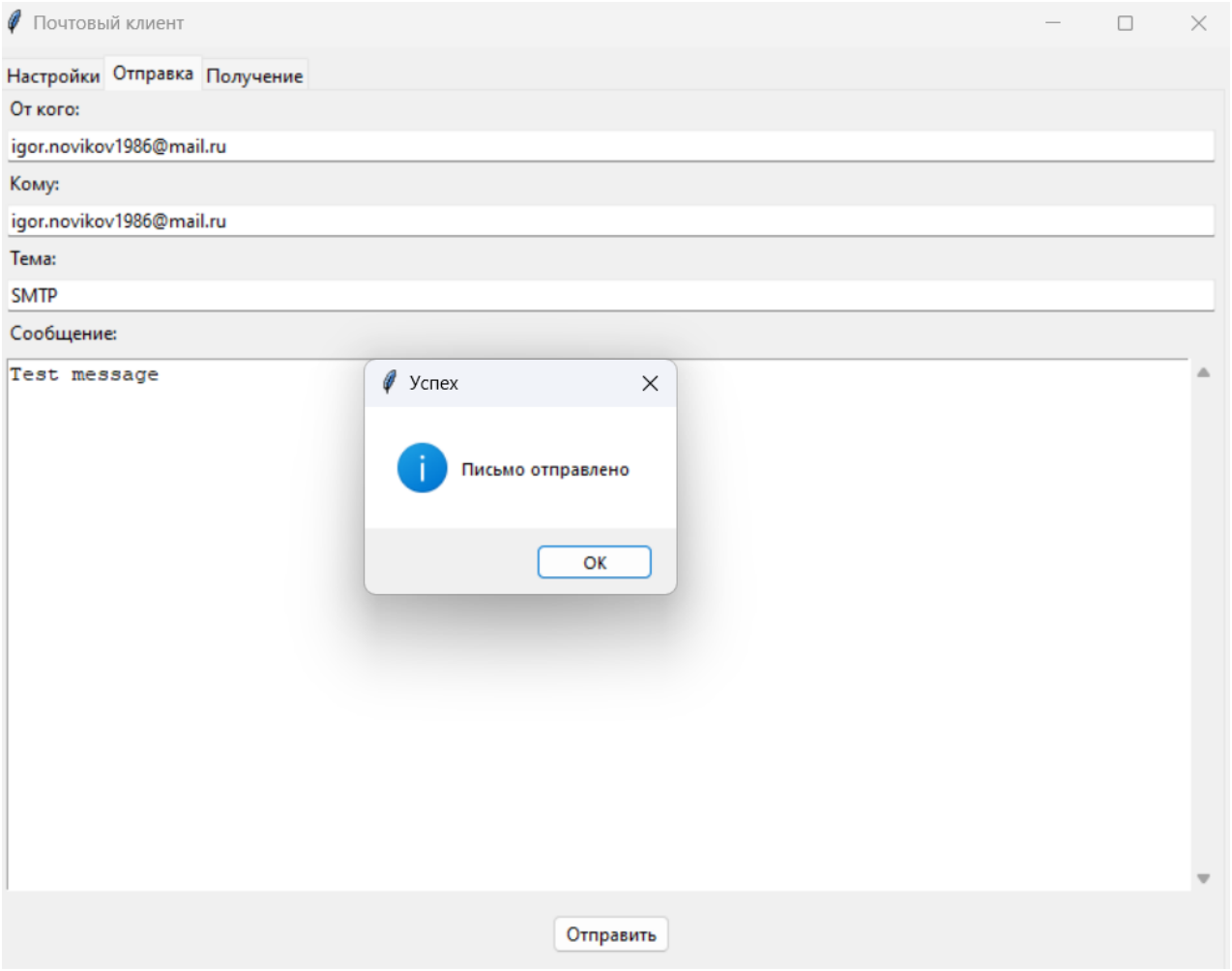


Рисунок 13 – Пример работы разработанного приложения.

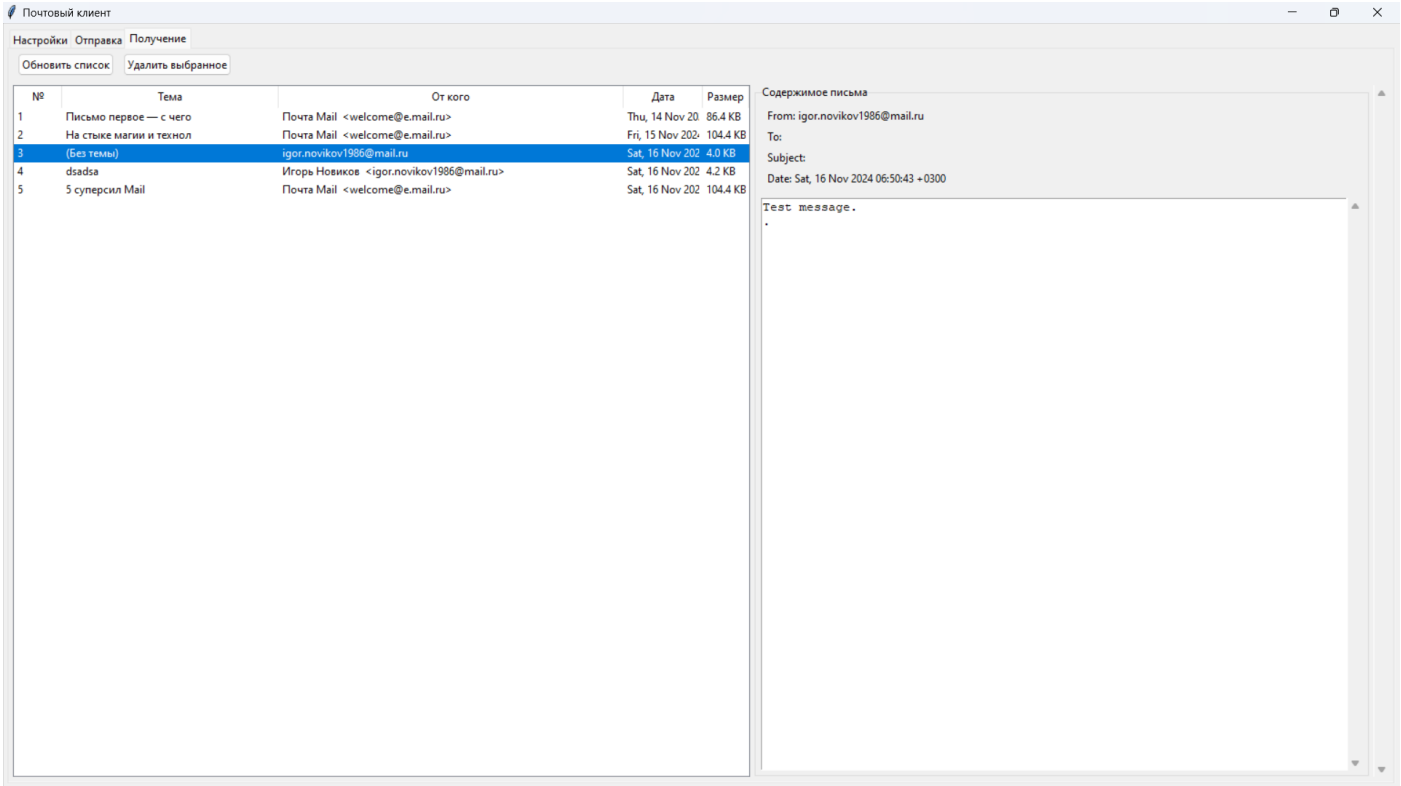


Рисунок 14 – Пример работы разработанного приложения.

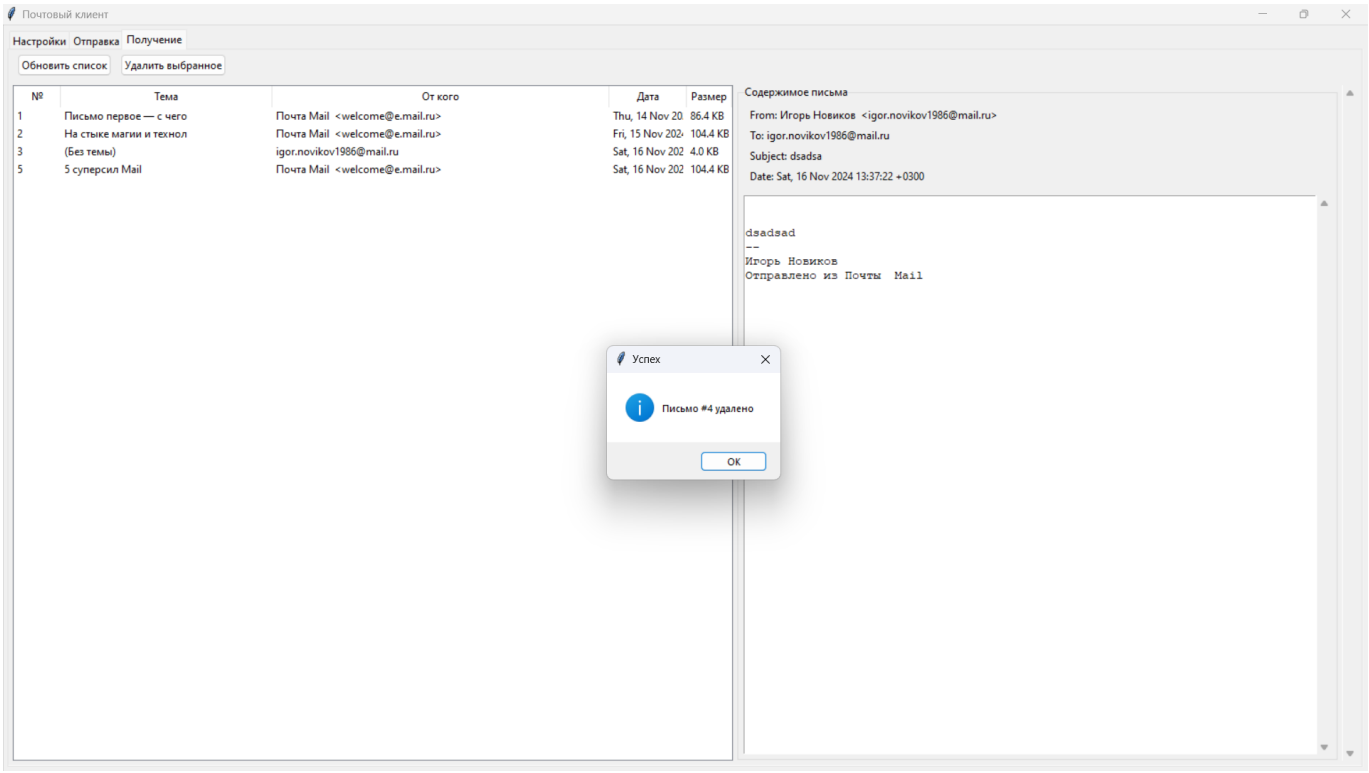


Рисунок 15 – Пример работы разработанного приложения.

4. Вывод

В ходе работы были изучены принципы работы протоколов SMTP и POP3, а также реализованы клиентские приложения для отправки и получения электронной почты. Особое внимание уделено безопасности, включая использование STARTTLS для шифрования сессий.

Проведен анализ сетевого трафика с помощью Wireshark, что подтвердило корректность взаимодействия приложений с почтовыми серверами. Проверка служебных заголовков сообщений доказала успешность отправки и получения писем.

Результатом работы стало создание функционального приложения с детальной настройкой взаимодействия, что позволило лучше понять работу сетевых протоколов.

5. Код программы

SMTP.py

```
import socket
import ssl
import datetime


class SMTPClient:
    def __init__(self):
        self.socket = None
        self.log_file = f"smtp_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.log"

    def log_message(self, message, direction=''):
```

```
    """Записывает сообщение в лог-файл"""
    timestamp = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    with open(self.log_file, 'a', encoding='utf-8') as f:
        f.write(f"{timestamp} {direction} {message}\n")
    print(f"{direction} {message}")

def receive_response(self):
    """Получает ответ от сервера"""
    try:
        response = self.socket.recv(1024).decode()
        self.log_message(response, '<--')
        return response
    except Exception as e:
        self.log_message(f"Ошибка при получении ответа: {str(e)}")
        return None

def send_command(self, command):
    """Отправляет команду серверу"""
    try:
        self.log_message(command, '-->')
        self.socket.send(f"{command}\r\n".encode())
        return self.receive_response()
    except Exception as e:
        self.log_message(f"Ошибка при отправке команды: {str(e)}")
        return None

def connect(self, server, port):
    """Устанавливает соединение с SMTP-сервером"""
    try:
        # Создаем обычный сокет
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.settimeout(10)  # Устанавливаем таймаут
        self.socket.connect((server, port))

        # Получаем приветственное сообщение
        initial_response = self.receive_response()

        if port == 587:  # Для STARTTLS
            # Отправляем EHLO
            self.send_command(f"EHLO {socket.gethostname()}")

            # Отправляем STARTTLS
            response = self.send_command("STARTTLS")
            if response and response.startswith('220'):
                # Создаем SSL контекст
                context = ssl.create_default_context()
                # Оборачиваем существующий сокет в SSL
                self.socket = context.wrap_socket(self.socket, server_hostname=server)
                # После STARTTLS нужно снова отправить EHLO
                self.send_command(f"EHLO {socket.gethostname()}")
            else:
                raise Exception("STARTTLS не поддерживается сервером")

        print("\nСоединение установлено. Теперь вы можете вводить SMTP команды.")
        print("Для завершения работы введите 'QUIT'\n")
        return True
```

```
        except Exception as e:
            self.log_message(f"Ошибка подключения: {str(e)}")
            return False

    def close(self):
        """Закрывает соединение"""
        if self.socket:
            self.socket.close()


def main():
    print("Добро пожаловать в SMTP клиент!")
    print("\nПримеры SMTP команд:")
    print("AUTH LOGIN")  # Добавлена команда аутентификации
    print("MAIL FROM:<sender@example.com>")
    print("RCPT TO:<recipient@example.com>")
    print("DATA")
    print("QUIT\n")

    # Запрашиваем параметры подключения
    server = input("Введите адрес SMTP-сервера: ")
    port = int(input("Введите порт: "))

    client = SMTPClient()

    if not client.connect(server, port):
        print("Не удалось подключиться к серверу")
        return

    # Основной цикл работы с командами
    while True:
        command = input("\nВведите SMTP команду: ")

        if not command:
            continue

        if command.upper() == 'DATA':
            # Особая обработка команды DATA
            client.send_command(command)
            print("\nВведите текст сообщения.")
            print("Для завершения введите точку (.) в отдельной строке.\n")

            message_lines = []
            while True:
                line = input()
                message_lines.append(line)
                if line == '.':
                    break

            message = '\r\n'.join(message_lines)
            client.send_command(message)

        elif command.upper() == 'QUIT':
            client.send_command(command)
            break
        else:
```

```
            client.send_command(command)

    client.close()
    print("\nСоединение закрыто. Сессия сохранена в файле", client.log_file)


if __name__ == "__main__":
    main()
```

POP3

```
import socket
import ssl
import sys
from datetime import datetime
import re
import base64
import quopri
import email
from email.header import decode_header
from email.parser import Parser


class POP3Client:
    def __init__(self, server, port, use_ssl=True):
        self.server = server
        self.port = port
        self.socket = None
        self.connected = False
        self.use_ssl = use_ssl
        self.log_file = f"pop3_{datetime.now().strftime('%Y%m%d_%H%M%S')}.log"

    def log_message(self, message, direction=""):
        timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        with open(self.log_file, 'a', encoding='utf-8') as f:
            f.write(f"[{timestamp}] {direction} {message}\n")
        print(f"{direction} {message}")

    def connect(self):
        try:
            plain_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            if self.use_ssl:
                context = ssl.create_default_context()
                self.socket = context.wrap_socket(plain_socket, server_hostname=self.server)
            else:
                self.socket = plain_socket

            self.socket.connect((self.server, self.port))
            response = self.socket.recv(1024).decode('utf-8')
            self.log_message(response, "SERVER:")
            self.connected = True
            return True
        except Exception as e:
            self.log_message(f"Connection error: {str(e)}", "ERROR:")
            return False

    def send_command(self, command):
```

```
    if not self.connected:
        self.log_message("Not connected to server", "ERROR:")
        return None

    try:
        self.log_message(command, "CLIENT:")
        self.socket.send(f"{command}\r\n".encode('utf-8'))
        response = self.socket.recv(1024).decode('utf-8')
        self.log_message(response, "SERVER:")
        return response
    except Exception as e:
        self.log_message(f"Error sending command: {str(e)}", "ERROR:")
        return None

def receive_multiline(self):
    try:
        response = ""
        while True:
            line = self.socket.recv(1024).decode('utf-8', errors='replace')
            response += line
            if line.endswith('\r\n.\r\n'):
                break
        self.log_message("Получено многострочное сообщение", "SERVER:")
        return response
    except Exception as e:
        self.log_message(f"Error receiving response: {str(e)}", "ERROR:")
        return None

def decode_message(self, message_data):
    try:
        # Парсим email сообщение
        email_message = email.message_from_string(message_data)

        print("\n=== Заголовки сообщения ===")
        # Декодируем и выводим основные заголовки
        for header in ['From', 'To', 'Subject', 'Date']:
            if header in email_message:
                value = email_message[header]
                decoded_header = decode_header(value)
                decoded_value = ''
                for part, charset in decoded_header:
                    if isinstance(part, bytes):
                        decoded_value += part.decode(charset or 'utf-8', errors='replace')
                    else:
                        decoded_value += part
                print(f"{header}: {decoded_value}")

        print("\n=== Содержимое сообщения ===")
        # Обрабатываем тело сообщения
        if email_message.is_multipart():
            for part in email_message.walk():
                if part.get_content_type() == "text/plain":
                    content = part.get_payload()
                    encoding = part.get('Content-Transfer-Encoding', '')

                    if encoding.lower() == 'base64':
```

```
                content = base64.b64decode(content).decode('utf-8', errors='replace')
            elif encoding.lower() == 'quoted-printable':
                content = quopri.decodestring(content).decode('utf-8', errors='replace')

            print(content)
            break
        else:
            content = email_message.get_payload()
            encoding = email_message.get('Content-Transfer-Encoding', '')

            if encoding.lower() == 'base64':
                content = base64.b64decode(content).decode('utf-8', errors='replace')
            elif encoding.lower() == 'quoted-printable':
                content = quopri.decodestring(content).decode('utf-8', errors='replace')

            print(content)

    except Exception as e:
        print(f"Ошибка при декодировании сообщения: {str(e)}")

  def close(self):
    if self.socket:
      self.send_command("QUIT")
      self.socket.close()
      self.connected = False


def print_available_commands():
  print("\nДоступные команды:")
  print("USER username - ввести имя пользователя")
  print("PASS password - ввести пароль")
  print("STAT - получить статистику почтового ящика")
  print("LIST - получить список сообщений")
  print("RETR n - получить сообщение номер n")
  print("DELE n - пометить сообщение n на удаление")
  print("RSET - отменить все помеченные на удаление")
  print("NOOP - проверка соединения")
  print("QUIT - завершить сессию")
  print("HELP - показать эту справку")


def main():
  server = input("Введите адрес POP3 сервера: ")
  use_ssl = input("Использовать SSL/TLS? (y/n): ").lower() == 'y'

  if use_ssl:
    default_port = 995
  else:
    default_port = 110

  port = int(input(f"Введите порт (по умолчанию {default_port}): ") or str(default_port))

  client = POP3Client(server, port, use_ssl)

  if not client.connect():
    print("Не удалось подключиться к серверу")
```

```
        return

    print_available_commands()

    while True:
        command = input("\nВведите команду (HELP для справки): ").strip()

        if command.upper() == "HELP":
            print_available_commands()
            continue

        if command.upper() == "QUIT":
            client.close()
            break

        if command.upper().startswith(("RETR")):
            response = client.send_command(command)
            if response and "+OK" in response:
                message_data = client.receive_multiline()
                if message_data:
                    client.decode_message(message_data)
        elif command.upper().startswith("LIST"):
            response = client.send_command(command)
            if response and "+OK" in response:
                list_data = client.receive_multiline()
                print(list_data)
        else:
            client.send_command(command)


if __name__ == "__main__":
    main()
```

SMTP_POP3

```
from datetime import datetime
from POP3.main import POP3Client
from SMTP.main import SMTPClient
import base64
from email_decoder import EmailDecoder


class EmailClient:
    def __init__(self):
        self.smtp_client = None
        self.pop3_client = None
        self.smtp_authenticated = False
        self.pop3_authenticated = False
        self.log_file = f"email_client_{datetime.now().strftime('%Y%m%d_%H%M%S')}.log"

    def setup_smtp(self, server, port, username, password, use_tls=True):
        try:
            self.smtp_client = SMTPClient()
            if not self.smtp_client.connect(server, port):
                return False

            if use_tls:
```

```
            self.smtp_client.send_command("STARTTLS")

        # Выполняем аутентификацию SMTP
        auth_string = base64.b64encode(f"\0{username}\0{password}".encode()).decode()
        self.smtp_client.send_command("AUTH PLAIN " + auth_string)

        self.smtp_authenticated = True
        self.log_message("SMTP аутентификация успешна", "ИНФО:")
        return True

    except Exception as e:
        self.log_message(f"Ошибка настройки SMTP: {str(e)}", "ОШИБКА:")
        return False

def check_smtp_auth(self):
    if not self.smtp_authenticated:
        self.log_message("Требуется аутентификация SMTP", "ОШИБКА:")
        return False
    return True

def check_pop3_auth(self):
    """
    Проверяет статус аутентификации POP3 и соединение с сервером.
    Возвращает True, если соединение активно и аутентификация выполнена успешно.
    """
    if not self.pop3_authenticated or not self.pop3_client:
        self.log_message("POP3 клиент не аутентифицирован или не подключен", "ОШИБКА:")
        return False
    try:
        # Проверяем соединение с помощью NOOP команды
        response = self.pop3_client.send_command("NOOP")
        if not response or "+OK" not in response:
            self.pop3_authenticated = False
            self.log_message("POP3 соединение потеряно", "ОШИБКА:")
            return False
        return True
    except Exception as e:
        self.pop3_authenticated = False
        self.log_message(f"Ошибка при проверке POP3 соединения: {str(e)}", "ОШИБКА:")
        return False

def setup_pop3(self, server, port, username, password, use_ssl=True):
    try:
        self.pop3_client = POP3Client(server, port, use_ssl)
        if not self.pop3_client.connect():
            return False

        # Выполняем аутентификацию POP3
        user_response = self.pop3_client.send_command(f"USER {username}")
        if not user_response or "+OK" not in user_response:
            self.log_message("Ошибка при отправке команды USER", "ОШИБКА:")
            return False

        pass_response = self.pop3_client.send_command(f"PASS {password}")
        if not pass_response or "+OK" not in pass_response:
            self.log_message("Ошибка при отправке команды PASS", "ОШИБКА:")
```

```
            return False

        self.pop3_authenticated = True
        self.log_message("POP3 аутентификация успешна", "ИНФО:")
        return True

    except Exception as e:
        self.log_message(f"Ошибка настройки POP3: {str(e)}", "ОШИБКА:")
        return False

def list_messages(self):
    """
    Получает список сообщений с сервера POP3.
    Возвращает список кортежей (номер, размер, заголовки).
    """
    if not self.pop3_client or not self.check_pop3_auth():
        self.log_message("POP3 клиент не настроен или не авторизован", "ОШИБКА:")
        return None

    try:
        # Получаем список сообщений
        response = self.pop3_client.send_command("LIST")
        if not response or "+OK" not in response:
            self.log_message("Ошибка при получении списка сообщений", "ОШИБКА:")
            return None

        messages_data = self.pop3_client.receive_multiline()
        if not messages_data:
            return []

        # Парсим список сообщений
        messages = []
        for line in messages_data.split('\n'):
            if line.strip():
                parts = line.strip().split()
                if len(parts) >= 2:
                    try:
                        msg_num = int(parts[0])
                        msg_size = int(parts[1])

                        # Получаем заголовки для каждого сообщения
                        headers = self.get_message_headers(msg_num)
                        messages.append((msg_num, msg_size, headers))
                    except ValueError:
                        continue

        self.log_message(f"Получено {len(messages)} сообщений", "ИНФО:")
        return messages

    except Exception as e:
        self.log_message(f"Ошибка при получении списка сообщений: {str(e)}", "ОШИБКА:")
        return None

def log_message(self, message, level="ИНФО:"):
    """Записывает сообщение в лог-файл"""
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

```
        log_entry = f"{timestamp} {level} {message}\n"
        try:
            with open(self.log_file, 'a', encoding='utf-8') as f:
                f.write(log_entry)
        except Exception as e:
            print(f"Ошибка записи в лог: {str(e)}")

    def get_message_headers(self, msg_number):
        if not self.pop3_client or not self.check_pop3_auth():
            return None

        try:
            response = self.pop3_client.send_command(f"TOP {msg_number} 0")
            if not response or "+OK" not in response:
                return None

            headers_data = self.pop3_client.receive_multiline()
            if not headers_data:
                return None

            # Парсим заголовки
            headers = {}
            current_header = None
            current_value = []

            for line in headers_data.split('\n'):
                line = line.strip()
                if not line:
                    continue

                if line[0] in [' ', '\t'] and current_header:
                    current_value.append(line.strip())
                else:
                    if current_header:
                        headers[current_header] = EmailDecoder.decode_header_value(' '.join(current_value))

                    if ':' in line:
                        current_header = line.split(':', 1)[0].strip()
                        current_value = [line.split(':', 1)[1].strip()]

            if current_header:
                headers[current_header] = EmailDecoder.decode_header_value(' '.join(current_value))

            return headers

        except Exception as e:
            self.log_message(f"Ошибка при получении заголовков сообщения {msg_number}: {str(e)}", "ОШИБКА:")
            return None

    def send_email(self, from_addr, to_addr, subject, message):
        if not self.smtp_client or not self.check_smtp_auth():
            print("SMTP клиент не настроен или не авторизован")
            return False

    def read_message(self, msg_number):
        if not self.pop3_client or not self.check_pop3_auth():
```

```
            return None

        try:
            response = self.pop3_client.send_command(f"RETR {msg_number}")
            if not response or "+OK" not in response:
                return None

            message_data = self.pop3_client.receive_multiline()
            if not message_data:
                return None

            # Декодируем и возвращаем содержимое сообщения
            return EmailDecoder.decode_message_content(message_data)

        except Exception as e:
            self.log_message(f"Ошибка при чтении сообщения {msg_number}: {str(e)}", "ОШИБКА:")
            return None

    def delete_message(self, msg_number):
        if not self.pop3_client or not self.check_pop3_auth():
            print("POP3 клиент не настроен или не авторизован")
            return

    def close(self):
        if self.smtp_client:
            if self.smtp_authenticated:
                self.smtp_client.send_command("QUIT")
            self.smtp_client.close()
        if self.pop3_client:
            if self.pop3_authenticated:
                self.pop3_client.send_command("QUIT")
            self.pop3_client.close()


def print_menu():
    print("\nПочтовый клиент - Главное меню")
    print("1. Настроить SMTP")
    print("2. Настроить POP3")
    print("3. Отправить письмо")
    print("4. Просмотреть список писем")
    print("5. Прочитать письмо")
    print("6. Удалить письмо")
    print("0. Выход")


def main():
    client = EmailClient()

    while True:
        print_menu()
        choice = input("\nВыберите действие (0-6): ")

        if choice == "0":
            client.close()
            print("Программа завершена")
            break
```

```
    elif choice == "1":
        server = input("Введите SMTP сервер: ")
        port = int(input("Введите порт SMTP: "))
        username = input("Введите имя пользователя: ")
        password = input("Введите пароль: ")
        if client.setup_smtp(server, port, username, password):
            print("SMTP успешно настроен и авторизован")
        else:
            print("Ошибка настройки или авторизации SMTP")

    elif choice == "2":
        server = input("Введите POP3 сервер: ")
        port = int(input("Введите порт POP3: "))
        username = input("Введите имя пользователя: ")
        password = input("Введите пароль: ")
        use_ssl = input("Использовать SSL? (д/н): ").lower() == 'д'
        if client.setup_pop3(server, port, username, password, use_ssl):
            print("POP3 успешно настроен и авторизован")
        else:
            print("Ошибка настройки или авторизации POP3")

    elif choice == "3":
        from_addr = input("От кого: ")
        to_addr = input("Кому: ")
        subject = input("Тема: ")
        print("Введите текст сообщения (для завершения введите пустую строку):")
        lines = []
        while True:
            line = input()
            if not line:
                break
            lines.append(line)
        message = "\n".join(lines)
        client.send_email(from_addr, to_addr, subject, message)

    elif choice == "4":
        client.list_messages()

    elif choice == "5":
        msg_num = input("Введите номер сообщения: ")
        client.read_message(msg_num)

    elif choice == "6":
        msg_num = input("Введите номер сообщения для удаления: ")
        client.delete_message(msg_num)

    else:
        print("Неверный выбор. Попробуйте снова.")


if __name__ == "__main__":
    main()
```