МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №7

по дисциплине: Компьютерные сети
тема: «Протоколы SMTP и POP3»

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович

Проверили:
Рубцов Константин Анатольевич

Белгород 2025 г.

Цель работы: изучить принципы и характеристику протоколов POP3 и SMTP и составить программу для приема/отправки электронной почты.

Краткие теоретические сведения

Протокол POP3

POP3 (англ. Post Office Protocol Version 3) - стандартный Интернет-протокол прикладного уровня, используемый клиентами электронной почты для извлечения электронного сообщения с удаленного сервера по TCP/IP-соединению.

В некоторых небольших узлах Интернет бывает непрактично поддерживать систему передачи сообщений (MTS - Message Transport System). Рабочая станция может не иметь достаточных ресурсов для обеспечения непрерывной работы SMTP-сервера [RFC-821]. Для “домашних ЭВМ” слишком дорого поддерживать связь с Интернет круглые сутки.

Но доступ к электронной почте необходим как для таких малых узлов, так и индивидуальных ЭВМ. Для решения этой проблемы разработан протокол POP3 (Post Office Protocol - Version 3, STD- 53. M. Rose, RFC-1939). Этот протокол обеспечивает доступ узла к базовому почтовому серверу.

POP3 не ставит целью предоставление широкого списка манипуляций с почтой. Почтовые сообщения принимаются почтовым сервером и сохраняются там, пока на рабочей станции клиента не будет запущено приложение POP3. Это приложение устанавливает соединение с сервером и забирает сообщения оттуда. Почтовые сообщения на сервере стираются.

POP3 поддерживает простые требования «загрузи-и-удали» для доступа к удаленным почтовым ящикам. Хотя большая часть POP- клиентов предоставляют возможность оставить почту на сервере после загрузки, использующие POP клиенты обычно соединяются, извлекают все письма, сохраняют их на пользовательском компьютере как новые сообщения, удаляют их с сервера, после чего разъединяются.

Другие протоколы, в частности IMAP, предоставляют более полный и комплексный удаленный доступ к типичным операциям с почтовым ящиком. Многие клиенты электронной почты поддерживают как POP, так и IMAP; однако, гораздо меньше интернет-провайдеров поддерживают IMAP.

POP3-сервер прослушивает порт 110. Шифрование связи для POP3 запрашивается после запуска протокола, с помощью либо команды STLS (если она поддерживается), либо POP3S, которая соединяется с сервером используя TLS или SSL по TCP-порту 995.

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

Обычно POP3-сервис устанавливается на 110-й TCP -порт сервера, который будет находиться в режиме ожидания входящего соединения. Когда клиент хочет воспользоваться POP3-сервисом, он просто устанавливает TCP-соединение с портом 110 этого хоста. После установления соединения сервис POP3 отправляет подсоединившемуся клиенту приветственное сообщение. После этого клиент и сервер начинают обмен командами и данными. По окончании обмена POP3-канал закрывается.

Команды POP3 состоят из ключевых слов, состоящих из ASCII- символов, и одним или несколькими параметрами, отделяемыми друг от друга символом "пробела" - <SP>. Все команды заканчиваются символами "возврата каретки" и "перевода строки" - <CRLF>. Длина ключевых слов не превышает четырех символов, а каждого из аргументов может быть до 40 символов.

Ответы POP3-сервера на команды состоят из строки статус- индикатора, ключевого слова, строки дополнительной информации и символов завершения строки. Длина строки ответа может достигать 512 символов. Строка статус-индикатора принимает два значения: положительное ("+OK") и отрицательное ("-ERR"). Любой сервер POP3 обязан отправлять строки статус-индикатора в верхнем регистре, тогда как другие команды и данные могут приниматься или отправляться как в нижнем, так и в верхнем регистрах. Ответы POP3-сервера на отдельные команды могут составлять несколько строк. В этом случае строки разделены символами <CRLF>. Последнюю строку информационной группы завершает строка, состоящая из символа "." (код — 046) и <CRLF>, т. е. последовательность "CRLF.CRLF".

POP3-сессия состоит из нескольких частей. Как только открывается TCP-соединение и POP3-сервер отправляет приветствие, сессия должна быть зарегистрирована - состояние аутентификации (AUTHORIZATION state). Клиент должен зарегистрироваться в POP3-сервере, т. е. ввести свой идентификатор и пароль. После этого сервер предоставляет клиенту его почтовый ящик и открывает для данного клиента транзакцию - состояние начала транзакции обмена (TRANSACTION state). На этой стадии клиент может считать и удалить почту своего почтового ящика.

После того как клиент заканчивает работу (передает команду QUIT), сессия переходит в состояние UPDATE - завершение транзакции. В этом состоянии POP3-сервер закрывает транзакцию данного клиента (на языке баз данных - операция COMMIT) и закрывает TCP-соединение. В случае получения неизвестной, неиспользуемой или неправильной команды, POP3-сервер должен ответить отрицательным состоянием индикатора. POP3-сервер может использовать в своей работе таймер контроля времени соединения. Этот таймер отсчитывает время "бездействия" ("idle") клиента в сессии от последней переданной команды. Если время сессии истекло, сервер закрывает TCP-соединение, не переходя в состояние UPDATE (иными словами, откатывает транзакцию или на языке баз данных — выполняет ROLLBACK) [25].

POP3-сервер может обслуживать группу клиентов, которые, возможно, присоединяются по коммутируемой линии, и, следовательно, необходимо иметь средство автоматического регулирования времени соединения. По спецификации POP3-таймер контроля состояния "idle" должен быть установлен на промежуток времени не менее 10 минут.

Команды протокола POP3:

- USER - идентифицирует пользователя с указанным именем;
- PASS - указывает пароль для пары клиент-сервер;
- QUIT - закрывает TCP-соединение;
- STAT - сервер возвращает количество сообщений в почтовом

ящике плюс размер почтового ящика;

- LIST - сервер возвращает идентификаторы сообщений вместе с

размерами сообщений;

- RETR - извлекает сообщение из почтового ящика;
- DELE - отмечает сообщение для удаления;
- NOOP - Сервер возвращает положительный ответ, но не

совершает никаких действий;

- LAST - Сервер возвращает наибольший номер сообщения из

тех, к которым ранее уже обращались;

- RSET - Отменяет удаление сообщения, отмеченного ранее

командой DELE.

Протокол SMTP

SMTP (Simple Mail Transfer Protocol) - широко используемый сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP. SMTP впервые был описан в RFC 821 (1982 год); последнее обновление в RFC 5321 (2008) включает масштабируемое расширение - ESMTP (Extended SMTP). В настоящее время под «протоколом SMTP», как правило, подразумевают и его расширения. Протокол SMTP предназначен для передачи исходящей почты, используя для этого порт TCP 25 [23].

Упрощенно схема взаимодействия представлена на рис. 7.1 (объемными стрелками показано направление движения почтовых сообщений).

Со стороны пользователя обычно одна и та же программа выступает в роли и POP3 клиента, и SMTP клиента отправителя. Наиболее распространенными на данный момент являются MS Outlook, The Bat, Netscape Messenger, Eudora, Pegasus mail, Mutt, Pine и др. При нажатии в них на кнопочку "отправить" происходит формирование очереди сообщений, и установление двустороннего сеанса общения с SMTP сервером провайдера. На схеме у пользователя есть клиентское ПО, а у провайдера – серверная часть приложения. На самом деле это немного не так. Протокол SMTP делает возможным смену сторон даже в ходе одного сеанса. Условно принято считать клиентом ту сторону, которая начинает взаимодействие и хочет отослать почту, а сервером ту, что принимает запросы. После того, как клиент посылает серверу несколько служебных команд и получает

положительные ответы на них, он отправляет SMTP серверу собственно тело сообщения. SMTP сервер получает сообщение, вносит в него дополнительные заголовки, указывающие на то, что он обработал данное послание, устанавливает связь со следующим SMTP сервером по пути следования письма. Общение между любыми SMTP серверами происходит по той же схеме. Инициирует переговоры клиент, сервер на них отвечает, а затем получает корреспонденцию и "ставит штампик" в теле письма (в его заголовочной части). Все это очень напоминает обычную бумажную почту, где работу по сортировке и отправке почты выполняют люди.

Если на каком-нибудь этапе передачи SMTP клиент обнаружит невозможность подключиться к следующему серверу (например, компьютер отправили на профилактику или аппаратура связи вышла из строя), он будет пытаться отправить сообщение через некоторое время – 1 час, 4 часа, день и т.д. до 4 суток в общем случае. Причем временные отрезки между попытками, как правило, зависят от настроек программы-пересыльщика почты. Одновременно, такой сервер должен уведомить отправителя сообщения о невозможности доставить почту, послав ему стандартное письмо "Failed delivery" (доставка невозможна) и рассказав о графике дальнейших попыток по продвижению исходного сообщения. Если канал связи не восстановится за указанный большой промежуток времени (например, 4 дня), посланная информация будет считаться утерянной.

Как только почта достигнет конечного пункта (SMTP сервера адресата сообщения), она будет сложена в почтовый ящик абонента, который всегда сможет в удобное для него время изъять ее по протоколам POP3 или IMAP, в зависимости от того, какой из них поддерживается провайдером.

Анализируя заголовок письма, можно узнать какими путями оно путешествовало, как долго длился сам путь, как называлась почтовая программа отправителя и многое другое. Получить эту информацию можно в "Свойствах письма", кликнув правой кнопкой мыши на самом письме в MS Outlook, нажав Ctrl+Shift+H в The Bat или совершить нечто подобное в других почтовых клиентах.

Рассмотрим клиент-серверное взаимодействие по протоколу SMTP. Программа пользователя, выбрав для связи соответствующий почтовый сервер, устанавливает с ним контакт на транспортном и сеансовом уровнях эталонной модели взаимодействия

открытых систем OSI/RM (в терминах TCP/IP (transmission control protocol / internet protocol) это – TCP уровень). Взаимодействие на более низких уровнях (канальном, сетевом) происходит прозрачно для обеих сторон. Протокол SMTP – протокол прикладного уровня и базируется поверх TCP. В его рамках не оговаривается ни размер сегментов данных, ни правила квитирования, ни отслеживание ошибок, возникающих при передаче информации.

По уже установленному соединению клиентское ПО передает команды SMTP серверу, ожидая тут же получить ответы. В арсенал SMTP клиента, равно как и сервера, входит около 10 команд, но, воспользовавшись только пятью из них, уже можно легально послать почтовое сообщение. Это HELO, MAIL, RCPT, DATA, QUIT. Их использование подразумевается именно в такой последовательности. HELO предназначена для идентификации отправителя, MAIL указывает адрес отправителя, RCPT – адрес назначения. После команды DATA и ответа на нее, клиент посылает серверу тело сообщения, которое должно заканчиваться строкой, содержащей лишь одну точку.

Непосредственно после установления соединения сервер выдает строчку с кодом ответа 220. В ответ на нее клиент может инициировать сеанс связи по протоколу SMTP, послав команду HELO и указав у нее в аргументах имя своего компьютера. По принятии команды HELO сервер обязан сделать запрос в DNS и, если это возможно, по IP адресу определить доменное имя компьютера клиента. (IP адрес уже известен на момент установления соединения по TCP протоколу).

Далее в команде "MAIL FROM:" клиент сообщает обратный адрес отправителя, который проверяется обычно только на корректность. После слов "RCPT TO:" следует набрать адрес электронной почты абонента на данном сервере. Клиент отсылает команду DATA и ждет приглашения начать пересылку тела письма (код 354).

Сообщение может быть достаточно длинным, но обязательно должно заканчиваться строкой, в которой есть одна-единственная точка. Это служит сигналом SMTP серверу о том, что тело письма закончилось. Он присваивает этому письму определенный идентификатор, и ждет команды QUIT, после чего сеанс считается завершенным.

Если клиент посылает сообщение, у которого в заголовочной части в поле CC указаны несколько e-mail адресов, первый по пути следования SMTP сервер должен будет в общем случае установить сеанс продвижения почты с каждым из серверов данного списка и

отослать точную копию письма каждому. В случае использования поля BCC клиент, формирующий сообщение, уничтожит запись BCC в теле сообщения и по количеству адресатов отошлет первому SMTP серверу команду "RCPT TO:" каждый раз с новым адресом в качестве аргумента. Таким образом, сервер получит указание разослать почту по многим адресатам. Причем, в этом случае получатели писем ничего не будут знать друг о друге, т.к. рассылка осуществляется посредством команд SMTP протокола.

Анализ применяемых функций

Для использования библиотеки применяется WSAStartUp, WSACleanUp и WSAGetLastError для отслеживания ошибок и логирования.

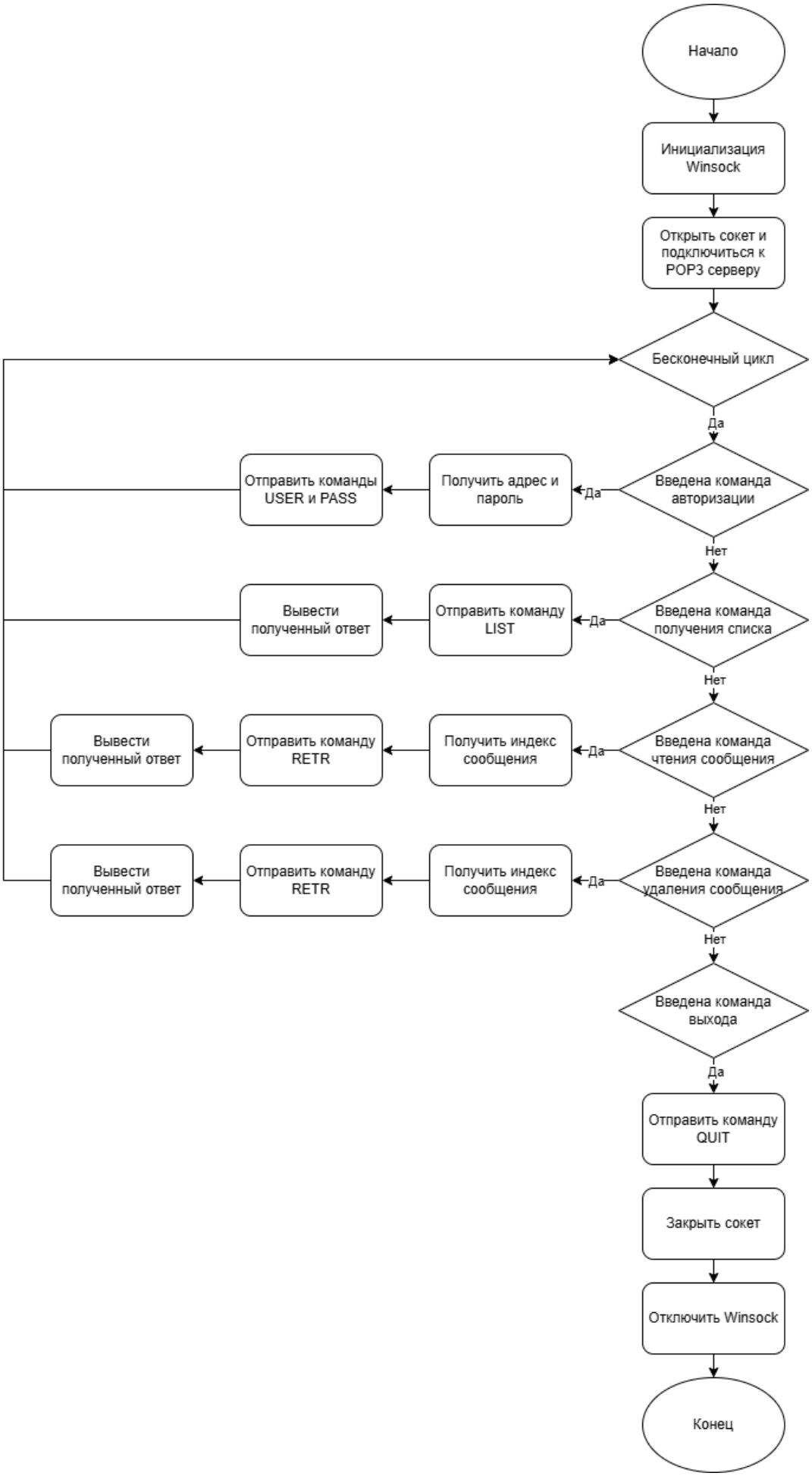
Для реализации подключения использованы функции socket и connect. На вход подавались адреса серверов SMTP и POP3, с соответствующими незащищёнными портами 25 и 110.

Для отправки команд используется send. Для получения ответа от сервера используется recv если это небольшие сообщения и recv в цикле если приходит множество строк.

Разработка программы. Блок-схемы программы

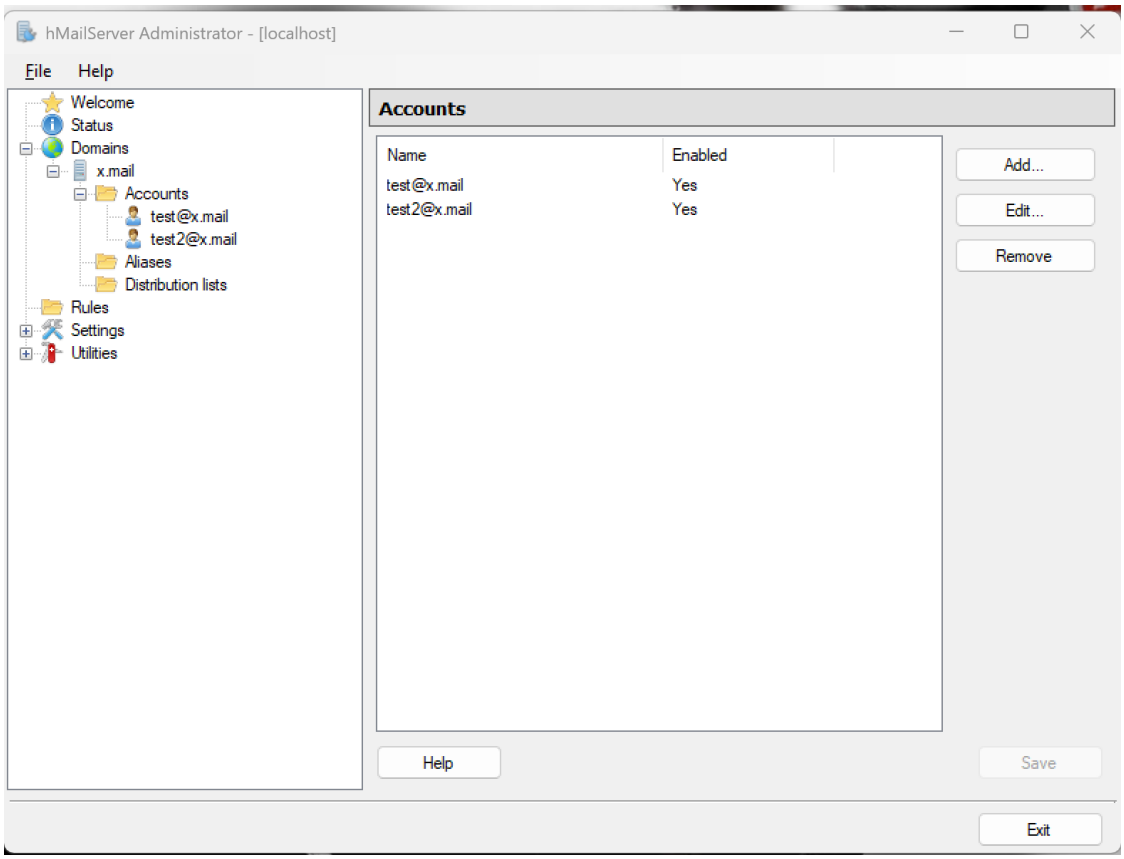
SMTP

POP3


Начало
Инициализация Winsock
Открыть сокет и подключиться к POP3 серверу
Бесконечный цикл
Да
Введена команда авторизации
Да
Получить адрес и пароль
Отправить команды USER и PASS
Нет
Введена команда получения списка
Да
Отправить команду LIST
Вывести полученный ответ
Нет
Введена команда чтения сообщения
Да
Получить индекс сообщения
Отправить команду RETR
Вывести полученный ответ
Нет
Введена команда удаления сообщения
Да
Получить индекс сообщения
Отправить команду RETR
Вывести полученный ответ
Нет
Введена команда выхода
Да
Отправить команду QUIT
Закрыть сокет
Отключить Winsock
Конец

Анализ функционирования программы

В качестве почтового сервера использовалась программа hMailServer, которая создаёт домен в локальной сети, хранит и передаёт сообщения.



Работа программы

SMTP

```
Input server domain: x.mail
Input yor addres: test@x.mail
Input yor login: test
Input yor password: test
Input destination address: test2@x.mail
Input topic: Message 1
Input massage: Hello
Input file name: mail.txt
250 Hello.

250 OK
M:<test@x.mail>

250 OK
<test2@x.mail>

354 OK, send.
x.mail>

250 Queued (0.016 seconds)

221 goodbye
.016 seconds)
```

POP3

```
+OK POP3

Select the command (Auth, List, Read, Delet, Quit): Auth
Input addres: test2@x.mail
+OK Send your password

Input password: test
+OK Mailbox locked and ready
```

```
Select the command (Auth, List, Read, Delet, Quit): List
+OK 1 messages (287 octets)


1 287
.
ages (287 octets)
```

```
Select the command (Auth, List, Read, Delet, Quit): Read
Select massage index: 1
+OK 287 octets
Return-Path: test@x.mail
Received: from Ë·I (Entik-Notebook [127.0.0.1])
        by ENTIK-NOTEBOOK with ESMTP
        ; Tue, 3 Jun 2025 03:41:27 +0300
Message-ID: <F5824330-1747-4B70-A1D0-A0388C65E2AE@ENTIK-NOTEBOOK>
From: test@x.mail
To: test2@x.mail

Topic: Message 1
Massage: Hello
File name: mail.txt
```

```
Select the command (Auth, List, Read, Delete, Quit): Delete
Select message index: 1
+OK msg deleted
```

```
Select the command (Auth, List, Read, Delete, Quit): List
+OK 0 messages (0 octets)
.
```

```
Select the command (Auth, List, Read, Delet, Quit): Quit
+OK POP3 server saying goodbye...
```

Вывод: на этой лабораторной работе были изучены принципы и характеристики протоколов POP3 и SMTP и составили программу для приема/отправки электронной почты.

Код программы:

Файл SMTP.cpp

```
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string>

using namespace std;

#define SMTP_PORT 25       // Порт SMTP
#define POP3_PORT 110      // Порт POP3
#define BUF_SIZE 4096      // Размер буфера для передачи данных

int main(){
    // Инициализация Winsock
    WORD wVersionRequested = MAKEWORD(2, 0);
    WSADATA wsaData;
    int error = WSAStartup(wVersionRequested, &wsaData);
    if (error != 0) {
        printf("WSAStartup failed with error: %d\n", WSAGetLastError());
        return 1;
    }

    // Ввод данных пользователя
    std::string servDom;
    std::cout << "Input server domain: ";
    std::cin >> servDom;

    std::string fromAddr;
    std::cout << "Input your address: ";
    std::cin >> fromAddr;

    std::string login;
    std::cout << "Input your login: ";
    std::cin >> login;

    std::string password;
    std::cout << "Input your password: ";
    std::cin >> password;

    std::string toAddr;
    std::cout << "Input destination address: ";
    std::cin >> toAddr;

    std::string topic;
    std::cout << "Input topic: ";
    std::cin.ignore();         // Очищаем буфер
    getline(cin, topic);       // Считываем строку с пробелами

    std::string massage;
    std::cout << "Input message: ";
    getline(cin, massage);

    std::string fileName;
    std::cout << "Input file name: ";
    std::cin >> fileName;

    // Подключение к SMTP-серверу
    int iProtocolPort = 0;
    char szBuffer[BUF_SIZE] = "";
    SOCKET hServer;
    LPHOSTENT lpHostEntry;
```

```
    SOCKADDR_IN SockAddr;

    // Получаем IP-адрес хоста
    lpHostEntry = gethostbyname("localhost");  // Или можно использовать servDom
    if (!lpHostEntry) {
        printf("Gethostbyname error\n");
        return false;
    }

    // Создаем сокет
    hServer = socket(PF_INET, SOCK_STREAM, 0);
    if (hServer == INVALID_SOCKET) {
        printf("Socket error\n");
        return false;
    }

    // Настраиваем структуру адреса сервера
    iProtocolPort = htons(SMTP_PORT);
    SockAddr.sin_family = AF_INET;
    SockAddr.sin_port = iProtocolPort;
    SockAddr.sin_addr = *((LPIN_ADDR)*lpHostEntry->h_addr_list);

    // Подключение к серверу
    if (connect(hServer, (PSOCKADDR)&SockAddr, sizeof(SockAddr))) {
        printf("Connect error\n");
        return false;
    }

    // Получаем приветственное сообщение от сервера
    recv(hServer, szBuffer, sizeof(szBuffer), 0);

    // Команды SMTP
    sprintf(szBuffer, "HELO %s\r\n", servDom.c_str());
    send(hServer, szBuffer, strlen(szBuffer), 0);
    recv(hServer, szBuffer, sizeof(szBuffer), 0);
    printf("%s\n", szBuffer);

    sprintf(szBuffer, "MAIL FROM:<%s>\r\n", fromAddr.c_str());
    send(hServer, szBuffer, strlen(szBuffer), 0);
    recv(hServer, szBuffer, sizeof(szBuffer), 0);
    printf("%s\n", szBuffer);

    sprintf(szBuffer, "RCPT TO:<%s>\r\n", toAddr.c_str());
    send(hServer, szBuffer, strlen(szBuffer), 0);
    recv(hServer, szBuffer, sizeof(szBuffer), 0);
    printf("%s\n", szBuffer);

    sprintf(szBuffer, "DATA\r\n");
    send(hServer, szBuffer, strlen(szBuffer), 0);
    recv(hServer, szBuffer, sizeof(szBuffer), 0);
    printf("%s\n", szBuffer);

    // Формируем тело письма
    sprintf(szBuffer, "From: %s\n", fromAddr.c_str());
    send(hServer, szBuffer, strlen(szBuffer), 0);

    sprintf(szBuffer, "To: %s\n\n", toAddr.c_str());
    send(hServer, szBuffer, strlen(szBuffer), 0);

    sprintf(szBuffer, "Topic: %s\n", topic.c_str());
    send(hServer, szBuffer, strlen(szBuffer), 0);

    sprintf(szBuffer, "Message: %s\n", massage.c_str());
    send(hServer, szBuffer, strlen(szBuffer), 0);

    sprintf(szBuffer, "File name: %s\n", fileName.c_str());
    send(hServer, szBuffer, strlen(szBuffer), 0);
```

```
    sprintf(szBuffer, "File data: ");
    send(hServer, szBuffer, strlen(szBuffer), 0);

    // Читаем и отправляем содержимое файла
    FILE *source = fopen(fileName.c_str(), "r");
    int bytes_read;
    if (!source) {
        printf("Failed to open file.\n");
    } else {
        while ((bytes_read = fread(szBuffer, sizeof(char), BUF_SIZE, source))) {
            send(hServer, szBuffer, bytes_read, 0);
        }
        fclose(source);
    }

    sprintf(szBuffer, "\n\n");
    send(hServer, szBuffer, strlen(szBuffer), 0);

    // Завершаем передачу письма
    sprintf(szBuffer, "\r\n.\r\n");
    send(hServer, szBuffer, strlen(szBuffer), 0);
    recv(hServer, szBuffer, sizeof(szBuffer), 0);
    printf("%s\n", szBuffer);

    // Закрываем соединение
    sprintf(szBuffer, "QUIT\r\n");
    send(hServer, szBuffer, strlen(szBuffer), 0);
    recv(hServer, szBuffer, sizeof(szBuffer), 0);
    printf("%s\n", szBuffer);

    closesocket(hServer);  // закрываем сокет
    cin.ignore();          // очищаем поток ввода
    getchar();             // ждем нажатия клавиши

    return 0;
}
```

Файл POP3.cpp

```
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string>

using namespace std;

#define POP3_PORT 110      // Порт по умолчанию для протокола POP3
#define BUF_SIZE 4096      // Размер буфера для обмена данными
#define RECV_TIMEOUT 5000  // Таймаут на прием данных в миллисекундах (5 секунд)

// Функция для установки таймаута на прием данных через сокет
void set_socket_timeout(SOCKET sock, int timeout_ms) {
    setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (const char*)&timeout_ms, sizeof(timeout_ms));
}

// Функция для получения многострочного ответа от сервера (например, для LIST или RETR)
bool receive_multiline(SOCKET hServer, char* szBuffer, int bufSize) {
    string full_response;
    bool completed = false;
    while (true) {
        memset(szBuffer, 0, bufSize); // Очищаем буфер перед каждым чтением
        int bytes_received = recv(hServer, szBuffer, bufSize - 1, 0); // Чтение данных из сокета

        if (bytes_received == SOCKET_ERROR) {
            // Обработка ошибок при приеме
```

```
            if (WSAGetLastError() == WSAETIMEDOUT) {
                printf("Receive timeout\n");
            } else {
                printf("Receive error: %d\n", WSAGetLastError());
            }
            return false;
        }

        if (bytes_received == 0) {
            // Сервер закрыл соединение
            printf("Connection closed by server\n");
            return false;
        }

        szBuffer[bytes_received] = '\0'; // Завершаем строку
        full_response += szBuffer;       // Добавляем полученные данные к общему ответу

        // Проверка на завершение ответа по стандарту POP3 (строка ".\r\n")
        if (full_response.find("\r\n.\r\n") != string::npos) {
            completed = true;
            break;
        }

        // Защита от бесконечного чтения слишком большого ответа
        if (full_response.length() > 100000) {
            printf("Response too large\n");
            return false;
        }
    }

    // Печатаем весь полученный ответ
    printf("%s\n", full_response.c_str());
    return completed;
}

int main() {
    WSADATA wsaData;
    WORD wVersionRequested = MAKEWORD(2, 0); // Версия Winsock

    // Инициализация библиотеки Winsock
    if (WSAStartup(wVersionRequested, &wsaData) != 0) {
        printf("WSAStartup failed with error: %d\n", WSAGetLastError());
        return 1;
    }

    // Создание TCP-сокета
    SOCKET hServer = socket(AF_INET, SOCK_STREAM, 0);
    if (hServer == INVALID_SOCKET) {
        printf("Socket error: %d\n", WSAGetLastError());
        WSACleanup();
        return 1;
    }

    // Установка таймаута на прием
    set_socket_timeout(hServer, RECV_TIMEOUT);

    // Получение IP-адреса хоста "localhost"
    HOSTENT* lpHostEntry = gethostbyname("localhost");
    if (!lpHostEntry) {
        printf("Gethostbyname error\n");
        closesocket(hServer);
        WSACleanup();
        return 1;
    }

    // Заполнение структуры адреса
    SOCKADDR_IN SockAddr;
```

```
SockAddr.sin_family = AF_INET;
SockAddr.sin_port = htons(POP3_PORT); // Преобразуем порт в сетевой порядок байт
SockAddr.sin_addr = *((LPIN_ADDR)*lpHostEntry->h_addr_list); // IP-адрес

// Установка соединения с сервером
if (connect(hServer, (PSOCKADDR)&SockAddr, sizeof(SockAddr)) != 0) {
    printf("Connect error: %d\n", WSAGetLastError());
    closesocket(hServer);
    WSACleanup();
    return 1;
}

// Получение приветственного сообщения от POP3-сервера
char szBuffer[BUF_SIZE] = {0};
recv(hServer, szBuffer, sizeof(szBuffer) - 1, 0);
printf("%s\n", szBuffer);

string input;
while (true) {
    // Предлагаем пользователю выбрать команду
    printf("Select the command (Auth, List, Read, Delete, Quit): ");
    cin >> input;

    if (input == "Auth") {
        // Аутентификация: USER и PASS
        printf("Input address: ");
        cin >> input;
        sprintf(szBuffer, "USER %s\r\n", input.c_str());
        send(hServer, szBuffer, strlen(szBuffer), 0);
        memset(szBuffer, 0, sizeof(szBuffer));
        recv(hServer, szBuffer, sizeof(szBuffer) - 1, 0);
        printf("%s\n", szBuffer);

        printf("Input password: ");
        cin >> input;
        sprintf(szBuffer, "PASS %s\r\n", input.c_str());
        send(hServer, szBuffer, strlen(szBuffer), 0);
        memset(szBuffer, 0, sizeof(szBuffer));
        recv(hServer, szBuffer, sizeof(szBuffer) - 1, 0);
        printf("%s\n", szBuffer);
    }
    else if (input == "List") {
        // Получение списка сообщений
        sprintf(szBuffer, "LIST\r\n");
        send(hServer, szBuffer, strlen(szBuffer), 0);
        if (!receive_multiline(hServer, szBuffer, sizeof(szBuffer))) {
            printf("Failed to receive LIST response\n");
        }
    }
    else if (input == "Read") {
        // Чтение конкретного сообщения
        printf("Select message index: ");
        cin >> input;
        sprintf(szBuffer, "RETR %s\r\n", input.c_str());
        send(hServer, szBuffer, strlen(szBuffer), 0);
        if (!receive_multiline(hServer, szBuffer, sizeof(szBuffer))) {
            printf("Failed to receive RETR response\n");
        }
    }
    else if (input == "Delete") {
        // Удаление сообщения
        printf("Select message index: ");
        cin >> input;
        sprintf(szBuffer, "DELE %s\r\n", input.c_str());
        send(hServer, szBuffer, strlen(szBuffer), 0);
        memset(szBuffer, 0, sizeof(szBuffer));
        recv(hServer, szBuffer, sizeof(szBuffer) - 1, 0);
```

```
            printf("%s\n", szBuffer);
        }
        else if (input == "Quit") {
            // Завершение сессии
            sprintf(szBuffer, "QUIT\r\n");
            send(hServer, szBuffer, strlen(szBuffer), 0);
            memset(szBuffer, 0, sizeof(szBuffer));
            recv(hServer, szBuffer, sizeof(szBuffer) - 1, 0);
            printf("%s\n", szBuffer);
            break;
        }
        else {
            // Обработка некорректного ввода
            printf("Input error\n");
        }
    }

    // Закрытие сокета и очистка ресурсов Winsock
    closesocket(hServer);
    WSACleanup();

    // Ожидание перед завершением
    cin.ignore();
    getchar();
    return 0;
}
```