ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КУРСОВОЙ ПРОЕКТ

по дисциплине “Сетевое программирование”

на тему

Разработка сетевого приложения. SMTP клиент.

Выполнил студент Павленко Павел Владимирович
Ф.И.О.

Группы ИВ-221

Работу принял _____ профессор д.т.н. К.В. Павский
подпись

Защищена _____ Оценка _____

Новосибирск – 2025

СОДЕРЖАНИЕ

Постановка задачи

Описание протокола

SMTP (Simple Mail Transfer Protocol) — это стандартный протокол прикладного уровня, предназначенный для передачи электронной почты в сетях TCP/IP. Разработанный в 1982 году (RFC 821), он до сих пор является основным механизмом доставки сообщений между почтовыми серверами. Протокол определяет формат команд и порядок взаимодействия между отправителем (MUA — Mail User Agent) и сервером (MTA — Mail Transfer Agent).

Принцип работы SMTP

Процесс отправки электронного письма с использованием SMTP включает следующие этапы:

1. **Формирование сообщения**

   Пользователь создаёт письмо, указывая адресата, тему и текст. Почтовый клиент кодирует данные в соответствии со стандартом **MIME (Multipurpose Internet Mail Extensions)**, что позволяет передавать не только текст, но и вложения, а также поддерживает различные кодировки символов.

2. **Установка TCP-соединения**

   Клиент инициирует соединение с SMTP-сервером через порт **25** (стандартный), **587** (с шифрованием STARTTLS) или **465** (SMTPS, SSL/TLS). Соединение устанавливается по протоколу TCP, обеспечивая надежную передачу данных.

3. **Начало сеанса (Handshake)**

   После установки соединения клиент отправляет команду **EHLO** (или **HELO** в устаревших реализациях), на что сервер отвечает списком поддерживаемых расширений (например, **STARTTLS**, **AUTH**, **SIZE**).

4. **Аутентификация (если требуется)**

   Если сервер требует авторизацию, клиент использует один из механизмов:

   - **LOGIN / PLAIN** (передача логина и пароля в Base64)
   - **CRAM-MD5** (более безопасный метод с хешированием)

5. **Указание отправителя и получателя**
    - Команда **MAIL FROM:** определяет обратный адрес.
    - Команда **RCPT TO:** задаёт получателя. Сервер может проверить валидность адреса.
6. **Передача данных**

    Команда **DATA** запускает отправку содержимого письма. Текст передаётся в формате **MIME**, а завершается последовательностью <CRLF>.<CRLF>.
7. **Доставка письма**

    Если получатель находится на другом сервере, MTA отправителя устанавливает соединение с MTA получателя и повторяет процесс. В случае ошибки письмо помещается в очередь для повторной отправки
8. **Завершение сеанса**

    Команда **QUIT** корректно закрывает соединение.

Преимущества SMTP

- **Стандартизация** — совместимость между различными почтовыми системами.
- **Поддержка MIME** — возможность передачи мультимедийных вложений.
- **Гибкость** — поддержка ретрансляции и маршрутизации писем через несколько серверов.
- **Детализация ошибок** — ответы сервера содержат числовые коды (например, **250 OK**, **550 Mailbox not found**).

Недостатки SMTP

- **Отсутствие шифрования по умолчанию** — базовый SMTP передаёт данные в открытом виде, что требует использования **TLS/SSL**.
- **Уязвимость к спаму и спуфингу** — необходимы дополнительные механизмы (**SPF**, **DKIM**, **DMARC**) для проверки подлинности отправителя.
- **Ограниченная поддержка больших вложений** — в чистом виде SMTP не оптимизирован для передачи объёмных файлов.

Описание реализации

Функция main() является ядром SMTP-клиента и реализует полный цикл отправки электронного письма через защищенное SSL-соединение. Рассмотрим ее работу пошагово:

1. Ввод пользовательских данных

```
std::string sender_email, recipient_email, subject, body, file, app_password;
std::vector<std::string> attachment_filenames;
std::cout << "Введите ваш email адрес...";
std::getline(std::cin, sender_email);
if(sender_email.empty()) sender_email = "pavlenkopasa382@gmail.com";
```

Особенности:

- Запрашивает все необходимые данные для отправки письма
- Предоставляет значения по умолчанию для обязательных полей
- Поддерживает множественные вложения через цикл while
- Использует std::cin.ignore() для корректной обработки многострочного ввода

2. Инициализация OpenSSL

```
SSL_library_init();
OpenSSL_add_all_algorithms();
SSL_load_error_strings();
const SSL_METHOD *method = TLS_client_method();
SSL_CTX *ctx = SSL_CTX_new(method);
```

Ключевые моменты:

1. SSL_library_init() - инициализирует библиотеку OpenSSL
2. OpenSSL_add_all_algorithms() - загружает все доступные алгоритмы шифрования
3. SSL_load_error_strings() - активирует текстовые описания ошибок
4. Создается контекст SSL с использованием TLS-метода (наиболее

безопасный вариант)

3. Создание и настройка сокета

```
int socket_desc = socket(AF_INET, SOCK_STREAM, 0);
struct hostent *server = gethostbyname(SMTP_SERVER.c_str());
struct sockaddr_in server_addr;
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(SMTP_PORT);
memcpy(&server_addr.sin_addr.s_addr, server->h_addr, server->h_length);
```

Архитектурные решения:

- Используется IPv4 (AF_INET) и потоковый протокол (SOCK_STREAM)
- DNS-запрос преобразует доменное имя в IP-адрес
- Структура sockaddr_in заполняется вручную для точного контроля параметров

4. Установка соединения

```
connect(socket_desc, (struct sockaddr *)&server_addr, sizeof(server_addr));
SSL *ssl = SSL_new(ctx);
SSL_set_fd(ssl, socket_desc);
SSL_connect(ssl);
```

Безопасность:

1. Сначала устанавливается TCP-соединение
2. Затем создается SSL-объект и привязывается к сокету
3. Выполняется SSL handshake для установки шифрованного канала

5. SMTP-сессия (основной рабочий цикл)

```
try {
    std::string response = read_response(ssl);
    std::cout << "Сервер: " << response << std::endl;
    check_response_code(response, 220);
    send_command(ssl, "EHLO example.com\r\n");
    // ... другие команды
```

Протокольные особенности:

- Строго соблюдается последовательность команд SMTP
- Каждый ответ сервера проверяется на ожидаемый код
- Используется механизм исключений для обработки ошибок

6. Формирование MIME-сообщения

```
std::string boundary = "----=_NextPart_001_" + std::to_string(time(0));
message += "Content-Type: multipart/mixed; boundary=\"" + boundary + "\"\r\n";
// Текст письма
message += "--" + boundary + "\r\n";
message += "Content-Type: text/plain; charset=UTF-8\r\n\r\n";
message += body + "\r\n";
// Вложения
for(const auto &filename : attachment_filenames) {
    message += "--" + boundary + "\r\n";
    message += "Content-Type: application/octet-stream\r\n";
    message += "Content-Transfer-Encoding: base64\r\n";
    message += "Content-Disposition: attachment; filename=\"" + filename + "\"\r\n\r\n";
    message += base64_encode_file(filename) + "\r\n";
}
message += "--" + boundary + "--\r\n";
```

Спецификация MIME:

1. Генерируется уникальная boundary на основе времени
2. Для текстовой части указывается кодировка UTF-8
3. Каждое вложение:
    - Имеет свой MIME-блок
    - Кодируется в Base64
    - Содержит оригинальное имя файла

7. Отправка данных письма

```
send_command(ssl, message + "\r\n.\r\n");
check_response_code(read_response(ssl), 250);
```

Важные детали:

- Точка в конце сообщения (\r\n.\r\n) - сигнал окончания данных
- Ожидается код 250 (Requested mail action okay)
- Все передается через SSL-шифрование

8. Завершение сеанса

```
send_command(ssl, "QUIT\r\n");
check_response_code(read_response(ssl), 221);
SSL_shutdown(ssl);
close(socket_desc);
SSL_CTX_free(ctx);
```

Корректное завершение:

1. Отправка команды QUIT (код 221 - закрытие соединения)
2. Постепенное освобождение ресурсов:
    - Сначала SSL, затем сокет
    - Очистка контекста SSL
3. Глобальная очистка OpenSSL

Теперь перейдем к основным внешним моментам, не в функции main(), и главным функциям.

Класс SMTPException

```
class SMTPException : public std::runtime_error {
public:
    explicit SMTPException(const std::string& message) : std::runtime_error(message) {}
};
```

Назначение:

Кастомный класс исключений для обработки ошибок SMTP-протокола.

Особенности:

- Наследуется от стандартного runtime_error
- Позволяет передавать текстовые сообщения об ошибках
- Используется во всех функциях для единообразной обработки ошибок

2. Функция read_response()

```
std::string read_response(SSL *ssl) {
    char buffer[BUFFER_SIZE] = {0};
    int bytes_received = SSL_read(ssl, buffer, BUFFER_SIZE - 1);
    if (bytes_received < 0) {
        throw SMTPException("Ошибка при чтении ответа от сервера...");
    }
    return std::string(buffer, bytes_received);
}
```

Назначение: Чтение ответа от SMTP-сервера через SSL-соединение.

Параметры:

- ssl: Указатель на SSL-соединение

Алгоритм работы:

1. Создает буфер фиксированного размера (4096 байт)
2. Читает данные с помощью SSL_read()
3. Проверяет код ошибки (при отрицательном значении генерирует исключение)
4. Возвращает ответ как строку

3. Функция send_command()

```
void send_command(SSL *ssl, const std::string& command) {
    std::cout << "Клиент: " << command;
    if (SSL_write(ssl, command.c_str(), command.length()) <= 0) {
        throw SMTPException("Ошибка при отправке команды...");
    }
}
```

Назначение: Отправка SMTP-команд на сервер.

Особенности:

- Автоматически добавляет вывод в консоль для отладки
- Использует SSL_write() для безопасной передачи
- Генерирует исключение при ошибках записи

4. Функция check_response_code()

```
void check_response_code(const std::string& response, int expected_code) {
    int code;
    std::stringstream ss(response);
    ss >> code;

    if (code != expected_code) {
        throw SMTPException("Неожиданный код ответа...");
    }
}
```

Назначение: Валидация кодов ответа SMTP-сервера.

Логика работы:

1. Извлекает числовой код из начала ответа (первые 3 цифры)
2. Сравнивает с ожидаемым кодом
3. При несоответствии генерирует исключение с полным текстом ответа

5. Функция base64_encode_file()

```
std::string base64_encode_file(const std::string& filename) {
    std::ifstream file(filename, std::ios::binary);
    if (!file.is_open()) {
        throw SMTPException("Не удалось открыть файл: " + filename);
    }

    std::string file_content((std::istreambuf_iterator<char>(file)),
                             std::istreambuf_iterator<char>());

    const std::string base64_chars =
            "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
            "abcdefghijklmnopqrstuvwxyz"
            "0123456789+/";

    std::string encoded_string;
    int i = 0;
    unsigned char char_array_3[3];
    unsigned char char_array_4[4];

    for (size_t pos = 0; pos < file_content.length(); pos++) {
        char_array_3[i++] = file_content[pos];
        if (i == 3) {
            char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
            char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] & 0xf0) >> 4);
            char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] & 0xc0) >> 6);
            char_array_4[3] = char_array_3[2] & 0x3f;

            for (i = 0; (i < 4); i++)
                encoded_string += base64_chars[char_array_4[i]];
            i = 0;
```

```
        }
    }

    if (i)
    {
        for (int j = i; j < 3; j++)
            char_array_3[j] = '\0';

        char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
        char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] & 0xf0) >> 4);
        char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] & 0xc0) >> 6);
        char_array_4[3] = char_array_3[2] & 0x3f;

        for (int j = 0; (j < i + 1); j++)
            encoded_string += base64_chars[char_array_4[j]];

        while((i++ < 3))
            encoded_string += '=';

    }

    return encoded_string;
}
```

Назначение: Кодирование файла в Base64 для вложений.

Особенности:

- Открывает файл в бинарном режиме
- Читает все содержимое в строку
- Использует ручную реализацию Base64 (без внешних зависимостей)
- Обрабатывает последний неполный блок (добавляет '=')

6. Функция base64_encode()

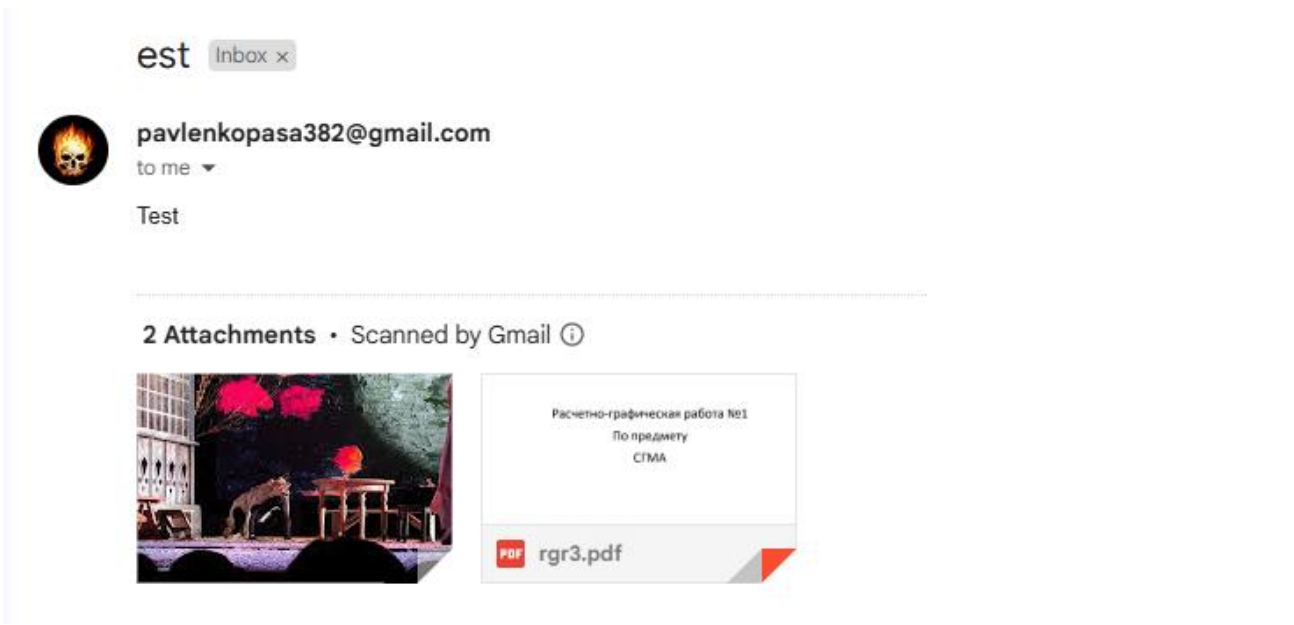
Отличие от base64_encode_file():

- Работает непосредственно со строковыми данными
- Используется для кодирования логина/пароля при аутентификации
- Идентичный алгоритм кодирования

Скан экрана работы программы

```
azard285@DESKTOP-N3ARCU8:/mnt/c/Users/Azard/3curs/seti/kurs2$ ./smtp_client
Введите ваш email адрес (Gmail) (по умолчанию pavlenkopasa382@gmail.com):
Введите пароль приложения (Gmail) (не нужно если не вводили свою почту):
Введите email адрес получателя (по умолчанию iv221s21@gmail.com):
Введите тему письма: Test
Введите текст письма: Test
Введите имя файла для вложения (или оставьте пустым, чтобы не вкладывать файл): Photo.jpg
Введите следующий файл или пропустите: rgr3.pdf
```

```
Клиент: EHLO example.com
Сервер: 250-smtp.gmail.com at your service, [2.63.201.55]
250-SIZE 35882577
250-8BITMIME
250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-CHUNKING
250 SMTPUTF8

Клиент: AUTH LOGIN
Сервер: 334 VXNlcm5hbWU6

Клиент: cGF2bGVua29wYXNhMzgyQGdtYWlsLmNvbQ==
Сервер: 334 UGFzc3dvcmQ6

Клиент: andidmh1ZXpqdnZzYXdxaA==
Сервер: 235 2.7.0 Accepted

Клиент: MAIL FROM:<pavlenkopasa382@gmail.com>
Сервер: 250 2.1.0 OK 38308e7fff4ca-317cfb484e3sm18534021fa.40 - gsmtp

Клиент: RCPT TO:<iv221s21@gmail.com>
Сервер: 250 2.1.5 OK 38308e7fff4ca-317cfb484e3sm18534021fa.40 - gsmtp

Клиент: DATA
Сервер: 354 Go ahead 38308e7fff4ca-317cfb484e3sm18534021fa.40 - gsmtp

Клиент: Date: Mon, 15 Nov 2023 14:30:00 +0000
From: <pavlenkopasa382@gmail.com>
To: <iv221s21@gmail.com>
Subject: est
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="-----=_NextPart_001_1745783925"

------=_NextPart_001_1745783925
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit

Test

Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit

Test
```


est
Inbox ×
pavlenkopasa382@gmail.com
to me
Test
2 Attachments • Scanned by Gmail
Расчетно-графическая работа №1
По предмету
СПМА
rgr3.pdf

```
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <stdexcept>
#include <vector>
#include <iomanip>

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>
#include <cstring>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <openssl/bio.h>


const int SMTP_PORT = 465;
const int BUFFER_SIZE = 4096;
const std::string SMTP_SERVER = "smtp.gmail.com";


class SMTPException : public std::runtime_error {
public:
    explicit SMTPException(const std::string& message) : std::runtime_error(message)
{}
};


std::string read_response(SSL *ssl) {
    char buffer[BUFFER_SIZE] = {0};
    int bytes_received = SSL_read(ssl, buffer, BUFFER_SIZE - 1);
    if (bytes_received < 0) {
        throw SMTPException("Ошибка при чтении ответа от сервера: " +
std::to_string(SSL_get_error(ssl, bytes_received)));
    }
    return std::string(buffer, bytes_received);
}


void send_command(SSL *ssl, const std::string& command) {
    std::cout << "Клиент: " << command;
    if (SSL_write(ssl, command.c_str(), command.length()) <= 0) {
        throw SMTPException("Ошибка при отправке команды на сервер: " +
std::to_string(SSL_get_error(ssl, 0)));
    }
}



void check_response_code(const std::string& response, int expected_code) {
    int code;
    std::stringstream ss(response);
    ss >> code;

    if (code != expected_code) {
        std::cerr << "Ошибка: " << response << std::endl;
        throw SMTPException("Неожиданный код ответа от сервера: " +
std::to_string(code) + " Response: " + response);
    }
}
```

```
std::string base64_encode_file(const std::string& filename) {
    std::ifstream file(filename, std::ios::binary);
    if (!file.is_open()) {
        throw SMTPException("Не удалось открыть файл: " + filename);
    }

    std::string file_content((std::istreambuf_iterator<char>(file)),
                             std::istreambuf_iterator<char>());

    const std::string base64_chars =
             "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
             "abcdefghijklmnopqrstuvwxyz"
             "0123456789+/";

    std::string encoded_string;
    int i = 0;
    unsigned char char_array_3[3];
    unsigned char char_array_4[4];

    for (size_t pos = 0; pos < file_content.length(); pos++) {
        char_array_3[i++] = file_content[pos];
        if (i == 3) {
            char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
            char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] &
0xf0) >> 4);
            char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] &
0xc0) >> 6);
            char_array_4[3] = char_array_3[2] & 0x3f;

            for (i = 0; (i < 4); i++)
                encoded_string += base64_chars[char_array_4[i]];
            i = 0;
        }
    }

    if (i)
    {
        for (int j = i; j < 3; j++)
            char_array_3[j] = '\0';

        char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
        char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] &
0xf0) >> 4);
        char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] &
0xc0) >> 6);
        char_array_4[3] = char_array_3[2] & 0x3f;

        for (int j = 0; (j < i + 1); j++)
            encoded_string += base64_chars[char_array_4[j]];

        while((i++ < 3))
            encoded_string += '=';

    }

    return encoded_string;
}

std::string base64_encode(const std::string& input) {
    const std::string base64_chars =
             "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
             "abcdefghijklmnopqrstuvwxyz"
             "0123456789+/";

    std::string encoded_string;
    int i = 0;
    unsigned char char_array_3[3];
```

```
    unsigned char char_array_4[4];

    for (size_t pos = 0; pos < input.length(); pos++) {
        char_array_3[i++] = input[pos];
        if (i == 3) {
            char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
            char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] &
0xf0) >> 4);
            char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] &
0xc0) >> 6);
            char_array_4[3] = char_array_3[2] & 0x3f;

            for (i = 0; (i < 4); i++)
                encoded_string += base64_chars[char_array_4[i]];
            i = 0;
        }
    }

    if (i)
    {
        for (int j = i; j < 3; j++)
            char_array_3[j] = '\0';

        char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
        char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] &
0xf0) >> 4);
        char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] &
0xc0) >> 6);
        char_array_4[3] = char_array_3[2] & 0x3f;

        for (int j = 0; (j < i + 1); j++)
            encoded_string += base64_chars[char_array_4[j]];

        while((i++ < 3))
            encoded_string += '=';

    }

    return encoded_string;
}


int main() {
    std::string sender_email, recipient_email, subject, body, file, app_password;
    std::vector<std::string> attachment_filenames;


    std::cout << "Введите ваш email адрес (Gmail) (по умолчанию
pavlenkopasa382@gmail.com): ";
    std::getline(std::cin, sender_email);

    if(sender_email.empty())
    {
        sender_email = "pavlenkopasa382@gmail.com";
    }

    std::cout << "Введите пароль приложения (Gmail) (не нужно если не вводили свою
почту): ";
    std::getline(std::cin, app_password);
    if(app_password.empty())
    {
        app_password = "jwbvhuezjvvsawqh";
    }

    std::cout << "Введите email адрес получателя (по умолчанию iv221s21@gmail.com):
";
    std::getline(std::cin, recipient_email);
    if(recipient_email.empty())
```

```
    {
        recipient_email = "iv221s21@gmail.com";
    }

    std::cout << "Введите тему письма: ";
    std::cin.ignore();
    std::getline(std::cin, subject);

    std::cout << "Введите текст письма: ";
    std::getline(std::cin, body);

    std::cout << "Введите имя файла для вложения (или оставьте пустым, чтобы не
вкладывать файл): ";
    std::getline(std::cin, file);

    while(!file.empty())
    {
        attachment_filenames.push_back(file);
        std::cout << "Введите следующий файл или пропустите: ";
        std::getline(std::cin, file);
    }



    SSL_library_init();
    OpenSSL_add_all_algorithms();
    SSL_load_error_strings();

    const SSL_METHOD *method = TLS_client_method();
    SSL_CTX *ctx = SSL_CTX_new(method);
    if (!ctx) {
        ERR_print_errors_fp(stderr);
        throw SMTPException("Не удалось создать SSL context");
    }


    int socket_desc = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_desc == -1) {
        std::cerr << "Не удалось создать сокет." << std::endl;
        return 1;
    }

    struct hostent *server = gethostbyname(SMTP_SERVER.c_str());
    if (server == nullptr) {
        std::cerr << "Не удалось разрешить имя хоста." << std::endl;
        close(socket_desc);
        SSL_CTX_free(ctx);
        return 1;
    }

    struct sockaddr_in server_addr;
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SMTP_PORT);
    memcpy(&server_addr.sin_addr.s_addr, server->h_addr, server->h_length);


    if (connect(socket_desc, (struct sockaddr *)&server_addr, sizeof(server_addr)) <
0) {
        std::cerr << "Не удалось подключиться к серверу." << std::endl;
        close(socket_desc);
        SSL_CTX_free(ctx);
        return 1;
    }

    SSL *ssl = SSL_new(ctx);
    if (!ssl) {
        ERR_print_errors_fp(stderr);
```

```
        close(socket_desc);
        SSL_CTX_free(ctx);
        return 1;
    }

    SSL_set_fd(ssl, socket_desc);

    if (SSL_connect(ssl) <= 0) {
        ERR_print_errors_fp(stderr);
        close(socket_desc);
        SSL_free(ssl);
        SSL_CTX_free(ctx);
        return 1;
    }

    try {

        std::string response = read_response(ssl);
        std::cout << "Сервер: " << response << std::endl;
        check_response_code(response, 220);


        send_command(ssl, "EHLO example.com\r\n");
        response = read_response(ssl);
        std::cout << "Сервер: " << response << std::endl;
        check_response_code(response, 250);


        send_command(ssl, "AUTH LOGIN\r\n");
        response = read_response(ssl);
        std::cout << "Сервер: " << response << std::endl;
        check_response_code(response, 334);


        std::string encoded_username = base64_encode(sender_email);
        send_command(ssl, encoded_username + "\r\n");
        response = read_response(ssl);
        std::cout << "Сервер: " << response << std::endl;
        check_response_code(response, 334);


        std::string encoded_password = base64_encode(app_password);
        send_command(ssl, encoded_password + "\r\n");
        response = read_response(ssl);
        std::cout << "Сервер: " << response << std::endl;
        check_response_code(response, 235);


        send_command(ssl, "MAIL FROM:<" + sender_email + ">\r\n");
        response = read_response(ssl);
        std::cout << "Сервер: " << response << std::endl;
        check_response_code(response, 250);


        send_command(ssl, "RCPT TO:<" + recipient_email + ">\r\n");
        response = read_response(ssl);
        std::cout << "Сервер: " << response << std::endl;
        check_response_code(response, 250);


        send_command(ssl, "DATA\r\n");
        response = read_response(ssl);
        std::cout << "Сервер: " << response << std::endl;
        check_response_code(response, 354);


        std::string message;
        message += "Date: Mon, 15 Nov 2023 14:30:00 +0000\r\n";
```

```
        message += "From: <" + sender_email + ">\r\n";
        message += "To: <" + recipient_email + ">\r\n";
        message += "Subject: " + subject + "\r\n";
        message += "MIME-Version: 1.0\r\n";
                std::string boundary = "----=_NextPart_001_" +
std::to_string(time(0));
                message += "Content-Type: multipart/mixed; boundary=\"" + boundary +
"\"\r\n\r\n";
                message += "--" + boundary + "\r\n";
                message += "Content-Type: text/plain; charset=UTF-8\r\n";
                message += "Content-Transfer-Encoding: 8bit\r\n\r\n";
                message += body + "\r\n\r\n";
        if (!attachment_filenames.empty()) {
            for(const auto &attachment_filename : attachment_filenames){
                std::string encoded_file = base64_encode_file(attachment_filename);
                message += "--" + boundary + "\r\n";
                message += "Content-Type: application/octet-stream; name=\"" + at-
tachment_filename + "\"\r\n";
                message += "Content-Transfer-Encoding: base64\r\n";
                message += "Content-Disposition: attachment; filename=\"" + attach-
ment_filename + "\"\r\n\r\n";
                message += encoded_file + "\r\n\r\n";
            }

                message += "--" + boundary + "--\r\n";
        } else {

            message += "Content-Type: text/plain; charset=UTF-8\r\n";
            message += "Content-Transfer-Encoding: 8bit\r\n\r\n";
            message += body + "\r\n";
        }


        send_command(ssl, message + "\r\n.\r\n");
        response = read_response(ssl);
        std::cout << "Сервер: " << response << std::endl;
        check_response_code(response, 250);


        send_command(ssl, "QUIT\r\n");
        response = read_response(ssl);
        std::cout << "Сервер: " << response << std::endl;
        check_response_code(response, 221);

    } catch (const SMTPException& e) {
        std::cerr << "Ошибка SMTP: " << e.what() << std::endl;
        ERR_print_errors_fp(stderr);
    } catch (const std::exception& e) {
        std::cerr << "Общая ошибка: " << e.what() << std::endl;
        ERR_print_errors_fp(stderr);
    }


    SSL_shutdown(ssl);
    close(socket_desc);
    SSL_free(ssl);
    SSL_CTX_free(ctx);
    EVP_cleanup();
    CRYPTO_cleanup_all_ex_data();
    ERR_free_strings();

    return 0;
}
```

Список источников

1. Что такое SMTP-протокол и как он устроен? / [Электронный ресурс]. – Режим доступа: URL: https://selectel.ru/blog/smtp-protocol/ . Дата обращения: 06.04.2025г
2. Что такое SMTP / [Электронный ресурс]. – Режим доступа: URL: https://mailganer.com/ru/glossary/smtp. Дата обращения: 03.04.2025г
3. Как настроить устройство или приложение для отправки почты через Google Workspace / [Электронный ресурс]. – Режим доступа: URL: https://support.google.com/a/answer/176600?hl=ru. Дата обращения: 08.04.2025г
4. Простой протокол передачи почты (SMTP) / [Электронный ресурс]. – Режим доступа: URL: https://www.ibm.com/docs/ru/i/7.3?topic=information-smtp. Дата обращения: 04.04.2025г