Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ РАБОТУ

НА ТЕМУ:

«Клиент-серверное приложение для отправки email-сообщений»

Студент ИУ7-76Б _____ Ф.Х.М. Ле
(Группа) (Подпись, дата) (И.О.Фамилия)

Студент ИУ7-76Б _____ Н. Хеттиараччи
(Группа) (Подпись, дата) (И.О.Фамилия)

Студент ИУ7-76Б _____ Г.С.Д.С. Хеламбаге
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ Н.О. Рогозин
(Подпись, дата) (И.О.Фамилия)

Москва, 2020 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ7
(Индекс)
_____ И.В. Рудаков
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового работа

по дисциплине Компьютерные сети

Студент группы ИУ7-76Б

Ле Фам Ха Ми
Хеламбаге Гавинду Саджиндра Де Сильва
Хеттиараччи Налиша
(Фамилия, имя, отчество)

Тема курсового проекта
Клиент-серверное приложение для отправки email-сообщений

Направленность КП (учебный, исследовательский, практический, производственный, др.)
учебный
Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения проекта: 25% к _._ нед., 50% к _._ нед., 75% к _._ нед., 100% к _._ нед.

Задание Данная работа посвящена разработке клиент-серверного приложения для отправки email сообщений на localhost по основе протоколу SMTP.

Оформление курсового проекта:

Расчетно-пояснительная записка на 20-30 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
На защиту проекта должна быть предоставлена презентация, состоящая из 15-20 слайдов.
На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс,

Дата выдачи задания « » _____ 2020г.

Руководитель курсового проекта _____ Н.О. Рогозин
(Подпись, дата) (И.О.Фамилия)

Студент _____ Ф.Х.М. Ле
(Подпись, дата) (И.О.Фамилия)

Студент _____ Н. Хеттиараччи
(Подпись, дата) (И.О.Фамилия)

Студент _____ Г.С.Д.С. Хеламбаге
(Подпись, дата) (И.О.Фамилия)

СОДЕРЖАНИЕ

РЕФЕРАТ 4
ВВЕДЕНИЕ 5
1. Аналитический раздел 6
1.1. Постановка задачи 6
1.2. Модель клиент-сервер 6
1.3 Сокеты 7
1.3.1 Основные принципы сокетов 7
1.3.2 Типы сокетов 7
1.4 Определение почтового сервера 10
1.4.1 Simple Mail Transfer Protocol 10
1.4.2 Почтовый сервер и DNS 11
1.5 Определение почтового клиента 12
1.5.1 Адресация в SMTP - системах 12
1.6 Заключение 14
2. Конструкторский раздел 15
2.1 Схема взаимодействия клиент - сервера 15
2.2. Клиентская часть 17
2.3 Серверная часть 18
3. Технологический раздел 23
3.1. Выбор среды разработки и языка программирования 23
3.2 Описание использованных модулей и библиотек 23
ЗАКЛЮЧЕНИЕ 25

РЕФЕРАТ

Данная работа посвящена разработке клиент-серверного приложения для отправки email-сообщений на localhost по основе протоколу SMTP.

ВВЕДЕНИЕ

Возможность передачи информации на расстоянии интересовала людей всегда. Для того чтобы обмениваться информацией не только при личной встрече, но и на больших расстояниях, люди изобретали всё новые технические средства, протягивали структурированные кабельные сети по всему миру, организовывали почтовые системы, запускали спутники связи. С развитием информационных технологий стали возможным еще более глобальные коммуникации. Почтому почта — одно из наиболее четко организованных учреждений в мире.

Работая в сети Internet мы очень часто встречаемся с разного рода многопользовательскими программами. Ими могут быть почтовые клиенты, чаты, форумы, FTP клиенты и т.п. Все эти приложения используют для своей работы разного рода протоколы: FTP, POP, SMTP, HTTP, и т.д.

Электронная почта является новым современным средством передачи информации. В отличие от обычной почты. по электронной передаются электронные копии сообщений, файлы, программы, различные данные — т.е. информация, обработанная с помощью компьютера.

На данный момент программы для сетевого общения на основе протокола SMTP представлены в широком ассортименте. разнообразны по функциональным возможностям и целям, для которых применяются. При этом на сегодняшний день на рынке программного Обеспечения не представлено качественных и удобных консольных программ для сетсвого общения, которые бы удовлетворяли требованиям большинства пользователей.

Целью данного курсового проекта является разработка клиент-серверного приложения на основе протокола SMTP.

Для достижения данной цели ставятся следующие задачи:

- Изучить предметную область и проанализировать существуйющие решения.
- Анализируйте протоколы SMTP
- Разработать программную структуру для серверной части
- Разработать программную структуру для клиентской части
- Внедрить программный комплекс

1. Аналитический раздел

В данном разделе производится постановка задачи, анализируется предметная область и на основе результатов исследования представляется вывод.

1.1. Постановка задачи

В соответствии с техническим заданием курсового проекта необходимо разработать клиент-серверное приложения для отправки email-сообшений. Для решения поставленной задачи необходимо изучить предметную область и проанализировать существующие решения.

1.2. Модель клиент-сервер

Сценарий модели клиент-сервер выглядит: сервер предлагает услуги, а клиент ими пользуется Программа, использующая сокеты, может выполнять либо роль сервера, либо роль клиента. Для того чтобы клиент мог взаимодействовать с сервером, ему нужно знать IP-адрес сервера и номер порта, через который клиент должен сообщить о себе.

Когда клиент начинает соединение с сервером, его система назначает данному соединению отдельный сокет, а когда сервер принимает соединение, сокет назначается со стороны сервера. После этого устанавливается связь между двумя зтими сокетами, по которой высылаются данные запроса к серверу. А сервер высылает клиенту. По тому жс соединению, готовые результаты согласно его запросу. Сервер не ограничен связыо только с одним клиентом. На самом деле он может обслуживать многих клиснтов. Каждому клиентскому сокету соответствует уникальный номер порта. Некоторые номера зарезервированы для так называемых стандартных служб. Таблица 1.1 содержит в себе данные о некоторых зарезервированных номерах портов.

Порт	Сервис	Описание
11	systat	показ зарегистрированных В систсме пользователей
21	FTP	доступ к файлам по сети
22	SSH	зашифрованное соединение к удаленному компьютеру
23	Telnet	удаленное соединение с компьютером
25	SMTP	отсылка электронной почты
80	HTTP	веб-сервер (иногда используются порты 8000 или 8080)
110	POP3	получение электронной почты
139	Netbios-SSN	разделение сетевых ресурсов
143	IMAP	пересылка электронной почты
194	IRC	Internet Relay Chat
442	HTTPS	зашифрованный HTTP

Таблица 1.1 Зарезервированные номера портов и их описание.

1.3 Сокеты

Сокеты (англ. socket — разъём) — название программного интерфейса для обеспечения обмена данными между процессами. процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет — абстрактный объект, представляющий конечную точку соединёния.

1.3.1 Основные принципы сокетов

Каждый процесс может создать слушающий сокет (серверный сокет) и привязать его к какому-нибудь порту операционной системы (в UNIX непривилегированные процессы не могут использовать порты меньше 1024).

Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется возможность проверить наличие соединений на данный момент, установить таймаут для операции и т.д.

Каждый сокет имеет свой адрес. ОС семейства UNIX могут поддерживать много типов адресов, но обязательными являются INET-адрес и UNIX-адрес. Если привязать сокет к UNIX-адресу, то будет создан специальный файл (файл сокета) по заданному пути. через который смогут сообщаться любые локальные процессы путём чтения/записи из него. Сокеты типа INET доступны из сети и требуют выделения номера порта.

Обычно клиент явно подсоединяется к слушателю, после чего любое чтение или запись через его файловый дескриптор будут передавать данные между ним и сервером.

1.3.2 Типы сокетов

Сокеты Беркли — интерфейс программирования приложений (API), представляющий собой библиотеку для разработки приложений на языке Си с поддержкой межпроцессного взаимодействия (IPC), часто применяемый в компьютерных сетях.

Сокеты Беркли (также известные как API сокетов BSD), впервые появились как API в операционной системе 4.2BSD Unix (выпущенной в 1983 году). Тем не менее, только в 1989 году Калифорнийский университет в Беркли смог начать выпускать версии операционной системы и сетевой библиотеки без лицензионных ограничений AT&T, действующих в защищённой авторским правом Unix.

API сокетов Беркли сформировал defacto стандарт абстракции для сетевых сокстов. Большинство прочих языков программирования используют интерфейс, схожий с API языка Си.

API Интерфейса транспортного уровня (TLI), основанный на STREAMS, представляет собой альтернативу сокетному API. Тем не менее, API сокетов Беркли Значительно преобладает в популярности и количестве реализаций.

Интерфейс сокета Беркли — API, позволяющий реализовывать взаимодействие между компьютерами или между процессами на одном компьютере Данная технология может работать со множеством различных устройств ввода/вывода и драйверов, несмотря на то, что их поддержка зависит от реализации операционной системы. Подобная реализация интерфейса лежит в основе TCP/IP, благодаря чему считается одной из фундаментальных технологий, на которых основывается интернет.

Программисты могут получать доступ к интерфейсу сокетов на трёх различных уровнях, наиболее мощным и фундаментальным из которых является уровень сырых сокетов. Довольно небольшое число приложений нуждается в ограничении контроля над исходящими соединениями, реализуемыми ими, поэтому поддержка сырых сокетов задумывалась быть доступной только на компьютерах, применяемых для разработки на основе технологий, связанных с интернет. Впоследствии, в большинстве операционных систем была реализована их поддержка, включая Window XP.

Unix domain socket (Доменный сокет Unix) или IPC-сокет (сокет межпроцессного взаимодействия) — конечная точка обмена данными. Схожая с Интернет-сокетом, но не использующая сетевой протокол для взаимодействия (обмена данными). Он используется в операционных системах, поддерживающих стандарт POSIX, для межпроцессного взаимодействия. Корректным термином стандарта POSIX является POSIX Local IPC Sockets.

Доменные соединения Unix являются по сути байтовыми потоками, сильно напоминая сетевые соединения, но при этом все данные остаются внутри одного компьютера (т.е. обмен данными происходит локально). UDS используют файловую систему как адресное пространство имен, т,е. они представляются процессами как иноды В файловой системе. Это позволяет двум различным процессам открывать один и тот же сокет для взаимодействия между собой. Однако, текущее взаимодействие (обмен данными) не использует файловую систему, а только буферы памяти ядра.

В дополнение к перееылаемым данным процессы могут отсылать файловые дескрипторы через соединение на основе UDS (включая файловые дескрипторы для доменных сокетов), используя системные вызовы sendmsg() и recvmsg(). Это означает, что доменные сокеты могут быть использованы как объектно-возможностная коммуникационная система.

Windows Sockets API (WSA), название которого было укорочено до Winsock. Это техническая спецификация, которая определяет, как сетевое программное обеспечение Windows будет получать доступ к сетевым сервисам, в том числе, TCP/IP. Он определяет стандартный интерфейс между клиентским приложением (таким как FTP клиент или веб- 9 браузер) и внешним стеком протоколов TCP/ПР, Он основывается на API модели сокетов Беркли, использующейся в BSD для установки соединения между программами.

Ранние операционные системы Microsoft, такие как MSDOS и Microsoft Windows имели ограничения по работе с сетью, которые были связаны с использованием протокола NetBIOS. В частности, Microsoft в то время не поддерживал работу со стеком протоколов TCP/IP. Несколько университетских групп и коммерческих фирм, включая MIT, FTP Software, SunMicrosystems, Ungermann-Bass, и Excelan, представляли свои решения для работы с TCP/IP в MS-DOS, часто как часть программно аппаратного комплекса. После выпуска Microsoft Windows 2.0 к этим разработчикам присоединились и другие, такие как Distinct и NetManage которые помогли в организации поддержки протоколов TCP/IP для Windows. Недостаток, с которыми столкнулись все вышеперечисленные разработчики, состоял, в том, что каждый из них использовал свои собственные API (ApplicationProgrammingInterface). Без единой стандартной модели программирования, трудно было убедить независимых разработчиков программного обеспечения в создании сетевых программ, которыс могли бы работать на основе реализации стека протоколов TCP/IP любого из разработчиков. Стало понятно, что необходима стандартизация.

Начиная с Windows 2000 Winsock работает через Transport Driver Interface Windows 8 включает в себя RIO (Registered IO), расширяющий возможности Winsock.

1.4 Определение почтового сервера

Почтовый сервер, сервер электронной почты, мейл-сервер — в системе пересылки электронной почты так обычно называют агент пересылки сообщений (англ. Mail transfer agent, MTA). Это компьютерная программа, которая передаёт сообщения от одного компьютера к другому. Обычно почтовый сервер работает «за кулисами», а пользователи имеют дело с другой программой — клиентом электронной почты (англ. Mail user agent, MUA). Взаимодействие клиент-сервера представлено на Рисунке No1.1.

Сервер, отправляющий электронные сообщения работает по протоколу SMTP (Simple Mail Transfer Protocol).

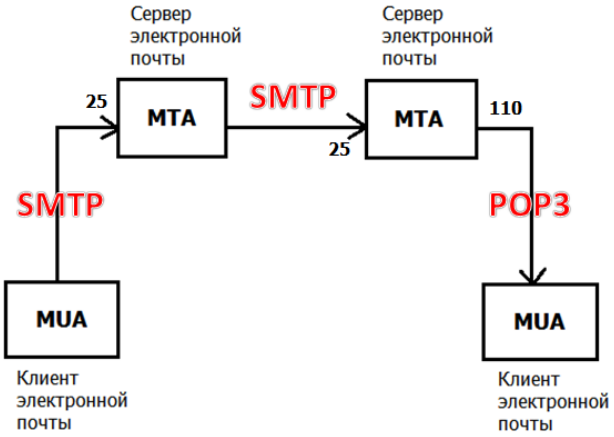

Рисунок 1.1. Схема взаимодействия клиента и почтового сервера.

1.4.1 Simple Mail Transfer Protocol

Основная задача протокола SMTP (Simple Mail Transfer Protocol) заключается в том, чтобы обеспечивать передачу электронных сообщений (почту). Для работы через протокол SMTP клиент создает TCP соединение с сервером через порт 25. Затем клиент и SMTP сервер обмениваются информацией пока соединение не будет закрыто или прервано. Основной процедурой в SMTP является передача почты (Mail Procedure). Далее идут процедуры форвардинга почты (Mail Forwarding). проверка имени почтового ящика и вывод списков почтовых групп. Самой первой процедурой является открытие канала передачи, а последней - его закрытие.

Команды SMTP указывают серверу, какую операцию хочет произвести клиент. Команды состоят из ключевых слов, за которыми следует один или более параметров. Ключевое слово состоит из 4-х символов и разделено от аргумента одним или несколькими пробелами. Каждая командная строка заканчивается символами CRLF. Синтаксис всех команд протокола SMTP представлен в Листинге кода 1.1.

```
HELO <SP> <domain> <CRLF>

MAIL <SP> FROM:<reverse-path> <CRLF>
RCPT <SF> TO:<forward-path> <CRLF>
DATA <CRLF>
RSET <CRLF>
SEND <SP> FROM:<reverse-path> <CRLF>
SOML <SP> FROM:<reverse-path> <CRLF>
SAML <SP> FROM:<reverse—path> <CRLF>
VRFY <SP> <string> <CRLF>
EXPN <SP> <string> <CRLF>
HELP <SP> <string> <CRLF>
NOOP <CRLF>
QUIT <CRLF>
```

Листинг кода 1.1. Синтаксис команд протокола SMTP.

1.4.2 Почтовый сервер и DNS

Почтовые сервера используют глобальную Систему Доменных Имён (DNS) для нахождения сетевого адреса компьютера получателя или почтового сервера получателя. Каждый домен (часть адреса электронной почты после символа @) должен иметь так называемую MX-запись в DNS. Эта запись указывает имя компьютера, который фактически получает почту для этого домена. Например, MX-записи могут указывать, что почта для домена company.com должна отправпяться на компьютер mail.company.com, а почта для домена enduser.com должна отправляться на компьютер provider.com.

Для одного домена может быть несколько MX-записей (с разными приоритетами). Если один из компьютеров (с наивысшим приоритетом) не может получать почту, то почта отправляется на компьютеры с более низким приоритетом (так называемые Запасные Почтовые Серверы). Запасные Почтовые Серверы попытаются доставить сообщение на основной сервер позднее

Когда имя компьютера получателя получено из DNS, отправляющий почтовый сервер обращается к DNS повторно, Теперь он использует DNS для того, чтобы преобразовать имя получающего почтового сервера в его сетевой адрес. Так называемые А-запиеи DNS содержат пары, которые связывают имя компьютера с его глобальным сетевым (IP) адресом в Интернет.

По получению из DNS сетевого адреса компьютера почтового сервера получателя, отправляющий почтовый сервер открывает SMTP соединение с этим сервером и передает сообщение (сообщения). После того, как все сообщения в этот домен переданы, соедипспие закрывается.

Когда сообщение имеет несколько адресов из одного домена, SMTP модуль может передать почтовому серверу, обслуживающему домен, только одну копию сообщения, и тот сервер самостоятельно доставит сообщепия всем получателям в этом домене. Но если В сообщение содержится слишком много адресов, то SMTP модуль может разбить их на несколько частей и отправить нёсколько копий сообщения, каждая из которых будет содержать только часть списквв адресов.

Если для одного домена есть несколько сообщений, то SMTP модуль может открыть несколько соединений с почтовым сервером, обслуживающим этот домен и отправить эти сообщения одновременно.

Для того, чтобы сообщения из Интернет получал зарегистрированный ранее почтовый сервер, то необходимо зарегистрировать домен и попросить провайдера зарегистрировать его имя в DNS. Записи в DNS должны указывать на компьютер, на котором работает почтовый сервер.

1.5 Определение почтового клиента

SMTP-клиент или почтовый клиент - это программа, обеспечивающая отправку электронных писем по SMTP протоколу на SMTP-сервер(ы). По способу работы их можно поделить на три группы:

- почтовые клиенты для приема почты (POP3 клиент);
- почтовые клиенты для отправки почты (SMTP клиент);
- почтовые клиенты для фильтрации (программа - фильтр почтовых сообщений)

1.5.1 Адресация в SMTP - системах

В системах на базе SMTP используется интуитивно понятная, простая и одновременно очень мощная иерархическая схема адресации, аналогичная той, что принята в службе имен Internet (Domain Name Services или DNS, подробнее раздел 1.4.2). Данная схема может обеспечить уникальность адреса практически неограниченному числу пользователей. Почтовый адрес SMTP записывается в следующем виде: mailbox@domain, где:

- mailbox- символическое имя почтового ящика пользователя, длинной до 63 символов;

- domain - уникальное имя (почтовый домен) системы, в которой зарегистрирован упомянутый пользователь, длинной до 255 символов.

Сочетание имени и домена образует уникальный идентификатор пользователя, Почтовый домен хранит полную информацию о положении системы в иерархии почтового пространства организации Каждый следующий уровень иерархии отделяется от предыдущего точкой. Разбор имени домена выполняётся справа налево.

Самый верхний уровень (top level), называемый корневым доменом (root domain), соответствует либо типу организации (com - для коммерческой, gov- для государственной. org - для общественной и т.п.), либо географическому региону (стране) (ru - для России, fr — для Франции и т.д,).

Следующими в иерархии идут домены первого уровня (first level), как правило, представляющие имя организации. Регистрацией имен доменов первого уровня занимается международный центр Internet (Internet Network Information Center или InterNIC).

За назначение имен доменов более низкого уровня чаще всего отвечают сами компании. Поскольку организациям не запрещается регистрировать для собственных нужд несколько параллельных доменов (например, CIT.MSK.RU и CIT.COM), пользователь может иметь более одного SMTP-адреса. Кроме того, современные ЗМТР-системы зачастую позволяют назначать псевдонимы для самого почтового ящика (например BMSTU_STUDENT@CIT.COM и BMSTU_STUDENT@CIT.MSK.RU). Рисунок 1.2 подробно описывает иерархическую структуру адресации по протоколу SMTP для примера.

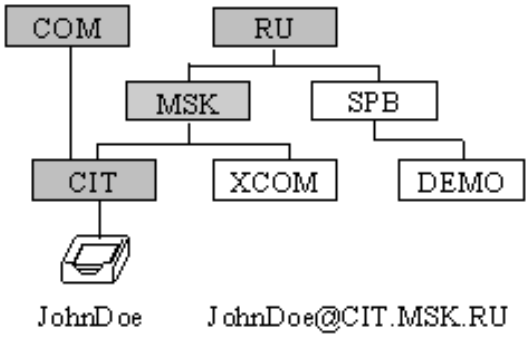

Рис 1.2. Иерархическая схема адресация no протоколу SMTP

1.6 Заключение

Главной задачей данного протокола является обеспечение передачи электронных сообщений (почты). Для организации работы с помощью протокола SMTP клиент проводит TCP соединение с сервером через порт 25. После этого происходит обмен информацией между клиентом и SMTP сервером до тех пор, пока соединение не закроется или не будет прервано.

В базовом протоколе передачи электронной почты SMTP есть недостаток: он не позволяет на должном уровне проводить аутент фикацию внешних отправителей При отправке письма в поле From можно подставить любой адрес. Например, пользователь, который получил письмо от мамы, не может быть уверен, что ему писала именно она. Подделку email можно распознать, но для этого нужно быть технически подкованным специалистом: необходимо сверять заголовки, служебную информацию, сервер и IP, с которых поступило письмо.

В 2002 г. для SMTP было разработано расширение STARTTLS, позволившее отправлять письма в зашифрованном виде.

После того как в 2013 г. Эдвард Сноудсн (Edward Snowden) рассказал о приемах Агенства национальной безопасности США, шифрование набрало популярность. В 2014 г. соцсеть Facebook, отправляющая миллиарды почтовых уведомлений в день, выяснила, что 58% этих уведомлений проходят по защищенным каналам. К августу того же года эта цифра возросла до 95%.

Расширение STARTTLS используется и по сей день, но оно не гарантирует защиту данных, так как содержит в себе ряд изъянов, позволяющих хакерам успешно выполнить свою работу.

Используя недостатки расширения, злоумышленник может фальсифицировать почтовый сервер и убедить приложение на ПК или другой почтовый сервер отправить сообщение в виде простого текста. Сдслать это можно путем принудительного отказа от шифрования (в штатном режиме такая необходимость может возникнуть тогда, когда принимающий сервер не поддерживает шифрование) либо путем использования поддельного сертификата.

2. Конструкторский раздел

2.1 Схема взаимодействия клиент - сервера

Согласно стандарту для SMTP взаимодействия, описанного в разделе 1.4.1, в листинге кода 2.1 описаны поддерживаемые сервером запросы. Представленная на рисунке 2.1 схема иллюстрирует взаимодействие клиента и сервера путем отправки запросов в соответствии с запросами из листинга 2.1.

```
state = {
    'HELO': False,
    'MAIL': False,
    'RCPT': False,
    'loop': True,
    'data': False,
    'recipient': "",
    'file':0,
    'domain': "",
    'completedTransaction': False
    }
```

Листинг кода 2.1 Инциалнзаиия состояний сервера из файла SMTP.py

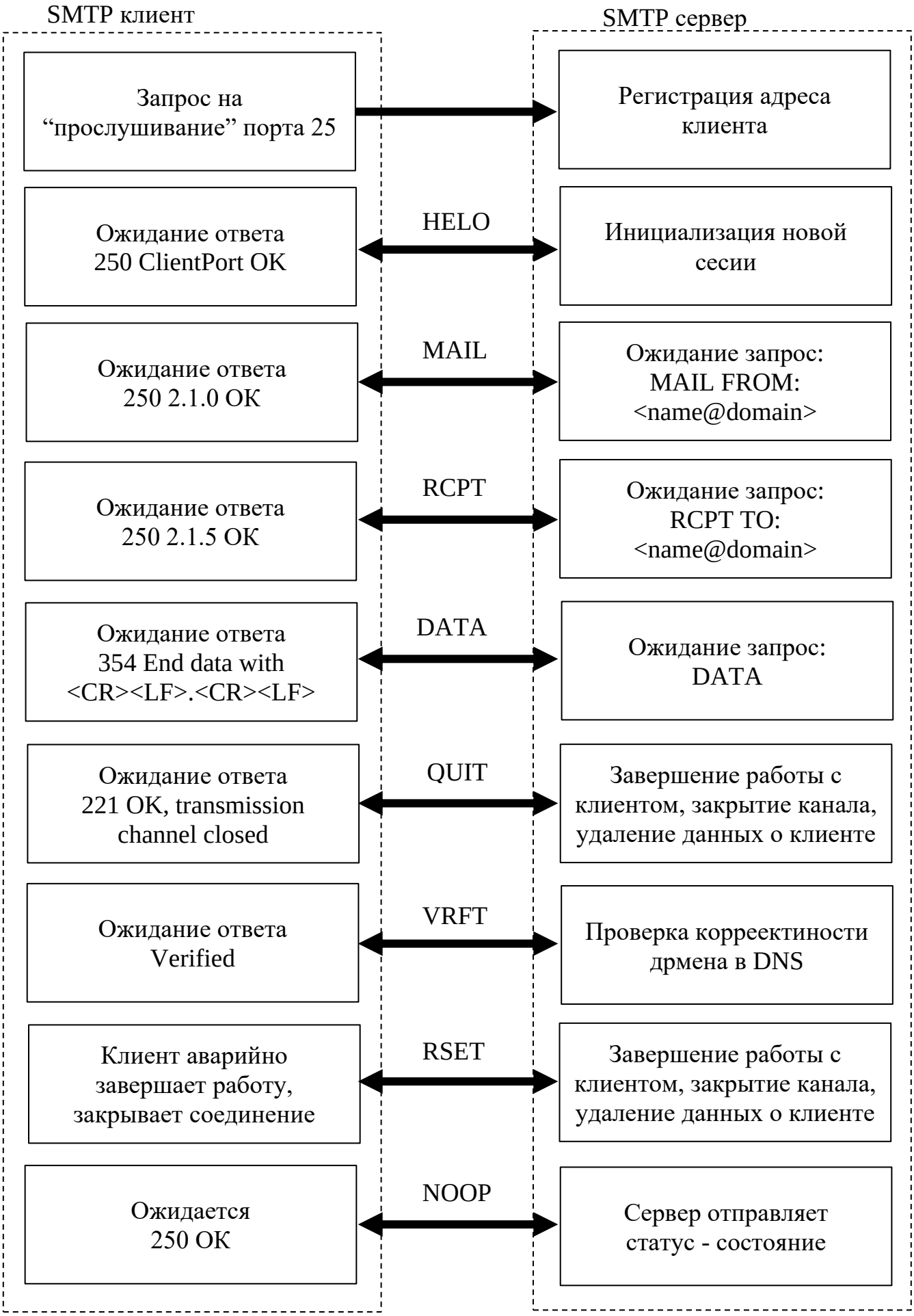

Рисунок 2.1 Схема взаимодействия клиент-сервера по протоколу SMTP

2.2. Клиентская часть

Клиент обрабатывает статус - коды от сервера, в соотвстствии с группами кодов статусов, приведенных в таблице 2.1.

Группа кодов статуса	Название группы	Описание
1xx	Информационная	Запрос получен, продолжение процесса
2xx	Выполнено	Действие успешно получено, проанализировано и принято
3xx	Псреадресация	Для выполнения запроса требуются дополнительные действия
4xx	Ошибка клиента	В запросе используется неверный или неразрешенный синтаксис
5xx	Ошибка сервера	Серверу не удалось выполнить явно корректный запрос

Таблица 2.1. Стандартные коды статусов SMTP.

Для реализации клиента необходимо создать объект класса socket, а затем вызвать метод bind(), передав в него, первым параметром, имя компьютера (или его IPадрес), а вторым — номер порта сервера. Объект класса socket сам попытается установить связь с сервером и, в случае успеха, отправит HELO запрос на сервер. В противном случае будет выслан сигнал error(int) с кодом ошибки, определенным в перечислении SMTP.HaloError. Это может произойти. например. в том случае, если на указанном компьютере не запущен сервер или не соотвстствует номер порта.

После устзновлсния соединения, объект класса socket можот высылать или считыввёгь данные сервера.

Затем на сервер отправляется запрос формата MAIL FROM:<address@domain>, в случае если сервер находит <address@domain> в списке DNS, клиент получает 250 OK, в противном случае обрабатывается ошибка SMTP.SMTPSenderRefused, и запрос MAIL FROM:<address@domain> запрашивает с потока ввода новое <address@domain>. Клиент поочередно отпрваляет запросы:

- RCPT TO:<address@domain> - посылает на сервер адрес получателя, ожидает статус 250 OK, в противном случае обрабатывает исключение SMTP.SMTPRecipiemRefused;

- Data - в случае успеха сервер отправляет статус 354 и запрашивает ввод данных, однако имеет дополнительные опции. в противном случае обрабатывается исключение ЗМТРВтаЕггог. Список дополнительных опций:

- SUBJECT: — указывается тема сообщения. ввод завершается <CR>;

- QUIT — ввод завершается комбинацией <CR><LF>.<CR><LF>, на сервер отправляется запрос QUIT, клиент- прерывает соединение.

Также дополнительно клиент может отправлять и обрабатывать следующие запросы:

- VRFY <address@domain> - отправить запрос на сервер для регистрации <address@domain> в DNS;

- RSET - аварийное завершение работы клиента, закрытие соелипспия, может прервать клиент-серверное взаимодействие в любой момент;

- NOOP - отправка запроса на сервер о его работоспособности, а также ожидания ответа от клиента-получателя 250 ОК.

2.3 Серверная часть

Функция HELO - отмечает начало сеанса с клиентом и имеет синтаксис HELO Hostname. Данный запрос должен быть принят только в том случае, если сервер не является расширенным сервером рассылки стаи-сообщений, в противном случае используется запрос EHLO. После получения правильного запроса сервер отвечает клиенту 250 ОК. Если получена ошибка в синтаксисе, сервер ответит 501 Syntax: HELO Hostname. Отправка команды HELO в любой момент почтового сеанса сбросит почтовый сеанс, состояние и очищает все буферы. Команда HELO - разрешает почтовую транзакцию. Инициализация сессии с новым клиентом, обработка SMTP - запроса HELO представлена в Листинге 2.2.

```
def HELO(args, s, client_address, state):
    fileName = str(client_address[1]) + '.txt'
    if len(args) != 2:
        s.send("501 Syntax: HELO hostname \n".encode())
        return

    if state['HELO'] == False:
        with open(fileName, 'w') as the_file:
            the_file.write(" ".join(args) + "\n")
        state['HELO'] = True
        state['file'] = client_address[1]
        state['domain'] = args[1]
        s.send(("250 "+ str(client_address[1]) + " OK \n").encode())

    else:
        open(fileName, 'w').close()
        with open(fileName, 'a') as the_file:
            the_file.write(" ".join(args) + "\n")
        state['HELO'] = False
        state['MAIL'] = False
        state['RCPT'] = False
        state['completedTransaction'] = False
        state['HELO'] = True
        s.send(("250 "+ str(client_address[1]) + " OK \n").encode())
```

Листинг кода 2.2 Функция обработки SMTP-запроса HELO.

Функция MAIL отмечает начало почтовой транзакции и представлена в листинге 2.3. Запрос MAIL имеет синтаксис MAIL FROM:<address@domain>. По получению правильного запроса сервер отвечает 250 OK, в противном случае 501 5.5.4 Syntax: MAIL FROM:<address>. Сервер проверяет формат адреса электронной почты путем использования регулярного выражения. Если клиент пытается инициализировать MAIL-запрос вложено, то сервер ответит 503 5.5.1 Error: nested MAIL command. В случае если MAIL - запрос используется перед HALO сервер возвращает ответ об ошибке последовательности инициализации клиента 503 5.5.1 Error: HELO/EHLO first.

```
def MAIL(args, s, client_address, state):
    fileName = str(state['file']) + '.txt'
    if state['HELO'] == False:
        s.send("503 5.5.1 Error: send HELO/EHLO first \n".encode())
    else:
        if state['MAIL'] == False:
            if len(args) != 2:
                s.send("501 5.5.4 Syntax: MAIL FROM:<address> \n".encode())
                return
            checkSyntax = re.match("(^FROM:<[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-
.]+>$)", args[1], re.IGNORECASE)
            if(checkSyntax):
                if state['data'] == False:
                    with open(fileName, 'a') as the_file:
                        the_file.write(" ".join(args) + "\n")
                    state['MAIL'] = True
                    s.send("250 2.1.0 Ok \n".encode())
                else:
                    state['file'] = state['file'] + 1
                    fileName = str(state['file']) + '.txt'
                    with open(fileName, 'a') as the_file:
                        the_file.write("helo " + state['domain'] + "\n")
                        the_file.write(" ".join(args) + "\n")
                    state['MAIL'] = True
                    state['completedTransaction'] = False
                    s.send("250 2.1.0 Ok \n".encode())
            else:
                s.send("501 5.1.7 Bad sender address syntax \n".encode())
        else:
            s.send("503 5.5.1 Error: nested MAIL command \n".encode())
```

Листинг кода 2.3 Функция обработки SMTP - запроса MAIL.

Функция RCPT - инициализирует имена отправигслсй, данный запрос может быть получен от клиента несколько раз, тогда будет проинициализирована сразу группа адресатов, реализация функции представлена в листинге 2.4. После получения от клиента правильного синтаксиса RCPT TO:<address@domain> сервер ответит 250 OK, в противном случае 501 5.5.4 Syntax: RCPT TO:<address>. Сервер проверяет формат адреса электронной почты, поскольку для процесса ретрансляции и отправки писем крайне важна корректность домена. Если команда RCPT будет использована до MAIL сервер вернет ответ об ошибке последовательности выполнения 503 5.5.1 Error: need mail command.

```
def RCPT(args, s, client_address, state):
    if state['MAIL'] == True and state['HELO'] == True:
        if len(args) != 2:
            s.send("501 5.5.4 Syntax: RCPT TO:<address> \n".encode())
            return
        checkSyntax = re.match("(^TO:<[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+>$)",
args[1], re.IGNORECASE)
        if(checkSyntax):
            state['recipient'] = checkSyntax.group()
            fileName = str(state['file']) + '.txt'
            with open(fileName, 'a') as the_file:
                the_file.write(" ".join(args) + "\n")
            state['RCPT'] = True
            s.send("250 2.1.5 Ok \n".encode())
        else:
            s.send("501 5.1.3 Bad recipient address syntax \n".encode())
    else:
        s.send("503 5.5.1 Error: need MAIL command \n".encode())
```

Листинг кода 2.4 Функция обработки SMTP-запроса RCPT.

Функция DATA представлена в листинге кода 2.5. Запрос DATA имеет синтаксис DATA. После получения синтаксически правильного запроса сервер отвечает клиенту 354 End data with <CR><LF>.<CR><LF>. Сервер будет получать данные до тех пор, пока не будет получено <CR><LF>.<CR><LF>, что является меткой об окончании блока DATA, после чего инициализируется операция отправки. Операция отправки email выполняется в новом потоке. Команда DATA может выполняться только после HELO, MAIL, RCPT. Каждая почтовая транзакция имеет только одну комзнду данных.

```
def DATA(args, s, client_address, state):
    fileName = str(state['file']) + '.txt'
    if state['MAIL'] == True and state['HELO'] == True and state['RCPT'] == True:
        s.send("354 End data with <CR><LF>.<CR><LF> \n".encode())
        data = recieveData(s, state)
        with open(fileName, 'a') as the_file:
            the_file.write("data \n")
            the_file.write(data)
            the_file.write("quit \n")
        state['MAIL'] = False
        state['RCPT'] = False
        s.send(("250 queued " + str(state['file']) + " \n").encode())
        state['data'] = True
        state['completedTransaction'] = True
        _thread.start_new_thread(relayData, (state['file'], state))
    elif state['MAIL'] == True and state['RCPT'] == False:
        s.send("554 5.5.1 Error: no valid recipients \n".encode())
    else:
        s.send("503 5.5.1 Error: need RCPT command \n".encode())
```

Листинг кода 2.5 Функция обработки SMTP-запроса DATA.

В листинге 2.6 представлены функции, реализующие обработку дополнительных запросов от клиента (детальное описание дополнительных запросов в разделе No2.2).

Запрос RSET сбрасывает почтовый сеанс. Отправка команды RSET в любой момент почтового сеанса сбросит сам сеанс, его состояние и очищает все буферы.

Запрос NOOP не оказывает влияния на параметры и ранее введенные команды. Не указывает никаких действий, кроме того, что получатель (или сервер, в случае если данные только передаются на сервер) подтверждают свою активность и отправляют ответ 250 OK.

Запрос QUIT завершает работу клиента-отправителя, клиент-получатель обязан отправить ответ 221 OK, затем закрыть канал передачит.

Запрос VRFY имеет синтаксис VRFY:<address@domain> - эта команда просит сервер подтвердить существование домена в списке DNS.

```
def NOOP(args, s, client_address, state):
    s.send("250 Ok \n".encode())

def QUIT(args, s, client_address, state):
    state['loop'] = False
    s.send("221 2.0.0 Bye \n".encode())
    s.close()
    if state['completedTransaction'] == False:
        fileName = str(state['file']) + '.txt'
        os.remove(fileName)

def VRFY(args, s, client_address, state):
    print(args)
    if len(args) > 2:
        s.send("501 5.5.4 Syntax: VRFY address \n".encode())
        return
    checkSyntax = re.match("TO:<\w+@\w+\.\w+>", args[1], re.IGNORECASE)
    if(checkSyntax):
        s.send("252 - VRFY'ed user \n".encode())
    else:
        s.send("450 4.1.2 Recipient address rejected: Domain not found \n".encode())

def RSET(args, s, client_address, state):
    fileName = str(state['file'] + '.txt')
    with open(fileName) as f:
        first_line = f.readline()
    open(fileName, 'w').close()
    with open(fileName, 'a') as the_file:
        the_file.write(first_line)
    state['MAIL'] = False
    state['RCPT'] = False
    s.send("250 OK \n".encode())
```

Листинг 2.6 Функции обработки дополнительных SMTP - запросов.

3. Технологический раздел

3.1. Выбор среды разработки и языка программирования

Языком программирования был выбран Python версии 3.7, поскольку он предоставляет удобный объектный C-подобный интерфейс для работы с сетевым сокетами.

В качестве среды разработки использовались Anaconda prompt.

Anaconda обеспечивает разработчика всеми необходимыми средствами для создания приложений и программирования на нескольких различных языках. Удобный пакетный менеджер и поддержка виртуальных оболочек позволяет с легкостью создавать все необходимые объекты пользовательского приложения, связывать их с источниками данных и создавать в оболочках свои txt с требованиями к версиям сторонних пакетов - requirements.txt. которые в случае распространения программного обеспечения легко установить на любое устройство с помощью встроенного пакетного менеджера conda.

3.2 Описание использованных модулей и библиотек

Модуль socket обеспечивает возможность работать с сокетами из Python. Сокеты используют транспортный уровень согласно ссмиуровнсвой модели OSI (Open Systems Interconnection, взаимодействие открытых систем), то есть относятся к более низкому уровню, чем большинство описываемых в этом разделе протоколов.

Каждый сокет относится к одному из коммуникационных доменов. Модуль socket поддерживает домены UNIX и Intemet. Каждый домен подразумевает свое семейство протоколов и адресацию, Данное изложение будет затрагивать только домен Internet, а именно протоколы TCP/IP и UDP/IP, поэтому для указания коммуникационного домена при создании сокета будет указываться константа socket.AF_INET.

Сервер открывает сокет на локальной машине на порту 25, и адресе 127.0.0.1. После этого он слушает (listen()) порт, Когда на порту появляются данные, принимается (accept()) входящее соединение. Метод accept() возвращает пару - Socket-объект и адрес удаленного компьютера, устанавливающего соединение (пара - IP-адрес, порт на удаленной машине). После этого применяются методы recv() и send() для общения между клиентом и сервером. В recv() задается число байтов в очередной порции. От клиента может прийти и меньшее количество данных.

Для работы с SMTP В стандартной библиотеке модулей имеется модуль smtplib. Для того чтобы начать SMTP-сосдинснис с сервером электронной почты, необходимо вначале создать объект для управления SMTP-сессией с помощью конструктора класса SMTP: smtplib.SMTP([host[, port]]) Параметры host и port задают адрес и порт SMTP-ссрвера, через который будет отправляться почта. По умолчанию, port=25. Если host задан, конструктор сам установит соединение, иначе придется отдельно вызывать метод conect().

Экземпляры класса SMTP имеют методы для всех распространенных команд SMTP протокола, но для отправки почты достаточно вызова конструктора и методов sendmail() и quit().

При работе с классом smtplib.SMTP могут возбуждаться различные исключения. Назначение некоторых из них приведено ниже:

- smtplib.SMTPExcepIion - базовый класс для всех исключсний модуля;

- smtplib.SMTPServerDisconnected - сервер неожиданно прервал связь(или связь с сервером не была установлена);

- smtplib.SMTPResponseException - базовый класс для всех исключений, которые имеют код ответа SMTP-сервера;

- smtplib.SMTPSenderRefused - отправитель отвергнут;

- smtplib.SMTPRecipientsRefused - все получатели отвергнуты сервером;

- smlplib.SMTPDataError - сервер ответил неизвестным кодом на данные сообщения;

- smtplibSMTPConncctError - ошибка установления соединения;

- smtplib.SMTPHeloError — сервер не ответил правильно на команду HELO или отверг её.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были выполнены следующие задачи:

- Изучить предметную область и проанализировать существуйющие решения.
- Анализируйте протоколы SMTP
- Разработать программную структуру для серверной части
- Разработать программную структуру для клиентской части
- Внедрить программный комплекс

Таким образом достигнута цель – разработан клиент-серверного приложение на основе протокола SMTP.

- ❖ Возможные улучшения
  - ➢ разрабатывать с продвинутым интерфейсом
  - ➢ добавление IMAP, POP3 к реализации для большего прогресса

Для разработки были выбраны такие средства как: Python, Anaconda prompt.

Список использованных источников

1. SMTP – Wikipedia [Электронный ресурс]
[https://ru.wikipedia.org/wiki/SMTP] (дата обращения: 20.12.2020)
2. How to configure an SMTP server [Электронный ресурс]
[https://www.serversmtp.com/smtp-configuration/] (дата обращения: 22.12.2020)
3. Сети и телекоммуникации [Электронный ресурс]
[http://network-journal.mpei.ac.ru/cgi-bin/main.pl?l=ru&n=24&pa=11&ar=1]
(дата обращения: 22.12.2020)
4. Протокол резервирования ресурсов RSVP // opennet.ru [Электронный ресурс].
[https://www.opennet.ru/docs/RUS/inet_book/4/44/rsv_4496.html]
(дата обращения: 08.12.2020).