Разработка SMTP-клиента как части MTA. Вариант №26

(Студент группы ИУ7-33М: Овчинников Владислав Александрович.)

26 декабря 2020 г.

Оглавление

Введение

Практически каждое вычислительное устройство должно обмениваться данными с другими вычислительными устройствами, среди которых можно выделить - компьютеры, серверы, маршрутизаторы и многие другие устройства, которые требуют данные извне. Система, обеспечивающая обмен между такими устройствами называется компьютерной сетью. Для функционирования компьютерных сетей используются сетевые протоколы.

Сетевой протокол - это набор программно-реализованных правил общения компьютеров, подключенных к сети. В настоящее время стандартом стало использование стека протоколов TCP/IP.

В стеке протоколов TCP/IP были выделены четыре следующих уровня передачи информации между процессами:

- Канальный уровень (Ethernet, PPP, HDLC) – предназначен для передачи данных между сетевыми адаптерами в одном сегменте сети. Также может использоваться для обнаружения и, возможно, исправления ошибок, возникших на физическом уровне.
- Сетевой уровень (IP) – предназначен для передачи данных между компьютерами в разных сегментах сети. Основная цель: определение пути передачи данных. Отвечает за трансляцию логических адресов в физические.
- Транспортный уровень (TCP, UDP) – предназначен для передачи данных между процессами на разных компьютерах. При этом неважно, какие данные передаются, т.е. представляет сам механизм передачи.
- Прикладной уровень (HTTP, SMTP) – обеспечивает взаимодействие сети и пользователя, разрешает приложениям пользователя иметь доступ к сетевым службам, таким как обработчик запросов к базам данных, доступ к файлам, пересылке электронной почты, просмотру веб-страниц и т.д.

В данной курсовой работе рассматривается разработка SMTP-клиента. Simple Mail Transfer Protocol (SMTP) - это широко используемый сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP. SMTP впервые был описан в RFC 821. Последнее обновление в RFC 5321 включает масштабируемое расширение

- ESMTP. Протокол SMTP предназначен для передачи исходящей почты с использованием порта TCP 25.

Целью курсовой работы является реализация SMTP-клиента (как части MTA), обеспечивающего удаленную доставку и поддерживающего очереди сообщении. Вариант лабораторной работы подразумевает реализацию многопоточного SMTP-клиента с использованием вызова pselect.

Глава 1

Аналитический раздел

1.1 Основные понятия протокола SMTP

В рамках курсовой работы требовалось разработать SMTP-клиент как часть MTA. Перед разработкой архитектуры был проведен анализ предметной области, связанной с решением поставленной задачи. Был рассмотрен SMTP-протокол, MTA, DNS.

SMTP (Simple Mail Transfer Protocol) - протокол передачи сообщений с компьютера на почтовый сервер для доставки конечному получателю. Этот протокол обеспечивает перенаправление почтовых сообщений с помощью записей типа MX (или записей программы обмена электронной почтой) и записей типа A (или записей сервера в системе DNS), форматирование почтовых сообщений и установление сеансов между почтовыми клиентами и почтовыми серверами. В протоколе SMTP в качестве транспортного протокола обычно используется TCP, но могут применяться и другие протоколы, как определено в документе RFC 821.

Обмен данными в рамках SMTP строится по принципу двусторонней связи, которая устанавливается между отправителем и получаетелем почтового сообщения. При этом отправитель инициирует соединение и посылает запросы на обслуживание, а получатель - отвечает на эти запросы. Фактически, отправитель выступает в роли клиента, а получатель - сервера. Канал связи устанавливается непосредственно между отправителем и получателем сообщения.

Протокол SMTP не несет никакой ответственности за прием почты. В спецификации этого протокола не определены способы настройки почтовых ящиков для отдельных пользователей, а также не упоминаются какие-либо иные задачи (такие как аутентификация), которые должны быть решены при приеме электронной почты. В этой спецификации просто указано, как должна осуществляться передача электронной почты от отправителя к получателю.

Основная задача протокола SMTP заключается в том, чтобы обеспечивать передачу электронных сообщений. 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 Mail Forwarding, проверка имен почтового ящика и вывод списка почтовых групп. Самой первой процедурой является открытие канала передачи, а последней - его закрытие.

MTA (Mail Transfer Agent) - самостоятельное, минимально достаточное для приема и отправки электронной почты программное обеспечение. Важнейшей частью почтового сервера является MTA (Mail Transfer Agent - агент пересылки почты), в задачи которого входит прием и передача почты.

MTA работает по протоколу SMTP и его одного достаточно для создания системы электронной почты. Работа MTA совмещает в себе одновременно функции внешней и локальной доставки и получения почты. MTA, получая письмо, помещает его в почтовый ящик пользователя на своем сервере, к которому последний должен получить доступ.

MDA (Mail Delivery Agent) - это агент доставки почты, его задача по запросу почтового клиента передать ему почту из почтового ящика на сервере. MDA может работать по протоколам POP3 (Post Office Protocol v3) или IMAP (Internet Message Access Protocol), в ряде случаев для "общения"почтового клиента и агента доставки могут применяться собственные протоколы, обладающие расширенной функциональностью, например MAPI (Messaging Application Programming Interface) в Exchange Server.

MTA (Mail Transfer Agent - агент пересылки почты) - отвечает за пересылку почты между почтовыми серверами, как правило, первый MTA в цепочки получает сообщения от MUA (почтовый агент пользователя), последний MTA передает сообщение к MDA.

MUA (Mail User Agent) - программа, обеспечивающая пользовательский интерфейс, отображающая полученные письма и предоставляющая возможность отвечать, создавать и перенаправлять письма.

Общение с SMTP сервером ведется при помощи команд. Команды SMTP указывают, 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 команда заканчивается символами CRLF. Обычный ответ SMTP-сервера состоит из номера ответа, за которым через пробел следует дополнительный текст. Номер ответа служит индикатором состояния сервера.

- EHLO – данная команда используется для начала диалога клиента с сервером и получения расширений ESMTP, которые доступны для данного сервера (устаревшая - HELO).
- HELO – устаревшая стандартная команда SMTP для начала диалога клиента с сервером (не позволяет получать расширения ESMTP).

- MAIL – определяет отправителя сообщения, используется для ответных сообщений в случае невозможности доставки письма. Для каждого письма команда MAIL должна быть выполнена только один раз.
- RCPT – определяет получателей сообщения. Доставка сообщения возможна тогда, когда указан хотя бы один доступный адрес получателя. Команда RCPT принимает в качестве аргумента только один адрес. Если нужно послать письмо большему числу адресатов, то команду RCPT следует повторять для каждого.
- DATA – определяет начало сообщения. С помощью этой команды серверу передается текст сообщения, состоящий из заголовка и отделенного от него пустой строкой тела сообщения. Команда DATA может быть выполнена только после успешного выполнения хотя бы одной команды RCPT.
- QUIT – остановка сеанса SMTP. Клиент заканчивает диалог с сервером. Сервер посылает подтверждение и закрывает соединение. Получив это подтверждение, клиент тоже прекращает связь.
- HELP – запрашивает список команд. Если команда HELP вызывается без параметров, сервер посылает клиенту список доступных команд. Если в качестве параметра передано название команды, то клиенту посылается описание этой команды.
- VRFY – проверяет имя пользователя системы. Используется для проверки наличия указанного в качестве аргумента почтового ящика. В ответ сервер посылает информацию о владельце ящика или сообщение об ошибке, свидетельствующее о том, что указанный ящик не существует.
- EXPN – используется для получения адресов, внесенных в список рассылки.
- NOOP – в ответ на данную команду сервер посылает подтверждение выполнения. Никаких действий на сервере не производится, параметры команды игнорируются.
- RSET – сброс SMTP-соединения. Данная команда аннулирует все переданные до нее на сервер данные. Процесс передачи сообщения следует начать заново с выполнения команды EHLO (HELO).
- TURN – смена направления передачи.

1.2 SMTP-сессия

По протоколу SMTP отправитель письма связывается с получателем при помощи командной строки и специальных каналов, роль которых обычно выполняет TCP-соединение. Любая SMTP-сессия состоит из двух ведующих компонентов: команд от клиента и соответствующих им ответов сервера. При открытой сессии обе этих составляющих обмениваются ее параметрами. Подобный обмен может включать как ноль, так и больше SMTP-операций (транзакций).

Классическая SMTP-сессия включает в себя следующие этапы:

1. Инициирование соединения. Клиент создает соединение с сервером. Сервер отвечает клиенту сообщением с кодом отклика 220 в случае готовности для продолжения работы или с кодом отклика 554 в случае отказа в открытии SMTP-сессии.

2. Инициирование работы с клиентом. Клиент передает команду EHLO (HELO). Сервер отправляет сообщение с кодом отклика 250. Если была отправлена команда EHLO, сервер также в сообщении возвращает список расширений, который он поддерживает. Сервер также может вернуть сообщение с кодом отклика 501, если не было передано в аргументах команды имя клиента (может быть любым, но обычно проверяется для борьбы со спамом с помощью PTR-записи).

3. SMTP-транзакция (в процессе SMTP-сессии их может быть несколько).

4. Завершение сессии. Если клиент желает завершить работу с сервером, то он посылает команду QUIT, на которую сервер отвечает сообщением с кодом отклика 221 и закрывает соединение. По данной команде можно определить, что клиент также должен освободить ресурсы под выделенное соединение.

Любая SMTP-транзакция представляет собой три последовательные этапа команда/ответ:

1. MAIL from. Определяет обратный адрес. Эта переменная необходима для возвращенных писем. Сервер в случае принятия данных отвечает сообщением с кодом отклика 250 в случае успеха. Но может также ответить с кодом отклика 501 (синтаксическая ошибка).

2. RCPT to. Определяет получателя текущего текстового сообщения. Команда может использоваться несколько раз, в зависимости от количества получателей. Сервер в случае принятия данных отвечает сообщением с кодом отклика 250 в случае успеха. Сообщение с кодом отклика 501 возвращается в случае синтаксической ошибки, а с кодом отклика 503 в случае невозможности принятия данных на данном этапе, с кодом отклика 550 - если пользователь не найден на сервере.

3. DATA. Определяется для последовательной отправки текстового сообщения. Включает в себя непосредственно содержимое письма, в отличие от оболочки. "DATA"несет в себе информацию о заголовке и теле сообщения (они разделяются пустой строкой). Ответ от сервера при передаче происходит в два этапа: на первом от отвечает конкретно на команду "DATA"(уведомление о готовности принять текстовое сообщения), а на втором - о принятии или отклонении всего письма в конце последовательности данных. После выполнения команды DATA сервер возвращает в случае успеха сообщение с кодом 354, что означает, что сервер готов принимать содержимое письма, на все последующие данные сервер не будет отвечать до того момента, пока не встретит последовательность из <CRLF>.<CRLF>. В случае успеха получения сервером всего текста письма он отвечает сообщением с кодом отклика 250.

Стоит отметить, что SMTP-сервер может разоравать SMTP-сессию в случае истечения времени ожидания. Тогда он вернет сообщение с кодом отклика 421, что свидетельствует о том, что сервер разорвал соединение, и клиент должен освободить ресурсы, выделенные под SMTP-сессию.

Любой SMTP-сервер выполняет несколько функций. Одной из них является проверка правильности настроек и выдача разрешения компьютеру, который пытается отправить электронное письма. Другая функция состоит в отправке исходящих писем на указанный адрес с последующей проверкой доставки. В качестве задачи, которую необходимо решить в рамках данной курсовой работы, является реализация функции отправки исходящих писем, т.е. реализация SMTP-клиента как части MTA.

1.3 Архитектура взаимодействия клиента с сервером

В процессе курсовой работы требовалось разработать SMTP-клиент как часть MTA для организации удаленной доставки письма после его получения и записи в каталог MAILDIR. Организация удаленной доставки может быть обеспечена следующим образом:

1. Определение имени сервера по пути сервера, в каталоге в MAILDIR которого находятся письма, для определения того, куда необходимо передавать доступные для удаленной доставки письма.
2. Открытие необходимого соединения SMTP-клиента с SMTP-сервером для последующей сессии.
3. Начало сессии и инициирование общения с ним с помощью команды HELO.
4. Считывание данных письма и последующий анализ дополнительных заголовков необходимых для последующей отправки.
5. Передача адреса почтового ящика отправителя в MAIL. Данный адрес считывается из дополнительных заголовков письма.
6. Попытка передачи всех возможных адресов получателей, которые были считаны из текста файла, содержащего текст письма.
7. Проверка корректности переданных данных с помощью команды DATA, так как в случае успеха данная команда вернет сообщение с кодом отклика 354.
8. Передача текста письма.
9. Передача флага текста письма и получение сообщения с откликом.
10. В случае успеха файл доставленного сообщения может быть удален (и удаляется). В случае неудачи, на каком-то из шагов удаление сообщения происходит при анализе ошибки. Если код ошибки 4xx, то в большинстве случаев это ошибка не связана с работой сервера и, желательно, повторить отправку. Если же код ошибки 5xx, то в большинстве случаев это означает ошибку, которую совершает клиент и сервер в любом случае не примет данное сообщение, следует удалить файл с текстом письма.

11. Если есть еще сообщения, то можем продолжить отправку, иначе закрыть соединение и завершить сессию до появления новых сообщений.

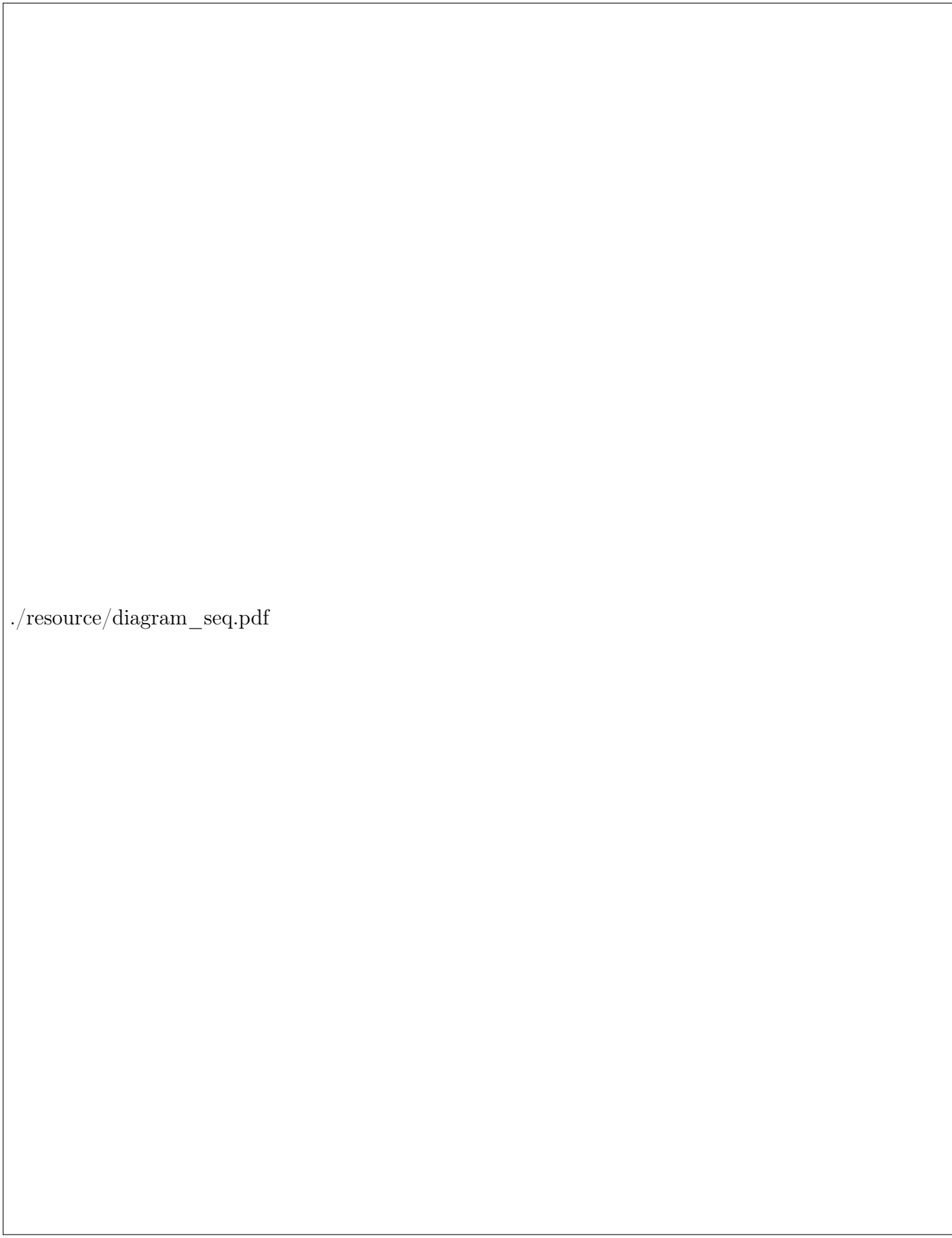


Рис. 1.1 Диграмма последовательности взаимодействия клиента с сервером

От SMTP-клиента как части MTA требуется способность работать с множеством исходящих подключений к удаленным SMTP-серверам. Для решения данной проблемы можно воспользоваться несколькими способами:

1. Один клиент - один поток (процесс). Одним из решений задачи обслуживания нескольких исходящих подключений является использование многопоточных возможностей ОС: можно создать отдельный поток для обслуживания одного клиента. Данный многопоточный подход имеет ряд недостатков. Создание нового потока - сравнительно затратная операция для ОС. В периоды пиковой нагрузки на сервер большое число одновременно запущенных потоков может исчерпать ресурсы системы, а непрерывное создание, переключение и завершение потоков только ухудшит ситуацию.

2. Пул потоков. Для решения проблемы траты ресурсов часто применяют пул заранее созданных потоков ограниченного размера, что в принципе при огромном количестве одновременных подключений не решает проблему.

3. Мультиплексирование при работе с сокетами: select, poll, pselect, epoll. Если потоки или процессы сервера большую часть времени проводят в ожидании сообщений от клиента или вспомогательных служб, а не занимаются сложными вычислениями, пользы от многопоточности будет мало. Альтернативой в этом случае является ожидание событий на множестве сокетов в одном потоке исполнения. Для этого используются функции select, poll, pselect или epoll. Общая идея такова: в функцию передается массив дескрипторов сокетов. Функция блокирует выполнение до тех пор, пока один из сокетов в массиве не станет доступен для неблокирующего чтения (т.е. поступили новые данные или новое соединение) или записи (освободился буфер отправки), либо пока не истечет указанный таймаут. Все взаимодействие с сокетами осуществляется в одном потоке исполнения, что снимает необходимость синхронизации доступа к разделяемым ресурсам.

В процессе выполнения курсовой работы требовалось разработать SMTP-клиент таким образом, чтобы он обрабатывал несколько исходящих соединений в одном потоке исполнения и при этом поддерживал многопоточную обработку. Требуется воспользоваться вызовом pselect.

Вызов pselect почти ничем не отличается от вызова select. Вызов pselect имеет аргумент, который содержит набор сигналов, которые ядро должно разблокировать (то есть удалить из маски сигналов, вызывающего потока) на время, пока вызывающий поток заблокирован в вызове pselect. В остальном он полностью схож с вызовом select, который имеет ряд недостатков:

- Вызов select (pselect) модифицирует передаваемые ему структуры fdsets, так что ни одну из них нельзя переиспользовать. Даже если, например, получив порцию данных, а требуется получить еще, структуры fdsets необходимо переинициализировать. Ну или копировать из заранее сохраненного бэкапа.

- Для выяснения того, какой именно дескриптор сгенерировал событие, требуется вручную опросить их все.
- Максимальное количество одновременно наблюдаемых дескрипторов ограниченно константной и может быть ограничено в 1024 дескриптора.
- Невозможно работать с дескрипторами из наблюдаемого набора из другого потока.

Исходя из вышеперечисленных недостатков можно сделать вывод, что следует учесть в проектировании SMTP-клиента максимальное количество допустимых дескрипторов.

Глава 2

Конструкторский раздел

2.1 Разработка SMTP-клиента

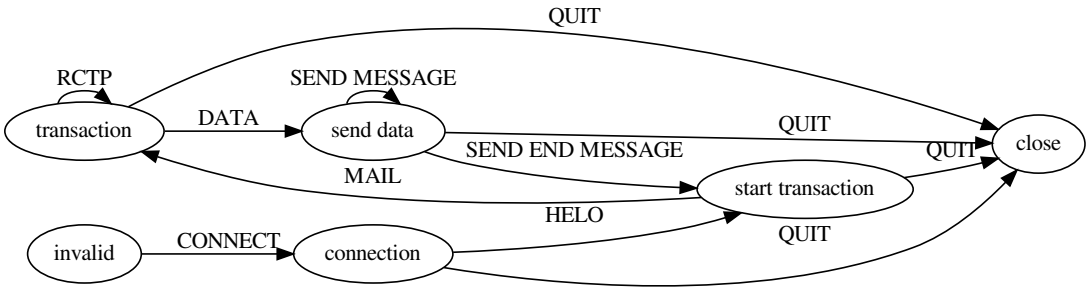


Рис. 2.1 Конечный автомат протокола SMTP для клиента

Каждая SMTP-сессия представляет собой совокупность состояний (т.е. что требуется сделать в данный момент или что будет требоваться далее) и множество переходов между этими состояниями (в данном случае это определяется с помощью команд при работе с SMTP-сервером). В связи с этим допустимо построить конечный автомат для протокола SMTP.

Конечный автомат - это математическая абстракция, которая состоит из трех основных элементов: множества внутренних состояний, множества входных сигналов, которые определяют переход из текущего состояния в следующее, множество конечных состояний, при переходе в которые автомат завершает работу.

Конечный автомат, представленный на рисунке 2.1, описывает состояния, изображенные в виде овалов, и переходы, изображенные на рисунке в виде дуг графа. Данный конечный автомат построен при помощи утилиты autofsm.

Были выделены следующие состояния конечного автомата:

1. invalid - данное состояние определяет, что клиент находится в невалидном для обмена данными состоянии, т.е. соединение с сервером не установлено, сокет не был открыт.

2. connection - данное состояние определяет, что клиент успешно подключился к серверу, и может выполнять доступные ему команды, в данном случае SMTP-клиент может выполнить только 2 команды HELO и QUIT. Выполнение команды QUIT происходит только тогда, когда SMTP-сервер ответил клиенту сообщением с кодом отклика, означающего ошибку, тогда клиент инициирует закрытие сокета.

3. start transaction - после выполнения команды HELO, когда клиент находится в состоянии connection, он переходит в состояние start transaction, которое обозначает, что можно начинать выполнять SMTP-транзакцию (т.е. подготовку для передачи данных и непосредственно передачу письма SMTP-серверу). Подготовка к передаче данных осуществляется путем выполнения команд MAIL, RCTP, DATA. Переход на следующую стадию подготовки к передаче производится путем выполнения команды MAIL, если сервер при ее выполнении возвращает сообщение с кодом отклика об успехе, иначе на данном этапе клиент произведен завершение сессии путем перехода в состояние close с помощью команды QUIT.

4. transaction - данное состояние определяет то, что MAIL прошел успешно и теперь требуется отправить адреса электронных ящиков получателей сообщения. Производится это путем выполнения команды RCTP. Выполнение команды QUIT производится на данной стадии при возникновении критических ошибок, которые вернул сервер в сообщении (например, таймаут или синтаксическая ошибка в команде). В случае сообщения об отсутствии адреса электронного ящика на сервере клиент не прекратит попытки отправки по причине того, что другие электронные ящики могут присутствовать на сервер и это сообщение будет доставлено кому-нибудь. Переход из данного состояние в следующее производится при помощи выполнения команды DATA и тогда, когда все электронные ящики были переданы. Если ни один из переданных адресов не был принят, сервер возвращает ошибку, которая приводит к инициированию закрытия соединения с помощью команды QUIT и переход в состояние close.

5. send data - данное состояние описывает процесс принятия данных сообщения. Переход, обозначающих отправку данных на рисунке обозначен в виде SEND MESSAGE. Части сообщения будут приниматься пока клиенту есть, что передавать. Сервер не отвечает на каждое успешно переданную часть сообщения. В случае каких-либо критических ошибок клиент инициирует закрытие соединения с помощью выполнения команды QUIT. Переход в следующее состояние производится путем передачи флага завершения сообщения CRLF.CRLF, данный флаг представлен на рисунке переходом в виде SEND END MESSAGE. В случае успешно переданного сообщения клиент переходит в состояние start transaction, иначе будет инициировано закрытие соединения с помощью выполнения команды QUIT.

2.2 Описание формата хранения писем в файловой системе

Для хранения текстов писем и дополнительной информации, которую принимает сервер во время SMTP-транзакции, используется файловая система. В разрабатываемой системе используется модифицированный Maildir.

Maildir - распространенный формат хранения электронной почты, нетребующий монопольного захвата файла для обеспечения целостности почтового ящика при чтении, добавлении или изменении сообщения. Каждое сообщение хранится в отдельном файле с уникальным именем, а каждая папка представляет собой каталог. Вопросами блокировки файлов при добавлении, перемещении и удалении файлов занимается локальная файловая система. Все изменения делаются при помощи атомарных файловых операций, таким образом, монопольный захват файла ни в каком случае не нужен.

В случае стандартного Maildir структура каталогов следующая:

```
-/
    -home
        -user1
            -Maildir
                -cur
                -new
                -tmp
        -user2
            -Maildir
                -cur
                -new
                -tmp
```

В связи с некоторыми требованиями по использованию стандартного Maildir, таким как создание множества пользователей на устройстве, где запущен SMTP-сервер, было решено модифицировать структуру Maildir таким образом, чтобы он содержал письма всех пользователей в едином каталоге. Поскольку также Maildir предназначен для доставки только локальной почты, а по условию требуется пересылать также удаленную почту, был добавлена новая директория для удаленной почты, где содержатся папки с почтой, предназначенной для удаленных SMTP-сервером. Структура каталогов модифицированного Maildir:

```
- maildir
    -user1
        -cur
        -tmp
        -new
    -user2
```

```
        -cur
        -tmp
        -new
    .OTHER_SERVERS
        -domain_server1
            -tmp
            -letter1
            -letter2
        -domain_server2
            -tmp
            -letter3
            -letter4
```

- maildir - корневой каталог Maildir.
- user1, user2 - имена и каталоги пользователей SMTP-сервера, работающего на данном устройстве.
- cur - папка, содержащая прочитанную почту пользователем.
- tmp - папка, содержащая письма на стадии доставки, запись в файл не является атомарной операцией, поэтому пока производится запись в файл отправка этих данных недопустима.
- new - папка, содержащая файлы писем, которые готовы к отправке.
- .OTHER_SERVERS - каталог, содержащий папки с письмами для удаленных SMTP-серверов.
- domain_server1, domain_server2 - каталоги, содержащие письма для конкретных удаленных сервером с доменным именем по имени данного каталога.
- letter1, letter2, letter3, letter4 - письма удаленной почты готовые для отправки, содержатся сразу в каталогах, предназначенных для удаленных SMTP-серверов.

Для того, чтобы SMTP-клиент определил от кого и кому доставляется письмо, SMTP-сервер кладет дополнительную информацию в виде заголовков в файл письма, предназначенного для удаленного SMTP-сервера.

Формат заголовков в файле письма:

```
X-Postman-From: <mailbox>
X-Postman-Date: <timestamp>
X-Postman-To: <mailbox> [, <mailbox> [...]]
<пустая строка (\r\n)>
<тело письма полученное во время почтовой транзакции>
```

SMTP-клиенту как части MTA достаточно читать письма из каталога .OTHER_SERVERS, а также удалять их из директории.

Работа с Maildir реализовано в файле maildir.

2.3 Основной процесс работы SMTP-клиента

Работа программы начинается с того, что запускается подсистема логгирования, которая представляет собой отдельный поток и очередь сообщений. В процессе выполнения потока исполнения, предназначенного для логгирования, выполняется получение сообщения из очереди и печать его на экран. Файл, в котором описаны все функции для работы с подсистемой логгирования, представлен в файле logs.c.

Далее производится загрузка конфигурации SMTP-клиента. Конфигурация представляет из себя параметры, которые позволяют регулировать количество потоков исполнения (application.threads, в которых производится мультиплексирование при работе с дескрипторами, идентифицирующими сокетное соединение. Полный путь до Maildir настраивается с помощью параметра application.maildir.path согласно конфигурации SMTP-сервера. Конфигурация позволяет включать режим отладки с помощью параметра application.debug, который активирует (или деактивирует) печать логов типа LOG_DEBUG. Также конфигурация позволяет настраивать доменное имя текущего сервера с помощью параметра hostname для того, чтобы удаленный SMTP-сервер мог проверять PTR-запись сервера с целью фильтрации спама, в этом случае данная проверка не позволяет нам отправлять корректные сообщения, добавление этого поля позволит настроить DNS-записи так, чтобы сервер определялся корректно. Параметр server_port является дополнительным и введен с целью выполнения подключения к локальном серверу для выполнения тестирования SMTP-клиента. Чтение конфигурационного файла реализовано посредством бибилотеки libconfig. В файле config.c функция loading_config реализует логику для заполнения глобальной структуры config_context, содержащая все необходимые конфигурационные данные для работы SMTP-клиента.

```
application: {
    threads: 10;
    maildir: {
        path: "/home/ubuntu/maildir"
    };
    debug: 1;
    hostname: "postman.local";
    server_port: "8081";
}
```

Если загрузка конфигурации произошла с ошибкой, то произойдет завершение работы программы путем освобождения ресурсов уже отданных под конфигурацию и остановка подсистемы логгирования.

После загрузки конфигурации производится инициализация обработчика сигналов в файле signal_handler.c в функции init_signals_handler. В данном случае регистрируется только обработчик на завершение работы программы для того, чтобы безопасно освободить все занятые ресурсы программой.

Далее производится инициализация основного контекста работы программы в файле context.c в функции init_context. На данном шаге создаются потоки исполнения и выделяются основные ресурсы для выполнения мультиплексирования, чтения Maildir и отправки письма.

Вся основная логика, реализующая конечный автомат находится в файле context.c. Процесс отправки писем производится в функции start$_thread.pselect$..switch case.$SMTP-$ smtp.c, network.c, .

Структура проекта была построена на базе Makefile с помощью утилит make2graph и dot и представлена на рис. 2.2. Схема структуры проекта содержит связи между компонентами при сборке проекта.

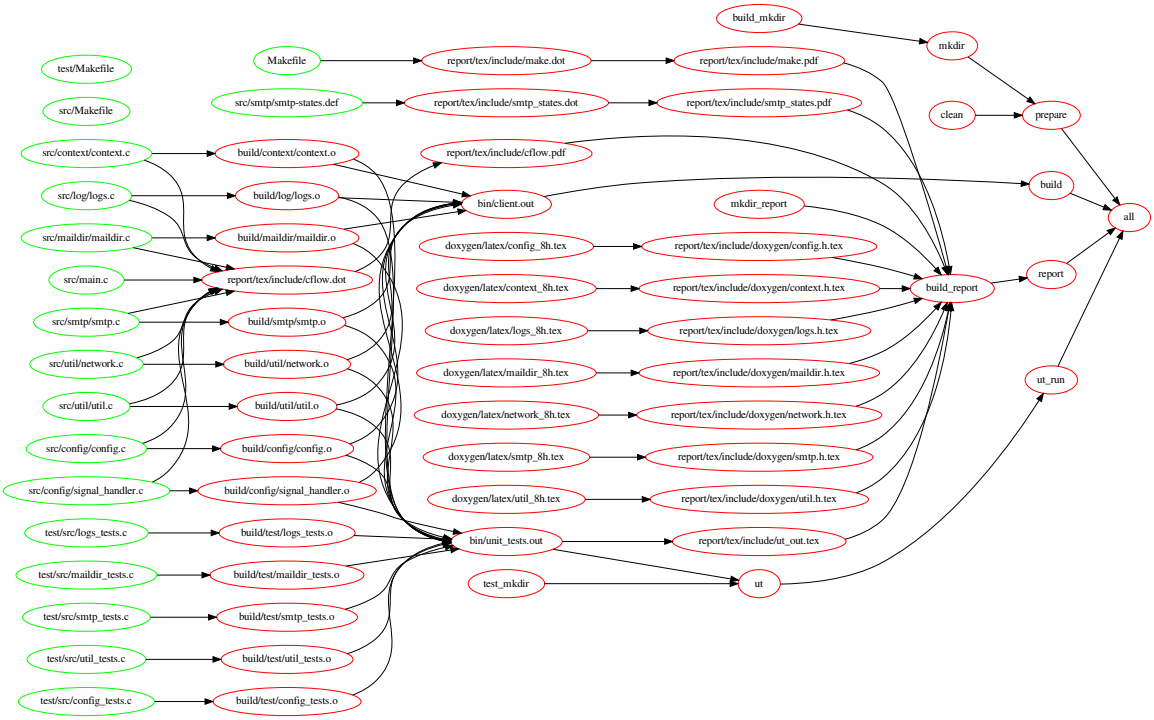
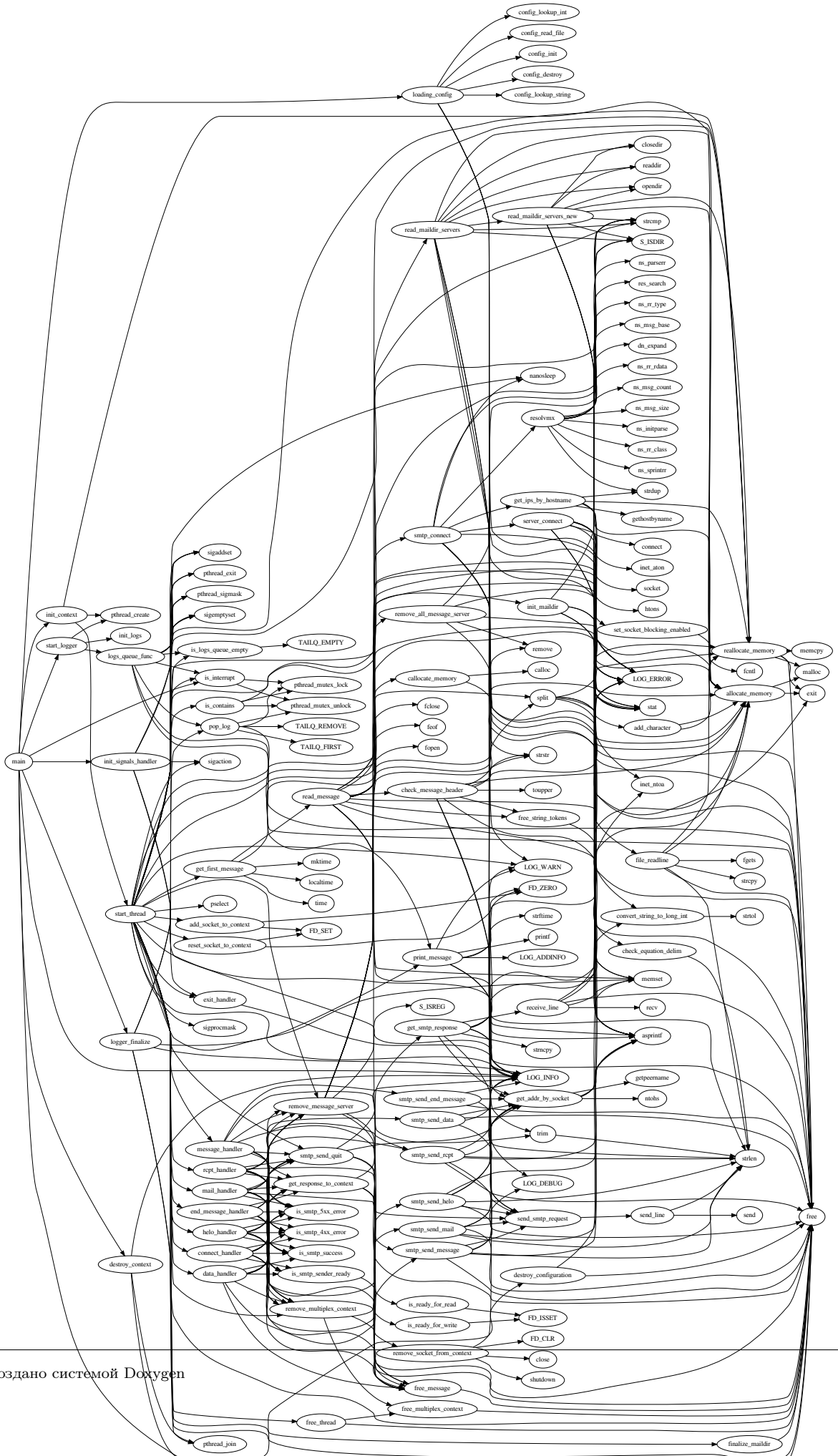


Рис. 2.2 Структура проекта.

Также с помощью утилиты cflow был построен граф вызовов функций (рис. 2.3), который отражает все взаимосвязи между модулями проекта.


config_lookup_int
config_read_file
config_init
config_destroy
config_lookup_string
loading_config
closedir
readdir
opendir
read_maildir_servers_new
read_maildir_servers
strcmp
S_ISDIR
ns_parserr
res_search
ns_rr_type
ns_msg_base
dn_expand
ns_rr_rdata
nanosleep
ns_msg_count
ns_msg_size
resolvmx
ns_initparse
ns_rr_class
ns_sprintrr
strdup
get_ips_by_hostname
gethostbyname
server_connect
smtp_connect
connect
inet_aton
socket
htons
sigaddset
pthread_exit
pthread_sigmask
sigemptyset
init_maildir
remove_all_message_server
set_socket_blocking_enabled
init_context
pthread_create
init_logs
start_logger
logs_queue_func
is_logs_queue_empty
TAILQ_EMPTY
remove
calloc
reallocate_memory
memcpy
fcntl
malloc
LOG_ERROR
allocate_memory
exit
is_interrupt
pthread_mutex_lock
callocate_memory
split
stat
is_contains
pthread_mutex_unlock
fclose
add_character
pop_log
TAILQ_REMOVE
feof
TAILQ_FIRST
fopen
main
init_signals_handler
sigaction
strstr
inet_ntoa
check_message_header
toupper
read_message
free_string_tokens
file_readline
fgets
strcpy
get_first_message
mktime
LOG_WARN
localtime
FD_ZERO
time
pselect
start_thread
strftime
convert_string_to_long_int
strtol
add_socket_to_context
FD_SET
printf
reset_socket_to_context
check_equation_delim
print_message
LOG_ADDINFO
memset
exit_handler
S_ISREG
receive_line
recv
sigprocmask
get_smtp_response
asprintf
logger_finalize
strncpy
LOG_INFO
getpeername
get_addr_by_socket
ntohs
remove_message_server
smtp_send_end_message
smtp_send_data
trim
message_handler
smtp_send_quit
smtp_send_rcpt
strlen
rcpt_handler
LOG_DEBUG
get_response_to_context
mail_handler
smtp_send_helo
end_message_handler
is_smtp_5xx_error
send_smtp_request
send_line
send
free
helo_handler
is_smtp_4xx_error
smtp_send_mail
connect_handler
is_smtp_success
smtp_send_message
destroy_context
data_handler
is_smtp_sender_ready
destroy_configuration
remove_multiplex_context
is_ready_for_read
is_ready_for_write
FD_ISSET
FD_CLR
remove_socket_from_context
close
shutdown
free_message
free_multiplex_context
free_thread
pthread_join
finalize_maildir

2.4 Тестирование

Для тестирования отдельных модулей сервера, было написаны unit-тесты с использованием библиотеки cunit. Результат работы тестиованяи представлен в листинге

```
    CUnit - A unit testing framework for C - Version 2.1-3
    http://cunit.sourceforge.net/


Suite: UtilTests
  Test: convert string to long int ...Тест конвертирования строки
passed
  Test: split string ...Тест получения токенов по строке
passed
  Test: trim string ...Тест удаления пробелов из строки
passed
Suite: SMTPTests
  Test: SMTP connect ...passed
  Test: SMTP send helo ...passed
  Test: SMTP send mail ...passed
  Test: SMTP send rcpt ...passed
  Test: SMTP send data ...passed
  Test: SMTP send message ...passed
  Test: SMTP send end message ...passed
  Test: SMTP send end message ...passed
  Test: SMTP send end message ...passed
  Test: SMTP send end message ...passed
  Test: SMTP send end message ...passed
  Test: SMTP send end message ...passed
  Test: SMTP send end message ...passed
Suite: LogsTests
  Test: log debug ...passed
  Test: log info ...passed
  Test: log error ...passed
  Test: log warn ...passed
  Test: log warn ...passed
Suite: ConfigTests
  Test: loading config ...passed
  Test: destroy config ...passed
Suite: MaildirTests
  Test: maildir init ...passed
  Test: update maildir ...FAILED
    1. test/src/maildir_tests.c:32  - CU_ASSERT_EQUAL(maildir->users_size,3)
    2. test/src/maildir_tests.c:34  - CU_ASSERT_EQUAL(maildir->users[0].messages_size,2)
```

```
  3. test/src/maildir_tests.c:37  - CU_ASSERT_EQUAL(maildir->servers[0].messages_size,2)
  4. test/src/maildir_tests.c:38  - CU_ASSERT_EQUAL(maildir->servers[1].messages_size,1)
Test: read maildir servers ...FAILED
  1. test/src/maildir_tests.c:55  - CU_ASSERT_EQUAL(maildir->servers[0].messages_size,2)
  2. test/src/maildir_tests.c:56  - CU_ASSERT_EQUAL(maildir->servers[1].messages_size,1)
Test: get first message ...FAILED
  1. test/src/maildir_tests.c:70  - CU_ASSERT_EQUAL(maildir->servers[0].messages_size,2)
Test: read message ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      5      5    n/a      0        0
               tests     28     28     25      3        0
             asserts     79     79     72      7      n/a

Elapsed time =    0.104 seconds

==18015== Memcheck, a memory error detector
==18015== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==18015== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==18015== Command: bin/unit_tests.out
==18015== Parent PID: 18014
==18015==
==18015==
==18015== HEAP SUMMARY:
==18015==     in use at exit: 0 bytes in 0 blocks
==18015==   total heap usage: 1,013 allocs, 1,013 frees, 547,475 bytes allocated
==18015==
==18015== All heap blocks were freed -- no leaks are possible
==18015==
==18015== For counts of detected and suppressed errors, rerun with: -v
==18015== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Так же было реализовано системное тестирование, на языке программирования python.

2.5 Основные функции программы

Данный раздел создан с помощью программы doxygen.

2.5.1 Файл include/config/config.h

```
#include <libconfig.h>
#include <string.h>
#include <stdbool.h>
#include <signal.h>
```

Классы

- struct maildir_config
- struct config

Макросы

- #define APPLICATION_CONFIG "resources/application.cfg"

Определения типов

- typedef struct maildir_config maildir_config
- typedef struct config config

Функции

- bool loading_config ()
- int init_signals_handler ()
- void destroy_configuration ()

Переменные

- config config_context

2.5.1.1 Макросы

2.5.1.1.1 APPLICATION_CONFIG

#define APPLICATION_CONFIG "resources/application.cfg"

2.5.1.2 Типы

2.5.1.2.1 config

typedef struct config config

2.5.1.2.2 maildir_config

typedef struct maildir_config maildir_config

2.5.1.3 Функции

2.5.1.3.1 destroy_configuration()

void destroy_configuration ( )

2.5.1.3.2 init_signals_handler()

int init_signals_handler ( )

2.5.1.3.3 loading_config()

bool loading_config ( )

2.5.1.4 Переменные

2.5.1.4.1 config_context

config config_context

2.5.2 Файл include/context/context.h

```
#include <stdlib.h>
#include <pthread.h>
#include <stdbool.h>
#include <memory.h>
#include <signal.h>
#include <sys/socket.h>
#include <sys/queue.h>
#include <unistd.h>
#include "smtp/smtp.h"
#include "maildir/maildir.h"
```

Классы

- struct multiplex_context
- struct thread
- struct context

Определения типов

- typedef struct multiplex_context multiplex_context
- typedef struct thread thread
- typedef struct context context

Функции

- int init_context ()
- int add_socket_to_context (int socket, thread *thr)
- int remove_socket_from_context (int socket, thread *thr)
- int reset_socket_to_context (thread *thr)
- void exit_handler (int sig)
- bool is_ready_for_read (int socket, thread *thr)
- bool is_ready_for_write (int socket, thread *thr)
- void destroy_context ()

Переменные

- context app_context

2.5.2.1 Типы

2.5.2.1.1 context

typedef struct context context

2.5.2.1.2 multiplex_context

typedef struct multiplex_context multiplex_context

2.5.2.1.3 thread

typedef struct thread thread

2.5.2.2 Функции

2.5.2.2.1 add_socket_to_context()

```
int add_socket_to_context (
            int socket,
            thread * thr )
```

2.5.2.2.2 destroy_context()

void destroy_context ( )

2.5.2.2.3 exit_handler()

```
void exit_handler (
            int sig )
```

2.5.2.2.4 init_context()

int init_context ( )

2.5.2.2.5 is_ready_for_read()

bool is_ready_for_read (
int socket,
thread * thr)

2.5.2.2.6 is_ready_for_write()

bool is_ready_for_write (
int socket,
thread * thr)

2.5.2.2.7 remove_socket_from_context()

int remove_socket_from_context (
int socket,
thread * thr)

2.5.2.2.8 reset_socket_to_context()

int reset_socket_to_context (
thread * thr)

2.5.2.3 Переменные

2.5.2.3.1 app_context

context app_context

2.5.3 Файл include/log/logs.h

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <pthread.h>
#include <sys/queue.h>
#include <time.h>
```

Классы

- struct log_message

Макросы

- #define COLOR_RESET "\x1b[0m"
- #define COLOR_BRIGHT "\x1b[1m"
- #define COLOR_DIM "\x1b[2m"
- #define COLOR_UNDERSCORE "\x1b[4m"
- #define COLOR_BLINK "\x1b[5m"
- #define COLOR_REVERSE "\x1b[7m"
- #define COLOR_HIDDEN "\x1b[8m"
- #define COLOR_BLACK "\x1b[30m"
- #define COLOR_RED "\x1b[31m"
- #define COLOR_GREEN "\x1b[32m"
- #define COLOR_YELLOW "\x1b[33m"
- #define COLOR_BLUE "\x1b[34m"
- #define COLOR_MAGENTA "\x1b[35m"
- #define COLOR_CYAN "\x1b[36m"
- #define COLOR_WHITE "\x1b[37m"
- #define BACKGROUND_BLACK "\x1b[40m"
- #define BACKGROUND_RED "\x1b[41m"
- #define BACKGROUND_GREEN "\x1b[42m"
- #define BACKGROUND_YELLOW "\x1b[43m"
- #define BACKGROUND_BLUE "\x1b[44m"
- #define BACKGROUND_MAGENTA "\x1b[45m"
- #define BACKGROUND_CYAN "\x1b[46m"
- #define BACKGROUND_WHITE "\x1b[47m"
- #define LOG_DEBUG(message, args...) log_debug(message, __FILE__, __L↩ INE__, args)
- #define LOG_INFO(message, args...) log_info(message, __FILE__, __LINE↩ __, args)

- #define LOG_ERROR(message, args...) log_error(message, __FILE__, __LI↩ NE__, args)
- #define LOG_WARN(message, args...) log_warn(message, __FILE__, __LI↩ NE__, args)
- #define LOG_ADDINFO(message, args...) log_addinfo(message, args)

Определения типов

- typedef enum log_type log_type
- typedef struct log_message log_message

Перечисления

- enum log_type {
  LOG_ERROR, LOG_INFO, LOG_DEBUG, LOG_WARN,
  LOG_ADDINFO }

Функции

- void init_logs ()
- void push_log (log_message *value)
- log_message * pop_log ()
- void log_debug (char *message, char *filename, int line,...)
- void log_info (char *message, char *filename, int line,...)
- void log_error (char *message, char *filename, int line,...)
- void log_warn (char *message, char *filename, int line,...)
- void log_addinfo (char *message,...)
- void start_logger ()
- void logger_finalize ()

Переменные

- int interrupt_thread_local

2.5.3.1 Макросы

2.5.3.1.1 BACKGROUND_BLACK

#define BACKGROUND_BLACK "\x1b[40m"

2.5.3.1.2 BACKGROUND_BLUE

#define BACKGROUND_BLUE "\x1b[44m"

2.5.3.1.3 BACKGROUND_CYAN

#define BACKGROUND_CYAN "\x1b[46m"

2.5.3.1.4 BACKGROUND_GREEN

#define BACKGROUND_GREEN "\x1b[42m"

2.5.3.1.5 BACKGROUND_MAGENTA

#define BACKGROUND_MAGENTA "\x1b[45m"

2.5.3.1.6 BACKGROUND_RED

#define BACKGROUND_RED "\x1b[41m"

2.5.3.1.7 BACKGROUND_WHITE

#define BACKGROUND_WHITE "\x1b[47m"

2.5.3.1.8 BACKGROUND_YELLOW

#define BACKGROUND_YELLOW "\x1b[43m"

2.5.3.1.9 COLOR_BLACK

#define COLOR_BLACK "\x1b[30m"

2.5.3.1.10 COLOR_BLINK

#define COLOR_BLINK "\x1b[5m"

2.5.3.1.11 COLOR_BLUE

#define COLOR_BLUE "\x1b[34m"

2.5.3.1.12 COLOR_BRIGHT

#define COLOR_BRIGHT "\x1b[1m"

2.5.3.1.13 COLOR_CYAN

#define COLOR_CYAN "\x1b[36m"

2.5.3.1.14 COLOR_DIM

#define COLOR_DIM "\x1b[2m"

2.5.3.1.15 COLOR_GREEN

#define COLOR_GREEN "\x1b[32m"

2.5.3.1.16 COLOR_HIDDEN

#define COLOR_HIDDEN "\x1b[8m"

2.5.3.1.17 COLOR_MAGENTA

#define COLOR_MAGENTA "\x1b[35m"

2.5.3.1.18 COLOR_RED

#define COLOR_RED "\x1b[31m"

2.5.3.1.19 COLOR_RESET

#define COLOR_RESET "\x1b[0m"

2.5.3.1.20 COLOR_REVERSE

#define COLOR_REVERSE "\x1b[7m"

2.5.3.1.21 COLOR_UNDERSCORE

#define COLOR_UNDERSCORE "\x1b[4m"

2.5.3.1.22 COLOR_WHITE

```
#define COLOR_WHITE "\x1b[37m"
```

2.5.3.1.23 COLOR_YELLOW

```
#define COLOR_YELLOW "\x1b[33m"
```

2.5.3.1.24 LOG_ADDINFO

```
#define LOG_ADDINFO(
                message,
                args... ) log_addinfo(message, args)
```

2.5.3.1.25 LOG_DEBUG

```
#define LOG_DEBUG(
                message,
                args... ) log_debug(message, __FILE__, __LINE__, args)
```

2.5.3.1.26 LOG_ERROR

```
#define LOG_ERROR(
                message,
                args... ) log_error(message, __FILE__, __LINE__, args)
```

2.5.3.1.27 LOG_INFO

```
#define LOG_INFO(
                message,
                args... ) log_info(message, __FILE__, __LINE__, args)
```

2.5.3.1.28 LOG_WARN

#define LOG_WARN(
message,
args...) log_warn(message, __FILE__, __LINE__, args)

2.5.3.2 Типы

2.5.3.2.1 log_message

typedef struct log_message log_message

2.5.3.2.2 log_type

typedef enum log_type log_type

2.5.3.3 Перечисления

2.5.3.3.1 log_type

enum log_type

Элементы перечислений

LOG_ERROR	
LOG_INFO	
LOG_DEBUG	
LOG_WARN	
LOG_ADDINFO	

2.5.3.4 Функции

2.5.3.4.1 init_logs()

```
void init_logs ( )
```

2.5.3.4.2 log_addinfo()

```
void log_addinfo (
            char * message,
             ... )
```

2.5.3.4.3 log_debug()

```
void log_debug (
            char * message,
            char * filename,
            int line,
             ... )
```

2.5.3.4.4 log_error()

```
void log_error (
            char * message,
            char * filename,
            int line,
             ... )
```

2.5.3.4.5 log_info()

```
void log_info (
            char * message,
            char * filename,
            int line,
             ... )
```

2.5.3.4.6 log_warn()

```
void log_warn (
            char * message,
            char * filename,
            int line,
             ... )
```

2.5.3.4.7 logger_finalize()

```
void logger_finalize ( )
```

2.5.3.4.8 pop_log()

```
log_message* pop_log ( )
```

2.5.3.4.9 push_log()

```
void push_log (
            log_message * value )
```

2.5.3.4.10 start_logger()

```
void start_logger ( )
```

2.5.3.5 Переменные

2.5.3.5.1 interrupt_thread_local

```
int interrupt_thread_local
```

2.5.4 Файл include/maildir/maildir.h

#include <stdio.h>

Классы

- struct message
- struct maildir_user
- struct maildir_other_server
- struct maildir_main

Определения типов

- typedef struct message message
- typedef struct maildir_user maildir_user
- typedef struct maildir_other_server maildir_other_server
- typedef struct maildir_main maildir_main

Функции

- maildir_main * init_maildir (char *directory)
- void update_maildir (maildir_main *maildir)
- void read_maildir_servers (maildir_main *maildir)
- void output_maildir (maildir_main *maildir)
- message * get_first_message (maildir_other_server *server)
- message * read_message (char *filepath)
- void finalize_maildir (maildir_main *maildir)
- void free_message (message *mess)
- void remove_all_message_server (maildir_other_server *server)
- void remove_message_server (maildir_other_server *server, message *mess)

2.5.4.1 Типы

2.5.4.1.1 maildir_main

typedef struct maildir_main maildir_main

2.5.4.1.2 maildir_other_server

typedef struct maildir_other_server maildir_other_server

2.5.4.1.3 maildir_user

typedef struct maildir_user maildir_user

2.5.4.1.4 message

typedef struct message message

2.5.4.2 Функции

2.5.4.2.1 finalize_maildir()

void finalize_maildir (
maildir_main * maildir)

2.5.4.2.2 free_message()

void free_message (
message * mess)

2.5.4.2.3 get_first_message()

message* get_first_message (
maildir_other_server * server)

2.5.4.2.4 init_maildir()

```
maildir_main* init_maildir (
        char * directory )
```

2.5.4.2.5 output_maildir()

```
void output_maildir (
        maildir_main * maildir )
```

2.5.4.2.6 read_maildir_servers()

```
void read_maildir_servers (
        maildir_main * maildir )
```

2.5.4.2.7 read_message()

```
message* read_message (
        char * filepath )
```

2.5.4.2.8 remove_all_message_server()

```
void remove_all_message_server (
        maildir_other_server * server )
```

2.5.4.2.9 remove_message_server()

```
void remove_message_server (
        maildir_other_server * server,
        message * mess )
```

2.5.4.2.10 update_maildir()

```
void update_maildir (
            maildir_main * maildir )
```

2.5.5 Файл include/smtp/smtp.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <time.h>
#include <fcntl.h>
#include <errno.h>
#include <netdb.h>
#include <stdbool.h>
```

Классы

- struct smtp_address
- struct smtp_header
- struct smtp_context
- struct smtp_response

Определения типов

- typedef enum smtp_state_code state_code
- typedef enum smtp_status_code status_code
- typedef struct smtp_address smtp_addr
- typedef struct smtp_header smtp_header
- typedef struct smtp_context smtp_context
- typedef struct smtp_response smtp_response

Перечисления

- enum smtp_state_code {
  SMTP_CONNECT = 0, SMTP_HELO = 1, SMTP_EHLO = 2, SMTP_MAIL = 3,
  SMTP_RCPT = 4, SMTP_DATA = 5, SMTP_MESSAGE = 6, SMTP_END↩
  _MESSAGE = 7,
  SMTP_RSET = 8, SMTP_QUIT = 9, SMTP_INVALID }
- enum smtp_status_code {
  SMTP_COMMAND_FOR_HUMAN = 214, SMTP_READY_FOR_WORK = 220, SMTP_CLOSING = 221, SMTP_SUCCESS = 250,
  SMTP_RECIPIENT_IS_NOT_LOCAL = 251, SMTP_NOT_VERIFIED = 252, SMTP_CONTEXT_INPUT = 354, SMTP_SERVER_IS_NOT_AVAIL↩
  ABLE = 421,
  SMTP_MAX_SIZE_MESSAGE = 422, SMTP_DISCONNECTION_DURING↩
  _TRANSMISSION = 442, SMTP_REQUESTED_COMMAND_FAILED = 450,
  SMTP_COMMAND_ABORTED_SERVER_ERROR = 451,
  SMTP_COMMAND_ABORTED_INSUFFICIENT_SYSTEM_STORAGE = 452, SMTP_SERVER_COULD_NOT_RECOGNIZE_COMMAND = 500, S↩
  MTP_SYNTAX_ERROR_COMMAND_ARGS = 501, SMTP_COMMAND_↩
  IS_NOT_IMPLEMENTED = 502,
  SMTP_SERVER_HAS_ENCOUNTERED_A_BAD_SEQUENCE_OF_CO↩
  MMANDS = 503, SMTP_COMMAND_PARAMETER_IS_NOT_IMPLEME↩
  NTED = 504, SMTP_HOST_NEVER_ACCEPTS_MAIL = 521, SMTP_CON↩
  TEXT_COULD_NOT_BE_DELIVERED_FOR_POLICY_REASONS = 541,
  SMTP_USER_MAILBOX_WAS_UNAVAILABLE = 550, SMTP_RECIPI↩
  ENT_IS_NOT_LOCAL_TO_THE_SERVER = 551, SMTP_ACTION_W↩
  AS_ABORTED_DUE_TO_EXCEEDED_STORAGE_ALLOCATION = 552,
  SMTP_MAILBOX_NAME_IS_INVALID = 553,
  SMTP_MAILBOX_DISABLED = 554, UNDEFINED_ERROR = 0, NOT_A↩
  NSWER = 1 }

Функции

- smtp_context * smtp_connect (char *server, char *port, smtp_context *context)
- state_code smtp_send_helo (smtp_context *context)
- state_code smtp_send_ehlo (smtp_context *context)
- state_code smtp_send_mail (smtp_context *context, char *from_email)
- state_code smtp_send_rcpt (smtp_context *context, char *to_email)
- state_code smtp_send_data (smtp_context *context)
- state_code smtp_send_message (smtp_context *context, char *message)
- state_code smtp_send_end_message (smtp_context *context)
- state_code smtp_send_rset (smtp_context *context)
- state_code smtp_send_quit (smtp_context *context)
- state_code send_smtp_request (smtp_context *context, char *str)
- smtp_response get_smtp_response (smtp_context *context)

- bool is_smtp_success (status_code)
- bool is_smtp_4xx_error (status_code status_code)
- bool is_smtp_5xx_error (status_code status_code)

2.5.5.1 Типы

2.5.5.1.1 smtp_addr

typedef struct smtp_address smtp_addr

2.5.5.1.2 smtp_context

typedef struct smtp_context smtp_context

2.5.5.1.3 smtp_header

typedef struct smtp_header smtp_header

2.5.5.1.4 smtp_response

typedef struct smtp_response smtp_response

2.5.5.1.5 state_code

typedef enum smtp_state_code state_code

2.5.5.1.6 status_code

typedef enum smtp_status_code status_code

2.5.5.2 Перечисления

2.5.5.2.1 smtp_state_code

enum smtp_state_code

Элементы перечислений

SMTP_CONNECT	
SMTP_HELO	
SMTP_EHLO	
SMTP_MAIL	
SMTP_RCPT	
SMTP_DATA	
SMTP_MESSAGE	
SMTP_END_MESSAGE	
SMTP_RSET	
SMTP_QUIT	
SMTP_INVALID	

2.5.5.2.2 smtp_status_code

enum smtp_status_code

Элементы перечислений

SMTP_COMMAND_FOR_HUMAN	
SMTP_READY_FOR_WORK	
SMTP_CLOSING	
SMTP_SUCCESS	
SMTP_RECIPIENT_IS_NOT_LOCAL	
SMTP_NOT_VERIFIED	
SMTP_CONTEXT_INPUT	
SMTP_SERVER_IS_NOT_AVAILABLE	
SMTP_MAX_SIZE_MESSAGE	
SMTP_DISCONNECTION_DURING_TRANSMISSION	
SMTP_REQUESTED_COMMAND_FAILED	
SMTP_COMMAND_ABORTED_SERVER_ERROR	
SMTP_COMMAND_ABORTED_INSUFFICIENT_SYSTEM_STORAGE	
SMTP_SERVER_COULD_NOT_RECOGNIZE_COMMAND	
SMTP_SYNTAX_ERROR_COMMAND_ARGS	
SMTP_COMMAND_IS_NOT_IMPLEMENTED	
SMTP_SERVER_HAS_ENCOUNTERED_A_BAD_SEQUENCE_OF_CO↩ MMANDS	
SMTP_COMMAND_PARAMETER_IS_NOT_IMPLEMENTED	
SMTP_HOST_NEVER_ACCEPTS_MAIL	

Элементы перечислений

SMTP_CONTEXT_COULD_NOT_BE_DELIVERED_FOR_POLICY_REASONS	
SMTP_USER_MAILBOX_WAS_UNAVAILABLE	
SMTP_RECIPIENT_IS_NOT_LOCAL_TO_THE_SERVER	
SMTP_ACTION_WAS_ABORTED_DUE_TO_EXCEEDED_STORAGE_ALLOCATION	
SMTP_MAILBOX_NAME_IS_INVALID	
SMTP_MAILBOX_DISABLED	
UNDEFINED_ERROR	
NOT_ANSWER	

2.5.5.3 Функции

2.5.5.3.1 get_smtp_response()

```
smtp_response get_smtp_response (
            smtp_context * context )
```

2.5.5.3.2 is_smtp_4xx_error()

```
bool is_smtp_4xx_error (
            status_code status_code )
```

2.5.5.3.3 is_smtp_5xx_error()

```
bool is_smtp_5xx_error (
            status_code status_code )
```

2.5.5.3.4 is_smtp_success()

```
bool is_smtp_success (
            status_code  )
```

2.5.5.3.5 send_smtp_request()

```
state_code send_smtp_request (
                smtp_context * context,
                char * str )
```

2.5.5.3.6 smtp_connect()

```
smtp_context* smtp_connect (
                char * server,
                char * port,
                smtp_context * context )
```

2.5.5.3.7 smtp_send_data()

```
state_code smtp_send_data (
                smtp_context * context )
```

2.5.5.3.8 smtp_send_ehlo()

```
state_code smtp_send_ehlo (
                smtp_context * context )
```

2.5.5.3.9 smtp_send_end_message()

```
state_code smtp_send_end_message (
                smtp_context * context )
```

2.5.5.3.10 smtp_send_helo()

```
state_code smtp_send_helo (
                smtp_context * context )
```

2.5.5.3.11 smtp_send_mail()

```
state_code smtp_send_mail (
            smtp_context * context,
            char * from_email )
```

2.5.5.3.12 smtp_send_message()

```
state_code smtp_send_message (
            smtp_context * context,
            char * message )
```

2.5.5.3.13 smtp_send_quit()

```
state_code smtp_send_quit (
            smtp_context * context )
```

2.5.5.3.14 smtp_send_rcpt()

```
state_code smtp_send_rcpt (
            smtp_context * context,
            char * to_email )
```

2.5.5.3.15 smtp_send_rset()

```
state_code smtp_send_rset (
            smtp_context * context )
```

2.5.6 Файл include/util/util.h

```
#include <sys/queue.h>
#include <stddef.h>
#include <stdarg.h>
#include <stdbool.h>
#include <stdio.h>
```

Классы

- struct pair
- struct string
- struct string_tokens

Макросы

- #define strsize(args...) snprintf(NULL, 0, args) + sizeof('\0')
- #define vstrsize(args...) snprintf(NULL, 0, args) + sizeof('\0')

Определения типов

- typedef struct pair pair
- typedef struct string string
- typedef struct string_tokens string_tokens

Функции

- long int convert_string_to_long_int (const char *str)
- string_tokens split (const char *const str, const char *const delim)
- void free_string_tokens (string_tokens *tokens)
- string * get_string_from_characters (string *str, char *characters)
- void free_string (string *str)
- void trim (char *str)
- void * allocate_memory (size_t bytes)
- void * reallocate_memory (void *buffer, size_t prev_size, size_t new_size)
- void * callocate_memory (size_t size, size_t bytes)
- char * file_readline (FILE *fp)
- bool is_interrupt ()

2.5.6.1 Макросы

2.5.6.1.1 strsize

```
#define strsize(
        args... ) snprintf(NULL, 0, args) + sizeof( '\0' )
```

2.5.6.1.2 vstrsize

```
#define vstrsize(
                args... ) snprintf(NULL, 0, args) + sizeof( '\0' )
```

2.5.6.2 Типы

2.5.6.2.1 pair

typedef struct pair pair

2.5.6.2.2 string

typedef struct string string

2.5.6.2.3 string_tokens

typedef struct string_tokens string_tokens

2.5.6.3 Функции

2.5.6.3.1 allocate_memory()

```
void* allocate_memory (
                size_t bytes )
```

2.5.6.3.2 callocate_memory()

```
void* callocate_memory (
                size_t size,
                size_t bytes )
```

2.5.6.3.3 convert_string_to_long_int()

```
long int convert_string_to_long_int (
                const char * str )
```

2.5.6.3.4 file_readline()

```
char* file_readline (
                FILE * fp )
```

2.5.6.3.5 free_string()

```
void free_string (
                string * str )
```

2.5.6.3.6 free_string_tokens()

```
void free_string_tokens (
                string_tokens * tokens )
```

2.5.6.3.7 get_string_from_characters()

```
string* get_string_from_characters (
                string * str,
                char * characters )
```

2.5.6.3.8 is_interrupt()

```
bool is_interrupt ( )
```

2.5.6.3.9 reallocate_memory()

```
void* reallocate_memory (
            void * buffer,
            size_t prev_size,
            size_t new_size )
```

2.5.6.3.10 split()

```
string_tokens split (
            const char *const str,
            const char *const delim )
```

2.5.6.3.11 trim()

```
void trim (
            char * str )
```

2.5.7 Файл include/util/network.h

```
#include <netdb.h>
#include <resolv.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
```

Классы

- struct ips

Определения типов

- typedef struct ips ips

Функции

- char * get_addr_by_socket (int socket)
- ips get_ips_by_hostname (char *hostname)
- int resolvmx (const char *name, char **mxs, int limit)
- char * receive_line (int socket_d)
- int send_line (int socket_d, char *message)
- bool set_socket_blocking_enabled (int socket, bool blocking)

2.5.7.1 Типы

2.5.7.1.1 ips

typedef struct ips ips

2.5.7.2 Функции

2.5.7.2.1 get_addr_by_socket()

```
char* get_addr_by_socket (
            int socket )
```

2.5.7.2.2 get_ips_by_hostname()

```
ips get_ips_by_hostname (
            char * hostname )
```

2.5.7.2.3 receive_line()

```
char* receive_line (
            int socket_d )
```

2.5.7.2.4 resolvmx()

```
int resolvmx (
        const char * name,
        char ** mxs,
        int limit )
```

2.5.7.2.5 send_line()

```
int send_line (
        int socket_d,
        char * message )
```

2.5.7.2.6 set_socket_blocking_enabled()

```
bool set_socket_blocking_enabled (
        int socket,
        bool blocking )
```

2.6 Заключение

В результате выполнения данной курсовой работы был изучен протокол SMTP. Разработан и реализован SMTP-клиент как часть MTA. Был изучен способ многопоточной обработки с использованием механизма мультиплексирования, когда несколько исходящих соединений обрабатываются в одном потоке исполнения. Разработка SMTP-клиента сопровождалась построением диаграмм, в том числе и конечного автомата для последующей разработке общения SMTP-клиента с удаленным SMTP-сервером. Проведено комплексное тестирование, включающее в себя юнит-тесты и интеграционные тесты в комплексе с ручным тестированием. Также были соблюдены все этапы жизненного цикла разработки ПО, включая в себя автоматизированную сборку проекта с использование CI.

В заключение можно сделать вывод, что все поставленные задачи в рамках курсовой работы по разработке многопоточного SMTP-клиента как части MTA были выполнены.

2.7 Список источников и литературы

1. http://rfc.com.ru/rfc2821.htm
2. http://rfc.com.ru/rfc1123
3. https://www.protocols.ru/WP/rfc5322/
4. RFC 1035 DOMAIN NAMES — IMPLEMENTATION AND SPECIFICATION https://www.protocols.ru/WP/rfc1035/
5. dovecot maildir https://wiki.dovecot.org/MailLocation/Maildir
6. qmail maildir https://cr.yp.to/proto/maildir.html