Расчетно-Пояснительная Записка

к курсовой работе на тему:
Серверная часть MTA SMTP

Сычев С.А. ИУ7-32М

15 января 2021 г.

Оглавление

Введение

Данная работа посвящена разработке серверной части почтового сервера MTA (англ. Message Transfer Agent), использующего протокол SMTP (англ. Simple Mail Transfer Protocol) - сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP.

Цель работы: спроектировать, разработать и протестировать серверную часть почтового SMTP-сервера (MTA), обеспечивающую локальную доставку сообщений и их добавление в очередь удаленной доставки.

Дополнительные условия:

- Использовать системный вызов `poll()`;
- Использовать рабочие процессы;
- Журналирование осуществлять в отдельном процессе.
- Требуется проверять обратную зону DNS.

Для достижения цели работы, необходимо выполнить следующие задачи:

1. Изучить и проанализировать предметную область, достоинства и недостатки требуемого способа разделения на процессы и опроса сокетов;
2. Спроектировать серверную часть MTA SMTP;
3. Реализовать серверную часть MTA SMTP;
4. Отладить и протестировать серверную часть MTA SMTP

Глава 1

Аналитический раздел

1.1 Протокол SMTP

SMTP – это сетевой протокол, предназначенный для передачи писем электронной почты в сетях TCP/IP. SMTP впервые был описан в RFC 821 (1982 год), а последнее обновление описано в RFC 5321 [1] и включает масштабируемое расширение протокола — ESMTP (Extended SMTP). В настоящее время под протоколом SMTP подразумеваются и его расширения. Протокол SMTP предназначен для передачи исходящей почты с использованием порта TCP 25.

Взаимодействие в рамках SMTP строится по принципу двусторонней связи, устанавливаемой между отправителем и получателем почтового сообщения. При этом отправитель инициирует соединение и посылает запросы, а получатель - отвечает на эти запросы. Таким образом, отправитель выступает в роли клиента, а получатель - сервера.

1.1.1 Команды протокола SMTP

Команды протокола SMTP начинаются с названий - ключевых слов, указывающих, какую операцию хочет осуществить клиент. За ключевым словом следуют, отделенные пробелом, параметры. Конец команды обозначается последовательностью символов \r\n - обозначаемой CRLF.

Ниже представлен список базовых SMTP-команд:

- **HELO [доменное имя клиента] CRLF** - Открывает SMTP сессию. В RFC 2821 рекомендуется использовать команду HELO, только если программное обеспечение не поддерживает команду EHLO.
- **EHLO [доменное имя клиента] CRLF** - Открывает ESMTP сессию. В ответ на эту команду сервер сообщает, готов ли он к продолжению диалога.
- **MAIL FROM: <адрес_отправителя> CRLF** - Сообщает адрес отправителя письма. Команда MAIL должна быть выполнена для письма только один раз и только после успешного выполнения команды EHLO или HELO.
- **RCPT TO: <адрес_получателя> CRLF** - Сообщает адрес получателя письма. Доставка сообщения возможна, только если указан хотя бы один адрес получателя. Команда RCPT может быть выполнена только после успешного выполнения команды MAIL и должна быть повторена для каждого получателя письма.

– **DATA CRLF** - Сообщает о готовности ввода текста письма. Команда DATA может быть выполнена только после успешного выполнения хотя бы одной команды RCPT. После успешного ответа сервера, клиент начинает передачу текста письма, с учетом заголовков. Сервер определяет окончание текста письма по последовательности: CRLF . CRLF

– **RSET CRLF** - Сброс SMTP соединения. Команда RSET аннулирует все переданные до нее на сервер данные.

– **VRFY <адрес> CRLF** - Проверяет наличия указанного в качестве аргумента почтового адреса.

– **QUIT CRLF** - Закрывает SMTP сессию.

Существуют также и другие команды ESMTP, однако в данной работе они не рассматриваются.

1.1.2 Ответы SMTP сервера

Ответ SMTP сервера состоит из кода ответа, за которым через пробел следует дополнительный текст. Код служит индикатором состояния сервера и делится на четыре группы:

– **2xx** - команда выполнена успешно;

– **3xx** - ромежуточный положительный результат. Команда принята, но сервер ожидает от клиента дополнительные данные для завершения операции;

– **4xx** - исполнение команды временно невозможно. Команда не может быть выполнена, но проблема может быть устранена;

– **5xx** - исполнение команды невозможно.

Если ответ сервера состоит из нескольких строк, то каждая из них начинается кодом, который отделяется от сопровождающего текста не пробелом, а символом "минус" (-). В последней строке номер отделяется от текста пробелом. Каждая строка ответа, как и строки команд, заканчивается последовательностью CRLF.

1.2 Преимущества и недостатки использования системного вызова poll.

Системный вызов poll, по сравнению с другими системными вызовами опроса сокетов имеет следующие достоинства [2]:

– Отсутствие активного ожидания на процессоре.

– Обрабатка больше 1024 клиентов по сравнению с системным вызовом `select()`.

– Не модифицируется структура pollfd, что даёт возможность её переиспользования между вызовами poll() — нужно лишь обнулить поле revents.

– Наблюдаемые события лучше структурированы. Например, можно определить отключение удалённого клиента без необходимости чтения данных из сокета.

К его недостаткам можно отнести [2]:

– Poll отсутствует на некоторых платформах (в основном старых).

– Невозможнось определить, какие именно дескрипторы сгенерировали события, без полного прохода по всем наблюдаемым структурам и проверки поля revents. (Проблема так-же в том, что в ядре данная операция реализована так-же).

– Как и при использовании select, нет возможности динамически менять наблюдаемый набор событий.

1.3 Преимущества и недостатки использования нескольких рабочих процессов.

Использование нескольких рабочих процессов имеет следующие преимущества:

– Позволяет повысить возможности сервера по одновременной обработке нескольких клиентов.

– Позволяет освободить высоконагруженные процессы, работающие с клиентами некоторых "тяжелых"задач, таких как операции файлового ввода/вывода.

– Позволяет более полно использовать возможности современных многоядерных процессоров.

К недостаткам использования многопроцессности можно отнести:

– повышение сложности разработки за счет необходимости корректного завершения всех процессов, ожидания, а также межпроцессного взаимодействия.

– Как и при использовании select, нет возможности динамически менять наблюдаемый набор событий.

1.4 Сущности предметной области

Для серверной части SMTP MTA можно выделить следующие основные сущности предметной области:

1. клиент (подключение) - это текущее активное соединение клиента с сервером, к нему можно отнести буферы ввода вывода и текущее состояние.

2. письмо - получаемое от клиента сообщение, содержащее информацию от отправителе и получателях, имеющее заголовки и тело.

3. почтовый адрес - это почтовый адрес, получатель, или один из отправителей письма.

ER-диаграмма сущностей предметной области приведена на 1.1

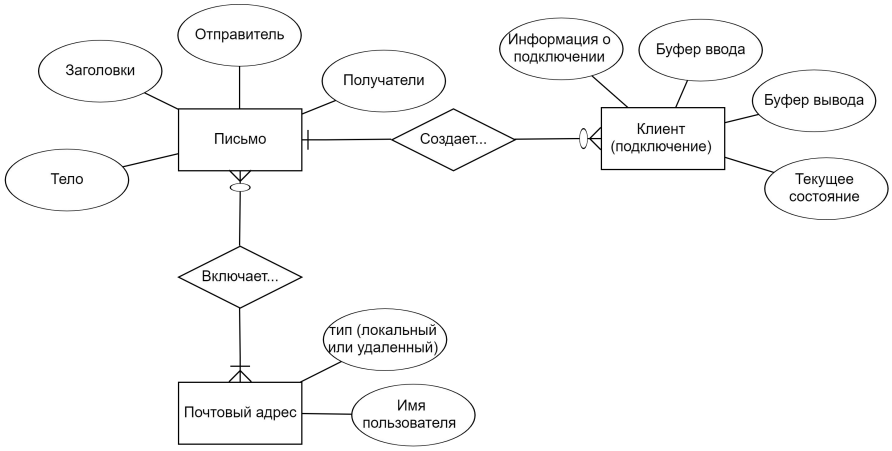

Рис. 1.1: ER-диаграмма сущностей предметной области

Глава 2

Конструкторский раздел

2.1 Описание основных структур данных

В результате проектирования системы, были получены следующие основные структуры данных:

- **server** - абстракция сервера, содержащая в себе все, что требуется для обработки соединений в одном потоке выполнения.
- **logger** - абстракция логгера, содержащая все необходимые структуры данных для работы логгера или отправки данных в логгер.
- **client_info** - абстракция информации о клиенте, содержит статус, буферы ввода вывода и другую информацию о клиенте.
- **mail** - абстракция письма, содержит адрес отправителя, адреса получателей и текст письма.
- **string** - абстракция строки, предоставляет ряд удобных функций по работе со строками. Осуществляет автоматическое перевыделение памяти, в случае необходимости.
- **address** - абстракция адреса, соредрит строку адреса, а также его тип.
- **process_info** - абстакция процесса, содержит информацию о процессе и его потомках, необходимую для корректного завершения программы.
- **server_parser** - абстракция обработчика входного буфера.
- **server_parser_result** - абстракция результата работы парсера.
- **compiled_regexp** - абстракция скомпилированного регулярного выражения.

На 2.1 приведена UML-диаграмма основных структур данных.

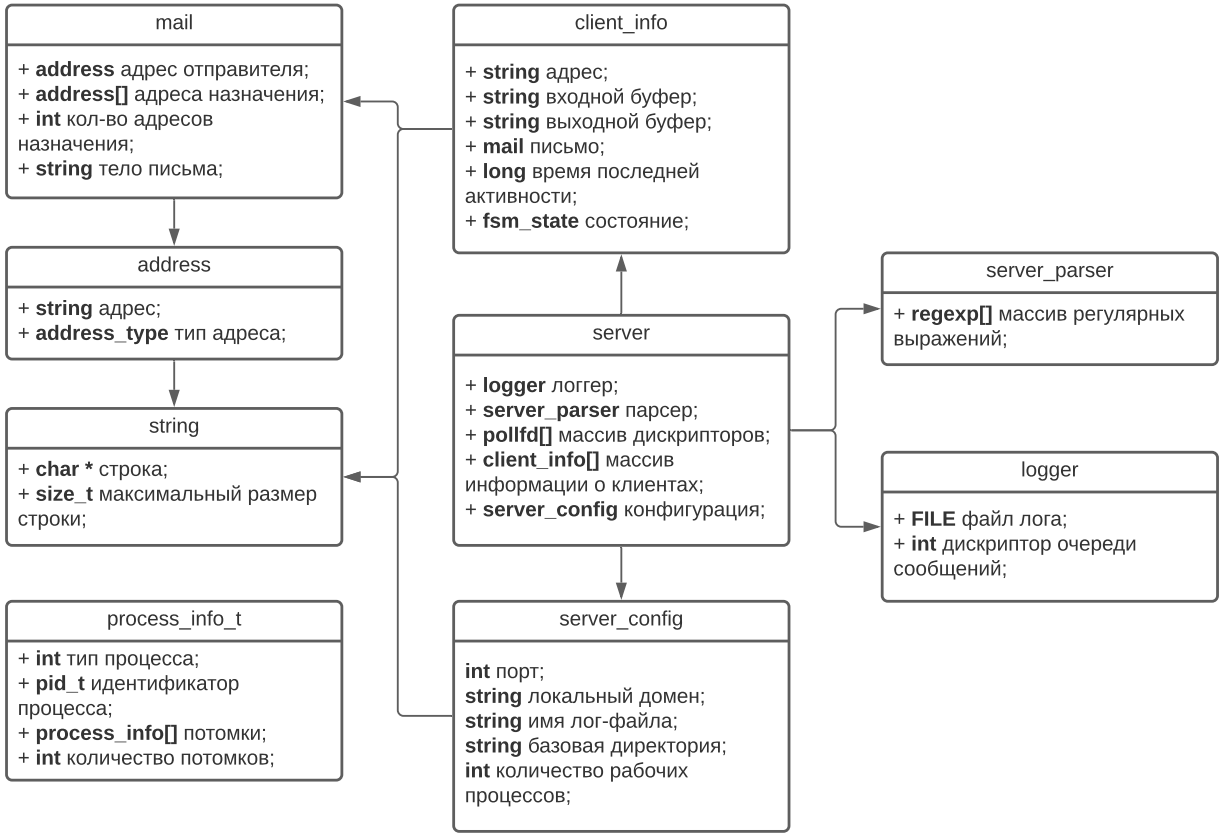

Рис. 2.1: Диаграмма основных структур данных

2.2 Обработка соединений в одном потоке выполнения

Ниже привезен псевдокод обработки соединений в одном потоке выполнения:

```
Основной цикл обработки:
  poll:
    Если пришло сообщение в дескриптор сигнала выхода:
      Завершить работу сервера, закрыв все клиентские сокеты

    Если пришло сообщение на слушающий сокет сервера:
      Добавить нового клиента обработав( подключение)
      Начать новую итерацию цикла (continue)

    Цикл по всем дескрипторам клиентских сокетов:
      Если на сокет клиента пришло сообщение:
        Обнулить таймер клиента
        Обработать сообщение от клиента
        Начать новую итерацию цикла (continue)
      Если есть неотправленное сообщение для клиента:
        Отослать сообщение клиенту
        Начать новую итерацию цикла (continue)

  Цикл по всем подключенным клиентам:
    Если таймер клиента превысил определенное значение:
```

```
21          Отправить клиенту сообщение об истечении таймера
22        слиЕ произошла внутренняя ошибка:
23          Отправить клиенту сообщение об ошибке
24        Если сессия с клиентом завершена:
25          Очистить данные о клиенте
26          Закрыть соединение с клиентом
```

2.3 Распределение задач по процессам

В данной работе предполагается использование фиксированного количества рабочих процессов. При этом по условию задания - журналирование также должен осуществлять отдельный процесс.

Основная задача SMTP сервера - обрабатывать подключающихся клиентов, сохраняя письма, присылаемые ими. Данная задача может выполняться в нескольких процессах параллельно. При этом, если каждого отдельного клиента, процесс обрабатывает полностью самостоятельно, не потребуется какого-либо взаимодействия.

Таким образом, разницы с точки зрения выполняемого серверного кода для основного родительского процесса и его потомков нет. Единственным отличием родительского процесса является то, что при завершении работы программы, он должен корректно дождаться завершения всех своих потомков.

Таким образом, SMTP сервер создает процессы следующих видов, со следующими задачами:

1. MASTER - родительский процесс, создающий остальных, после чего выполняющий основной серверный код и обрабатываюший подключающихся клиентов. При завершении работы программы ожидает завершения своих потомков.

2. WORKER - дочерний процесс, выполняющий основной серверный код и обрабатываюший подключающихся клиентов.

3. LOGGER - дочерний процесс журналирования, выполняет непосредственную запись логов в файл и консоль.

На рис. 2.2 синими стрелками отмечены отношения типа родитель-потомок между процессами.

2.4 Межпроцессное взаимодействие

В силу того, что отдельные клиенты обрабатываются в отдельных процессах изолировано, межпроцессное взаимодействие необходимо только для передачи логов процессу журналирования. На рис. 2.2 черными прерывистыми линиями показана передача данных от процессов друг другу.

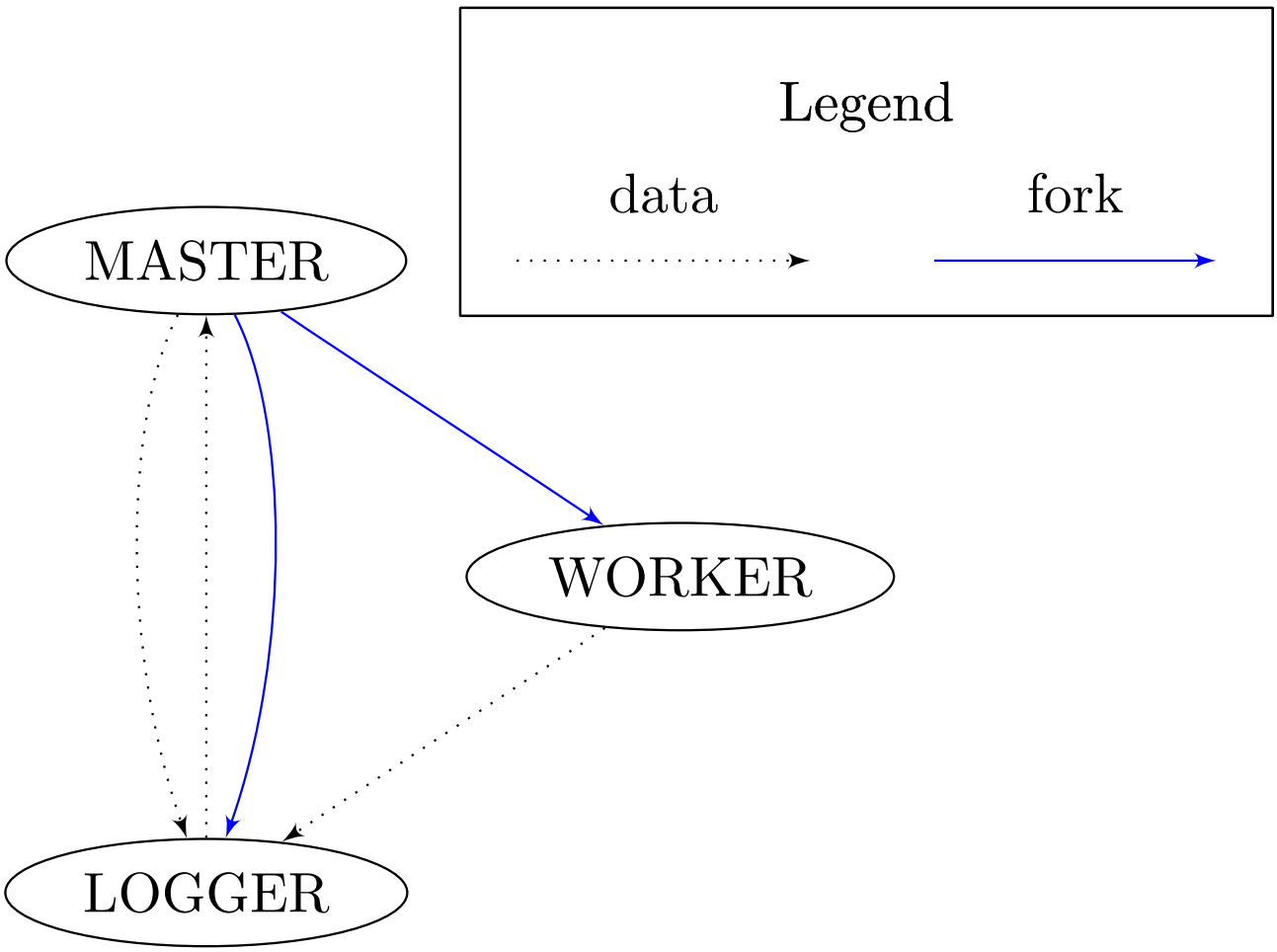

Рис. 2.2: Диаграмма межпроцессного взаимодействия

Для передачи логов используются очереди сообщений и системные вызовы msgrcv msgsend и др.

Родительский процесс, создавая процесс журналирования, перед дальнейшей работой, ожидает от него подтверждения готовности (сообщение особого типа). После чего, в очередь сообщений отправляются логи.

Для остановки процесса журналирования также используется сообщение специального типа, отправляемое родительским процессом.

Рабочим процессам не требуется отдельно передавать сигнал завершения, так как они наследуют от родительского файловый дискриптор для получения этого сигнала, и получат сигнал без каких либо дополнительных действий.

2.5 Конечный автомат состояний сервера

На рис. 2.3 изображен конечный автомат разрабатываемой серверной части SMTP сервера.

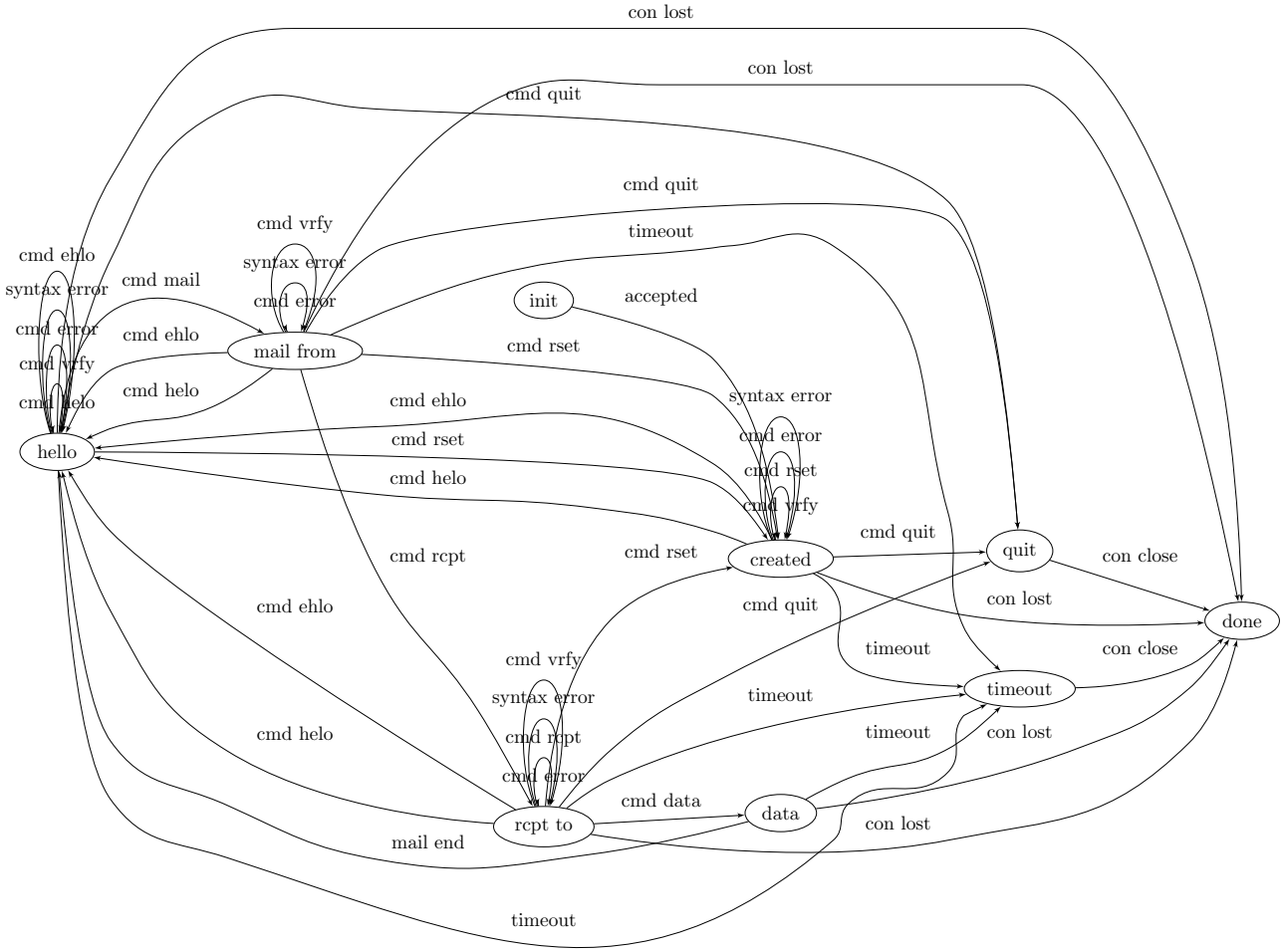

Рис. 2.3: Конечный автомат состояний сервера

2.6 Регулярные выражения для парсинга SMTP команд

Регулярные выражения для парсера SMTP команд приведены на листинге ниже:

```
#define PARSER_EOL "\r\n"
#define PARSER_EOD "\r\n.\r\n"

#define PARSER_EOL_SIZE (sizeof(PARSER_EOL) - 1)
#define PARSER_EOD_SIZE (sizeof(PARSER_EOD) - 1)

#define DOMAIN_REGEXP "(\\s+\\S+)?"
#define ADDRESS_REGEXP "\\s*( \
(?:[a-z0-9!#$%&\'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&\'*+/=?^_`{|}~-]+)
    *|\"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\
\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])
    *\")@(?:(?:[a-z0-9](?:\
[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])
    ?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9]\
[0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0
    -9]:(?:[\\x01-\\x08\\x0b\\x0c\
```

```
\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])+)
    \\])|\
\
<(?:[a-z0-9!#$%&\'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&\'*+/=?^_`{|}~-]+)
    *|\"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\
\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])
    *\")@(?:(?:[a-z0-9](?:\
[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])
    ?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9]\
[0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0
    -9]:(?:[\\x01-\\x08\\x0b\\x0c\
\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])+)
    \\])>|\
\
\"(?:[a-z0-9!#$%&\'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&\'*+/=?^_`{|}~-]+)
    *|\"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\
\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])
    *\")@(?:(?:[a-z0-9](?:\
[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])
    ?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9]\
[0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0
    -9]:(?:[\\x01-\\x08\\x0b\\x0c\
\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])+)
    \\])\")\\s*"

#define ADDRESS_REGEXP_WITH_EMPTY "\\s*(<>|\
(?:[a-z0-9!#$%&\'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&\'*+/=?^_`{|}~-]+)
    *|\"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\
\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])
    *\")@(?:(?:[a-z0-9](?:\
[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])
    ?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9]\
[0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0
    -9]:(?:[\\x01-\\x08\\x0b\\x0c\
\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])+)
    \\])|\
\
<(?:[a-z0-9!#$%&\'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&\'*+/=?^_`{|}~-]+)
    *|\"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\
\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])
    *\")@(?:(?:[a-z0-9](?:\
[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])
    ?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9]\
[0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0
    -9]:(?:[\\x01-\\x08\\x0b\\x0c\
\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])+)
    \\])>|\
\
\"(?:[a-z0-9!#$%&\'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&\'*+/=?^_`{|}~-]+)
    *|\"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\
\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])
    *\")@(?:(?:[a-z0-9](?:\
[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])
    ?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9]\
[0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0
    -9]:(?:[\\x01-\\x08\\x0b\\x0c\
```

```
44 \\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])+)
       \\])\")\\s*"
45 // see https://emailregex.com/ for this interesting regex for email
46
47 #define HELO_REGEXP "[Hh][Ee][Ll][Oo]" DOMAIN_REGEXP PARSER_EOL
48 #define EHLO_REGEXP "[Ee][Hh][Ll][Oo]" DOMAIN_REGEXP PARSER_EOL
49 #define MAIL_REGEXP "[Mm][Aa][Ii][Ll] [Ff][Rr][Oo][Mm]:"
       ADDRESS_REGEXP_WITH_EMPTY PARSER_EOL
50 #define RCPT_REGEXP "[Rr][Cc][Pp][Tt] [Tt][Oo]:" ADDRESS_REGEXP PARSER_EOL
51 #define DATA_REGEXP "[Dd][Aa][Tt][Aa]" PARSER_EOL
52 #define RSET_REGEXP "[Rr][Ss][Ee][Tt]" PARSER_EOL
53 #define VRFY_REGEXP "[Vv][Rr][Ff][Yy]\\s+\\S*" PARSER_EOL
54 #define QUIT_REGEXP "[Qq][Uu][Ii][Tt]" PARSER_EOL
```

Листинг 2.1: parser/server_parser.h

2.7 Хранение почты

Для хранения локальной почты, то есть почты, предназначенной для клиентов с почтовым доменом сервера, используется формат Maildir. Maildir - формат хранения электронной почты, не требующий монопольного захвата файла для обеспечения целостности почтового ящика при чтении, добавлении или изменении сообщений. Каждое сообщение хранится в отдельном файле с уникальным именем. 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.

Путь до директории с письмами, адресованными пользователю user будет выглядеть следующим образом:

```
1 base_maildir/user/maildir/new/
```

Для хранения почты, преднозначенной другим почтовым серверам также используется формат Maildir. Единственное отличие - отсутствие в пути к файлу имени клиента т.е.:

```
1 base_maildir/maildir/new/
```

Имена файлов с письмами генерируюстя также в соответствии с Maildir и содержат случайное число, текущее время (UNIX Time) и префикс сервера.

Ниже приведен пример такого имени:

```
1 1610708049.10777.smtp_serv.846930886:2,
```

Глава 3

Технологический раздел

3.1 Язык программирования и библиотеки

Для написания исходного кода SMTP сервера использовался язык C стандарта C99. Помимо стандартных библиотек были использованы:

- libconfig - библиотека для чтения файлов конфигурации [3].
- Autogen - библиотека для автоматической генерации конечного автомата состояний.
- PCRE - библиотека для работы с регулярными выражениями [4].
- CUnit - библиотека для модульного тестирования [5].

3.2 Платформы и компиляторы

Для сборки использовался компилятор gcc, со следующими флагами:

```
1 -std=c99 -g -g3 -O0 -Wall -Werror -pedantic -D\_GNU\_SOURCE
```

Программное обеспечение разрабатывалось, собиралось и тестировалось на виртуальной машине с ОС Ubuntu 16.04, с двухядерным процессором и 4 ГБ оперативной памяти.

3.3 Сборка

Сборка осуществляется при помощи утилиты Make. Применен иерархический подход. Сценарии сборки описаны в трех Makefile-ах.

1. Makefile для сборки и тестирования сервера - расположен в каталоге **/server/src** и отвечает за непосредственную сборку и тестирование сервера.
2. Makefile для сборки отчета - расположен в каталоге **/server/report** и отвечает за сборку данного отчета.
3. Корневой Makefile - расположен в каталоге **/server**, объединяет функции двух предыдущих, имеет общие цели для сборки сервера, его тестирования и генерации отчета, по итогам тестирования.

3.4 Визуализация сценариев сборки

В данном разделе приведена визуализиция Makefile-ов, используемых для сборки и тестирования сервера и отчета. Визуализация корневого Makefile-а не приводится так как она во многом повторяет указанные ранее.

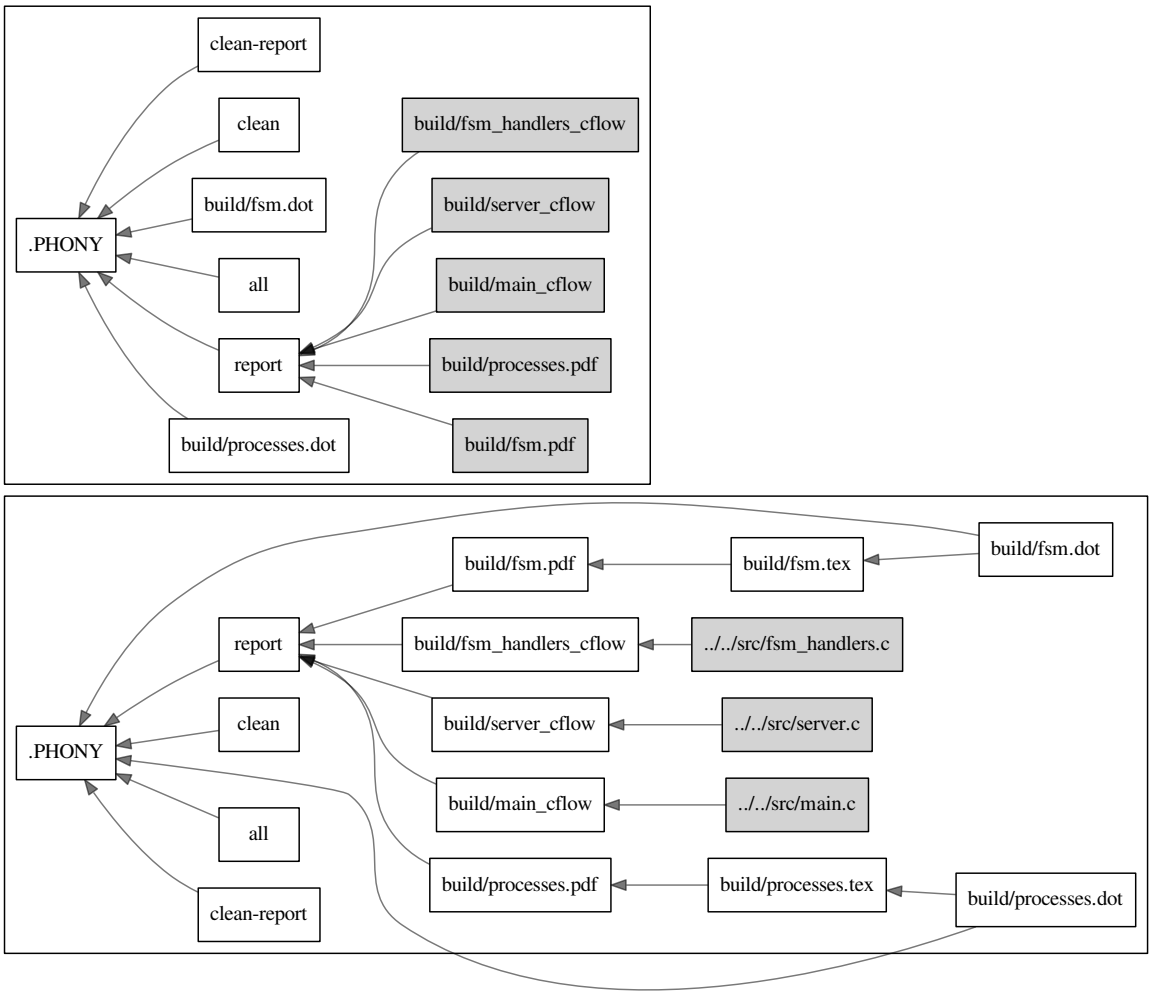

Рис. 3.1: Визуализация сценария сборки отчета

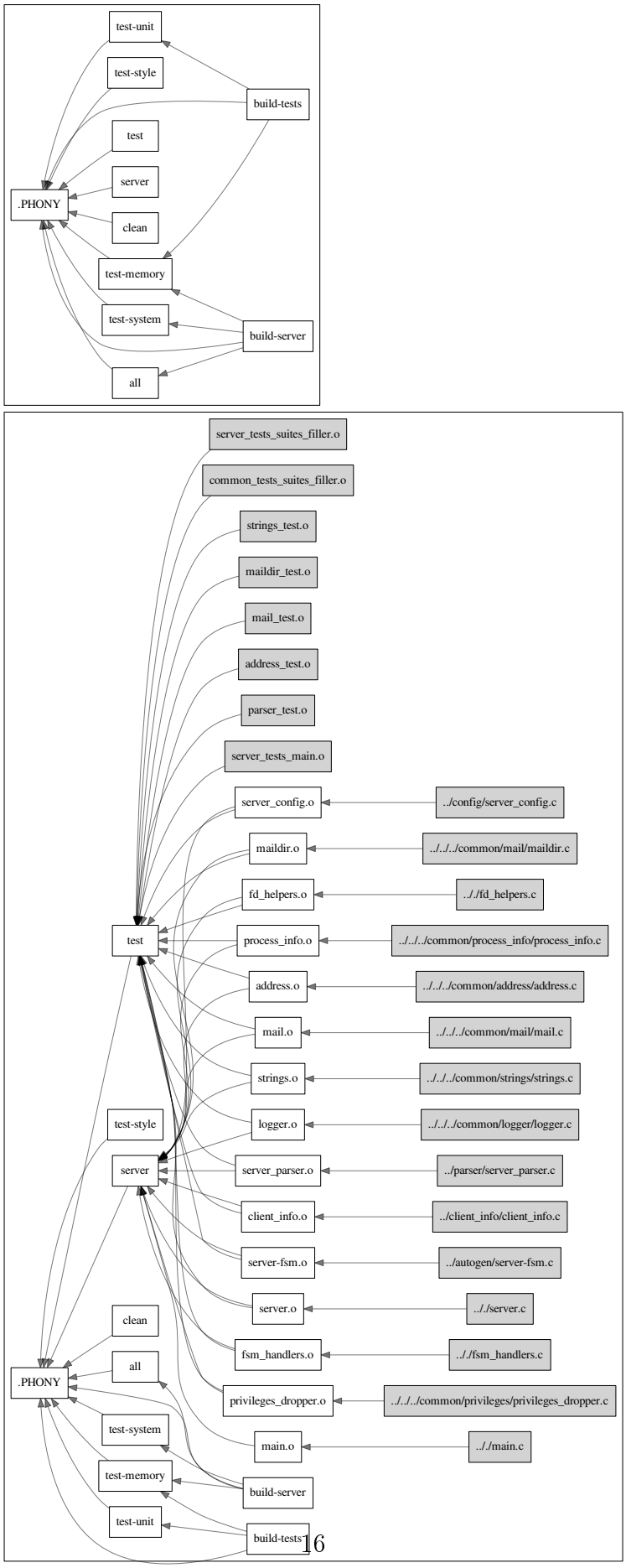

Рис. 3.2: Визуализация сценария сборки и тестирования сервера

3.5 Кодогенерация

Для ускорения разработки конечного автомата сервера была применена библиотека autogen.

Из .def файла с описанием конечного автомата был сгенерирован код, осуществляющий все переключения между состояниями.

Далее потребовалось лишь написать обработчики переходов.

Файл, на основании которого был сгенерирован автомат приведен на листинге ниже:

```
1  Autogen Definitions fsm;
2
3  method = case;
4  type   = reentrant;
5  prefix = server_fsm;
6
7  cookie = "void *server";
8  cookie = "int client_id";
9  cookie = "void *data_str";
10
11 /* Состояния init, invalid, done в fsn есть по умолчанию */
12 /* Состояние invalid - используется для отслеживания ошибок */
13 state  =
14         created,         /* Состояние готовности к приему подключений */
15         hello,           /* Состояние принята" команда HELO/EHLO" */
16         mail_from,       /* Состояние принята" команда MAIL FROM" */
17         rcpt_to,         /* Состояние принята" команда RCPT TO" */
18         data,            /* Состояние принята" команда MAIL DATA" осуществляется( прием данных) */
19         quit,            /* Состояние принята" команда MAIL QUIT" */
20         timeout;         /* Состояние выхода по таймауту */
21
22 event  =
23         accepted,        /* Принят новый клиент */
24         cmd_helo,        /* Получена команда HELO */
25         cmd_ehlo,        /* Получена команда EHLO */
26         cmd_mail,        /* Получена команда MAIL FROM */
27         cmd_rcpt,        /* Получена команда RCPT TO */
28         cmd_data,        /* Получена команда DATA */
29         cmd_rset,        /* Получена команда RSET */
30         cmd_vrfy,        /* Получена команда VRFY */
31         cmd_quit,        /* Получена команда QUIT */
32         mail_end,        /* закончена секция DATA */
33         con_lost,        /* соединение потеряно по( вине клиента) */
34         timeout,         /* таймаут */
35         con_close,       /* соединение закрыто */
36         syntax_error,    /* синтаксическая ошибка ввода */
37         cmd_error;       /* ошибка последовательности команд */
38
39 transition =
40 { tst = created, hello, mail_from, rcpt_to;            tev = cmd_helo;        next = hello;   ttype = helo;
          },
41 { tst = created, hello, mail_from, rcpt_to;            tev = cmd_ehlo;        next = hello;   ttype = ehlo;
          },
42 { tst = created, hello, mail_from, rcpt_to;            tev = cmd_rset;        next = created; ttype = rset;
          },
43 { tst = created, hello, mail_from, rcpt_to;            tev = cmd_quit;        next = quit;    ttype = quit;
          },
44 { tst = created, hello, mail_from, rcpt_to, data;     tev = timeout;         next = timeout; ttype = timeout
     ;    },
45 { tst = created, hello, mail_from, rcpt_to, data;     tev = con_lost;        next = done;    ttype = lost;
          },
46 { tst = timeout, quit;                                tev = con_close;       next = done;    ttype = close;
          };
47
48
49 transition =
50 { tst = init;          tev = accepted;       next = created;     ttype = accepted;     };
51
52 transition =
53 { tst = created;       tev = cmd_vrfy;       next = created;     ttype = vrfy;         },
54 { tst = created;       tev = cmd_error;      next = created;     ttype = cmd_error;    },
55 { tst = created;       tev = syntax_error;   next = created;     ttype = syntax_error; };
56
57 transition =
58 { tst = hello;         tev = cmd_mail;       next = mail_from;   ttype = mail;         },
59 { tst = hello;         tev = cmd_vrfy;       next = hello;       ttype = vrfy;         },
60 { tst = hello;         tev = cmd_error;      next = hello;       ttype = cmd_error;    },
61 { tst = hello;         tev = syntax_error;   next = hello;       ttype = syntax_error; };
62
63 transition =
64 { tst = mail_from;     tev = cmd_rcpt;       next = rcpt_to;     ttype = rcpt;         },
65 { tst = mail_from;     tev = cmd_vrfy;       next = mail_from;   ttype = vrfy;         },
66 { tst = mail_from;     tev = cmd_error;      next = mail_from;   ttype = cmd_error;    },
67 { tst = mail_from;     tev = syntax_error;   next = mail_from;   ttype = syntax_error; };
68
69 transition =
70 { tst = rcpt_to;       tev = cmd_rcpt;       next = rcpt_to;     ttype = rcpt;         },
71 { tst = rcpt_to;       tev = cmd_data;       next = data;        ttype = data;         },
72 { tst = rcpt_to;       tev = cmd_vrfy;       next = rcpt_to;     ttype = vrfy;         },
73 { tst = rcpt_to;       tev = cmd_error;      next = rcpt_to;     ttype = cmd_error;    },
```

```
74 { tst = rcpt_to;       tev = syntax_error;   next = rcpt_to;      ttype = syntax_error; };
75
76 transition =
77 { tst = data;          tev = mail_end;       next = hello;        ttype = mail_received;};
```

3.6 Конфигурация

Для конфигурирования сервера, используется конфигурационный файл, путь до которого необходимо передавать как параметр командной строки при запуске сервера.

Для чтения файла конфигурации используется библиотека: **libconfig**.

В файле конфигурации задаются:

- version - версия файла конфигурации, необходима, чтобы отличить новые версии конфига, в случае дальнейшей работы над проектом.
- port - порт, на котором будет слушать сервер.
- local_domain - почтовый домен, который сервер будет считать локальным (и письма будут записываться в соответствии с maildir).
- log_filename - имя файла с логом.
- maildir - базовая директория для записи писем.
- max_process_cnt - максимальное количество рабочих процессов (без учета логгера).

Ниже привезен пример файла конфигурации:

```
#configuration

version = "1.0";

port = 8080;
local_domain = "local.com";
log_filename = "log.log";
maildir = "/home/netroot/test_mail";
max_process_cnt = 4;
```

3.7 Графы вызова функций

Ниже приведены графы вызова функций для точки входа (*main.c*), основного серверного кода (*server.c*) и обработчиков переходов состояний (*fsm_handlers.c*). Из графов для большей наглядности удалены обращения к следующим функциям:

```
printf
fprintf
strstr
strlen
memcpy
memset
free
malloc
log_info
```

```
log_debug
log_warning
log_error
```

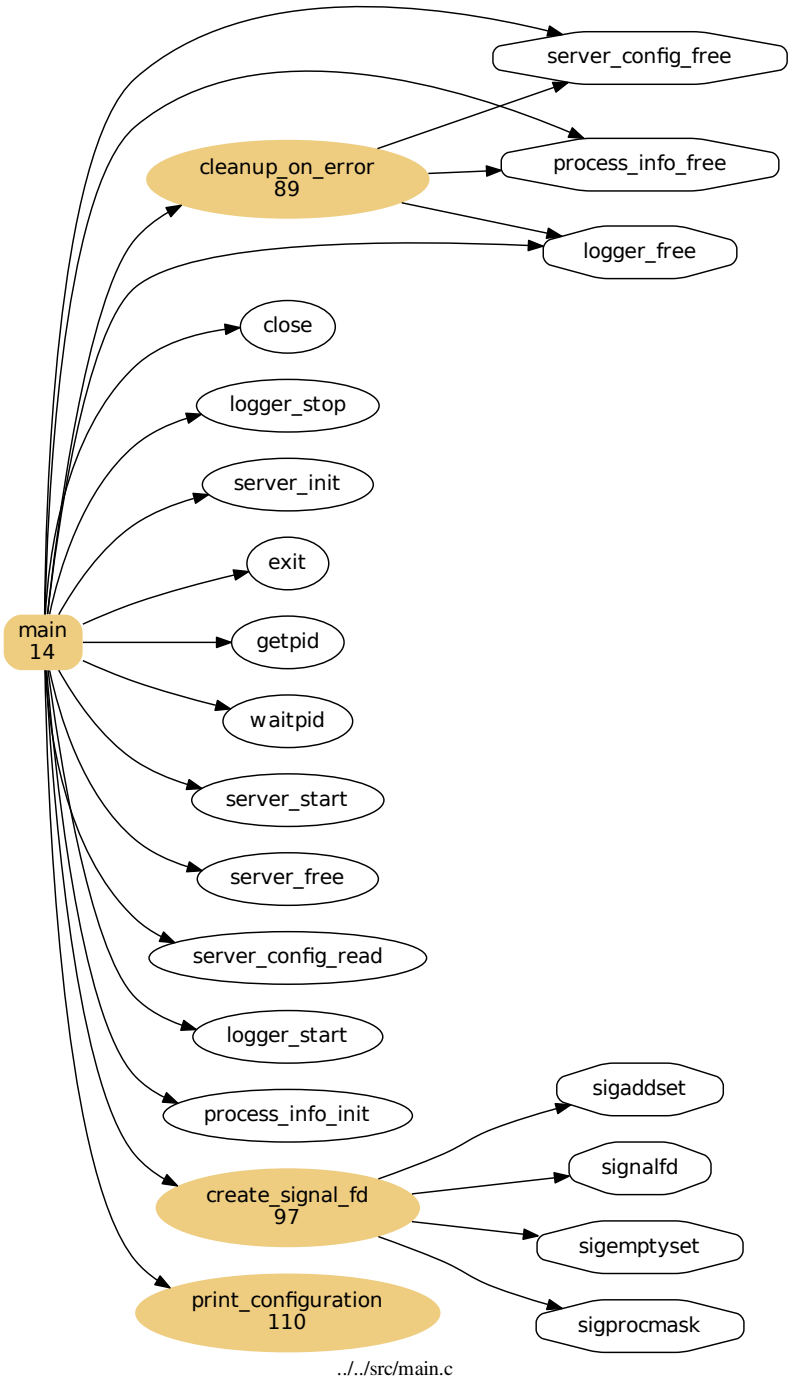

Рис. 3.3: Граф вызовов. Точка входа

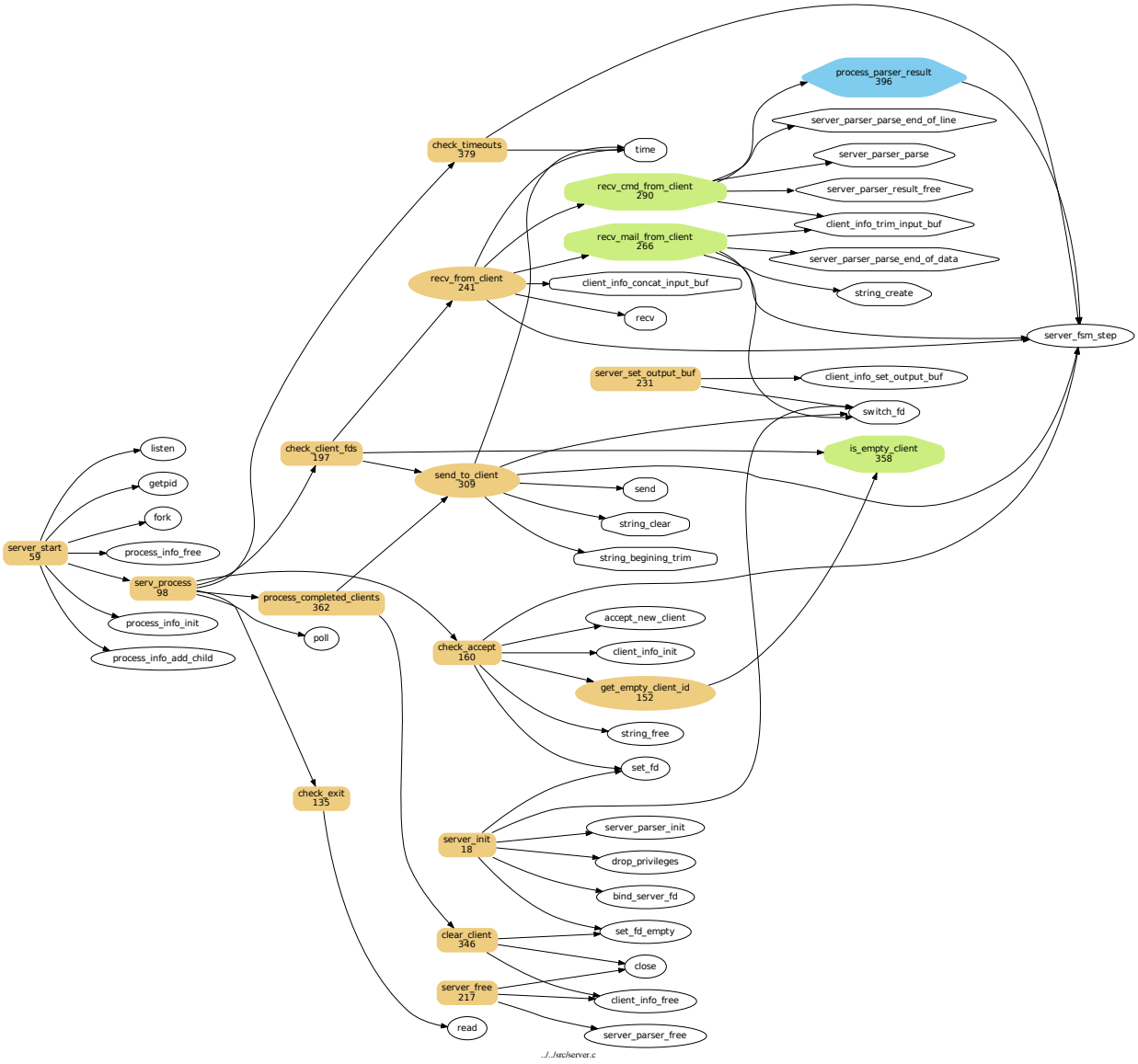

Рис. 3.4: Граф вызовов. Основной код сервера

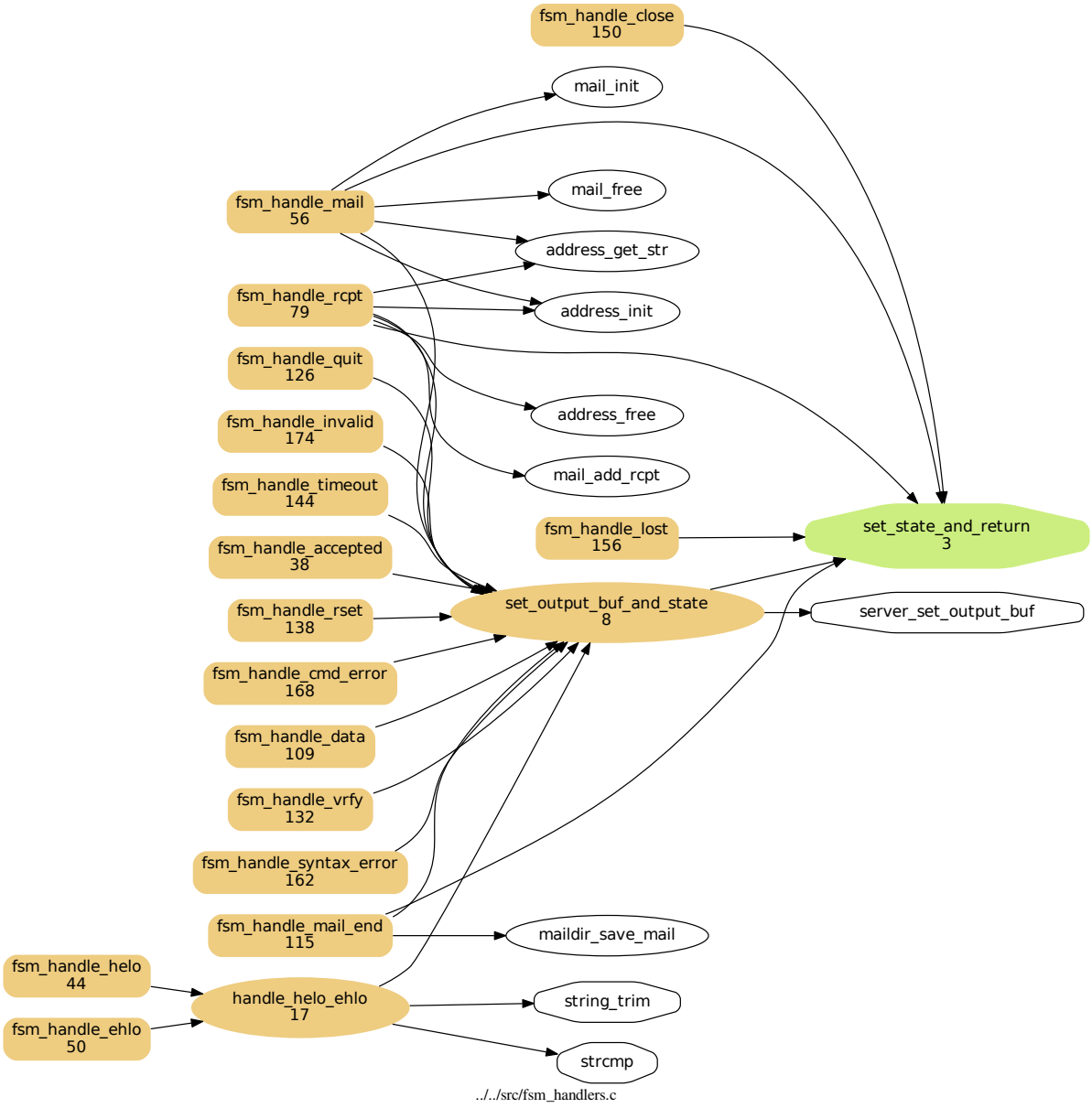

Рис. 3.5: Граф вызовов. Обработчики переходов состояний

3.8 Модульное тестирование

Для модульного тестирования функций сервера в работе используется библиотека CUnit.

Модульными тестами покрыто подавляющее большинство функций из библиотек, размещенных в common, т.к. они широко используются в коде серверной части.

Результаты тестирования привидены на листинге ниже:

```
Starting Server tests..


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


Suite: Address unit tests
  Test: address_get_str test ...passed
  Test: address_get_username test ...passed
  Test: address_init local test ...passed
  Test: address_init remote test ...passed
  Test: address_init trim test ...passed
  Test: address_copy test ...passed
  Test: address_free test for null ...passed
Suite: Strings unit tests
  Test: string_init test ...passed
  Test: string_create test ...passed
  Test: string_expand_memory_to test ...passed
  Test: string_concat test ...passed
  Test: string_begining_trim test ...passed
  Test: string_set simple test ...passed
  Test: string_set test with offset ...passed
  Test: string_set test with set smaller str ...passed
  Test: string_clear test ...passed
  Test: string_free test for null ...passed
  Test: string_trim test with empty string ...passed
  Test: string_trim test with spaces string ...passed
  Test: string_trim test with tabs string ...passed
  Test: string_trim test for string that dont need trim ...passed
  Test: string_trim test with string from spaces and tabs ...passed
Suite: Mail unit tests
  Test: mail_init test ...passed
  Test: mail_add_rcpt test ...passed
  Test: address_init local test rcpts over max ...passed
  Test: mail_free test for null ...passed
Suite: Maildir unit tests
  Test: maildir_get_dir test with local address ...passed
  Test: maildir_get_dir test with remote address ...passed
  Test: concat_dir_and_filename test ...passed
Suite: Parser unit tests
  Test: parser_error_test test ...passed
  Test: parser_helo_test test ...passed
  Test: parser_helo_test_with_addr test ...passed
  Test: parser_ehlo_test test ...passed
  Test: parser_ehlo_test_with_addr test ...passed
  Test: parser_mail_from_test test ...passed
  Test: parser_mail_from_test_empty test ...passed
```

```
  Test: parser_rcpt_to_test test ...passed
  Test: parser_rcpt_to_error_test test ...passed
  Test: parser_data_test test ...passed
  Test: parser_vrfy_test test ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      5      5    n/a      0        0
               tests     40     40     40      0        0
             asserts    325    325    325      0      n/a

Elapsed time =    0.003 seconds
Finish Server tests .
```

3.9 Системное тестирование

Системное тестирование реализовано в виде скрипта на языке Python. Использован SMTP клиент из стандартной библиотеки Python.

Все тесты устроены следующим образом:

- Подготавливаются данные для отправки письма/писем.
- SMTP клиент подключается к серверу и выполняет отправку одного или нескольких писем.
- Производится проверка корректности записи писем (вплоть до посимвольной проверки содержимого файла).

При этом до запуска тестов, автоматически запускается сервер, а после завершения - отправляется сигнал SIGINT, для корректного завершения работы.

Реализованы следующие системные тесты:

- Простой тест с отправкой небольшого письма одному получателю.
- Тест с отправкой письма, размером 1МБ.
- Тест с отправкой письма с несколькими получателями, в т.ч. с локальным/удаленным почтовым доменом.
- Тест с отправкой нескольких писем в рамках одной SMTP сессии.

Результаты системного тестирования приведены ниже:

```
Start SMTP server system tests...
Start SMTP server...
base maildir removed
simple_mail_test OK
many_mails_test OK
one_mb_mail_test OK
many_rcpt_mail_test OK
send Ctrl+C to SMTP server ...
SMTP server is shutdown ...
Finish SMTP server system tests.
```

3.10 Тестирование утечек памяти

Для тестирования утечек памяти использовалась утилита valgrind.

При выполнении основного сценария сборки и тестирования проводится автоматический запуск модульных и системных тестов под valgrind.

Результаты проверки модульных тестов приведены на листинге ниже, утечек памяти не обнаружено:

```
==44925== Memcheck, a memory error detector
==44925== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==44925== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==44925== Command: ./test_build/test
==44925== Parent PID: 44924
==44925==
==44925==
==44925== HEAP SUMMARY:
==44925==     in use at exit: 0 bytes in 0 blocks
==44925==   total heap usage: 968 allocs, 968 frees, 191,962 bytes allocated
==44925==
==44925== All heap blocks were freed -- no leaks are possible
==44925==
==44925== For counts of detected and suppressed errors, rerun with: -v
==44925== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Результаты проверки системных тестов приведены на листинге ниже, утечек памяти не обнаружено:

```
==44935== Memcheck, a memory error detector
==44935== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==44935== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==44935== Command: ./build/server server.ini
==44935== Parent PID: 44934
==44935==
==44937==
==44937== HEAP SUMMARY:
==44937==     in use at exit: 0 bytes in 0 blocks
==44937==   total heap usage: 207 allocs, 207 frees, 81,555 bytes allocated
==44937==
==44937== All heap blocks were freed -- no leaks are possible
==44937==
==44937== For counts of detected and suppressed errors, rerun with: -v
==44937== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==44939==
==44939== HEAP SUMMARY:
==44939==     in use at exit: 0 bytes in 0 blocks
==44939==   total heap usage: 73 allocs, 73 frees, 40,961 bytes allocated
==44939==
==44939== All heap blocks were freed -- no leaks are possible
==44939==
==44939== For counts of detected and suppressed errors, rerun with: -v
==44939== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==44938==
==44938== HEAP SUMMARY:
==44938==     in use at exit: 0 bytes in 0 blocks
==44938==   total heap usage: 289 allocs, 289 frees, 96,778 bytes allocated
==44938==
==44938== All heap blocks were freed -- no leaks are possible
```

```
==44938==
==44938== For counts of detected and suppressed errors, rerun with: -v
==44938== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==44936==
==44936== HEAP SUMMARY:
==44936==     in use at exit: 0 bytes in 0 blocks
==44936==   total heap usage: 51 allocs, 51 frees, 31,796 bytes allocated
==44936==
==44936== All heap blocks were freed -- no leaks are possible
==44936==
==44936== For counts of detected and suppressed errors, rerun with: -v
==44936== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==44935==
==44935== HEAP SUMMARY:
==44935==     in use at exit: 0 bytes in 0 blocks
==44935==   total heap usage: 220 allocs, 220 frees, 5,333,404 bytes
    allocated
==44935==
==44935== All heap blocks were freed -- no leaks are possible
==44935==
==44935== For counts of detected and suppressed errors, rerun with: -v
==44935== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3.11 Тестирование стиля кодирования

В качестве стиля кодирования был выбран Google C++ style [https://google.github.io/styleguide/cppguide.html] с несколькими незначительными изменениями.

Для проверки стиля кодирования использовалась утилита от Google с открытым исходным кодом: cpplint.py [6].

В утилиту были внесены незначительные изменения (для отмены нескольких правил, которые были сочтены излишними). Также применялся флаг для установки максимальной длины строки в 120 символов, вместо стандартных 80.

В ходе автоматической проверки стилевых ошибок не обнаружено, результаты проверки приведены на листинге ниже:

```
Done processing ./fsm_handlers.c
Done processing ./fd_helpers.c
Done processing ./main.c
Done processing ./server.c
Done processing ./tests/unit/server_tests_suites_filler.c
Done processing ./tests/unit/server_tests_main.c
Done processing ./tests/unit/parser_test.c
Done processing config/server_config.c
Done processing client_info/client_info.c
Done processing ../../common/logger/logger.c
Done processing ../../common/strings/strings.c
Done processing ../../common/mail/mail.c
Done processing ../../common/mail/maildir.c
Done processing ../../common/address/address.c
Done processing ../../common/tests/address_test.c
Done processing ../../common/tests/mail_test.c
Done processing ../../common/tests/maildir_test.c
```

```
Done processing ../../common/tests/strings_test.c
Done processing ../../common/tests/common_tests_suites_filler.c
Done processing ../../common/privileges/privileges_dropper.c
Done processing ../../common/process_info/process_info.c
Done processing ./fsm_handlers.h
Done processing ./smtp.h
Done processing ./fd_helpers.h
Done processing ./server.h
Done processing ./tests/unit/server_tests_suites_filler.h
Done processing ./tests/unit/parser_test.h
Done processing config/server_config.h
Done processing client_info/client_info.h
Done processing ../../common/logger/logger.h
Done processing ../../common/strings/strings.h
Done processing ../../common/mail/mail.h
Done processing ../../common/mail/maildir.h
Done processing ../../common/address/address.h
Done processing ../../common/tests/maildir_test.h
Done processing ../../common/tests/strings_test.h
Done processing ../../common/tests/common_tests_suites_filler.h
Done processing ../../common/tests/address_test.h
Done processing ../../common/tests/mail_test.h
Done processing ../../common/privileges/privileges_dropper.h
Done processing ../../common/process_info/process_info.h
Total errors found: 0
```

Заключение

В ходе работы, были выполнены все поставленные задачи, а именно:

1. Изучена и проанализирована предметная область, достоинства и недостатки требуемого способа разделения на процессы и опроса сокетов;
2. Спроектирована серверная часть MTA SMTP;
3. Реализована серверная часть MTA SMTP;
4. Отлажена и протестирована серверная часть MTA SMTP.

Таким образом, цель работы достигнута.

Список литературы

1. http://www.faqs.org/rfcs/rfc5321.html - RFC 5321 (Спецификация протокола SMTP)
2. Статья "select/poll/epoll: практическая разница"[Электронный ресурс] Режим доступа: URL: https://habr.com/ru/company/infopulse/blog/415259/
3. https://habr.com/ru/post/148948/ - libconfig "Конфигурационные файлы. Библиотека libconfig"
4. https://www.pcre.org/ - офф сайт pcre
5. http://cunit.sourceforge.net/documentation.html - документация cunit
6. https://github.com/google/styleguide/tree/gh-pages/cpplint - github cpplint

Листинги