Расчетно-Пояснительная Записка

к курсовой работе на тему:

Клиентская часть MTA SMTP

Москвичев Николай Владимирович

28 декабря 2020 г.

Введение

Данная расчетно-пояснительная записка содержит информацию о протоколе SMTP (англ. **S**imple **M**ail **T**ransfer **P**rotocol) и реализации серверной части MTA (англ. **M**essage **T**ransfer **A**gent) SMTP в рамках курсовой работы.

Задание: необходимо было создать SMTP-клиент, как часть MTA, обеспечивающий удаленную доставку и поддерживающий очереди сообщений. Дополнительные условия:

- Используется системный вызов select();
- Используется несколько рабочий процессов;
- Журналирование в отдельном процессе.

Цель работы: реализовать клиентскую часть MTA SMTP.

Основные задачи:

1. Анализ и изучение протокола SMTP, способов мультиплексирования;
2. Программная реализация SMTP клиента на языке программирования Си под Unix-подобные операционные системы;
3. Тестирование и отладка написанного клиента;
4. Оформление расчетно-пояснительной записки по результатам работы.

Глава 1

Аналитический раздел

1.1 Протокол SMTP

SMTP (англ. **S**imple **M**ail **T**ransfer **P**rotocol) – это широко используемый 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 (2008 год) 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, которая устанавливается 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 Базовые команды SMTP

Каждая команда SMTP начинается с ключевого слова – названия команды, указывающего какую операцию хочет произвести клиент. За ним могут следовать параметры, отделенные пробелом. Конец строк в протоколе SMTP обозначается последовательностью символов "возврат каретки" (\r) и "перевод строки" (\n) - эта последовательность обозначается CRLF. Сервер начинает выполнение команды только получив от клиента строку, завершающуюся последовательностью CRLF.

Обычный ответ SMTP сервера на команды клиента состоит из номера ответа, за которым через пробел следует дополнительный текст. Номер ответа служит индикатором состояния сервера и делится на четыре группы:

– Команда выполнена успешно (код 2xx);

– Промежуточный положительный результат. Команда принята, но сервер ожидает от клиента дополнительные данные для завершения операции (код 3xx);

– Исполнение команды временно невозможно. Команда не может быть выполнена, но проблема может быть устранена (код 4xx);

– Исполнение команды невозможно (код 5xx).

Если ответ состоит из нескольких строк, то каждая из них начинается номером, 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.

Ниже представлен список базовых SMTP-команд:

– EHLO доменное_имя_клиента CRLF - Открывает ESMTP сессию. В ответ на эту команду сервер сообщает, готов ли он к продолжению диалога.

– HELO доменное_имя_клиента CRLF - Открывает SMTP сессию. В RFC 2821 рекомендуется использовать команду HELO, только если программное обеспечение не поддерживает команду EHLO. Отличие этой команды только в том, что она делает невозможным использование расширений ESMTP. Передача почты возможна только после выполнения одной из двух перечисленных выше команд.

– MAIL FROM: <адрес_отправителя> CRLF - Сообщает адрес отправителя письма. Для каждого письма команда MAIL должна быть выполнена только один раз. Адрес может быть оставлен пустым: <>. Команда MAIL может быть выполнена 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 принимает в качестве аргумента только один адрес. Если нужно послать письмо большему числу адресатов, то команду RCPT следует повторять для каждого. Команда RCPT может быть выполнена только после успешного выполнения команды MAIL.

– DATA CRLF - Определяет начало письма. С помощью этой команды серверу передается текст сообщения, состоящий из заголовка и отделенного от него пустой строкой тела сообщения. В ответ на правильно введенную команду DATA сервер сообщает о готовности к приему или об ошибке, если прием сообщения невозможен. Передача самого сообщения заканчивается строкой, состоящей из одной точки. Эта строка не является частью сообщения и удаляется на приемной стороне. Команда DATA может быть выполнена только после успешного выполнения хотя бы одной команды RCPT.

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

– VRFY CRLF - Проверяет наличия указанного в качестве аргумента почтового ящика.

– QUIT CRLF - Закрыть SMTP сессию. Командой QUIT клиент заканчивает диалог с сервером. Сервер посылает подтверждение и закрывает соединение. Получив это подтверждение, клиент тоже прекращает связь.

1.2 Плюсы и минусы использования многопроцессорного подхода обработки подключений при помощи мультиплексирования и системного вызова select()

В рамках задания необходимо реализовать клиентскую часть MTA SMTP с использованием системного вызова select() в нескольких рабочих процессах. К плюсам данного подхода можно отнести:

– Возможность распределения нагрузки по нескольким процессам

– системный вызов select() удобен для простых задач, не требуется работать с динамической памятью, как в poll()

К недостаткам данного подхода относятся:

– Системный вызов select() не способен обрабатывать количество сокетов, большее 1024

– Создание отдельных процессов - дополнительные расходы и нагрузка на систему, особенно в случае создания процессов для обработки небольшого количества писем

1.3 Сущности предметной области

Для SMTP клиента можно выделить две основные сущности, которыми необходимо оперировать (обрабатывать, сохранять):

1. Соединение - текущее активное соединение с информацией о подключении (статус подключения, буферы ввода-вывода, размеры буферов)

2. Письмо - передаваемое клиентом сообщение на сервер с заголовками (отправитель, получатель, адрес получателя и т.д.) и данными (сам текст письма).

Глава 2

Конструкторский раздел

2.1 Конечный автомат состояний сервера

На рис. 2.1 изображен конечный автомат состояний клиентской части MTA SMTP.

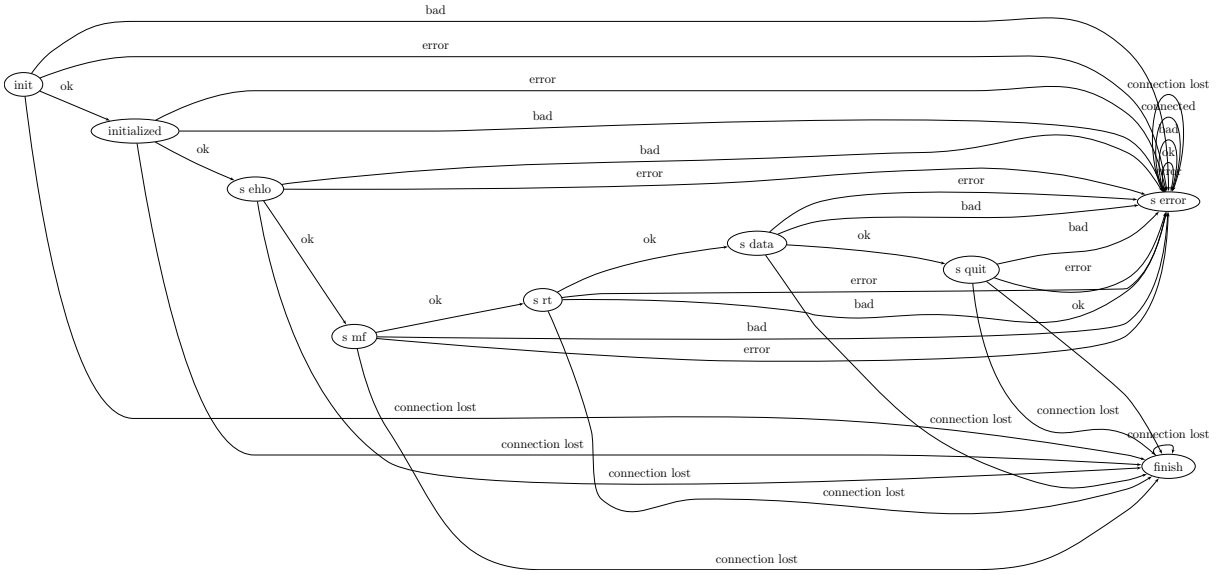

Рис. 2.1 Конечный автомат состояний клиента

Все команды отправки письма по протоколу SMTP выполняются последовательно. В случае возникновения ошибки или разрыва соединения, текущее соединение переходит в состояние ошибки.

2.2 Описание основных структур данных

Основные структуры данных клиента:

– Mail_files - структура с именами файлов в очереди сообщений

– Mail - структура самого письма со всей информацией о отправителе, получателе, адресе сервера получателя и текстом письма

– SMTP_Connection - структура соединения, с буферами приема, отправки, письмом и состоянием соединения

2.3 Обработка соединений в нескольких рабочих процессах

Ниже представлен псевдокод обработки клиентом писем в нескольких процессах:

```
Считать все имена файлов из очереди сообщений

Разделить письма на несколько процессов
Цикл для каждого процесса:
    Запустить процесс, запустить обработку нескольких переданных писем
    Считать письма
    Для каждого получателя создать соединение

    Цикл пока все соединения не перейдут в одно из конечных состояний:

    Ожидание срабатывания select
    Цикл по всем дескрипторам сокетов соединений:
        Если пришло сообщение в файловый дескриптор для выхода
            Выйти

        Если пришло сообщение на сокет соединения
            Обработать ответ сервера
        Если нужно отправить сообщение серверу
            Отослать сообщение серверу

    Проверка завершения работы соединений
```

2.4 Связь основной программы и процесса журналирования

На рис. 2.2 изображена связь между основной программой и процессом журналирования с использованием очереди сообщений.

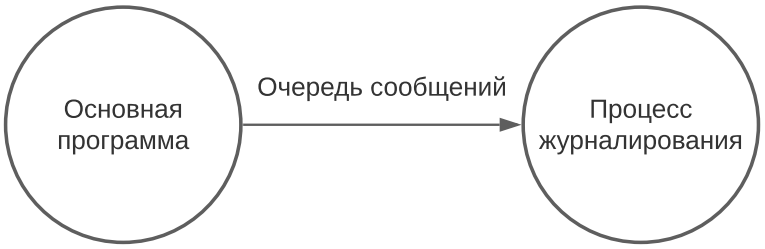

Рис. 2.2 Связь основной программы и процесса журналирования

2.5 Хранение почты

Для хранения локальной почты, то есть предназначенной для клиентов с почтовым доменом сервера, используется упрощенный 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, 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.

Для почты, предназначенной другим почтовым серверам также используется файловая система. Каждое сообщение аналогично хранится в отдельном файле с уникальным именем. Первоначально файл заполняется во временном месте, чтобы клиент не мог его прочитать пока тот не будет полностью записан. Затем файл переносится в отдельную заранее указанную директорию, из которой в дальнейшем клиент считывает письма для их пересылки.

Для того, чтобы имена файлов были уникальными, в их качестве используется слегка измененный формат Unix time (рус. Unix-время) - система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах. В отличие от оригинального Unix time в данной работе вместо секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года, используются миллисекундыя, так как секунд недостаточно для удостоверения уникальности имени файла.

Глава 3

Технологический раздел

3.1 Написание исходного кода

Для написания исходного кода клиентской части MTA SMTP использовался язык C стандарта C99 со стандартными системными библиотеками (ввода/вывода, сокетов, строк, времени, сигналов и т.д.), а так же дополнительными библиотеками и утилитами такими как:

- Autoopts - для автоматической генерации кода чтения параметров командной строки.
- Autofsm - для автоматической генерации конечного автомата состояний сервера.
- CUnit - для модульного тестирования написанного кода приложения.

3.2 Платформы и компиляторы

Разработанное приложение SMTP сервера проверялось на операционной системе Ubuntu-20.04 в подсистеме WSL (англ. **W**indows **S**ybsystem **L**inux) с использованием компилятора GCC и стандарта языка Си C99.

Для связи с сервером и проверки его работоспособности в реальном рабочем окружении использовалось приложение Microsoft Outlook.

3.3 Сборка программы

Сборка системы разделена на две части и описана в файлах Makefile системы сборки make.

Первая часть, верхнеуровневая, описывает описывает сборку всего MTA, включая и клиентскую часть и серверную. Однако также можно указать сборку только одной части. В данном файле указываются флаги для сборки сервера и клиента, а также директория, где будут лежать запускамые бирнарные файлы. В качестве аргумента при вызове make принимается параметр build_type, который может быть равен debug или release. По умолчанию стоит debug. Пример вызовов:

```
make server
make build_type=release
```

Вторая часть относится непосредственно к сборке приложения клиента и запуску тестов и вызывается из верхнеуровнего файла сборки. В этом файле указываются имя результирующего бинарного файла, флаги компилятора, линковщика и папки с исходным кодом программы. Все зависимости make ищет сам в рамках указанных папок.

3.4 Основные функции программы

Данный раздел сгеренерирован при помощи doxygen из части комментированных исходников программы. В файле конфигурации **doxyggen.cfg** был отключён параметр **HAVE_DOT**, поскольку для рисования графов вызовов используется *cflow*.

Здесь описываются основные функции клиента, отвечающие за его работоспособность.

3.4.1 Файл main.c

Main entry point file.

Структуры данных

- struct options_struct

  Client options struct.

Определения типов

- typedef struct options_struct options_t

  Client options struct.

Функции

- static void **close_handler** (int signal)
- int main_loop ()

  Main client loop func.
- options_t **fill_options** ()
- int **validate_options** (options_t options)
- int **main** (int argc, char **argv)

Переменные

- static volatile int **run** = 1
- static logger_t * **logger**
- static int **pipeDescrs** [2] = { 0, 0 }
- options_t **client_options**

3.4.1.1 Подробное описание

Main entry point file.

3.5 Основные структуры программы

Данный раздел аналогично прошлому сгеренерирован при помощи doxygen и описывает основные структуры, использующиеся в коде сервера и на которые ссылаются его основные функции.

3.6 Описание параметров командной строки

Ниже приведено описание используемых сервером обязательных и необязательных параметров командной строки в формате autoopts:

```
flag = {
    name      = home_mode;
    value     = h;
    arg-type  = number;
    arg-range = "0->1";
    max       = 1;
    min       = 1;
};
flag = {
    name      = proc_count;
    value     = p;
    arg-type  = number;
    arg-range = "1->10";
    max       = 1;
    min       = 1;
};
flag = {
    name      = log_dir;
    value     = l;
    arg-type  = string;
    max       = 1;
    min       = 0;
};
flag = {
    name      = mail_dir;
    value     = d;
    arg-type  = string;
    max       = 1;
    min       = 0;
};
```

3.7 Графы вызова функций

Графы вызова функций разбиты на два рисунка. Ниже на рис. 3.1 показаны основные функции, связанные с работоспособностью клиента. На рис. 3.2 в свою очередь показаны функции обработки ответов от сервера и конечного автомата состояний подключения.

Сами графы были созданы с помощью утилит cflow, cflow2dot и dot.

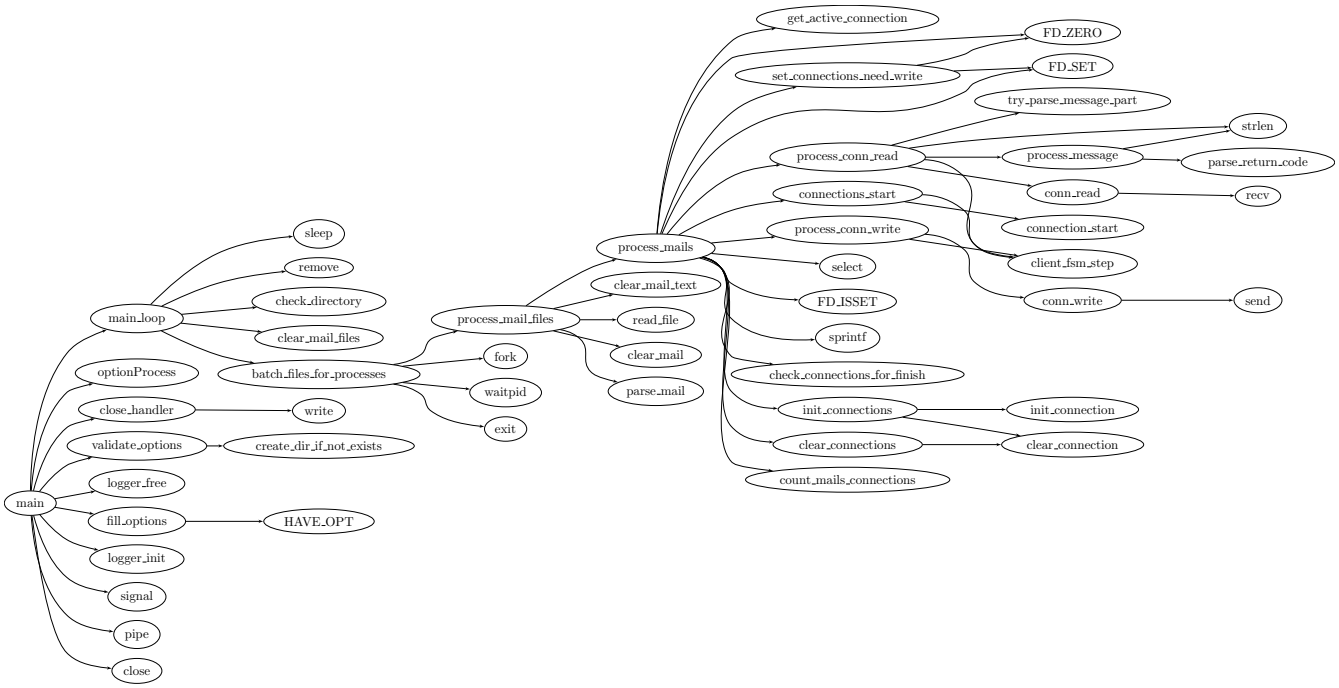

Рис. 3.1 Граф вызовов. Основные функции

3.8 Модульное тестирование

Для модульного тестирования функций сервера в работе используется библиотека CUnit. Тестировались основные модули, такие как: модуль обработки директории, чтения, парсинга писем и получения адресов получателей. Для каждого из них было написано несколько тестов с ассертами. Ниже приведены сводные результаты тестирования:

```
Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      2      2    n/a      0        0
               tests      5      5      5      0        0
             asserts     18     18     18      0      n/a

Elapsed time =    0.002 seconds
Client tests passed
```

3.9 Тестирование утечек памяти

Для тестирования утечек памяти использовалась утилита valgrind. По результатам тестов, при помощи описанного выше скрипта, она не обнаружила никаких потенциально возможных мест с утечками памяти. Окончательные результаты ее работы приведены ниже:

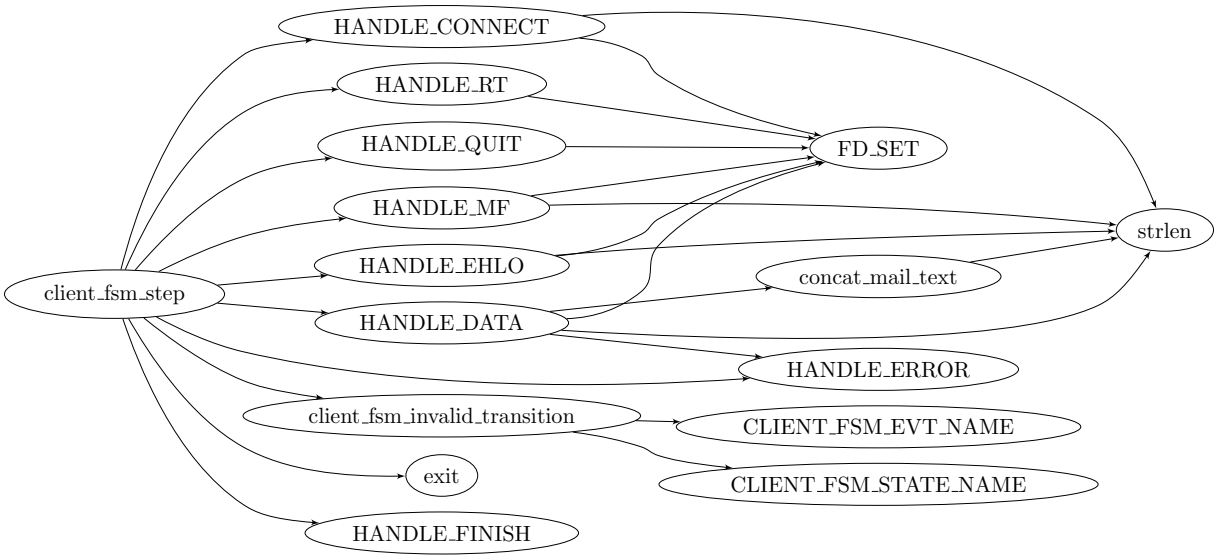

Рис. 3.2 Граф вызовов. Функции обработки команд

```
==1533== LEAK SUMMARY:
==1533==    definitely lost: 0 bytes in 0 blocks
==1533==    indirectly lost: 0 bytes in 0 blocks
==1533==      possibly lost: 0 bytes in 0 blocks
==1533==    still reachable: 4,192 bytes in 4 blocks
==1533==         suppressed: 0 bytes in 0 blocks
```

Заключение

В процессе выполнения работы была написана программная реализация клиентской части MTA SMTP. Был изучен протокол передачи электронной почты SMTP и способы мультиплексирования. Были закреплены и получены навыки в написании сетевых приложений на языке Си для Unix-подобных операционных систем. Было проведено тестирование и отладка разработанного серверного приложения и в итоге по результатам проведенной работы была оформлена расчетно-пояснительная записка.