Оглавление

Введение

0.0.1 Цели и задачи

Цель: Разработать **SMTP-сервер** с использованием рабочих процессов и select.
Задачи:

- Проанализировать архитектурное решение
- Разработать подход для обработки входящих соединений и хранения входящих писем в maildir
- Рассмотреть **SMTP**-протокол
- Реализовать программу для получения писем по протоколу **SMTP**

Глава 1

Аналитический раздел

Предметная область

ER-диаграмма предметной области

Согласно обозначенному протоколу в рамках данной работы, в системе устанавливаются отношения "отправитель - получатель причем отправитель может отправить несколько писем, указав себя в качестве источника сообщения (единственного). Основная единица данных, передаваемая по протоколу - письмо, которое включает в себя отправителя и получателя, причем получателей может быть несколько. Также письмо содержит в себе единственное тело, которое может быть использовано как для последующей передачи, так и для хранения на сервере. Таким образом, в рамках предметной области можно выделить 4 вида сущностей:

- 1. Отправитель
- 2. Получатель
- 3. Письмо
- 4. Тело письма

Зависимость между сущностями предметной области может быть описана следующей диаграммой (1.1):

Сервер

Преимущества и недостатки условия задачи

Согласно условию задачи, в работе сервера предлагается исполльзовать многопроцессную систему. Данный тип системы является самым простым в плане разработки при условии, что на каждую пользовательскую сессию или даже любой пользовательский запрос создается новый процесс. Данная архитектура имеет следующие преимущества:

- 1. Простота разработки. Фактически, мы запускаем много копий однопоточного приложения и они работают независимо друг от друга. Можно не использовать никаких специфически многопоточных API и средств межпроцессного взаимодействия.

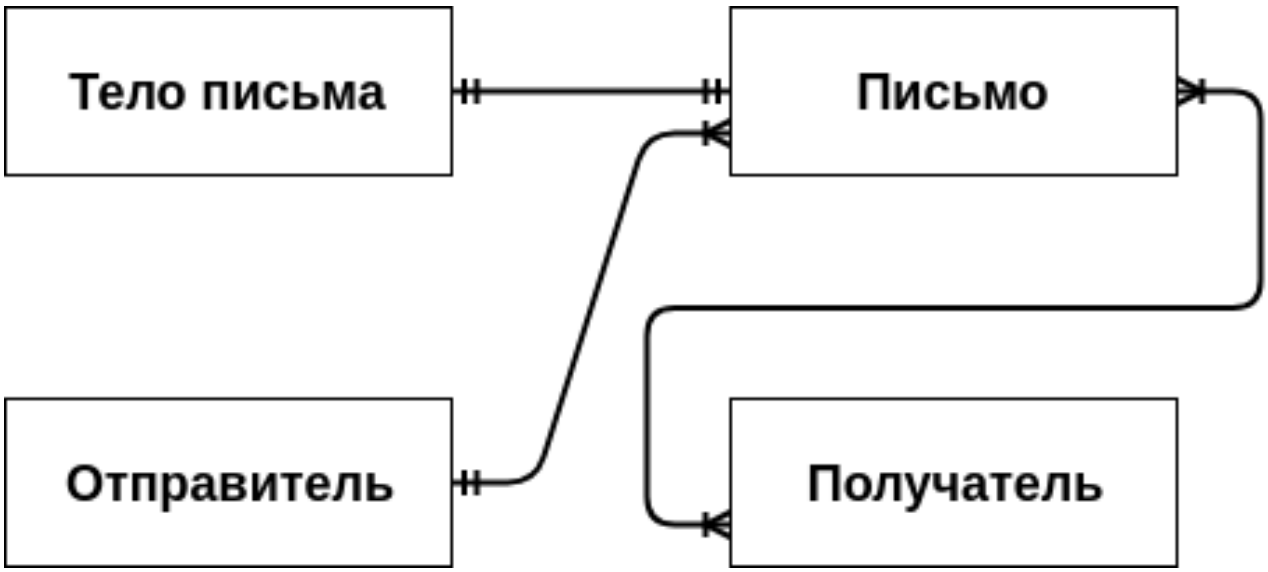

Рис. 1.1: ER-диаграмма сущностей

– 2. Высокая надежность. Аварийное завершение любого из процессов никак не затрагивает остальные процессы.

– 3. Хорошая переносимость. Приложение будет работать налюбой многозадачной ОС

– 4. Высокая безопасность. Разные процессы приложения могут запускаться от имени разных пользователей. Таким образом можно реализовать принцип минимальных привилегий, когда каждый из процессов имеет лишь те права, которые необходимы ему для работы. Даже если в каком-то из процессов будет обнаружена ошибка, допускающая удаленное исполнение кода, взломщик сможет получить лишь уровень доступа, с которым исполнялся этот процесс.

При этом данная архитектура имеет следующие недостатки:

– 1. Далеко не все прикладные задачи можно предоставлять таким образом. Например, эта архитектура годится для сервера, занимающегося раздачей статических HTMLстраниц, но совсем непригодна для сервера баз данных и многих серверов приложений.

– 2. Создание и уничтожение процессов – дорогая операция, поэтому для многих задач такая архитектура неоптимальна.

Поэтому для минимизации операций создания и уничтожения процессов предлагается архитектурное решение, представляющее собой пул процессов, создаанных заранее. Это позволит фиксировать число операций создания процесса. При этом слушающие сокеты сервера должны наследоваться каждым создаваемым процессом. Это делается для решения проблемы распределения соединений между процессами одной группы. В ходе исследования предметной области и реализации сервера с заданной архитектурой, было выяснено, что открытые файловые дескрипторы процесса не могут быть переданы посредством очередей сообщений. (при передаче открытого дескриптора возникает ошибка EBADF). Однако система с наследованием имеет недостаток в отсутствии возможности распределения соединений между процессами - соединение принимает тот, кто быстрее успел.

Глава 2

Конструкторский раздел

2.1 Конечный автомат состояний сервера

Конечный автомат состояний сервера представлен на Рис. 2.1

2.1.1 Синтаксис команд протокола

Ниже приведен формат команд сообщений протокола в виде регулярных выражений

1. **EHLO**: *EHLO [w+]+*
2. **HELO**: *HELO [w+]+*
3. **MAIL**: *MAIL FROM <[\w]+@[\w]+[\.\w]+>*
4. **RCPT**: *RCPT <[\w]+@[\w]+[\.\w]+>*
5. **DATA**: *DATA*
6. **VRFY**: *VRFY [w+]+*
7. **RSET**: *RSET*
8. **QUIT**: *QUIT*

Представление данных

Ниже приведены диаграммы представления данных в системе - логическая и физическая.

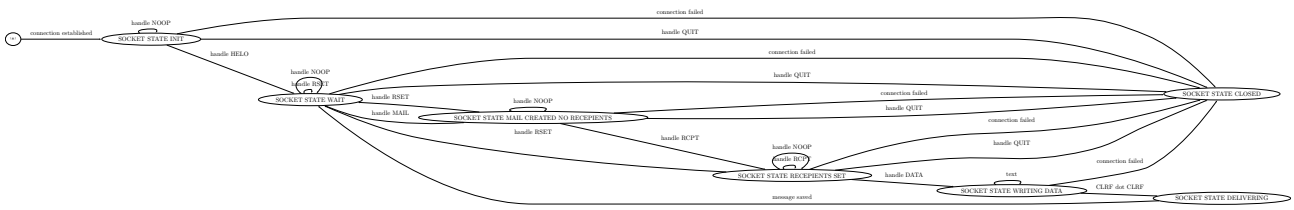

Рис. 2.1: Состояния сервера

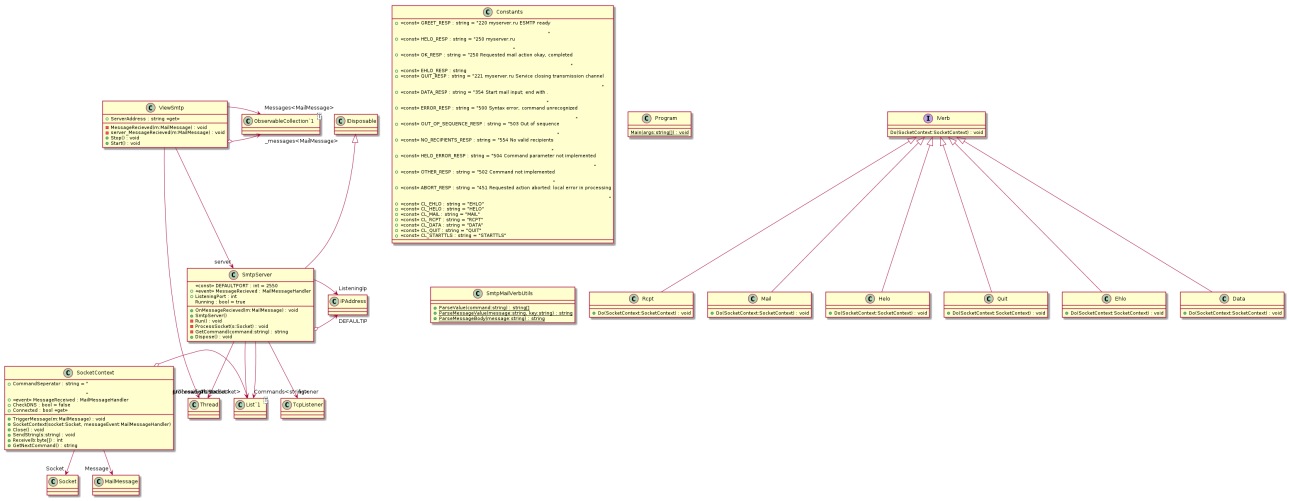

Рис. 2.2: Логическая диаграмма сущностей

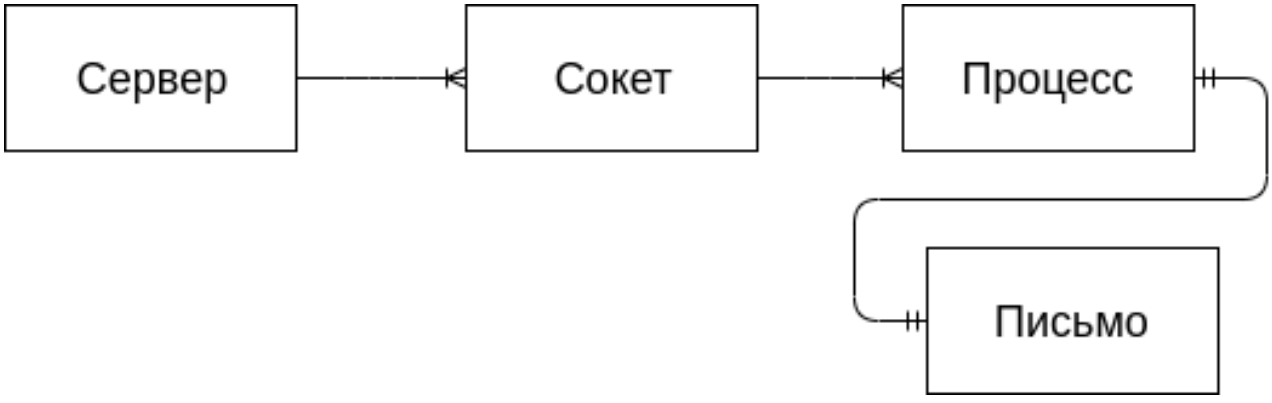

Рис. 2.3: Физическая диаграмма сущностей

2.1.2 Алгоритм обработки соединений

```
ПОКА (процесс_работает == 1)
     Добавить дескрипторы слушающих сокетов в сет читателей
     Добавить дескрипторы клиентских сокетов в сет читателей
     Ожидать соединения на одном из сокетов (время = 5с)
     ЕСЛИ есть запрос ТО
       ДЛЯ каждого слушающего сокета
          ЕСЛИ действие на одном из слушающих сокетов ТО
              Принять новое соединение
              Инициализировать новый сокет
              Отправить приветствие
              Установить неблокирующий режим для принятого соединения
          КОНЕЦ ЕСЛИ
       КОНЕЦ ДЛЯ
       ДЛЯ каждого клиентского сокета
          ЕСЛИ действие на одном из клиентских сокетов ТО
             Обработать действие в соответствии с протоколом
          КОНЕЦ ЕСЛИ
       КОНЕЦ ДЛЯ
     КОНЕЦ ЕСЛИ
КОНЕЦ ПОКА
```

Глава 3

Технологический раздел

3.1 Сборка программы

Сборка программы описана в файле *Makefile* системы сборки *make*. Рис. 3.1 нагенерили *makefile2dot*.

3.2 Графы вызова функций

Поскольку функций много, графы вызовов разбиты на два рисунка. На рис. 3.2 показаны основные функции, на рис. 3.3 – функции обработки команд.

Графы созданы с помощью *nDepend*.

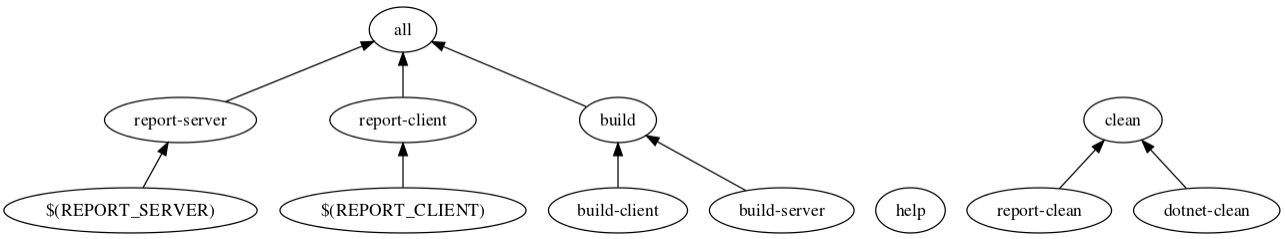

Рис. 3.1: Сборка программы

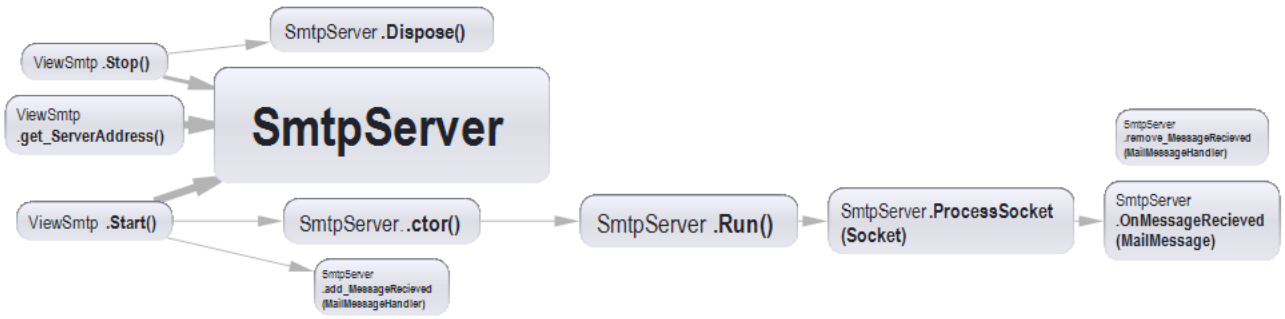

Рис. 3.2: Граф вызовов, основные функции

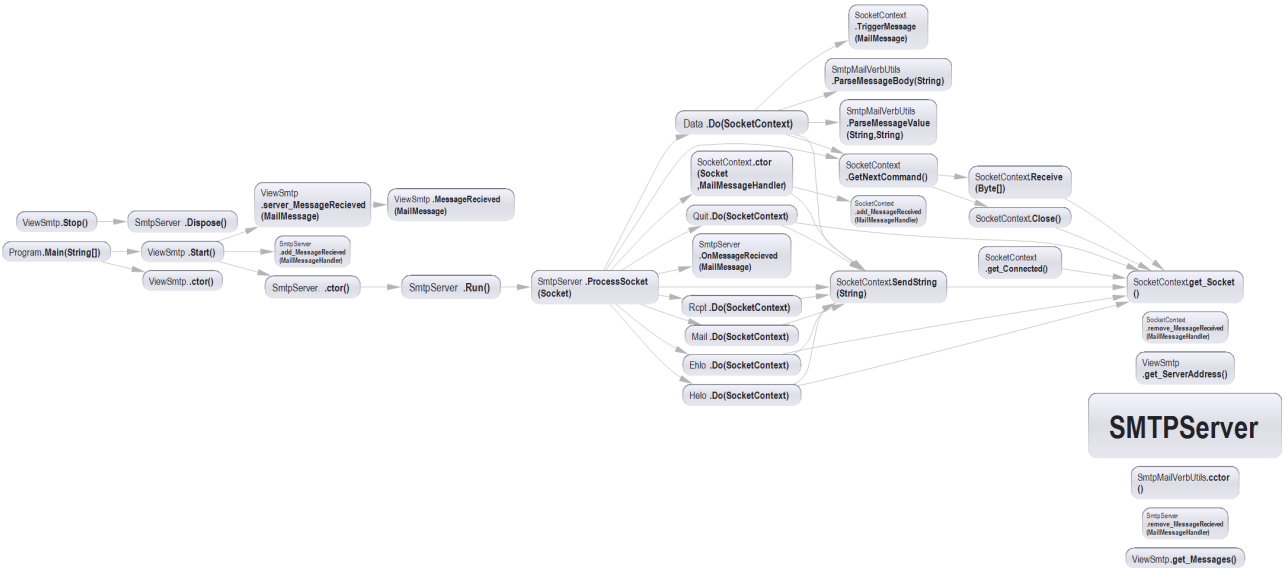

Рис. 3.3: Граф вызовов, функции обработки команд

3.3 Тестирование

Ниже приведён отчет о модульном тестировании.

```
Server: 504 Command parameter not implemented
HELO Test: success
Server: 504 Command parameter not implemented
EHLO Test: success
Server: 250 Requested mail action okay, completed
MAIL Test1: success
Server: 500 Syntax error, command unrecognized
MAIL Test2: success
Server: 500 Syntax error, command unrecognized
RCPT Test1: success
Server: 250 Requested mail action okay, completed
RCPT Test2: success
Server: 250 Requested mail action okay, completed
RCPT Test3: success
Server: 252 Cannot VRFY user, but will accept message and attempt delivery
VRFY Test: success
Server: 503 Out of sequence
DATA Test1: success
Server: 554 No valid recipients
DATA Test2: success
Server: 354 Start mail input; end with .
250 Requested mail action okay, completed
DATA Test3: success
Server: 354 Start mail input; end with .
500 Syntax error, command unrecognized
DATA Test4: success
Server: 250 Requested mail action okay, completed
RSET Test1: success
Server: 250 Requested mail action okay, completed
RSET Test2: success
Server: 221 myserver.ru Service closing transmission channel
QUIT Test: success

RunTime 00:00:00.14
```

Выводы

В рамках предложенной работы нами были реализованы два компонента современного SMTP-сервера, взаимодействующих между самой по самописному протоколу в соответствии со стандартами RFC. В ходе работы реализованы следующие задачи:

1. Проанализировали архитектурное решение
2. Разработали подход для обработки входящих соединений и хранения входящих писем в maildir
3. Рассмотрели **SMTP**-протокол
4. Реализовали программу для получения писем по протоколу **SMTP**
5. Рассмотрели работу с неблокирующими сокетами и их взаимодействие
6. Разработали системы, работающие в многозадачном режиме, столкнувшись с проблемами взаимодействия процессов, которые были решены в ходе работы
7. Познакомились с утилитами автоматической сборки и тестирования