Оглавление

Введение

0.0.1 Цели и задачи

Цель: Разработать **SMTP-сервер** используя вызов select и рабочие процессы. Журналирование в отдельном процессе.

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**
- Реализовать метод журналирования в отдельном процессе**SMTP**

Глава 1

Аналитический раздел

Предметная область

Согласно обозначенному протоколу в рамках данной работы, в системе устанавливаются отношения "отправитель - получатель причем отправитель может отправить письмо нескольким получателям. 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 3 вида сущностей:

- 1. Сервер
- 2. Клиент
- 3. Письмо

Сервер

Преимущества и недостатки условия задачи

Согласно условию задачи, в работе сервера предлагается исполльзовать многопроцессную систему. 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.
- 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.

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. **RSET**: *RSET*
7. **QUIT**: *QUIT*

Представление данных

Ниже приведена диаграмма представления данных в системе

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
```

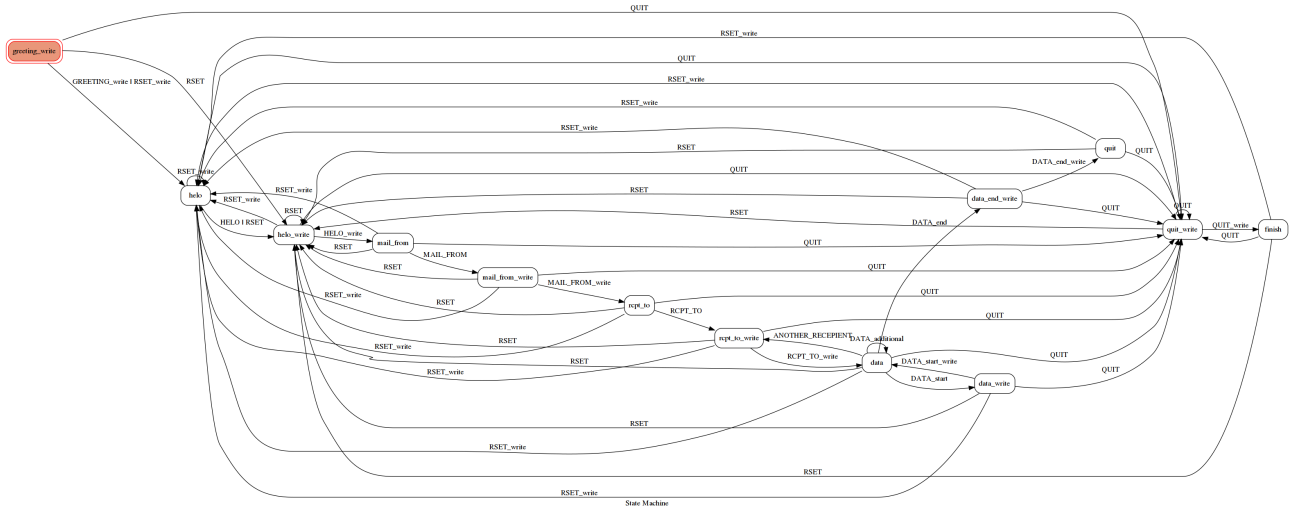

Рис. 2.1: Состояния сервера

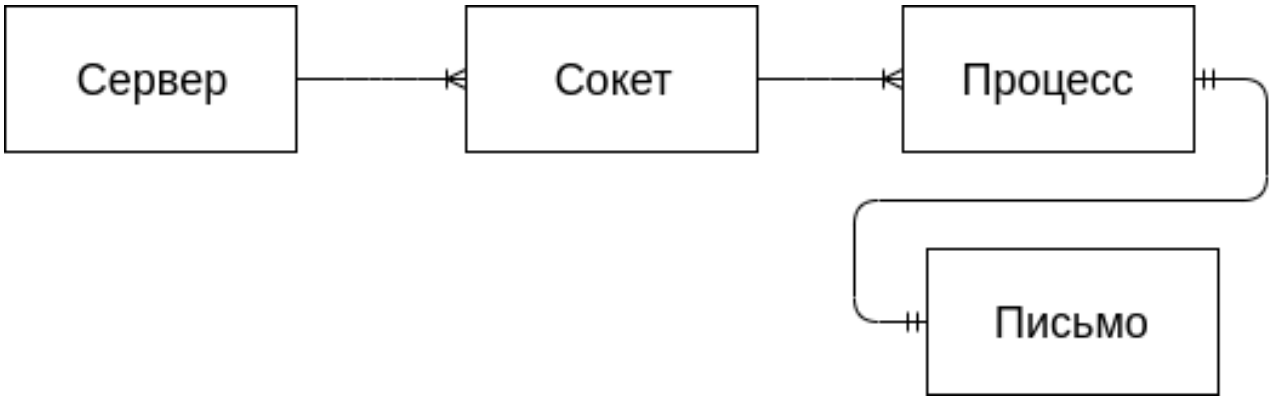

Рис. 2.2: Физическая диаграмма сущностей

```
          ЕСЛИ действие на одном из слушающих сокетов ТО
              Принять новое соединение
              Инициализировать новый сокет
          КОНЕЦ ЕСЛИ
          ЕСЛИ действие на одном из клиентских сокетов ТО
           Обработать действие в соответствии с протоколом
          КОНЕЦ ЕСЛИ
        КОНЕЦ ДЛЯ
        ДЛЯ каждого сокета записи
          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. Изображение с конечным автоматом генерируется срествами библиотеки transitions.

3.2 Графы вызова функций

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

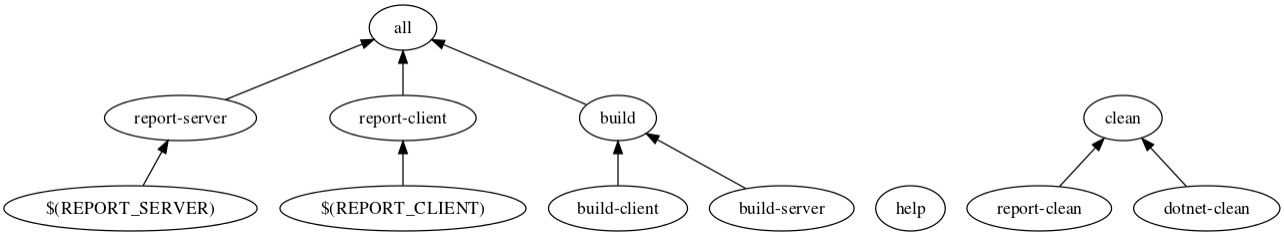

Рис. 3.1: Сборка программы

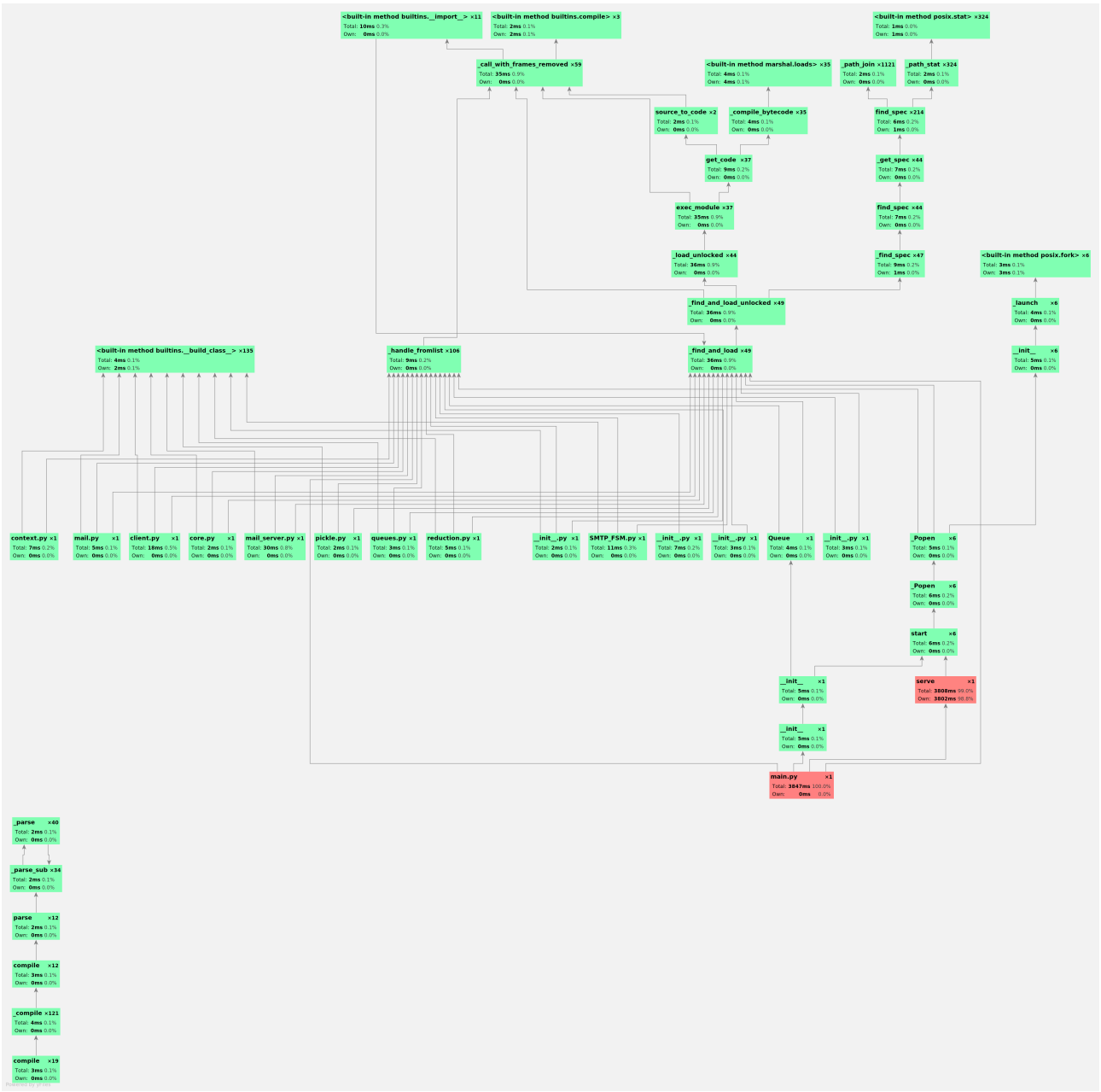

Рис. 3.2: Граф вызовов

3.3 Тестирование

Ниже приведён отчет о модульном тестировании.

```
Launching pytest with arguments /home/v/programming/c/smtp-course-work in /home/v/progra

============================= test session starts ==============================
platform linux -- Python 3.7.5, pytest-5.3.2, py-1.8.0, pluggy-0.13.1 -- /home/v/progran
cachedir: .pytest_cache
rootdir: /home/v/programming/c/smtp-course-work
plugins: cov-2.8.1
collecting ... collected 13 items

server/tests/test_code.py::test_connection_to_server
server/tests/test_code.py::test_send_mail_return
server/tests/test_integr.py::test_send_simple_message_smtplib
server/tests/test_integr.py::test_send_simple_message_socket
server/tests/test_integr.py::test_send_message_two_recepients
server/tests/test_maildir.py::test_maildir_duplicate
server/tests/test_maildir.py::test_maildir_oversize
server/tests/test_re.py::test_hello_pattern
server/tests/test_re.py::test_email_pattern
server/tests/test_re.py::test_email_patternf
server/tests/test_re.py::test_data_end_patternf
server/tests/test_re.py::test_data_end_patternf_should_match
server/tests/test_sock.py::test_send_simple_message_socket

============================== 13 passed in 2.50s==============================
```

Выводы

В рамках предложенной работы нами был реализован SMTP-сервер 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. Реализовали метод журналирования в отдельном процессе**SMTP**
7. Разработали систему работающую в многозадачном режиме
8. Познакомились с утилитами автоматической сборки и тестирования

В ходе работы получены следующие навыки:

1. проектирования реализации сетевого протокола по имеющейся спецификации;
2. реализации сетевых приложений на языке программирования;
3. реализации сетевой службы без создания нити на каждое соединение;
4. автоматизированного системного тестирования ПО сетевой службы;
5. групповой работы с использованием системы контроля версий;