Московский государственный технический университет
имени Н. Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Программное обеспечение ЭВМ
и информационные технологии»

Расчётно-пояснительная записка
к курсовому проекту по курсу «Проектирование вычислительных сетей»
на тему

Создание почтового сервера (MTA), работающего по протоколу SMTP

Выполнили: студент ИУ7-31М Пенской И. С. _____

студент ИУ7-31М Чиж И. С. _____

Руководитель: Корниенко В. В. _____

Москва, 2019

Содержание

1. Введение

Курсовой проект в рамках курса "Проектирование вычислительных сетей"ориентирован на проектирование, реализацию и тестирование почтового сервера (MTA). Поскольку задания образуют пары (прием почты-передача почты), то проект выполняется совместно.

1.1. Сервер

Задание. 2 Используется вызов select и рабочие процессы. Журналирование в отдельном процессе. Нужно проверять обратную зону днс. **Цели и задачи** Цель: Разработать **SMTP-сервер** с использованием рабочих процессов и select. Задачи:

1) Проанализировать архитектурное решение
2) Разработать подход для обработки входящих соединений и хранения входящих писем в maildir
3) Рассмотреть **SMTP**-протокол
4) Реализовать программу для получения писем по протоколу **SMTP**
5) Провести тестирование разработанного ПО

1.2. Клиент

Задание. 2 Используется вызов select и рабочие процессы. Журналирование в отдельном процессе. Пытаться отправлять все сообщения для одного MX за одну сессию. **Цели и задачи** Цель: Разработать **SMTP-сервер** с использованием рабочих процессов и select. Задачи:

1) Проанализировать архитектурное решение
2) Разработать подход для организации логирования
3) Рассмотреть **SMTP**-протокол
4) Реализовать программу для отправки писем по протоколу **SMTP**
5) Провести тестирование разработанного ПО

2. Аналитическа часть

2.1. 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 загаловок, содержащий имя отправителя, получателя и дополнительную информацию о письме (тема, дата), 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 следущие сущностей:

1) Отправитель
2) Получатель
3) Письмо
4) Загаловок письма
5) Тело письма

Зависимость между сущностями предметной области может быть описана следующей диаграммой (1):

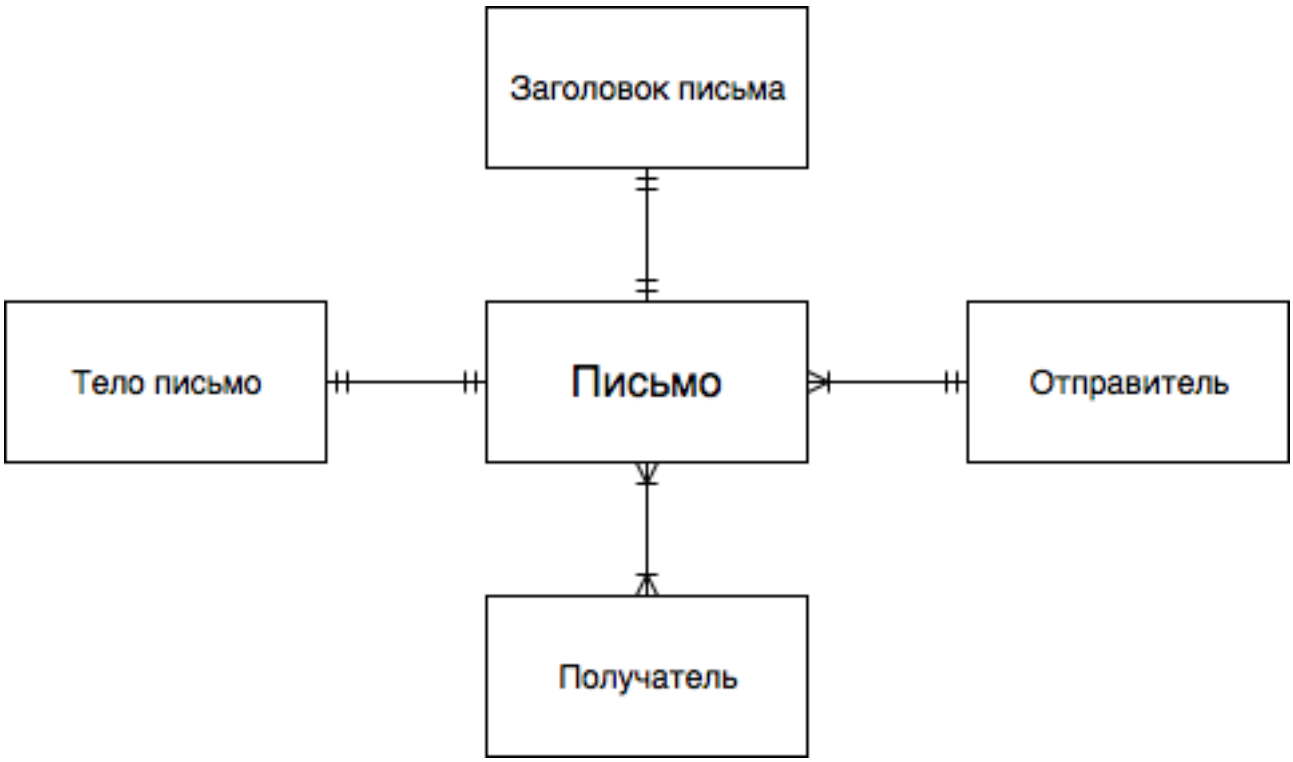


Рис. 1. ER-диаграмма сущностей

2.2. Преимущества и недостатки поставленного целью подхода

Согласно условию задачи, в работе клиента/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.

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) Усложнение процесса разработки и реализации алгоритмов

2) Увеличение вероятности ошибки

3) 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 клиента/сервера должны наследоваться каждым создаваемым процессом. Это делается для решения проблемы распределения соединений между процессами одной группы.

2.3. Способ балансирования нагрузки

Поскльку конечный алгоритм является параллельным, необходимо стараться обеспечить равномерную загрузку обработчиков. Для этого существует множество алгоритмов балансировки нагрузки. Примеры алгоритмов планирования:

1) случайный выбор;

2) круговой выбор;

3) планирование с приоритетами.

Самым простым и достаточно эффективным для поставленны целей реализации является алгоритм кругового выбора. Именно он и будет исспользоваться для определения следующего работника.

3. Конструкторский раздел

3.1. Конечный автомат состояния SMTP

На рисунке (2)представлен конечный автомат, описывающие возможные состояния SMTP - клиента и переходы между этими состояниями. На рисунке (3)представлен конечный автомат, описывающие возможные состояния SMTP - сервера и переходы между этими состояниями.

3.2. Синтаксис команд протокола

Протокол SMTP в данной реализации использует следующие команды:

- **EHLO** <имя клиента> *EHLO [w+]+* - команда приветствия, с которой начинается установка соединения (возможно оповещение о расширениях сервера);
- **HELO** <имя клиента> *HELO [w+]+ - команда приветствия, с которой начинается установка соединения;*
- ***MAIL FROM:*** *<from address> <[\w]+@[\w]+[\w]+> - от кого пришло письмо (обратный путь);*
- ***RCPT TO:*** *<to addredd> <[\w]+@[\w]+[\w]+> - кому предназначается письмо, команда отправляется для каждого адреса в отдельности (прямой путь);*

– ***DATA*** *DATA - начало отправки данных письма (заголовков и тела);*

– ***NOOP*** *NOOP - пустая операция;*

– ***RSET*** *RSET - сброс (прерывание текущей почтовой транзакции);*

– ***QUIT*** *QUIT - завершение сеанса SMTP.*

3.3. Принцип работы клиента

Работу клиента можно разделить на 2 отдельные сущности: работа родительского процесса, работа дочерних процессов. В родительском процессе решено производить следующие действия

1) Циклический поиск новых писем

2) Распределение писем по дочерним процессам

Работу дочернего процесса можно описать следующим образом

1) Создание массива подключений

2) Получение писем от родительского процесса

3) Распределение писем по подключениям

4) Обработка писем в соответствии с порядком, определенным конечным автоматом клиента

3.4. Принцип работы сервера

Работа сервера делится на 4 этапа.

1) Запуск процесса логирования

2) Создание и инициализация слушающих сокетов

3) Создание дочерних процессов, наследующих слушающие сокеты

4) Совместная, независимая работа процессов по обработке входящих соединений (конкурентное ожидание)

3.5. Конечный автомат состояния сообщения(клиент)

На рисунке (4)представлен конечный автомат, описывающие возможные состояния сообщений клиента и переходы между этими состояниями.

3.6. Алгоритм поключения к сокету

На рисунке 5 представлен алгоритм подключения по сокету к SMTP - серверу.

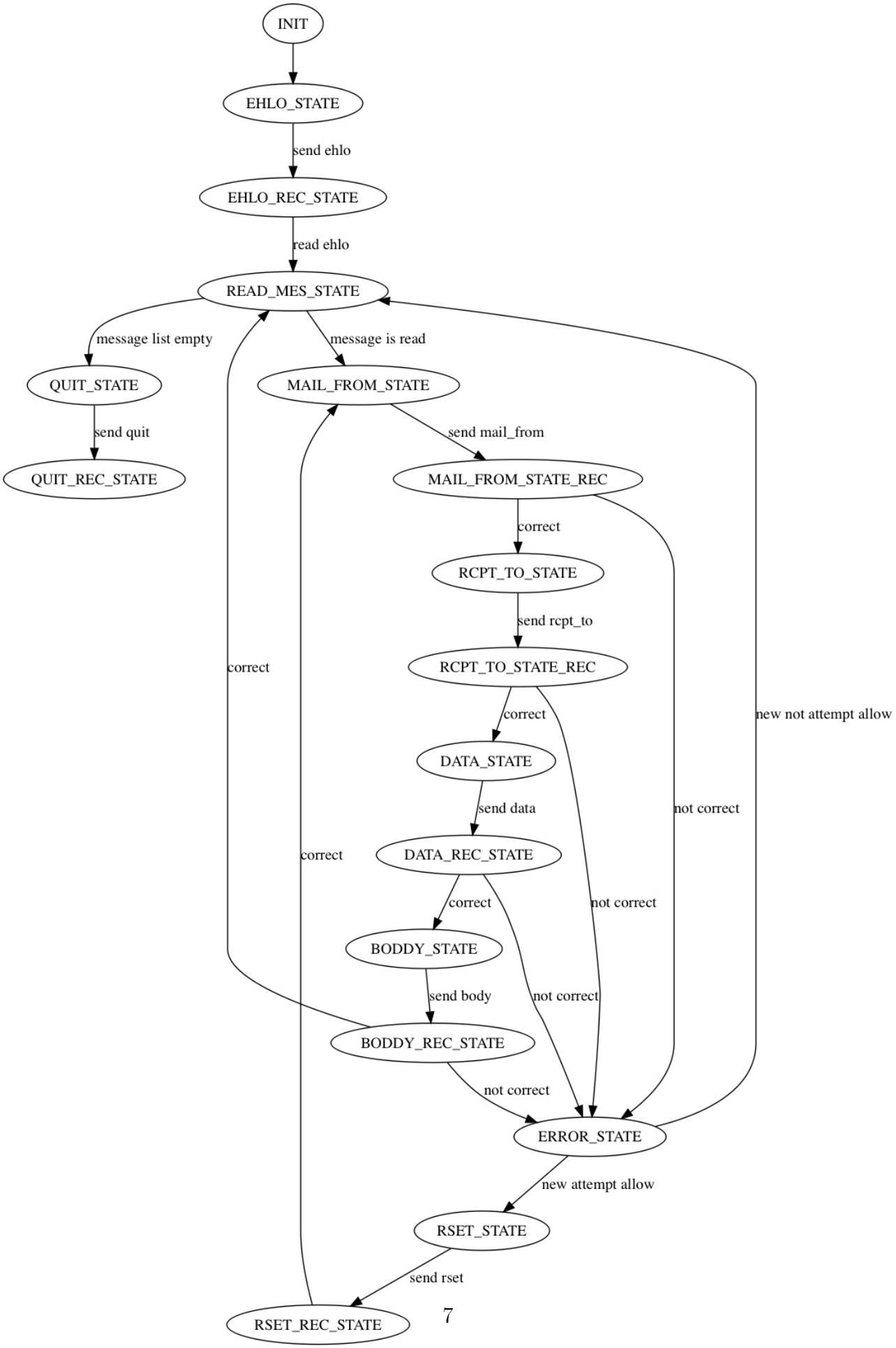


Рис. 2. Конечный автомат клиента

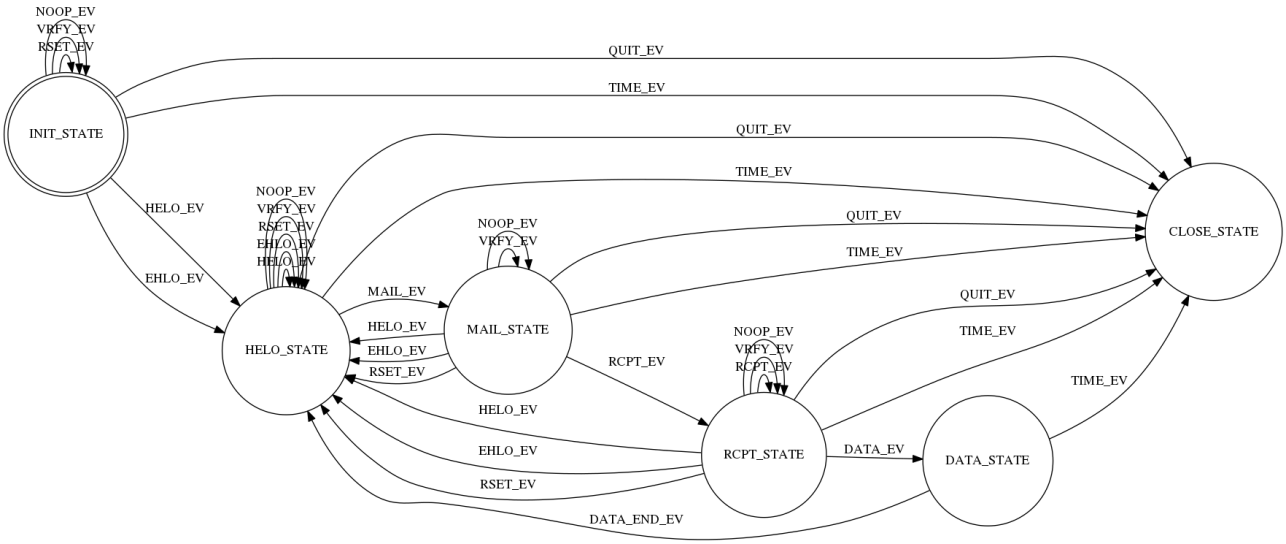


Рис. 3. Конечный автомат сервера

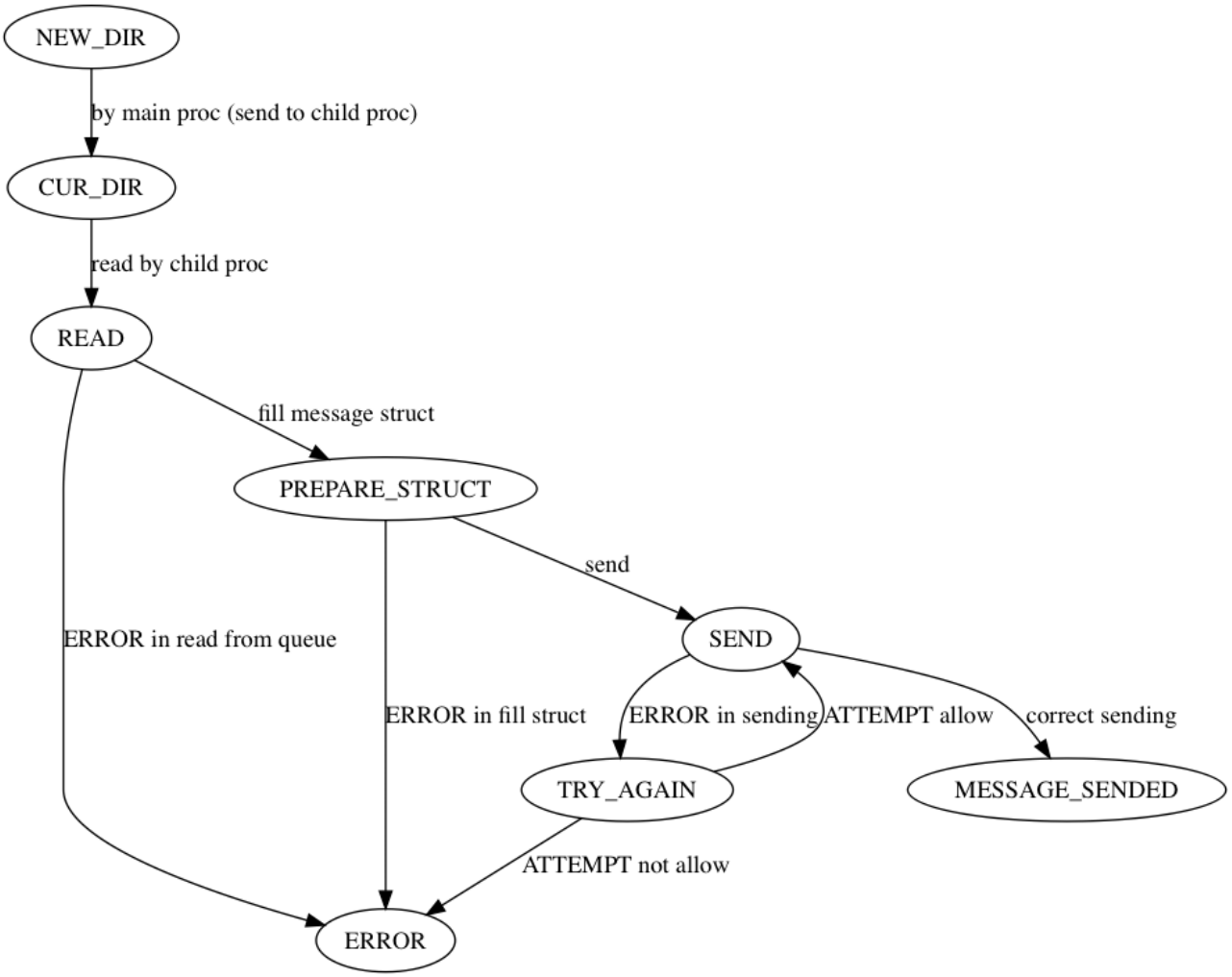


Рис. 4. Конечный автомат сервера

3.7. Ипользуемые структуры данных

Диаграмма связей между основными структурами данных представлена на рисунке 6

3.8. Графы вызова функций

3.9. Выделенные подсистемы

В ходе анализа возможной архитектуры проектируемого решения было выделено 3 подситемы:

- файловая подсистема, которая является внешней по отношению к разрабатываемому приложению;
- подсистема работы с сокетами и соединениями;
- подсистема обработки данных по протоколу SMTP.

3.10. Организация взаимодействия между процессами для SMTP-клиента

Исходя из поставленной хадачи, необходимо организовать взаимодействие основного процесса процесса, осуществляющего логирование, и дочерних процессов. Для этой цели был выбран механизм обмена сообщениями. Каждый процесс создает очередь сообщение с именем, равным ID процесса.

3.11. Способ хранения почты

Для хранения почты и разделения доступа к файлам используется формат **Maildir**. Поддиректория **еьз** - используется SMTP-сервером как временное хранилище для новых писем. Поддиректория **new** - содержит новые, только что поступившие письма. Поддиректория **cur** - содержит письма, взятые в обработку клиентом. Как только в **new** появляются новые файлы, подсистема мониторинга входящей почты перемещает их в поддиректорию **cur**. Откуда, в свою очередь, файлы берутся на обработку.

4. Технологический раздел

4.1. Платформа разработки

Данное решение разрабатывалось на ОС Linux. Использовался дистрибутив Ubuntu 16.04.

4.2. Выбор языка программирования

Для написания программы использовался язык C стандарта gnu99. Программы, написанные таким образом, переносимы на уровне исходных текстов на любую ОС, поддерживающую компилятор GCC с опцией -std=gnu99.

4.3. Файлы конфигурации

Файл конфигурации клиента содержит:

1) корневой каталог для очередей сообщений new;

2) корневой каталог для очередей сообщений cur;

3) имя файла журнала (лог);

4) общее время на попытки отправить письмо;

5) минимальное время между попытками повтора;

6) максимальное число рабочих процессов.

Файл конфигурации сервера содержит:

1) каталог maildir;

2) имя файла журнала (лог);

3) порт

4) id пользователя

5) id группы

6) имя хоста

7) максимальное число дочерних процессов.

4.4. Процесс генерации кода

Сборка программы, как для сервера, так и для клиента, осуществляется с помощью Makefile (GNU make), объединяющего в себе несколько целей:

1) сборка исполняемого файла сервера/клиента (по умолчанию)

2) сборка и запуск тестов

3) сборка отчета

Makefile клиента

```
    CFSM_DIR = cfsm_lib
CFLAGS = -Wall -Werror -g3
OBJ = main.o logging.o message_list.o smtp_fsm.o programm_manager.o read_message.o m
client: $(OBJ)
    gcc -L/usr/lib/ -o $@ $^ -lrt -lconfig -lresolv
%.o: %.c *.h
    gcc -c $< $(CFLAGS)
fsm.c: fsm.fsm
    cd $(CFSM_DIR) && make --silent
```

```
    $(CFSM_DIR)/cfsm/cfsm -t $(CFSM_DIR)/cfsm/ -d fsm.fsm -o fsm.c
    $(CFSM_DIR)/cfsm/cfsm -t $(CFSM_DIR)/cfsm/ -g fsm.fsm -o fsm.dot
test_style: apply_style
    diff -u <(cat *) <(clang-format -style=file ./*)
apply_style:
    clang-format -i -style=file ./*
clean:
    rm -f client $(OBJ)
    cd $(CFSM_DIR) && make clean --silent
.PHONY: clean
```

Makefile сервера

```
    EXE = notsmtp
    UNIT_EXE = test_notsmtp

    SRC_DIR = src
    OBJ_DIR = obj
    INCLUDE_DIR = include
    TEST_DIR = test

    AUTOOPTS_DIR = src/shell_opt
    AUTOOPTS_CFLAGS = # -DTEST_NOTSMTP_OPTS `autoopts-config cflags`
    AUTOOPTS_LDFLAGS = $(shell autoopts-config ldflags)

    # AUTOFSM_DIR = src/fsm
    FSM_DIR = src/fsm
    CFSM_DIR = cfsm_lib
    CFSM_CFLAGS = -std=gnu99

    SRC = $(wildcard $(SRC_DIR)/*.c)

    TEST_FILES =\
    $(wildcard $(TEST_DIR)/*.c)\
    $(SRC_DIR)/client.c\
    $(SRC_DIR)/config.c\
    $(SRC_DIR)/logging.c\
    $(SRC_DIR)/server.c\
    $(SRC_DIR)/smtp_cmd_utils.c\
    $(SRC_DIR)/smtp_cmd.c\
    $(SRC_DIR)/smtp.c\
    $(SRC_DIR)/socket_utils.c\
    $(SRC_DIR)/socket.c\
    $(SRC_DIR)/sds.c\
    $(FSM_DIR)/fsm.o

    OBJ = $(SRC:$(SRC_DIR)/%.c=$(OBJ_DIR)/%.o)
```

```
CC = gcc

CPPFLAGS += -I$(INCLUDE_DIR)
CFLAGS += -std=gnu99 -Wall -Werror
LDFLAGS +=
LDLIBS += -lconfig -lrt -lpcre -lfiredns $(AUTOOPTS_LDFLAGS)

.PHONY: all clean

all: clean $(EXE)

$(EXE): $(AUTOOPTS_DIR)/shell_opt.o $(FSM_DIR)/fsm.o $(OBJ)
    $(CC) $(LDFLAGS) $^ $(LDLIBS) -o $@

$(OBJ_DIR)/%.o: $(SRC_DIR)/%.c
    $(CC) $(CPPFLAGS) $(CFLAGS) -c $< -o $@

# AUTOOPTS #

$(AUTOOPTS_DIR)/shell_opt.c: $(AUTOOPTS_DIR)/shell_opt.def
    cd $(AUTOOPTS_DIR) && SHELL=/bin/sh autogen shell_opt.def
    mv -f $(AUTOOPTS_DIR)/shell_opt.h $(INCLUDE_DIR)

$(AUTOOPTS_DIR)/shell_opt.o: $(AUTOOPTS_DIR)/shell_opt.c
    $(CC) -c -I$(INCLUDE_DIR) $(AUTOOPTS_CFLAGS) -o $@ $<

# FSM #

$(FSM_DIR)/fsm.o: $(FSM_DIR)/fsm.c
    $(CC) -c -I$(INCLUDE_DIR) $(CFSM_CFLAGS) -o $@ $<

$(FSM_DIR)/fsm.c: $(FSM_DIR)/fsm.fsm
    cd $(CFSM_DIR) && make --silent
    $(CFSM_DIR)/cfsm/cfsm -t $(CFSM_DIR)/cfsm/ -d $(FSM_DIR)/fsm.fsm -o fsm.c
    mv -f fsm.c $(FSM_DIR)
    mv -f fsm.h $(INCLUDE_DIR)
    $(CFSM_DIR)/cfsm/cfsm -t $(CFSM_DIR)/cfsm/ -g $(FSM_DIR)/fsm.fsm -o $(FSM_DI

##############################################################################

# AUTOFSM #

# $(AUTOFSM_DIR)/server_fsm-fsm.c: $(AUTOFSM_DIR)/server_fsm.def
#  cd $(AUTOFSM_DIR) && autogen -L$(AUTOFSM_DIR) server_fsm.def
#  mv -f $(AUTOFSM_DIR)/server_fsm-fsm.h $(INCLUDE_DIR)
```

```
# $(AUTOFSM_DIR)/server_fsm-fsm.o: $(AUTOFSM_DIR)/server_fsm-fsm.c
#  $(CC) -c -I$(INCLUDE_DIR) -o $@ $<

################################################################################

test_units: $(EXE) $(UNIT_EXE)
    ./$(UNIT_EXE)

$(UNIT_EXE): $(TEST_FILES)
    $(CC) -I$(INCLUDE_DIR) $(CFLAGS) $^ -lconfig -lrt -lpcre -lcunit -o $@

test_memory: test_units
    valgrind ./$(UNIT_EXE)

test_system:
    echo "MAKE SYSTEM TESTS"

test_style:
    echo "MAKE STYLE TESTS"

tests: test_units test_memory test_system test_style

report:
    echo "MAKE REPORT"

clean:
    $(RM) $(OBJ)
    $(RM) $(AUTOOPTS_DIR)/*.o $(AUTOOPTS_DIR)/*.c $(AUTOOPTS_DIR)/*.h
    $(RM) $(FSM_DIR)/*.o $(FSM_DIR)/*.c $(FSM_DIR)/*.h $(FSM_DIR)/*.dot
    $(RM) $(INCLUDE_DIR)/fsm.h $(INCLUDE_DIR)/shell_opt.h
    cd $(CFSM_DIR) && make clean --silent
    $(RM) $(UNIT_EXE)
```

Генерация отчета по данному проекту осуществляется в единственном экземпляре и файл отчета содержит в себе информацию как о сервере, так и о клиенте. Для этой цели был создан отдельный Makefile с целями для генерации непосредственно отчета, а также включаемых данных (диаграммы, рисунки и др.). Генерация конечных автоматов происходит с использованием библиотеки **cfsm**

4.5. Тестирование

Для модульных тестов используется библиотека cunit. Модульные тесты находятся в поддиректории test. Тестирование утечек памяти проводилось помощью программы valgrind. Вывод результатов тестирования утечек памяти:

Вывод запуска модульных тестов на клиенте

```
parallels@parallels-vm:/media/psf/Home/xcode/smtp_client/smtp_client$ make test_unit
cc -I. -Wall -Werror -g3 test/test_smtp.c test/test_utils.c test/main.c read_message
./test_smtp

     CUnit - A Unit testing framework for C - Version 2.1-0
     http://cunit.sourceforge.net/

Suite: suite_test
  Test: read_file_correct ... passed
  Test: read_file_incorrect ... passed
  Test: fsm_correct ... passed
  Test: fsm_incorrect ... passed
  Test: connection_correct ... passed
  Test: connection_incorrect ... passed
  Test: ehlo_correct ... passed
  Test: ehlo_incorrect ... passed
  Test: ehlo_incorrect_send ... passed
  Test: mail_send ... passed
  Test: mail_send_rset_fsm ... passed

--Run Summary: Type      Total     Ran  Passed  Failed
               suites        1       1     n/a       0
               tests        11      11      11       0
               asserts      81      81      81       0
```

5. Заключение

В рамках предложенной работы нами были реализованы два компонента почтового сервера (MTA), взаимодействующих между собой по протоколу SMTP в соответствии со стандартами RFC 5321 и смежными. В ходе работы реализованы следующие задачи:

1) Сервер- Проанализировано архитектурное решение
2) Сервер- Разработан подход для обработки входящих соединений и хранения входящих писем в maildir
3) Сервер- Рассмотрен и реализован **SMTP**-протокол
4) Сервер- Реализована программа получения почты по протоколу **SMTP**
5) Клиент- Разработан подход для обработки писем в maildir
6) Клиент- Рассмотрен и реализован **SMTP**-протокол
7) Клиент- Проанализированы способы получения и обработки **MX**-записей

8) Клиент- Реализована программу для отправки почты по протоколу **SMTP**

9) Общее- Рассмотрена работа с неблокирующими сокетами и их взаимодействие поверх протокола TCP

10) Общее- Разработаны подходы взаимодействия процессов при их совместной работе

11) Общее- Произведена работа с утилитами для сборки и тестирования ПО

12) Общее- Разработаны сценарии тестирования ПО

6. Список использованной литературы

1) Курс лекций по дисциплине "Проектирование вычислительных сетей". Корниенко В.В. М-.2018

2) Курс лекций по дисциплине "Проектирование вычислительных сетей". Рязанова Н.Ю М-.2015

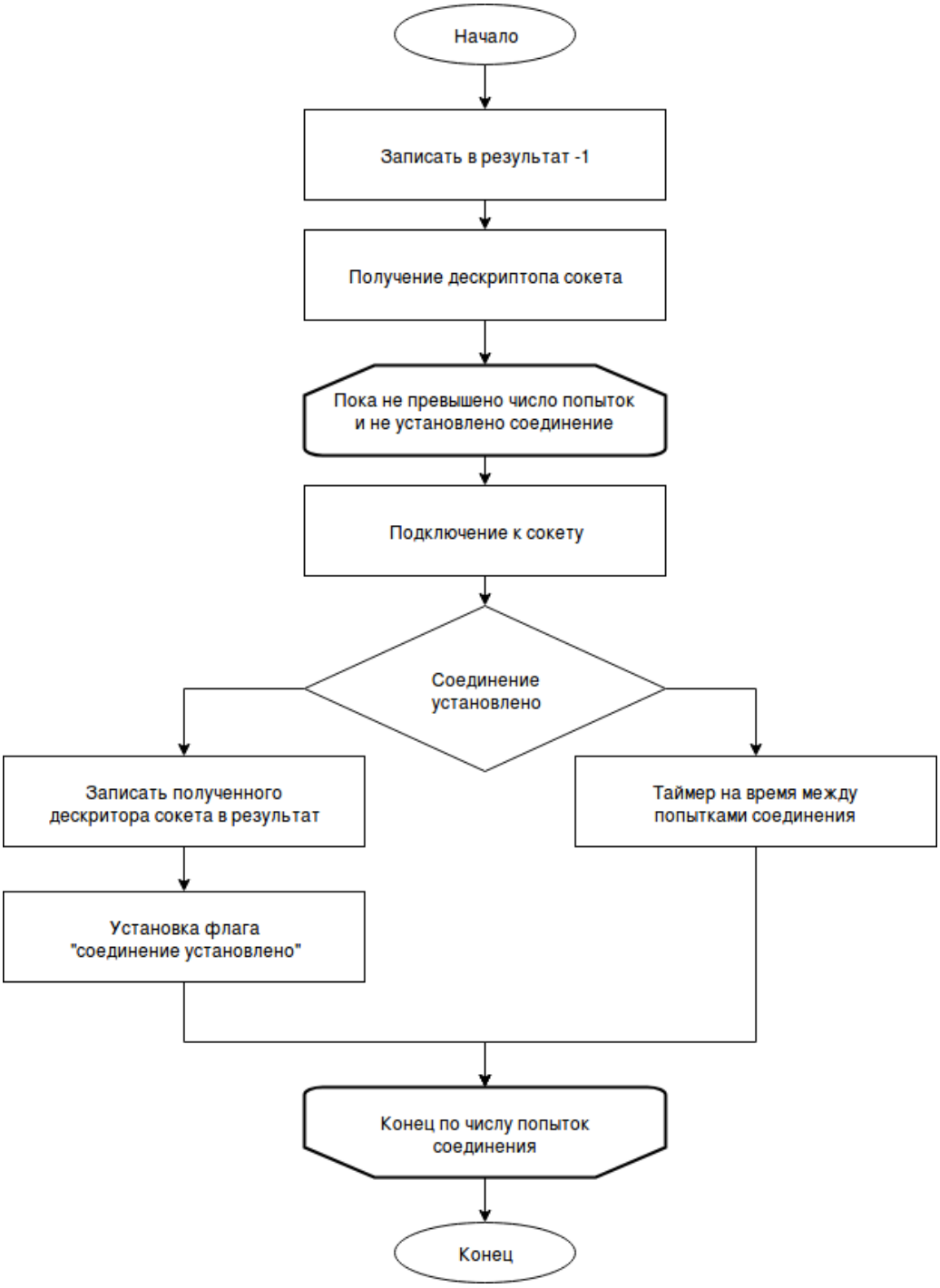


Рис. 5. Алгоритм работы SMTP - клиента

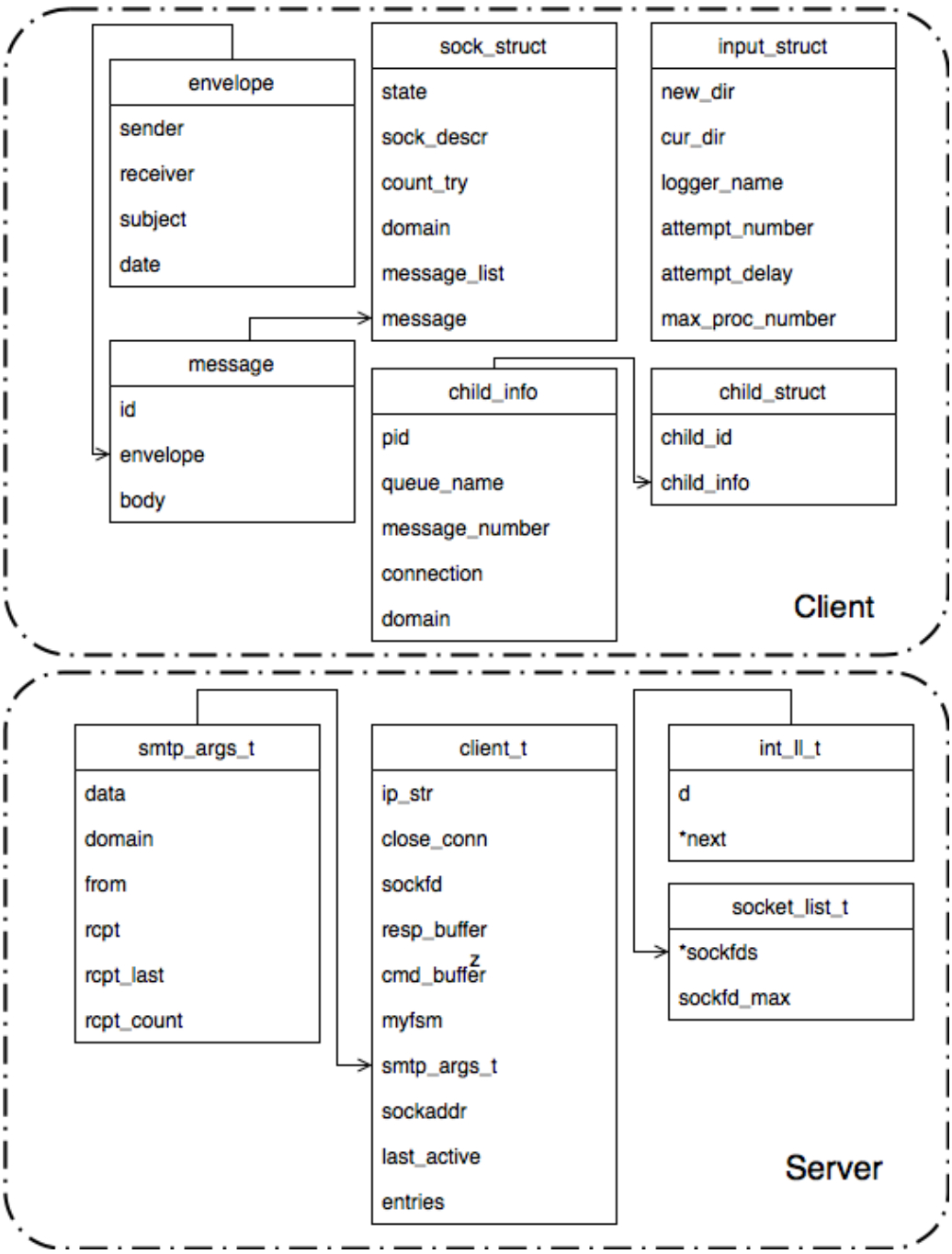
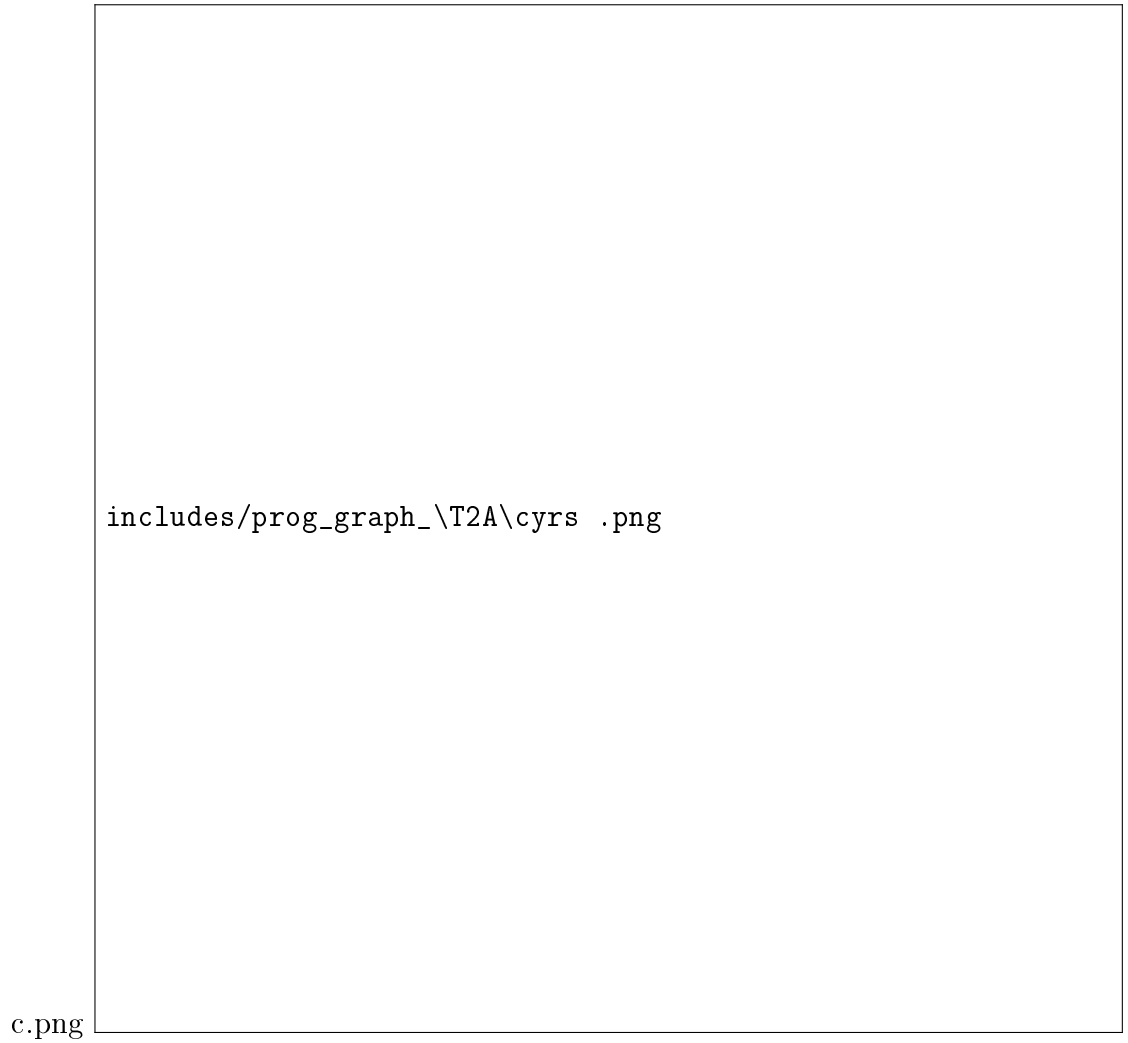


Рис. 6. Диаграмма связей между основными структурами данных

Рис. 7. Граф вызова функций сервера

includes/prog_graph_\T2A\cyrs .png



c.png

Рис. 8. Граф вызова функций клиента