РПЗ ПВС

Ершов Константин ИУ7-37

22 января 2016 г.

Оглавление

1 ВВЕДЕНИЕ

1.1 Задание на курсовую работу

В рамках задания к курсовой работе по курсу "Протоколы вычислительных сетей"реализовать SMTP-клиент для отправки сообщений электронной почты. Вариант задания - 21подрузамевает использование вызова poll(), для организации обработки множества входящих соединений в размках одного процесса, и рабочих процессов. Также необходимо предусмотреть возможность ведения журнала работы программы. Журналирование должно быть реализованно в отдельном процессе.

1.2 Цель и задачи работы

Целью данной курсовой работы является разработка SMTP-клиента для отправки сообщений электронной почты.

Задачи работы представлены ниже:

- Создание SMTP-клиента (как части MTA), обеспечивающего удаленную доставку и поддерживающего очереди сообщений.
- Все варианты предполагают обработку нескольких исходящих соединений в одном потоке выполнения (т.е., одном процесс или одном потоке).
- На один удалённый MX надо создавать не более одного сокета (допустимый вариант — на один удалённый IP не более одного сокета).
- Следует использовать отдельную очередь собщений для каждого MX.

2 АНАЛИТИЧЕСКИЙ РАЗДЕЛ

В данном разделе представлено описание предметной области, выделены сущности предметной области, представлен способ разделения нагрузки между рабочими процессами.

2.1 Предметная область

В соответствии с выданным вариантом, необходимо реализовать SMTP-сервер для получения сообщений электронной почты с использованием poll() и рабочих процессов.

Поскольку обычно ввод-вывод осуществляется с использованием блокирующих системных вывозов, то при попытке считывания или записи данных программа будет блокироваться на этой операции и ожидать появления некоторых данных. Если программа производит работу над локальным файлом, то блокировка, обычно, не занимает долго времени. Однако при использовании сетевых сокетов, когда программа использует вызовы accept(), recv(), write() ожидание данных может занимать продолжительное время, поскольку возможна работа в условиях медленного соединения.

Для решения описанной выше проблемы возможно использование вызова poll(). При использовании poll() регистрируются множества файловых дескрипторов ожидающих записи или чтения. Когда появляется возможность записи или чтения в отслеживаемом дескрипторе, программа может выполнить требуемую операцию, проверив состояние дескриптора в соответствующем множестве.

Использование poll() позволяет производить обработку нескольких активных соединений в рамках одного процесса, и особенно эффективено при небольшом числе одновременно активных сокетов (передача в ядро и обратно по три байта на сокет). Также poll() выл портирован на множество различных ОС, что позволяет использовать его в переносимых программах.

Ограничением работы с poll() являются максимальное количество отслеживаемых файловых дескрипторов равное 1024 (Linux). В некоторых ОС возможно увеличение этого значения изменив параметр FD_SETSIZE, однако если необходима высокая переносимость программы, то этого значения может не хватать.

Получение и обработка запросов выполняется в рабочих процессах программы. Каждый процесс обрабатывает новые подключения, принимает и отправляет сообщения клиентам.

Многопроцессный подход к организации обработки сетевых соединений имеет неколько преимуществ перед однопроцессным, т.к. возникновение сбоев в работе одного процесса не приводит к завершению всей программы. При наличии реализации однопроцессной программы с FSM архитектурой, перевод ее на работу с несколькими процессами достаточно прост. Также с увеличением числа процессоров такие программы хорошо масштабируются. С использованием FSM архитектуры логики обработки заявок такой подход может быть высокоэффективным. Однако в некоторых ОС (Solaris, HP_UX), где обработка потока программы производится быстрее процесса, эффективнее было бы использование потоков, вместо процессов.

2.2 Сущности предметной области

Необходимо выделить основные сущности указанной предметной области. К ним относятся:

- **Почтовый клиент** – отправитель письма. Им может выступать как почтовый клиент пользователя, так и другой SMTP-сервер. Имеет адрес, с которого устанавливается соединение с сервером. Отправитель формирует письмо и отправляет его на SMTP-сервер;
- **Письмо** – сообщение, передаваемое отправителем SMTP-серверу. Содержит поля TO - адрес получателя, FROM - адрес отправителя и текст письма. Письмо формируется отправителем и поступает на SMTP-сервер;
- **SMTP-сервер** – сервер, получающий письма от отправителей. Имеет атрибуты: адрес сервера и порт. Служит для получения писем от отправителей;

В этой системе можно выделить следующие субъекты и соответствующие им прецеденты:

SMTP-сервер - инициирует один из прецедентов: получение данных письма, создание файла копии письма, завершение соединения.

Прецеденты:

- **Получение данных письма** – при установлении подключения соединения к SMTP-серверу, клиент производит отправку данных письма(или нескольких писем в рамках одной сессии): данные об отправителе, получателе (или нескольких получателях), текст письма. В ответ на передаваемые серверу команды, он отправляет сообщения о результатах выполнения;

- **Создание файла копии письма** – после получения всех необходимых данных письма, SMTP-сервер осуществляет создание файла письма, который в дальнейшем используется SMTP-клиентом для пересылки почты. Данный прецедент может включать создание директорий пользователя, при проведенной ранее аутентификации;

- **Завершение соединения** – В случае возникновения ошибок в обмене сообщениями с клиентом, или по завершению обмена, SMTP-сервер разрывает соединение с клиентом, отправляя сообщение о разрыве соединения.

2.3 Разделение нагрузки между рабочими процессами

При разработке сервера с использованием нескольких рабочих процессов возникает задача распределения нагрузки по обработке входящих соединений между ними.

В данной работе было предложено равномерное распределение нагрузки между рабочими процессами. При установке нового подключения его обработкой должен заниматься тот процесс, который обрабатывает наименьшее число активных соединений. Такой подход обеспечит равномерное распределение обработки соединений между рабочими процессами.

Алгоритм распределения нагрузки между рабочими процессами представлен следующим образом:

- При инициализации программы главным процессом создается указанное число рабочих процессов;

- Каждому процессу назнается его ID - уникальный идентификатор. Главный процесс имеет ID = 0, рабочие процессы ID = 1,2,3... по порядку;

- Создается массив, хранящий текущее количество активных соединений каждого рабочего процесса. Начальное значение каждого элемента массива равно нулю;

- Объявляется переменная nproc, хранящая ID рабочего процесса, который должен обрабатывать новое подключениею. Начальное значение равно единице;

- Главный процесс, используя массив количества активных подключений, выбирает рабочий процесс с наименьшим числом активных подключений и записывает его ID в переменную nproc;

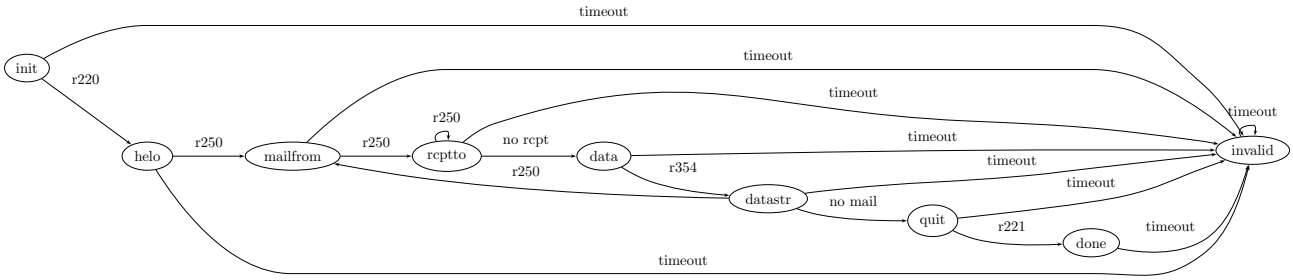

Рис. 1: Состояния сервера

– При установлении нового подключения, его обработкой займется процесс с ID = nproc. В момент установления нового подключения он увеличивает на единицу счетчик активных подключений;

– При разрыве соединения, или завершении цикла получения письма от получателя счетчик активных подключений уменьшается на единицу;

– Три описанные выше шага поворяются до момента завершения работы программы.

Возможно возникновение ситуации, когда один из рабочих процессов был аварийно завершен (в случае возникновения в нем ошибок). Главный процесс, считывая массив активных подключений рабочих процессов, не зная о завершении процесса, будет назначать его кандидатом на обработку нового подключения. В данной ситуации такое поведение ошибочно и будет приводить к задержкам обработки или даже невозможности установления новых подключений. Для избежания этого была предусмотрена логика проверки состояния рабочего процесса с использованием вызова kill(2) перед изменением переменной nproc.

3 Конструкторский раздел

3.1 Конечный автомат состояний клиента

Процессы работы SMTP-сервера представлены в виде конечного автомата на Рис. 1, который описан в файле client.def.

3.2 Синтаксис команд протокола

220 `^220?:\s+.+?\r\n`

221 `^221?:\s+.+?\r\n`

250 `^250?:\s+.+?\r\n`

354 `^354?:\s+.+?\r\n`

data `^DATA\r\n`

mail from `^MAIL FROM:\s*<.+>\r\n`

rcpt to `^RCPT TO:\s*<.+@.+>\r\n`

mx dns `MX\s+\d+\s*.+$`

4 Технологический раздел

4.1 Сборка программы

Сборка программы описана в файле *Makefile* системы сборки *make*. Рис. 2 нагенерили самодельные *makesimple* и *makefile2dot*, а также *dot2tex* и *dot*.

Отмечу, что за исключения целей типа *all*, *install*, *clean*, *tests*, все имена целей в файле систем сборки *make* обычно совпадают с именами файлов (такой вот низкоуровневый инструмент). То есть вместо цели *lexer* следует использовать цель *src/lexer.c*.

4.2 Основные функции программы

Весь это раздел сгеренерировал doxygen из части комментированных исходников программы. В файле конфигурации **doxyggen.cfg** был отключён параметр **HAVE_DOT**, поскольку для рисования графов вызовов используется *cflow*.

4.3 Файл include/key-listener.h

Ожидание сигнала для завершения программы

Функции

int **keyboard_listener_fork** ()

int **keyboard_listener_final** ()

int **quit_key_pressed** ()

Подробное описание

Ожидание сигнала для завершения программы Ожидание символа 'Q' ведется в отдельном потоке. При нажатии на эту клавишу основному процессу посылается сингла о том, что выполнение программы пора прекратить. Проверка на завершение следует производить через вызов функции quit_key_pressed().

4.4 Файл src/key-listener.c

Функции и переменные для разбора команд.

Функции

int **keyboard_loop** ()

int **keyboard_listener_fork** ()

int **quit_key_pressed** ()

int **keyboard_listener_final** ()

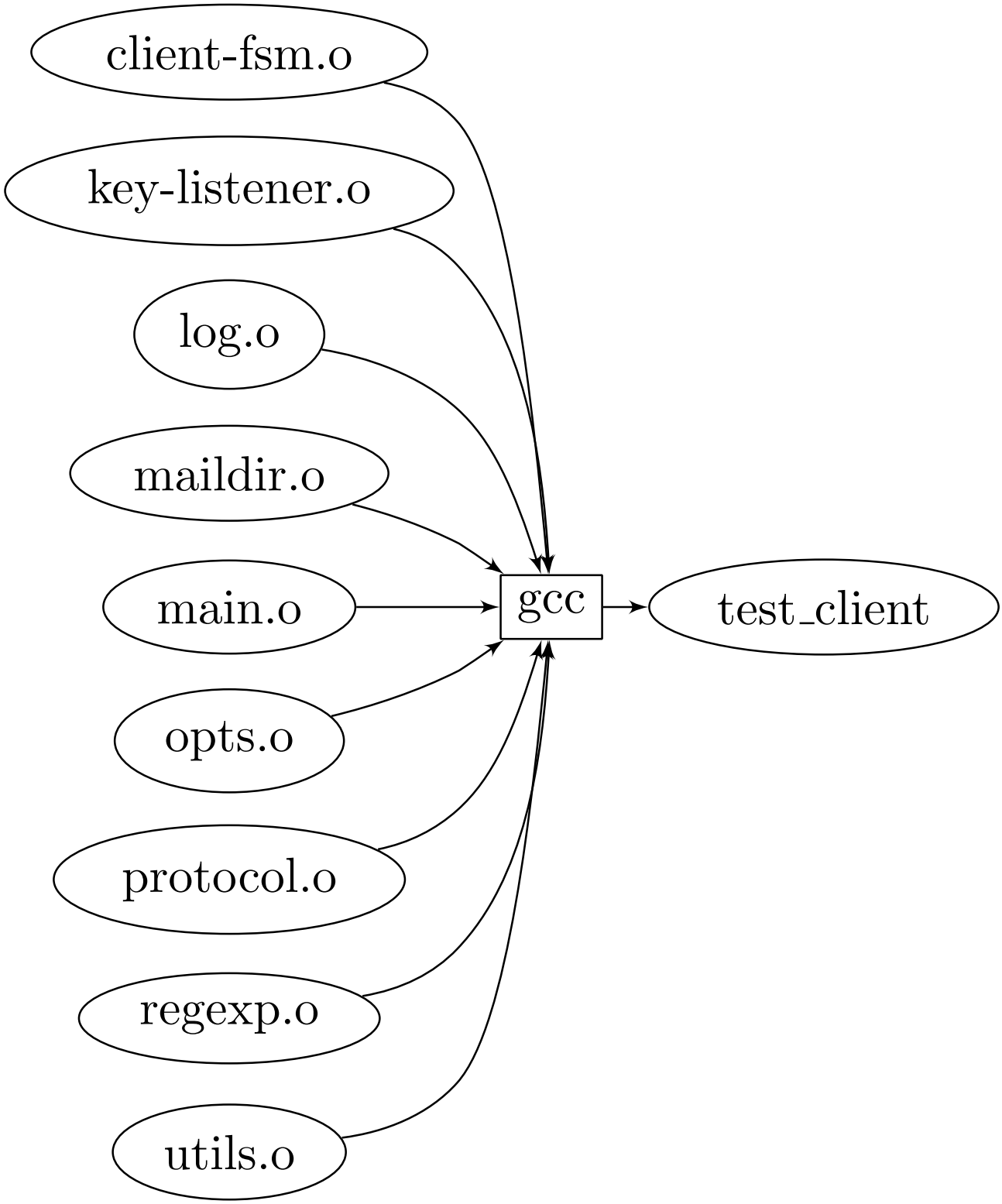

Рис. 2: Сборка программы

Переменные

int key_sock

Socket to pass exit signal.

Подробное описание

Функции и переменные для разбора команд. В этом файле описаны функции необходимые для работы key listener

4.5 Файл include/log.h

Логирование сообщений в отдельном процессе.

Функции

int **fork_log** ()

int close_log ()

int **send_log** (char *msg)

Подробное описание

Логирование сообщений в отдельном процессе. Передача сообщений ведется через пару сокетов. Возможна раскраска сообщений. Три вида сообщений: отладка, ошибки, обычные.

Есть возможность включения/выключения режима отладки (пропадают все сообщения типа DLOG), работы лога в отдельном процессе (в таком случае все сообщения выводятся сразу же) и раскраски (все цвета не выводятся).

Ограничения: 1) первый аргумент любого макроса - строка вида "..."; использование переменных в качестве первого параметра недопустимо; 2) максимальная длина сообщения установлена как MAX_LOG_MESSAGE_SIZE; 3) для запуска лога следует использовать функцию fork_log(), а для завершения - close_log(); при этом основной процесс будет ожидать, пока не будут выведены все накопленные сообщения. 4) для логирования следует использовать макросы ELOG, DLOG и LOG; использование функции send_log() недопустимо.

Функции

int close_log ( ) Send stop signal to log from main process and wait till log process finishes his work; if log is not forked, function just returns

4.6 Файл src/log.c

Логирование сообщений в отдельном процессе.

Функции

int **log_message** (const char *buf, char type)

void **log_loop** ()

int **fork_log** ()

int close_log ()

int **send_log** (char *msg)

Переменные

int msg_sock

Подробное описание

Логирование сообщений в отдельном процессе. Передача сообщений ведется через пару сокетов. Возможна раскраска сообщений. Три вида сообщений: отладка, ошибки, обычные.

Функции

int close_log ( ) Send stop signal to log from main process and wait till log process finishes his work; if log is not forked, function just returns

Переменные

int msg_sock Socket: 1) to get messages (from log); 2) to send messages (from main process)

4.7 Файл include/maildir.h

Файл со структурами и функциями для работы с сообщениями

Структуры данных

struct rcpt

Структура для хранения имени и домена получателя

struct mail

Структура для хранения писем

Перечисления

enum **maildir_dir** {
DIR_ROOT, **DIR_NEW**, **DIR_CUR**, **DIR_NOTSENT**,
maildir_count }

Функции

TAILQ_HEAD (rcpt_list, rcpt)

TAILQ_HEAD (mail_list, mail)

int **maildir_init** ()

int **maildir_final** ()

void **free_mail** (struct mail *m)

void **free_mail_list** (struct mail_list *ml)

int **new_mail_exist** ()

void **move_mail** (const char *filename, maildir_dir from_dir, maildir_dir to_dir)

void **copy_mail** (const char *filename, maildir_dir from_dir, maildir_dir to_dir)

void **delete_mail** (const char *filename, maildir_dir dir)

int **filter_my_mail** (struct mail_list *ml)

int **read_all_mail** (struct mail_list *ml)

struct mail * **read_mail_file** (const char *filename)

int **read_mail_from** (FILE *f, struct mail *m)

int **read_mail_to** (FILE *f, struct mail *m)

int **read_mail_data** (FILE *f, struct mail *m)

Подробное описание

Файл со структурами и функциями для работы с сообщениями

4.8 Файл src/maildir.c

Файл со структурами и функциями для работы с сообщениями

Функции

int **maildir_init** ()

int **maildir_final** ()

int **new_mail_exist** ()

int **read_all_mail** (struct mail_list *ml)

struct mail * **read_mail_file** (const char *filename)

int **read_mail_from** (FILE *f, struct mail *m)

int **read_mail_to** (FILE *f, struct mail *m)

int **read_mail_data** (FILE *f, struct mail *m)

int **filter_my_mail** (struct mail_list *ml)

void **free_mail** (struct mail *m)

void **free_mail_list** (struct mail_list *ml)

void **move_mail** (const char *filename, maildir_dir from_dir, maildir_dir to_dir)

void **copy_mail** (const char *filename, maildir_dir from_dir, maildir_dir to_dir)

void **delete_mail** (const char *filename, maildir_dir dir)

Переменные

char * **maildir_path** [maildir_count]

Подробное описание

Файл со структурами и функциями для работы с сообщениями

4.9 Файл include/protocol.h

Файл содержащий структуры и функции для работы по протоколу SMTP.

Структуры данных

struct mx_conn

struct domain

Функции

TAILQ_HEAD (mx_conn_list, mx_conn)

TAILQ_HEAD (domain_set, domain)

int smtp_client_loop ()

Основная функция, которая мониторит директорию с почтой, чтобы ее отправить

int conn_init ()

Initialize structures for connections with several SMTP servers.

void **conn_loop** ()

int conn_final ()

Frees all structures used in mail transfer.

void domain_add (struct domain_set *domains, char *new_domain_name)

Adds another domain into domain set if it is not present there and if it is not local domain (MY_DOMAIN in maildir.h)

int **rcpt_is_from_domain** (struct rcpt *r, struct domain *d)

int **mail_has_rcpts_from_domain** (struct mail *m, struct domain *d)

int **check_dns** (char *d, char *output_address)

struct mx_conn * **create_connection** (struct domain *dom)

struct mx_conn * **get_conn_by_socket** (struct mx_conn_list *cl, int sock)

int **wait_for_response** ()

int **parse_response** (struct mx_conn *conn, char *str, int length)

void **invalidate_connection** (struct mx_conn *conn)

int **send_hello** (struct mx_conn *conn)

int **send_mailfrom** (struct mx_conn *conn)

int **send_rcptto** (struct mx_conn *conn)

int **send_data** (struct mx_conn *conn)

int **send_datastr** (struct mx_conn *conn)

int **send_quit** (struct mx_conn *conn)

Подробное описание

Файл содержащий структуры и функции для работы по протоколу SMTP.

Функции

void domain_add (

domains,

new_domain_name) Adds another domain into domain set if it is not present there and if it is not local domain (MY_DOMAIN in maildir.h)

Аргументы *domains* – список доменов куда нужно добавить

---

new_domain_name – имя домена для добавления

4.10 Файл src/protocol.c

Файл содержащий структуры и функции для работы по протоколу SMTP.

Функции

int smtp_client_loop ()

Основная функция, которая мониторит директорию с почтой, чтобы ее отправить

int conn_init ()

Initialize structures for connections with several SMTP servers.

int **free_connection** (struct mx_conn *conn)

int **free_domain** (struct domain *d)

int conn_final ()

Frees all structures used in mail transfer.

void domain_add (struct domain_set *domains, char *new_domain_name)

Adds another domain into domain set if it is not present there and if it is not local domain (MY_DOMAIN in maildir.h)

int **rcpt_is_from_domain** (struct rcpt *r, struct domain *d)

int **mail_has_rcpts_from_domain** (struct mail *m, struct domain *d)

int **connect_with_timeout** (int sock, struct addrinfo *addr, int ms)

int **connect_to_any_with_timeout** (struct addrinfo *info, int ms)

struct mx_conn * **create_connection** (struct domain *dom)

int **check_dns** (char *d, char *output_address)

struct mx_conn * **get_conn_by_socket** (struct mx_conn_list *cl, int sock)

void **conn_loop** ()

int **parse_response** (struct mx_conn *conn, char *str, int length)

void **invalidate_connection** (struct mx_conn *conn)

int **wait_for_response** ()

int **send_hello** (struct mx_conn *conn)

int **send_mailfrom** (struct mx_conn *conn)

int **send_rcptto** (struct mx_conn *conn)

int **send_data** (struct mx_conn *conn)

int **send_datastr** (struct mx_conn *conn)

int **send_quit** (struct mx_conn *conn)

Переменные

struct mail_list * **mails**

struct domain_set * **domains**

struct mx_conn_list * **connections**

Подробное описание

Файл содержащий структуры и функции для работы по протоколу SMTP.

Функции

domain_add (

domains,

new_domain_name) Adds another domain into domain set if it is not present there and if it is not local domain (MY_DOMAIN in maildir.h)

Аргументы *domains* – список доменов куда нужно добавить

---

new_domain_name – имя домена для добавления

4.11 Файл include/regexp.h

Набор функций для работы с регулярными выражениями. smtp_re_name перечисляет все представленные регулярные выражения.

Перечисления

enum **smtp_re_name** {
r220, **r221**, **r250**, **r354**,
RE_data, **RE_mail_from**, **RE_rcpt_to**, **RE_rcpt_domain**,
RE_mx_dns, **RE_mx_dns_prio**, **smtp_re_count** }

Функции

int **re_init** ()

int **re_final** ()

int **re_match** (smtp_re_name re_name, char *str, int length)

smtp_re_name **re_match_any** (char *str, int length)

int **re_match_and_fill_substring** (smtp_re_name re_name, char *str, int length, char *substr)

char * **re_match_and_alloc_substring** (smtp_re_name re_name, char *str, int length)

Подробное описание

Набор функций для работы с регулярными выражениями. smtp_re_name перечисляет все представленные регулярные выражения. 1) Для инициализации работы необходим вызов функции re_init(), а для кооректного завершения - re_final().

2) re_match() возвращает 1, если строка подходит под регулярку, и 0 во всех других случаях.

3) re_match_any() возвращает номер регулярки, если строка подходит под любую из всех доступных (номер равен номеру первой подошедшей). Если ни одна регулярка не подошла, возвращает -1.

4) re_match_and_fill_substring() и re_match_and_alloc_substring() заполняют или выделяют память под строку, соответствующей первой группе скобок в регулярке соответственно. Если нет соответствия реулярке, то возвращается 0.

4.12 Файл src/regexp.c

Функции

int **re_compile** (smtp_re_name re_name)

int **re_init** ()

int **re_final** ()

int **re_match** (smtp_re_name re_name, char *str, int length)

char * **re_match_and_alloc_substring** (smtp_re_name re_name, char *str, int length)

int **re_match_and_fill_substring** (smtp_re_name re_name, char *str, int length, char *substr)

smtp_re_name **re_match_any** (char *str, int length)

Переменные

const char * **patterns** [smtp_re_count]

pcre * **smtp_re** [smtp_re_count]

Переменные

const char* patterns[smtp_re_count] **Инициализатор** = { "220(? : \\s+.+)?\\r\\n
, "221(? : \\s+.+)?\\r\\n
, "250(? : \\s+.+)?\\r\\n
, "354(? : \\s+.+)?\\r\\n
, "DATA\\r\\n
, "MAILFROM : \\s*<.+>\\r\\n
, "RCPTTO : \\s*<(.+@.+)>\\r\\n
, "RCPTTO : \\s*<.+@(.+)>\\r\\n
, "MX\\s+\\d+\\s*(.+)\\.
, "MX\\s+(\\d+)\\s*.+\\.
}

4.13 Графы вызова функций

Поскольку функций много, графы вызовов разбиты на два рисунка. На рис. 3 показаны основные функции, на рис. 4 – функции обработки команд. Файл **cflow.ignore** содержит список функций (точнее, шабловнов поиска), использыемых программой *grep* для удаления малоинтересных стандартных функций[1].

Графы созданы с помощью *cflow*, *cflow2dot*, *dot*.

[1] Функции по работе с сокетами, ipc и привилегиями к малоинтересным ни в коем случае не относятся.

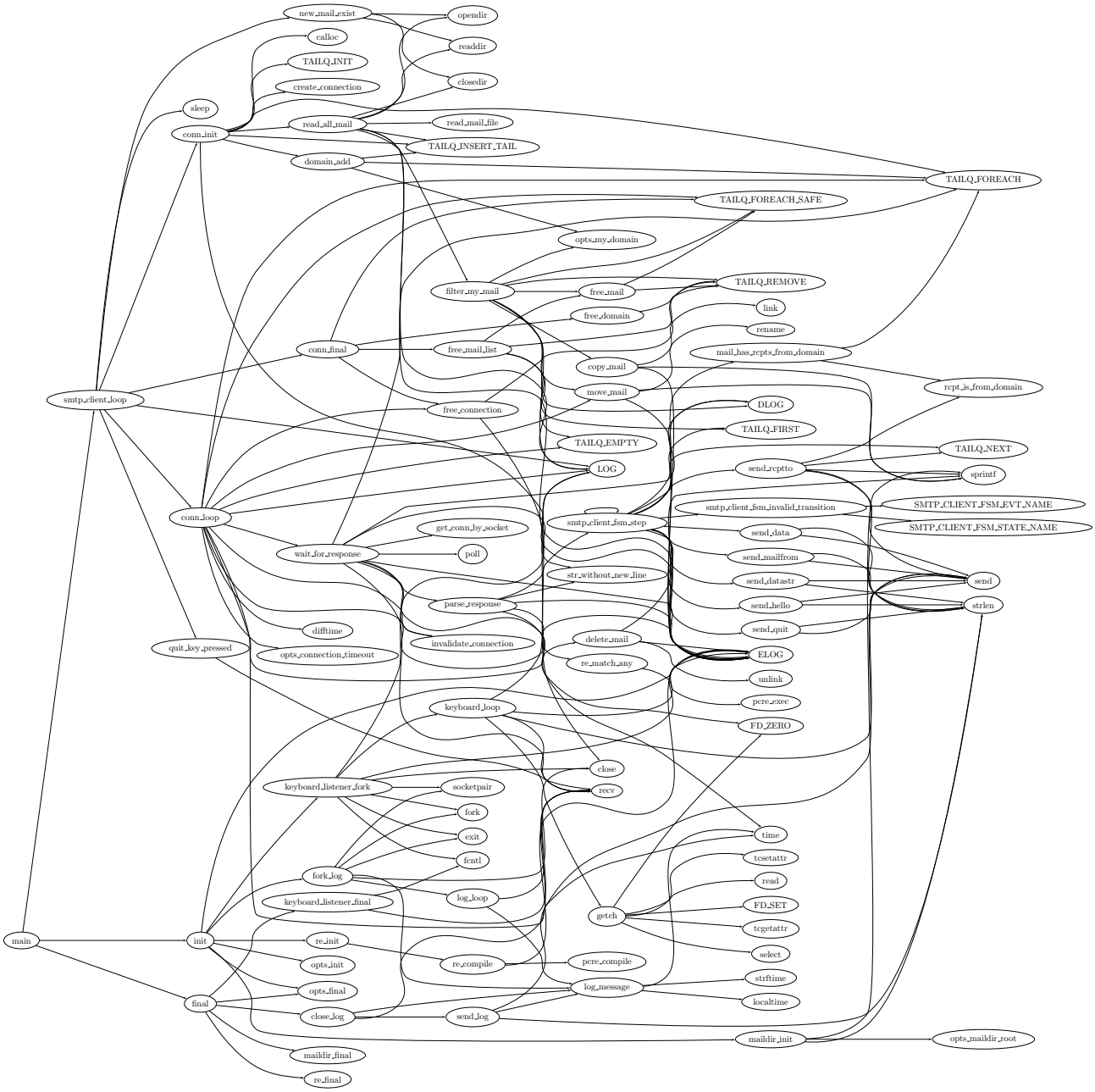

Рис. 3: Граф вызовов, основные функции

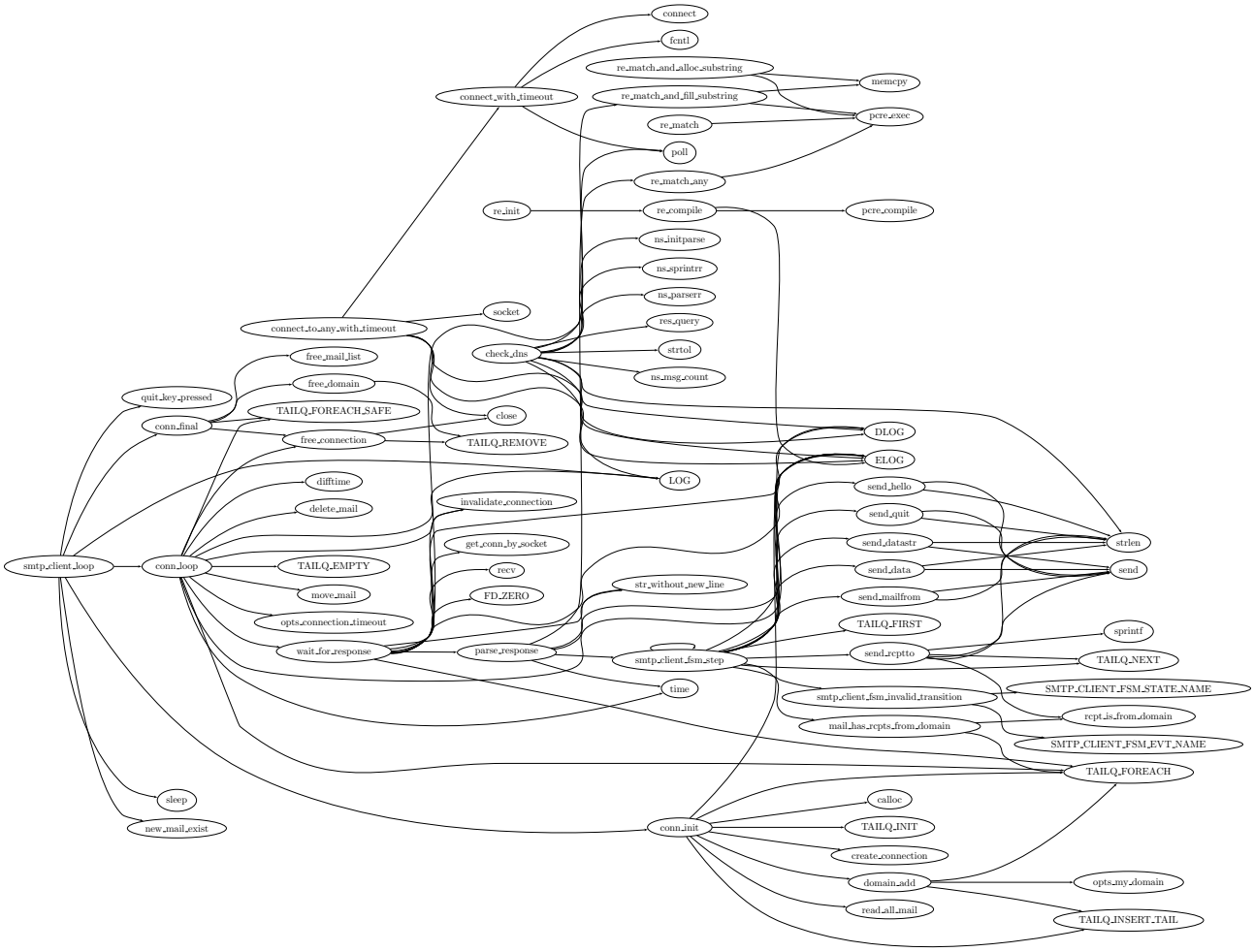

Рис. 4: Граф вызовов, функции обработки команд

5 ЗАКЛЮЧЕНИЕ

Результатом данной работы является разработанная программа - SMTP-сервер, позволяющий производить получение электронной почты по протоколу SMTP.

Также были решены все поставленные задачи:

- Реализован КА логики работы протокола SMTP клиента
- Создан алгоритм обработки нескольких соединений в рамках одного процесса с использованием вызова poll();
- Разработана программа - SMTP-клиент;
- Представлено описание реализованных функций программы, графа вызовов функций, процесса и графа сборки программы, реализованных алгоритмов для синхронизации доступа к данным;
- Проведенно модульное, системное тестирование, а также тестирование утечек памяти полученного SMTP-сервера. Выполнена оценка полученных результатов.

Использование вызова poll() позволило организовать обработку нескольких соединений в рамках одного рабочего процесса, а равномерное распределение подлючений между рабочими процессами позволило эффективно производить обслуживание множества подключений.

Программа Autogen с библиотеками AutoFSM и AutoOpts позволила быстро получить код КА для работы программы, а также функционал распознавания параметров командной строки.

По результатам проведенного модульного тестирования, с использования CUnit, а также системного тестирования были выявлены и устранены некоторые ошибки в функциях разработанной программы. А использование Valgrind позволило выявить и устранить места утечек памяти программы.