Разработка почтового сервера

0. ВВЕДЕНИЕ

В рамках курса “Протоколы вычислительных сетей” велась работа над проектированием, реализацией и тестированием почтового сервера (MTA). Такой сервер состоит из 2х частей: SMTP-сервер, который обеспечивает прием почты от клиентов MUA; и SMTP-клиент, ответственный за удаленную передачу почты.

Целью работы является получение или закрепление следующих навыков:

- проектирования реализации сетевого протокола по имеющейся спецификации;
- реализации сетевых приложений;
- созданием реализацией сетевой службы без создания нити на каждое соединение;
- автоматизированного системного тестирования ПО сетевой службы;
- создания сценариев сборки ПО;
- групповой работы с использованием VCS и CI.

0.1. Сервер

Задание варианта #11: Используется вызов select и единственный рабочий поток. Журналирование в отдельном потоке.

Цель: реализовать SMTP-сервер с использованием единственного рабочего потока и select.

Основные решаемые задачи:

- проанализировать архитектурное решение;
- разработать подход для обработки входящих соединений и хранения входящих писем;
- рассмотреть SMTP-протокол;
- реализовать программу для получения писем по протоколу SMTP.

SMTP-сервер должен:

- реализовывать получение писем по протоколу SMTP;
- реализовывать структуру хранения входящих писем;
- реализовывать алгоритм обработки соединений в одном потоке с использованием вызова select;
- подерживать команды HELO и EHLO, MAIL, RCPT, DATA, RSET, QUIT, VERIFY протокола SMTP

- реализовывать журналирование в отдельном потоке

0.2. Клиент

Задание варианта #32: Используется вызов select и единственный рабочий поток (или процесс). Журналирование в отдельном потоке. Пытаться отправлять все сообщения для одного MX за одну сессию.

Основные решаемые задачи:

- Изучить работу SMTP-протокола применительно для клиентской части системы
- Проанализировать архитектурное решение для клиентской части системы
- Продумать способы интеграции с серверной частью
- Разработать алгоритм обработки соединений в одном потоке
- Реализовать программу для отправки писем по протоколу SMTP

SMTP-клиент должен:

- представлять программу для отправки писем по протоколу SMTP
- реализовывать алгоритм обработки соединений в одном потоке с использованием вызова select
- на один удалённый MX создавать не более одного сокета
- использовать отдельную очередь сообщений для каждого MX
- поддерживать набор команд, достаточный для отправки почты как минимум одной крупной почтовой публичной службе с веб-интерфейсом (в данном случае выбран Yandex)
- пытаться отправлять все сообщения для одного MX за одну сессию
- реализовывать журналирование в отдельном потоке

1. АНАЛИТИЧЕСКИЙ РАЗДЕЛ

1.1. Предметная область

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.

В рамках данной системы предусмотрены следующие сущности:

- **Письмо** - основная единица данных. Состоит из заголовков и тела письма. В заголовках указываются отправитель (From), получатель (To), а также могут быть указаны тема письма (Subject) и другие атрибуты. Обязательными в рамках данного протокола являются From и To. Получателей может быть несколько.

- **Получатель-хост (MX-сервер)** - сервер, предназначенный для получения и хранения почтовых сообщений. Например, почтовый сервер Яндекса (доступен по адресу mx.yandex.ru).

1.2. Плюсы и минусы указанного в задании способа решение проблемы «развязывания» потоков выполнения сервера и соединений

Потоки или нити приложения исполняются в пределах одного процесса. Все адресное пространство процесса разделяется между потоками. На первый взгляд кажется, что это позволяет организовать взаимодействие между потоками вообще без каких-либо специальных API. В действительности, это не так – если несколько потоков работает с разделяемой структурой данных или системным ресурсом, и хотя бы один из потоков модифицирует эту структуру, то в некоторые моменты времени данные будут несогласованными.

Преимущества:

- Высокая производительность. На большинстве Unix-систем, создание нити требует в десятки раз меньше процессорного времени, чем создание процесса.
- Эффективный произвольный доступ к разделяемым данным. В частности, такая архитектура пригодна для создания серверов баз данных.

- Высокая переносимость и легкость переноса ПО из-под других ОС, реализующих многопоточность.

Недостатки:

- Высокая вероятность опасных ошибок. Все данные процесса разделяются между всеми нитями. Иными словами, любая структура данных может оказаться разделяемой, даже если разработчик программы не планировал этого. Ошибки при доступе к разделяемым данным – ошибки соревнования – очень сложно обнаруживать при тестировании.
- Высокая стоимость разработки и отладки приложений.
- Низкая надежность. Разрушение структур данных, например в результате переполнения буфера или ошибок работы с указателями, затрагивает все нити процесса и обычно приводит к аварийному завершению всего процесса. Другие фатальные ошибки, например, деление на ноль в одной из нитей, также обычно приводят к аварийной остановке всех нитей процесса.
- Низкая безопасность. Все нити приложения исполняются в одном процессе, то есть от имени одного и того же пользователя и с одними и теми же правами доступа. Невозможно реализовать принцип минимума необходимых привилегий, процесс должен исполняться от имени пользователя, который может исполнять все операции, необходимые всем нитям приложения.
- Создание нити – все-таки довольно дорогая операция. Для каждой нити в обязательном порядке выделяется свой стек, который по умолчанию занимает 1 мегабайт ОЗУ на 32-битных архитектурах и 2 мегабайта на 64-битных архитектурах, и некоторые другие ресурсы. Поэтому данная архитектура оптимальна не для всех приложений.
- Невозможность исполнять приложение на многомашинном вычислительном комплексе. Приемы, такие, как отображение на память разделяемых файлов, для многопоточной программы не применимы.

В целом можно сказать, что многопоточные приложения имеют почти те же преимущества и недостатки, что и многопроцессные приложения, использующие разделяемую память. Однако стоимость исполнения многопоточного приложения ниже, а разработка такого приложения в некоторых отношениях проще, чем приложения, основанного на разделяемой памяти.

1.3. Рабочие потоки

Приложение SMTP-сервера использует два независимых друг от друга потока:

- Поток логгера
- Поток, обрабатывающий все подключения клиентов по протоколу SMTP

Приложение SMTP-клиента использует 4 независимых друг от друга потока:

- Поток логгера
- Поток, обрабатывающий все соединения с почтовыми серверами
- Поток-крон, подготавливает сообщения из очередей к отправке
- Поток-крон, вычитывает новые сообщения из Maildir и распределяет их в очереди

1.4. Выбор способа хранения писем для SMTP-сервера

SMTP-сервер сохраняет полученные сообщения в директорию, соответствующую формату Maildir. Maildir - распространенный 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.

Каталог Maildir имеет три подкаталога: tmp, new и cur. При доставке сообщения оно помещается в файл в подкаталоге tmp. Имя файла может формироваться из текущего времени, имени хоста, идентификатора процесса, создавшего этот файл, и некоторого случайного числа — таким образом гарантируется уникальность имен файлов. После записи в файл всего сообщения обычно создается жесткая ссылка на этот файл в каталоге new, а текущая ссылка из tmp удаляется, но в некоторых реализациях просто используется системный вызов rename, — всё это делается для того, чтобы никакой другой процесс не смог прочитать содержимое сообщения до тех пор, пока оно не будет записано полностью.

1.5. Выбор способа хранения писем в очередях для SMTP-клиента

SMTP-клиент вычитывает новые сообщения из Maildir и распределяет их по соответствующим очередям. Распределение ведется по домену почтового адреса получателя. Например, если письмо адресовано для noreply@test.ru, то оно попадет в очередь, соответствующую домену test.ru. Добавление новых очередей осуществляется динамически.

2. КОНСТРУКТОРСКИЙ РАЗДЕЛ

2.1. Сервер

2.1.1 Конечный автомат состояний сервера

Конечный автомат состояний сервера представлен на рисунке 1.

Выделены следующие состояния:

- SS_INIT;
- SS_WAIT;
- SS_MAIL_SET;
- SS_RECEPIENTS_SET;
- SS_WRITING_DATA;
- SS_DELIVERING;
- SS_CLOSED.

Переходы, соответствующие командам SMTP-протокола и возможным событиям:

- handle_HELO;
- handle_EHLO;
- handle_MAIL;
- handle_RCPT;
- handle_RSET;
- handle_DATA;
- handle_QUIT;
- handle_NOT_IMPLEMENTED;
- handle_TEXT;
- connection_established;
- connection_failed;
- message_saved;
- text;
- CLRF_dot_CLRF.

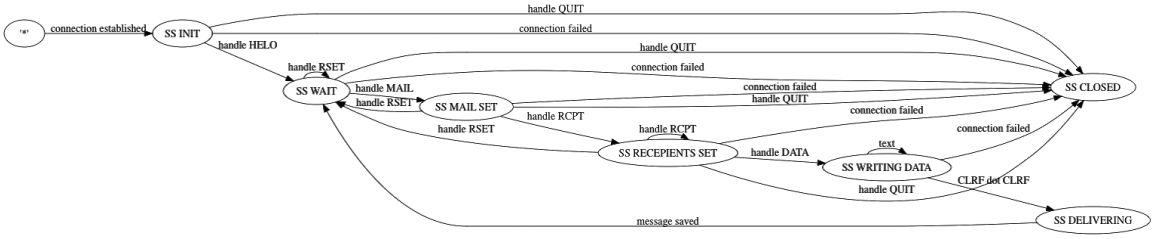

Рис. 1. Конечный автомат состояний сервера

2.1.2 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 TO:* <*\w+@\w+\.\w+*>
5. **RSET**: *RSET*
6. **DATA**: *DATA*
7. **VERIFY**: *VRFY \w+*
8. **QUIT**: *QUIT*

2.1.3 Архитектура решения

Приложение SMTP-клиента использует 2 независимых друг от друга потока, которые порождает главныq поток. Главный поток инициализирует воркеры через пул потоков и обрабатывает их корректное освобождение в случае завершения работы приложения. Во время корректного завершения главный поток по очереди оповещает все воркеры с помощью сигнала и ожидает их завершения.

- Поток логгера (LogWriter). Вычитывает сообщения из очереди для лога и пишет их в консоль.

- Поток, обрабатывающий все соединения с почтовыми серверами (MessageReceiver). Использует вызов select. Обрабатывает операции ACCEPT, READ.

Осуществление механизма для мониторинга одного или нескольких каналов сокета реализовано через Selector из библиотеки NIO. Graceful closing

осуществляется при помощи извещения и закрытия всех доступных сокетов данного Selector'a. Команда VRFY не обрабатывается, как отдельная команда - она обрабатывается так, если бы это был любой другой текст, используемый в качестве команды. Команды должны посылаться на сервер строго как 4 символа в верхнем регистре - нет перевода между регистрами

2.1.4 Алгоритм обработки соединений в одном процессе

Для обработки все соединений с почтовыми серверами в одном потоке используется вызов **select**. Ниже приведена реализация такого обработчика:

```
 ssc = ServerSocketChannel.open();

ssc.configureBlocking(false);
ssc.socket().bind(bindAddress);

final Selector acpt_sel = Selector.open();
ssc.register(acpt_sel, SelectionKey.OP_ACCEPT);

while (!stopped) {
  Set<SelectionKey> selectedKeys = acpt_sel.selectedKeys();
  Iterator<SelectionKey> iterator = selectedKeys.iterator();

  while (iterator.hasNext()) {
     SelectionKey key = iterator.next();
      iter.remove();

     if (!key.isValid()) {
         LOG.error("Error!!! Invalid key!!!");
        return;
     } else if (key.isAcceptable()) {
        accept(key);
     } else if (key.isReadable()) {
        read(key);
     }
  }
}
ssc.socket().close();
```

2.2. КЛИЕНТ

2.2.1. Архитектура решения

Приложение SMTP-клиента использует 4 независимых друг от друга потока (воркера), которые порождает главным поток. Главный поток инициализирует воркеры через пул потоков и обрабатывает их корректное освобождение в случае завершения работы приложения. Во время корректного завершения главный поток по очереди оповещает все воркеры с помощью сигнала и ожидает их завершения.

- Поток логгера (LogWriter). Вычитывает сообщения из очереди для лога и пишет их в консоль.

- Поток, обрабатывающий все соединения с почтовыми серверами (MessageSenderService). Использует вызов select. Обрабатывает операции CONNECT, READ и WRITE.

- Поток-крон, подготавливает сообщения из очередей к отправке (MessageQueueReaderScheduler). Запускается каждые 2 секунды.

- Поток-крон, вычитывает новые сообщения из Maildir и распределяет их в очереди (MessageFileReaderScheduler). Запускается каждые 2 секунды.

2.2.2. Обработка новых писем

Крон MessageFileReaderScheduler вычитывает все сообщения из директории указанной в конфигурации приложения, в данном случае используется Maildir/new. В каждом файле хранится сообщение, которое может быть адресовано нескольким получателям, в том числе и получателям с разными почтовыми доменами. Оригинальные получатели и отправитель указаны соответственно в заголовках X-Original-To и X-Original-From. Если получателей несколько, то клиент разбивает данный файл с сообщением на несколько сообщений с одним конкретным получателем. После чего направляет считанные и обработанные данные в виде списка сообщений в дальнейшую обработку и затем распределяет их по соответствующим очередям. Распределение ведется по домену почтового адреса получателя. Например, если письмо адресовано для noreply@test.ru, то оно попадет в очередь, соответствующую домену test.ru.

Крон MessageQueueReaderScheduler опрашивает очереди сообщений для каждого домена. Если очередь не пуста, то проверяет доступен ли сокет для отправки. Если доступен, то запускает стейт-машину для отправки всех текущих сообщений для выбранного домена.

2.2.3. Синтаксис поддерживаемых команд протокола

В данном разделе приведены поддерживаемые клиентом команды протокола SMTP:

- HELO [w+]+
- MAIL FROM: [\w]+@[\w]+ [\w]+
- RCPT TO: [\w]+@[\w]+ [\w]+
- DATA
- RSET
- QUIT

2.2.4. Машина состояний

Машина состояний отвечает за корректную отправку сообщений почтовым серверам. Для контроля обмена информаций сохраняется некий контекст в рамках данной группы сообщений, который передается из одного состояния машины в другое. Таким образом исключается возможность непредвиденных переходов, отправки некорректного сообщения, а также исключается неправильная очередность между отправки или получения сообщений между двумя системами. Состояния соединений и переходы между ними изображены на рисунке 2.

Отправка команд протокола SMTP сопровождается проверкой ответов от сервера. Ниже приведены успешные коды ответов для каждой из команд:

- Установка соединения: **220**
- HELO: **250**
- MAIL FROM: **250**
- RCPT TO: **250**
- DATA: **354**
- Запись данных: **250**
- QUIT: **221**

При получении кода ответа, отличного от представленных выше, машина переходит в состояние завершения.

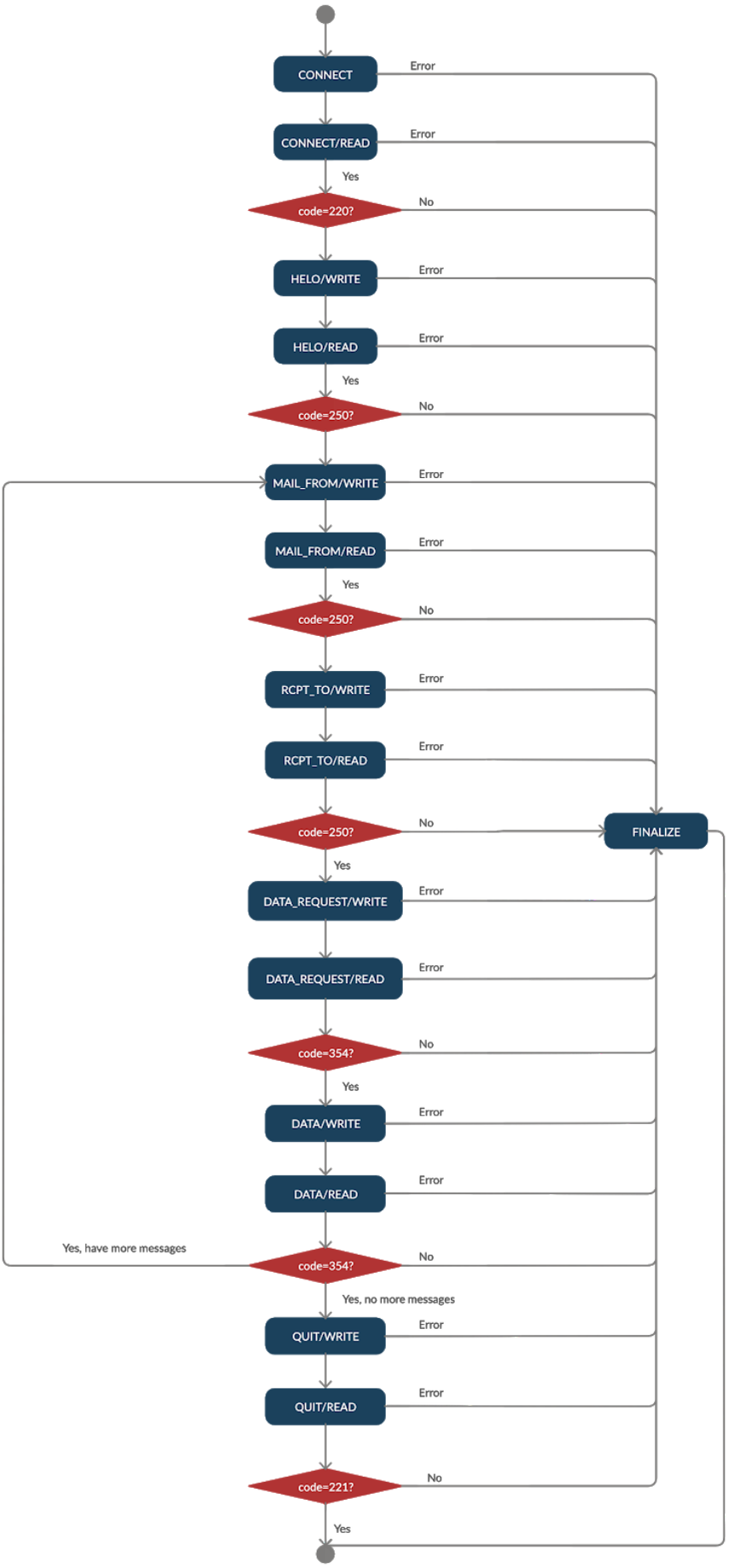

Рисунок 2. Машина состояний клиента

2.2.5. Обработка соединений в одном потоке

Для обработки все соединений с почтовыми серверами в одном потоке используется вызов **select**. Ниже приведена реализация такого обработчика:

```
while (!stopped) {
  selector.selectNow();

  Set<SelectionKey> selectedKeys = selector.selectedKeys();
  Iterator<SelectionKey> iterator = selectedKeys.iterator();

  while (iterator.hasNext()) {
     SelectionKey key = iterator.next();

     if (key.isConnectable()) {
        connect(key);
     } else if (key.isReadable()) {
        read(key);
     } else if (key.isWritable()) {
        write(key);
     }

     iterator.remove();
  }
}
```