SMTP сервер№1

(Агеев А. В.)

20 декабря 2020 г.

Оглавление

Введение

Для функционирования компьютерных сетей, на оборудовании устанавливается программное обеспечение реализующий различные протоколы взаимодействия. Протоколы различаются по назначению. В данное время для обеспечения сети интернет используется стек протоколов TCP/IP, который состоит из протоколов выполняющий каждый свою задачу:

- Канальный уровень (например Ethernet) – беспечивают отправку и прием данных данных через среду передачи.
- Сетевой уровень (ip) – Канальный уровень работает с множеством устройств, которые объединены в одну группу (сеть). В данной группе устройства «видят» друг друга напрямую. Протоколы сетевого уровня предназанчены для обеспечения взаимодействия устройст из разных групп. Две сети объединяются маршрутизатором, а с помощью протокола сетевого уровня выплняется адресация устройст. В этом случае, между устройствами разных групп существует посредник – маршрутизатор
- Транспортный уровень (TCP, UDP) – на современном оборудовании работает множество программ, для определения того, какой программе адресованы прешедшые данные из сети, используются протоклы транспротного уровня. Их основная цель – адресация процессов на устройстве.
- Прикладной уровень – данные протоколы реализуются приложениями, которую выполняют некоторую задачу.

Целью курсовой работы является реализация протокола прикладоного уровня для получения и доставки электронной почты – Simple Mail Transfer Protocol (SMTP). А именно, части, которая выполняет прием почты и выполняет ее передачу на следующий этап – отправку почты.

Глава 1

Аналитический раздел

1.1 Основные понятия протокола SMTP

SMTP протокол основан на клиент-серверной архитектуре. В данном случае клиентом выступает программа, которая хочет отправить почту, а сервером является программа для приема почты. Протокол поддреживает маршрутизацию почты, то есть серверу может придти письмо, которое адресовано клиенту на другом сервере. В этом случае серверное программное обеспечение принимает роль клиента и отправлет почту другому серверу.

Протокол состоит из текстовых сообщений, которые передают друг другу клиент и сервер при взаимодействии. Каждое сообщение прдеставляет из себя команду с параметрами, которые выполняются сервером. На какждую команду серверв выдает отклик. При организации надежного соединения (например посредством протокола TCP) клиент инициирует почтову транзакцию, которая состоит из последовательности команд, задающих отправителя и получателя сообщения, а так же передается содержательная часть письма. После чего клиент может завершить сеанс или начать новую почтовую транзакцию для передачи очередного письма.

Объекты электронной почты: ({Конверт; Содержимое})

- Конверт
  - Адрес отправителя – определяется командой MAIL FROM, которая так же начинает почтовую транзакцию.
  - Адрес получателей - с помощью команды RCPT TO определяется один получатель и маршрут почты до этого получателя (в RFC2821 указано, что лучше механизм маршрутизации почты игнорировать). Данная команда может быть передана несоклько раз для указания списка получателей одного письма.
  - Дополнительные заголовки. Протокол SMTP поддреживает расширения - добавление новых заголовков и параметров к стандартным заголовкам.
- Содержимое – передается после отправки команды DATA
  - Заголовок - список полей вида <ключ>:<значение>, спецификация которых описана в RFC5322
  - Тело сообщения - это непосредственное содержимое письма, которая представляет из себя текстовый набор данных соответсвущий спецификации форматов разны типов объектов MIME (Multipurpose Internet Mail Extensions)

Все элементы описваются с исопльзованием 7-битной кодировки US-ASCII, но это ограничение может быть снято с использованием расширения протокола 8BITMIE

Получатель и отправитель:

Протокол SMTP работает в 2 стороны. Получателем и отправителем может выступать как почтовая служба на сервере так и клиентское программное обеспечение. В протоколе выделяются следующие понятия:

- Клиент – Отправлющая сторона в текущей почтовой транзакции.
- Сервер – Принимающая сторона в текущей почтовой транзации.
- Агент доставки почты (Mail Transfer Agent, MTA) – Клиент и сервер SMTP обеспечивающее почтовый трансопртный сервис.
- Пользовательский почтовый агент (Mail User Agent, MUA) – Программное обеспечение выступающее в качетсве исходных отправителей и конечных получателей почтовых сообщений

$$MUA \rightarrow MTA \rightarrow MTA \rightarrow MUA$$

Типы агентов SMTP:

- Система отрпавки (originator) – Вносит сообщение в среду передачи данных, в котором находится транспортный сервис.
- Система доставки (delivery) – Принимает почту от транспортного серивса и передает ее пользовательскому агенту или размещает ее в хранилище.
- Транслятор (relay) – Получает почту от клиента и передает ее другому серверу.
- Шлюз (gateway) – Система получающие письма от одной транспортной среды и передающие письма сереверу находящейся в другой транспортной среде.

1.2 SMTP сеанс

При подключении клиента к серверу начинается SMTP сеанс, в течении которого выполняется взаимодействие клиента и сервера по доставки писем.

1. Инициирование соединения

   Клиент: создает соединение с сервером

   Сервер: Отправляет отклик

   - 220 в случае готовности
   - 554 в случае отказа в SMTP сервисе

2. Инициирование клиента (сеанса)

   Клиент: передает команду HELO/EHLO. HELO - содание SMTP сеанса. EHLO - создание SMTP сеанса с поддержкой расширений протокола (Extend Hello).

   Сревер: Отрпавляет отклик 250. Если бла отправлена команда EHLO, то сервер так же возращает список расширений, который он поддерживает (расширения далее не рассматриваются)

3. Почтовая транзакция (Транзакцию нельзя сделать вложеной в другую транзакцию)

   (a) Начало транзакции

   Клиент: Отправлет команду MAIL FROM. Команда говорит о запуске новой почтовой транзакции и передает адрес отправителя. Если в процессе передачи возникнет ошибка, на этот адрес будет отправлено уведомление.

   Сервер: Отклик 250

   (b) Определение спика получателей

   Для определение списка получателей клиент отправлет несколько команд RCPT TO, на каждую из которых сервер отправляет отклик 250.

   Если команда RCPT TO отправлена до начала почтовой транзакции, то сервер отправлет отклик 503

   (c) Передача тела письма

   Клиент: Отправлет команду DATA

   Сервер: отправлет отклик 354, что свидетельствует о том, что сервер готов принимать содержимое письма

   Клиент: Отрпавлет все почтовые данные. После завершения отправки тела письма, клиент должен отправить точку на отдельной строке (<CRLF>.<CRLF> – послеовательность кончания данных письма)

   Сервер: Должен воспринимать все присилаемые данные, как тело письма. Как только он получает последовательность конца данных (<CRLF>.<CRLF>) сервер должен инициировать процесс доставки письма. А клиенту отправить отклик 250

4. Завершение сеанса или новая транзакция

   - Если клиент желает завершить работу с сервером, то он должен послеть команду QUIT, на которую сервер должен ответить откликом 221 и закрыть соединение.
   - Если клиент желает продолжить работу с сервером, то он должен создать новую почтовую транзакцию. Для этого необходимо перейти на шаг 3а

Дополнительные команды:

1. VRFY
2. EXPN
3. RSET – прерывание текущей почтовой транзакции. Отклик сервера: 250
4. HELP
5. NOOP

1.3 Синтаксис команд

```
ehlo = "EHLO" SP Domain CRLF
helo = "HELO" SP Domain CRLF
ehlo-ok-rsp  = ("250" domain [SP ehlo-greet] CRLF)
             | ("250-" domain [SP ehlo-greet] CRLF
             | *("250-" ehlo-line CRLF)
             | ("250" SP ehlo-line CRLF)

ehlo-greet = 1*(%d0-9 / %d11-12 / %d14-127)
ehlo-line = ehlo-keyword *( SP ehlo-param )
ehlo-keyword = (ALPHA / DIGIT) *(ALPHA / DIGIT / "-")
ehlo-param   = 1*(%d33-127)
"MAIL FROM:" ("<>" / Reverse-Path) [SP Mail-parameters] CRLF
"RCPT TO:" ("<Postmaster@" domain ">" /
           "<Postmaster>" / Forward-Path)
           [SP Rcpt-parameters] CRLF
"DATA" CRLF
"RSET" CRLF
"VRFY" SP String CRLF
"EXPN" SP String CRLF
"HELP" [ SP String ] CRLF
"NOOP" [ SP String ] CRLF
"QUIT" CRLF
Reverse-path = Path
Forward-path = Path
Path = "<" [ A-d-l ":" ] Mailbox ">"
A-d-l = At-domain *( "," A-d-l )
At-domain = "@" domain
Mail-parameters = esmtp-param *(SP esmtp-param)
Rcpt-parameters = esmtp-param *(SP esmtp-param)
esmtp-param     = esmtp-keyword ["=" esmtp-value]
esmtp-keyword   = (ALPHA / DIGIT) *(ALPHA / DIGIT / "-")
esmtp-value     = 1*(%d33-60 / %d62-127)
Keyword  = Ldh-str
Argument = Atom
Domain = (sub-domain 1*("." sub-domain)) / address-literal
sub-domain = Let-dig [Ldh-str]
address-literal = "[" IPv4-address-literal /
                      IPv6-address-literal /
                      General-address-literal "]"
Mailbox         = Local-part "@" Domain
Local-part      = Dot-string / Quoted-string
Dot-string      = Atom *("." Atom)
Atom            = 1*atext
Quoted-string   = DQUOTE *qcontent DQUOTE
String          = Atom / Quoted-string
IPv4-address-literal    = Snum 3("." Snum)
IPv6-address-literal    = "IPv6:" IPv6-addr
```

```
General-address-literal = Standardized-tag ":" 1*dcontent
Standardized-tag        = Ldh-str
Snum                    = 1*3DIGIT
Let-dig                 = ALPHA / DIGIT
Ldh-str                 = *( ALPHA / DIGIT / "-" ) Let-dig
IPv6-addr               = IPv6-full / IPv6-comp / IPv6v4-full / IPv6v4-comp
IPv6-hex  = 1*4HEXDIG
IPv6-full = IPv6-hex 7(":" IPv6-hex)
IPv6-comp = [IPv6-hex *5(":" IPv6-hex)] "::" [IPv6-hex *5(":"IPv6-hex)]
IPv6v4-full = IPv6-hex 5(":" IPv6-hex) ":" IPv4-address-literal
IPv6v4-comp = [IPv6-hex *3(":" IPv6-hex)] "::"
              [IPv6-hex *3(":" IPv6-hex) ":"] IPv4-address-literal
```

1.4 Тестирование

Для тестирования функциональности сервера, было написаны unit-тесты с использованием библиотеки cunit. Результат работы тестиованяи представлен в листинге

```
     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/


Suite: EventsQueue
  Test: add element in queue ...passed
  Test: add two elements of equals type in queue ...passed
  Test: pop element from queue ...passed
  Test: pop not existing element from queue ...passed
Suite: RegisteredEventsQueue
  Test: push element accept type in queue ...passed
  Test: push element read type in queue ...passed
  Test: push element write type in queue ...passed
  Test: pop element accept type in queue ...passed
  Test: pop element read type in queue ...passed
  Test: pop element write type in queue ...passed
  Test: get bitmask of registered events for socket ...passed
Suite: EventLoop
  Test: test init event loop ...passed
  Test: create pollfd from event_loop structure ...passed
  Test: process pollin ...passed
Suite: Vector
  Test: test init vector ...passed
  Test: test get element by wrong index ...passed
  Test: test create full copy ...passed
  Test: test create sub vector as first part ...passed
  Test: test create sub vector as second part ...passed
```

```
Suite: Smtp regex
  Test: regex hello ...passed
  Test: regex IPv4 ...passed
  Test: regex domain route list ...passed
  Test: mail from ...passed
  Test: rcpt to ...passed
  Test: parsing command 'hello' ...passed
  Test: parsing command 'mail from' ...passed
  Test: parsing command 'rcpt to' ...passed
  Test: check good command sequence ...passed
Suite: Users list
  Test: add element and find ...passed
Suite: Server
  Test: smtp session ...passed
  Test: substr iterate by sep ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      7      7    n/a      0        0
               tests     31     31     31      0        0
             asserts    147    147    147      0      n/a

Elapsed time =    0.466 seconds

==14488== Memcheck, a memory error detector
==14488== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==14488== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==14488== Command: bin/unit_tests.out
==14488== Parent PID: 14487
==14488==
==14488==
==14488== HEAP SUMMARY:
==14488==     in use at exit: 0 bytes in 0 blocks
==14488==   total heap usage: 8,121 allocs, 8,121 frees, 1,078,458 bytes allocated
==14488==
==14488== All heap blocks were freed -- no leaks are possible
==14488==
==14488== For lists of detected and suppressed errors, rerun with: -s
==14488== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Так же было выполнено системное тестирование отчет по покрытию тестами представлен в следующей таблице Отчет по покрытию тестами не строится в gitlab!

Для реализации smtp протокола использовались регулярные выражения. Которые представленны в следующем листинге

,Ь,,Ы,,Ъ,

,Ь,,Ы,,Ъ,,Ь,,Ы,,Ъ,

```
RE_DOMAIN := ([[:alnum:]]+\\.)+[[:alnum:]]+
RE_IPv4 := ((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(25[0-5]|2[0-4][0-9]|[01]?
RE_IPv6 :=
RE_GENERAL_ADDRESS_LITERAL :=
RE_ADDRESS_LITERAL := \\[ RE_IPv4 \\]
RE_DOMAIN_LITERAL := < RE_DOMAIN >
RE_AT_DOMAIN := @ RE_DOMAIN
RE_MAILBOX := [[:alnum:]]+ RE_AT_DOMAIN // TODO (ageev) если имя содеожит '_' то ре
RE_ROUTE_PATH := (( RE_AT_DOMAIN [[:space:]]*,[[:space:]]*)* RE_AT_DOMAIN [[:space:
RE_PATH := < RE_ROUTE_PATH RE_MAILBOX >
RE_SERVER_NAME := ( RE_ADDRESS_LITERAL )|( RE_DOMAIN_LITERAL );
RE_HELLO := (helo)|(ehlo)
RE_EMPTY_PATH := <>
RE_MAIL_FROM_PATH := ( RE_EMPTY_PATH )|( RE_PATH )
RE_MAIL_FROM := mail[[:space:]]+from[[:space:]]*:[[:space:]]*
RE_RCPT_DOMAIN := <(postmaster RE_AT_DOMAIN )|(postmaster)|( RE_PATH )>
RE_RCPT_TO := rcpt[[:space:]]+to[[:space:]]*:[[:space:]]*
RE_DATA := data
RE_RSET := rset
RE_VRFY := vrfy
RE_EXPN := expn
RE_HELP := help
RE_NOOP := noop
RE_QUIT := quit
```

1.5 Список источников и литературы

1. http://rfc.com.ru/rfc2821.htm
2. http://rfc.com.ru/rfc1123
3. https://www.protocols.ru/WP/rfc5322/
4. RFC 1035 DOMAIN NAMES — IMPLEMENTATION AND SPECIFICATION https://www.pr