Санкт-Петербургский государственный институт точной механики и оптики (технический университет)

Кафедра “Компьютерные технологии”

К.А. Агафонов, Д.С. Порох, А.А. Шалыто

Реализация протокола “SMTP” на основе SWITCH-технологии

Проектная документация

Проект создан в рамках
“Движения за открытую проектную документацию”
http://is.ifmo.ru

Санкт-Петербург
2003

Содержание

ВВЕДЕНИЕ

Предлагаемая программная документация описывает комплекс программных средств, моделирующих и визуализирующих работу почтового протокола *SMTP* в соответствии со спецификацией *RFC 821*[1].

В состав программного комплекса входят четыре приложения, демонстрирующих работу протокола *SMTP*:

- два консольных приложения для операционных систем *Unix* и *Win32*;
- *Win32*-приложение с графическим интерфейсом пользователя (GUI);
- приложение, эмулирующее работу почтового сервера по протоколу *SMTP* с помощью демона *inetd* под управлением ОС *Unix*.

Все четыре приложения основаны на общем ядре.

Ядро программного комплекса моделирует работу протокола *SMTP*. Оно написано на языке *C*. При создании кода ядра для повышения централизации управления функциональные алгоритмы и локальные задачи управления были отнесены в различные части программы - системонезависимую и системозависимую соответственно.

Функциональные алгоритмы разработаны и реализованы с использованием **SWITCH-технологии [2,3]**, базирующейся на применении конечных автоматов. Алгоритмы ядра представляют собой четыре взаимосвязанных автомата, взаимодействующих по вложенности.

Главный автомат отвечает за общую работу протокола (прием команд, вывод сообщений об ошибках), а остальные автоматы выполняют различные подзадачи, например, разбор команд или прием текста письма.

Приложения и системозависимая часть ядра созданы в учебных целях, однако автоматы, созданные в настоящей работе, используются в двух коммерческих системах электронной почты для работы с протоколом *SMTP*. Использование автоматов значительно упростило отладку и поддержку следующих версий программ.

При этом отметим, что применение конечных автоматов является весьма традиционным при реализации протоколов **[4]**, однако применяемые в настоящей работе автоматы имеют более простую нотацию, а в рамках используемой технологии разработана открытая проектная документация, что не характерно для других подходов**.**

---

[1] SMTP- команды реализованы в соответствии в разделом 4.5.1. Minimum Implementation спецификации.

1. Описание команд протокола

Команды протокола *SMTP* определяют функции передачи письма или системные функции, запрашиваемые пользователем. Каждая команда – строка символов, заканчивающаяся последовательностью <CRLF>(перевод строки).

Команда протокола состоит из двух частей – кода команды и аргумента. Код команды – строка из четырех символов английского алфавита, заканчивающаяся символом <SP> (пробел). Аргумент – последовательность символов, заканчивающаяся символом <CRLF>. Важно отметить, что регистр вводимых символов не учитывается при разборе команды сервером – команды можно вводить большими и/или маленькими буквами.

1.1. Команда HELO (HELLO)

Используется для идентификации клиента[2] (инициатора диалога[3]) сервером[4]. Поле аргумента может содержать имя клиента.

Положительный ответ[5] на данную команду означает, что сервер готов к работе – отсутствуют незавершенные транзакции, все переменные и автоматы находятся в начальном состоянии.

1.2. Команда MAIL

Применяется для инициализации почтовой транзакции[6]. Поле аргумента содержит ключевое слово “FROM:” и адрес отправителя письма.

1.3. Команда RCPT (RECIPIENT)

Используется для идентификации одного получателя сообщения. Допускается многократное применение данной команды, для того, чтобы разослать сообщение нескольким адресатам. Поле аргумента содержит ключевое слово “TO:” и адрес получателя письма.

1.4. Команда DATA

Все строки текста, последующие за этой командой, должны восприниматься сервером как тело письма. Все полученные данные должны быть переданы в, так называемый, почтовый буфер[7].

Тело письма дожно заканчиваться последовательностью символов “<CRLF>.<CRLF>”. Эта последовательность прекращает работу команды DATA. Очевидно, что такая последовательность может встретиться и в самом теле письма. Поэтому в спецификации определены действия клиента и сервера в подобной ситуации. Для того, чтобы передать строку, состоящую из n точек, клиент передает n+1 точку, а сервер “выкидывает” одну из них при приеме тела письма.

---

[2] Sender-SMTP

[3] Процесс отправки электронного письма - это диалог по *TCP/IP-каналу* между клиентом и сервером(подробнее в спецификации *RFC 821*, раздел 2 “The SMTP Model”). В учебных примерах диалог происходит через консоль, а роль клиента выполняет пользователь приложения.

[4] Receiver-SMTP

[5] OK replay

[6] Mail transaction

[7] Mail data buffer

1.5. Команда RSET (RESET)

Предназначена для прерывания почтовой транзакции.

Все уже принятые данные (отправитель, адресат и тело письма) должны быть обнулены. Все переменные и автоматы должны быть установлены в исходное состояние. Сервер должен вернуть подтверждение того, что операция завершилась успешно.

1.6. Команда NOOP

Команда не требует никаких параметров предварительно выполненных команд и не выполняет никаких действий, за исключением возврата положительного ответа от сервера.

1.7. Команда QUIT

По этой команде сервер должен выдать положительный ответ и закрыть канал.

2. Описание команд управления и логирования

Для обеспечения наглядности в приложения было внедрено несколько команд управления логированием. Эти команды находятся вне ядра, обрабатываются отдельной частью приложений и управляют количеством выводимой в протокол информации. Для версии под ОС *Win32* управление процессом логирования осуществляется через элементы GUI.

В приложениях доступны три уровня логирования:

1. “Silent” – логирование работы автоматов отключено;
2. “Start/End” – логирование запуска и завершения работы автоматов, а также значения переменных;
3. “Action” – полное логирование. В файл протокола записывается также и информация о событиях.

Команды управления логированием:

_baN/_eaN – включить/отключить логирование работы автомата N;
__lM – установить уровень логирования равным M (1,2 или 3).

При моделировании и визуализации работы протокола SMTP обычно требуется обработка исключительных ситуаций. В приложениях имеется возможность создавать две исключительные ситуации – разрыв соединения клиентом и завершения работы сервера. Первое событие эмулируется по команде “**___e**”, а второе – по команде “**___b**”.

3. Описание пользовательского интерфейса приложений

Как отмечено выше, в работе созданы три приложения, демонстрирующих работу протокола *SMTP* – *Win32*-приложение с графическим интерфейсом пользователя, а также два консольных приложения для операционных систем *Windows* и *Unix*. Четвертое приложение является рабочим и пользовательского интерфейса не имеет.

3.1. Win32-приложение

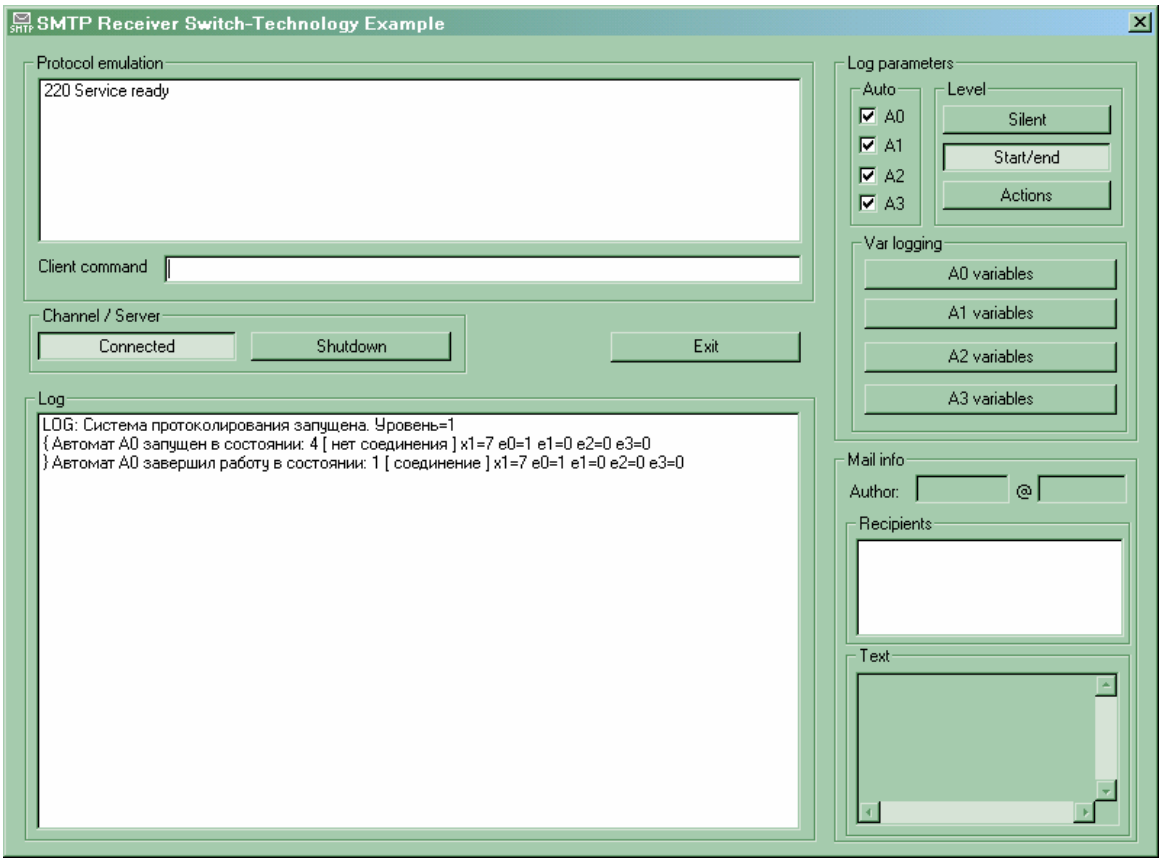


Рис. 1. Внешний вид приложения с графическим интерфейсом

Основное окно приложения состоит из нескольких областей (рис.1):

- эмуляции работы протокола (“Protocol emulation”);
- визуализации результатов работы протокола (“Mail info”);
- отображения протоколов работы (“Log”);
- управления логированием (“Log parameters”);
- управления каналом и сервером (“Channel/Server”);
- кнопки выход(“Exit”).

Область эмуляции работы протокола (“Protocol emulation”) разделена на две части. В нижней части (поле “Client command”) пользователь может вводить разрешенные команды, а в верхней - отображаются сами команды и сообщения сервера.

Область визуализации результатов работы протокола (“Mail info”). В этой области окна программы пользователь может увидеть текущий статус принимаемого письма. В статус письма входят имя и домен автора, список адресов получателей (*Recipients*) и текст письма (*Text*). Данные вносятся автоматически при получения соответствующих команд сервером.

В *области отображения протоколов работы (“Log”)* отображаются: изменения в состояниях автоматов, для которых разрешено логирование, а также системные сообщения об изменениях в настройках системы логирования.

В *области управления логированием (“Log parameters”)* расположены элементы GUI, управляющие процессом логирования работы эмулятора протокола *SMTP*. По своему выбору пользователь может:

- изменять уровни логирования;
- выбирать автоматы, работа которых будет логироваться;
- для каждого автомата выбирать какие воздействия и значения, каких переменных логировать.

Область управления каналом и сервером(“Channel/Server”) имеет две кнопки – “Connected” и “Shutdown”. Кнопка “Connected” управляет состоянием канала. Нажатие на эту кнопку означает установление соединения между клиентом и сервером (установление канала) и начало диалога, а повторное нажатие – разрыв соединения. Кнопка “Shutdown” управляет работой сервера. Нажатие на кнопку “Shutdown” соответствует завершению работы сервера, повторное нажатие перезапускает сервер. Важно отметить, что когда сервер прекратил работу, установить с ним соединение (открыть канал) невозможно.

3.2. Консольные приложения

Консольные приложения для операционных систем *Win32* и *Unix* не имеют графического интерфейса. Поэтому, если сравнивать их с GUI-приложением, то они имеют только область эмуляции работы протокола. При этом логирование работы эмулятора происходит в файл “log.txt”, расположенный в каталоге программы.

Управление логированием, а также моделирование исключительных ситуаций в консольных приложениях происходит при помощи команд, описанных в разделе “Описание команд протокола”. При вводе команд, описанных в этом разделе, исключая команду “**___b**”, эмулятор работы протокола выведет сообщение “*500 Command unrecognized*” в соответствии со спецификацией *RFC821* протокола *SMTP*. При этом настройки системы логирования будут автоматически изменены в зависимости от пришедшей команды и соответствующее сообщение появится в *log*-файле.

4. Пример работы с приложением

Ниже в качестве примера приведены 28 строк, которые представляют собой фрагмент диалога между клиентом и сервером в процессе передачи электронного письма на сервер. Сообщения сервера выделены жирным шрифтом, а клиента – курсивом. Номера строк добавлены для удобства разбора и в реальном диалоге не используются.

```
1:  220 Service ready
2:  helo
3:  250 Requested mail action okay, completed
4:  mail from: <konst@int.spb.ru>
5:  Author name:konst
6:  Author domain:int.spb.ru
7:  250 Requested mail action okay, completed
8:  rcpt to: <poroh@int.spb.ru>
9:  Recepient name:poroh
10: Recepient domain:int.spb.ru
11: 250 Requested mail action okay, completed
12: rcpt to: <lex@int.spb.ru>
13: Recepient name:lex
14: Recepient domain:int.spb.ru
15: 250 Requested mail action okay, completed
16: data
17: 354 Start mail input; end with <CRLF>.<CRLF>
18: US Congress has authorized the President of the US to go to
19: war against
20: Iraq.
21: Please consider this an urgent request. A UN Petition for
22: Peace. A
23: Stand for Peace. Islam is not the Enemy. War is NOT the
24: Answer.
25: .
26: 250 Requested mail action okay, completed
27: quit
28: 221 Service closing transmission channel
```

В строке 1 приведено стандартное сообщение о готовности сервера принимать команды клиента. Это сообщение появляется при запуске сервера. После получения такого сообщения клиент может начать диалог. Первая команда, приходящая от клиента, команда *helo* (строка 2). На эту команду сервер отвечает положительно (строка 3). Следовательно, клиент может приступать к передаче письма на сервер.

При передаче письма обязательно указать отправителя и адресата или адресатов, причем информация об отправителе должна быть передана сначала. В строке 4 с помощью команды *mail* клиент указывает адрес отправителя *konst@int.spb.ru*. В строках 5, 6 и 7 сервер сообщает о том, что разбор адреса успешно завершен (строки 5 и 6) и выдает положительный ответ (строка 7). Аналогичным образом (строки 8 и 12) клиент сообщает двух адресатов *poroh@int.spb.ru* и *lex@int.spb.ru* соответственно. Оба раза сервер сообщает об успешной обработке команд.

После установки адресатов сообщения можно начинать передачу текста письма, что клиент и делает командой *data* (строка 16). В строке 17 сервер сообщает о том, что начал принимать текст письма, а завершением передачи будет последовательность “<CRLF>.<CRLF>”. Строки с 18 по 24-ую клиент передает текст письма, а в строке 25 он

заканчивает передачу, поставив точку в новой строке и передав еще один символ новой строки (“<CRLF>”). В строке 26 сервер сообщает об успешной обработке команды *data*. Передача письма на сервер завершена. После этого клиент может осуществить сброс текущей транзакции командой *rset* или завершить диалог командой *quit*. Клиент передает серверу команду *quit* (строка 27) и сервер сообщает о закрытии канала (строка 28).

5. Перечень и нумерация событий (e)

0 – Соединение.
1 – Разрыв соединения.
2 – Ошибка.
3 – Завершение работы.

6. Перечень и нумерация входных переменных (x)

1 – Команда.
Переменная x1 может принимать значения:
0 – HELO,
1 – MAIL,
2 – RCPT,
3 – RSET,
4 – QUIT,
5 – NOOP,
6 – DATA,
7 – неизвестная команда.
2 – Символ команды (кодовая таблица ANCII).
3 – Магазин = "MAIL".
4 – Магазин = "RCPT".
5 – Магазин = "FROM".
6 – Магазин = "TO".
7 – Нет больше символов.
8 – Предыдущий символ '.'
9 – Символ <CR>.
10 – Символ <LF>.
11 – Символ '.'

7. Перечень и нумерация выходных переменных (z)

1 – Записать информацию об инициаторе соединения.
2 – Выдать информацию об ошибке в канал и получить следующую команду.
Функция, реализующая выходную переменную z2, вызывается с параметром в зависимости от типа ошибки:
421: "Service not available, closing transmission channel" - сервис недоступен, канал передачи закрывается;
500: "Command unrecognized" – неизвестная команда;
501: "Syntax error in parameters or arguments" – синтаксическая ошибка;
503: "Bad sequence of commands" – неправильная последовательность команд;
3 – Выдать информацию об успешном завершении операции в канал и получить следующую команду.
Функция, реализующая выходную переменную z3, вызывается с параметром в зависимости от типа и результата операции:

220: "Service ready" – сервер на линии;
221: "Service closing transmission channel" – сервер закрывает канал передачи данных;
250: "Requested mail action okay, completed" – запрошенное действие выполнено;
354: "Start mail input; end with <CRLF>.<CRLF>" – сервер готов принять данные письма.

4 – Сбросить почтовый буфер.
5 – Сохранить информацию об авторе.
6 – Сохранить информацию о получателе.
7 – Сохранить тело письма.
8 – Разорвать соединение.
9 – Добавить символ в магазин.
10 – Очистить магазин.
11 – Взять следующий символ.
12 – Очистить адресную информацию.
13 – Добавить значение переменной x2 как символ в имя пользователя.
14 – Добавить точку в имя пользователя, если x2 != '.'
15 – Добавить значение переменной x2 как символ в имя домена.
16 – Добавить точку в имя домена, если x2 != '.'
17 – Прочитать символ из канала.
18 – Записать текущий символ.
19 – Записать точку.
20 – Записать <CR>.
21 – Записать <LF>.

8. Схема взаимодействия автоматов

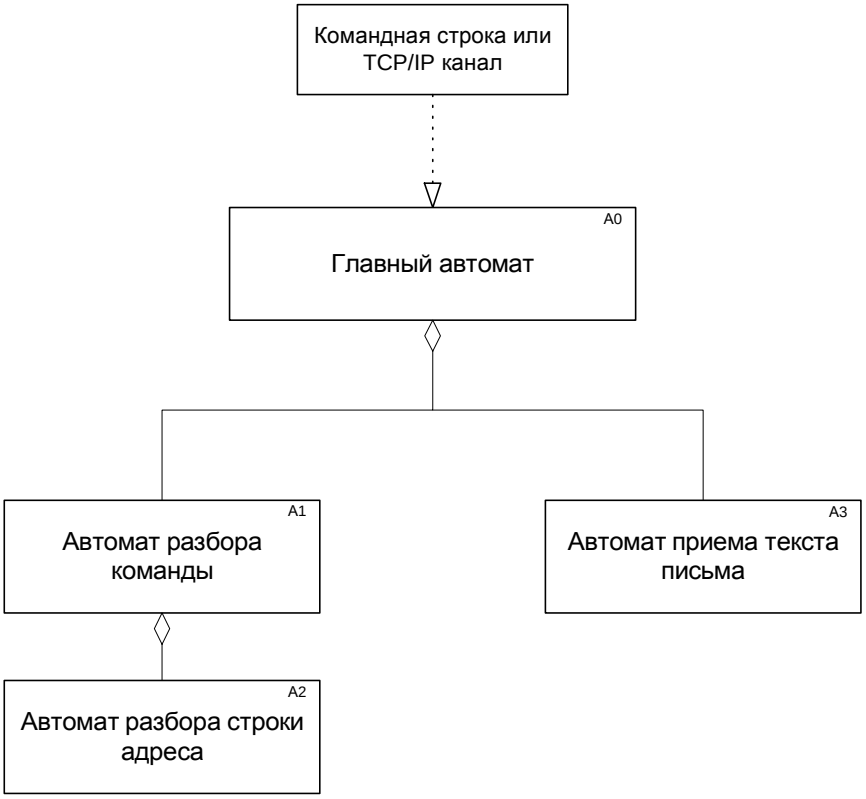


Рис. 2. Схема взаимодействия автоматов

В этой схеме используется символика потока данных и вложености, предложенная в языке UML.

9. Системонезависимая часть

9.1. Главный автомат (A0)

9.1.1. Словесное описание автомата A0

Данный автомат является главным. Работа автомата начинается при установлении соединения и завершается при обрыве соединения или выключении сервера.

Автомат управляет процессом получения данных из канала и перенаправляет их на обработку соответствующему вложенному автомату (A1 или A3). Также автомат обеспечивает запись информации о письме в почтовый буфер и запись системных сообщений в канал.

Отметим, что при отправке письма некоторые команды должны поступать в определенной последовательности. Команда DATA может поступить, если уже была успешно обработана команда RCPT. Последняя, в свою очередь, может поступить только после того как была принята и обработана команда MAIL. При этом между этими командами может быть сколько угодно команд типа NOOP. По команде RSET автомат возвращается в состояние 1 к ожиданию команды MAIL.

9.1.2. Схема связей автомата A0

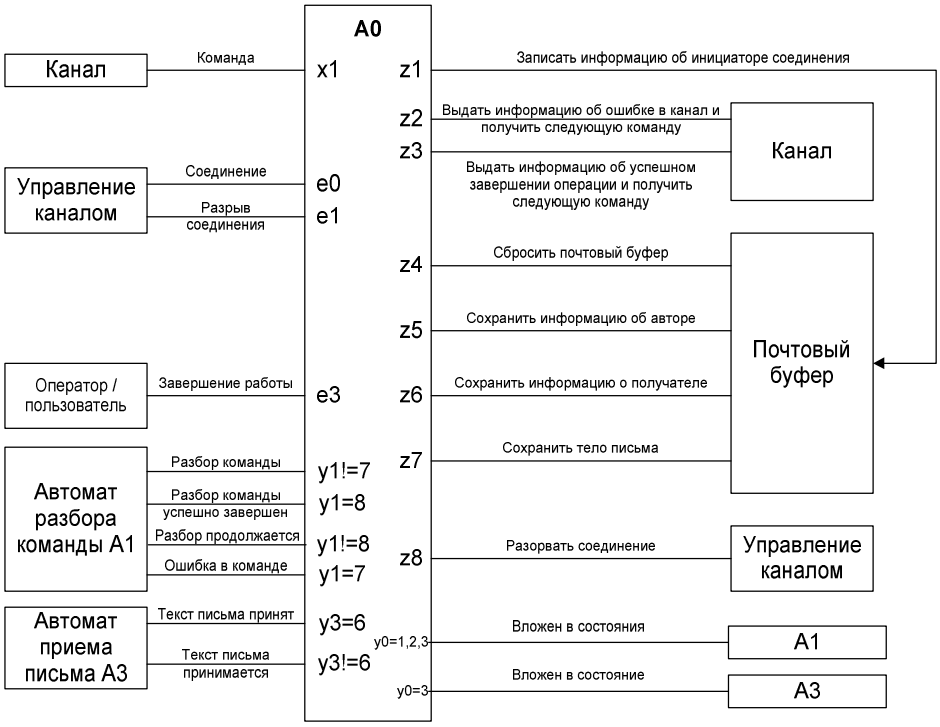


Рис. 3. Схема связей автомата A0

9.1.3. Граф переходов автомата A0

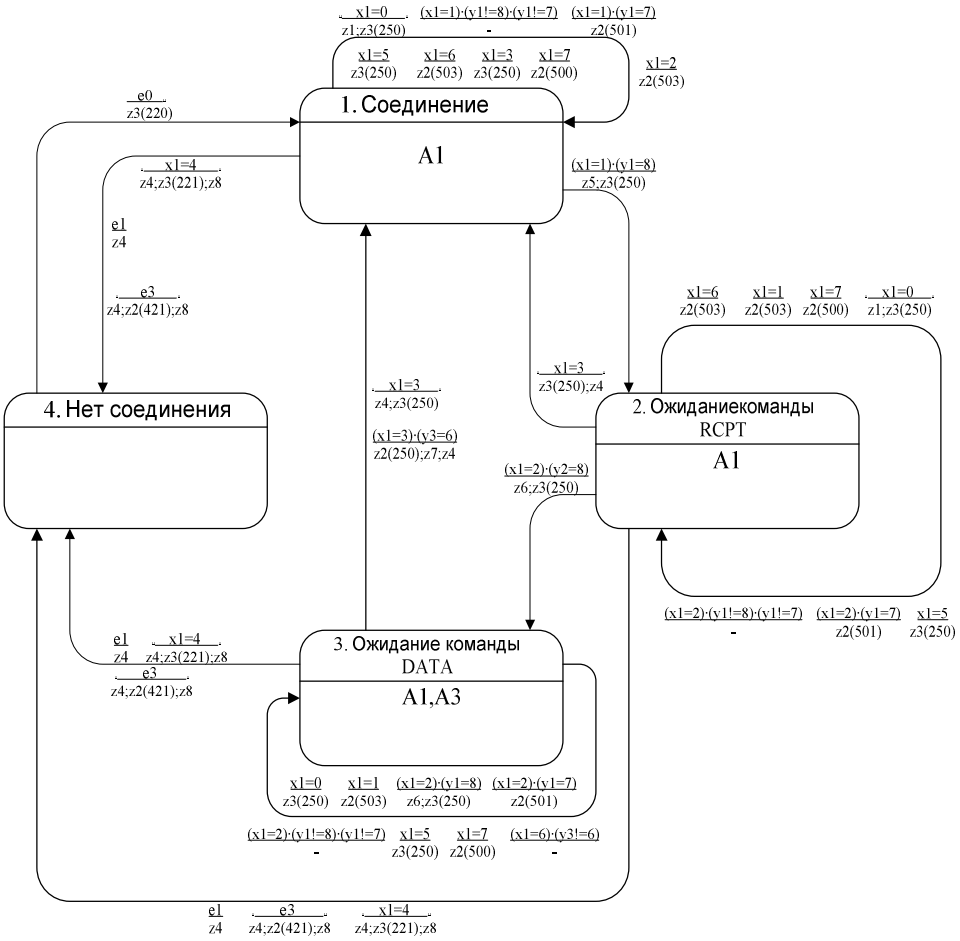


Рис. 4. Граф переходов автомата A0

9.1.4. Текст функции, реализующей автомат A0

```
void A0()
{
    switch( y0 )
    {
      case 1:
        A1();
        if (e1)                       { z4();                y0=4; break; }
        if (e3)                       { z4();z2(421);z8();   y0=4; break; }
        if (x1==0)                    { z1();z3(250);              break; }
        if (x1==1 && y1!=8 && y1!=7) {                       y0=1; break; }
        if (x1==1 && y1==8)           { z5();z3(250);        y0=2; break; }
        if (x1==1 && y1==7)           { z2(501);                   break; }
        if (x1==2)                    { z2(503);                   break; }
        if (x1==3)                    { z3(250);                   break; }
        if (x1==4)                    { z4();z3(221);z8();   y0=4; break; }
        if (x1==5)                    { z3(250);                   break; }
        if (x1==6)                    { z2(503);                   break; }
        if (x1==7)                    { z2(500);                   break; }
      break;

      case 2:
        A1();
        if (e1)                       { z4();                y0=4; break; }
        if (e3)                       { z4();z2(421);z8();   y0=4; break; }
        if (x1==0)                    { z1();z3(250);              break; }
        if (x1==1)                    { z2(503);                   break; }
        if (x1==2 && y1!=8 && y1!=7) {                       y0=2; break; }
        if (x1==2 && y1==8)           { z6();z3(250);        y0=3; break; }
        if (x1==2 && y1==7)           { z2(501);                   break; }
        if (x1==3)                    { z3(250);z4();        y0=1; break; }
        if (x1==4)                    { z4();z3(221);z8();   y0=4; break; }
        if (x1==5)                    { z3(250);                   break; }
        if (x1==6)                    { z2(503);                   break; }
        if (x1==7)                    { z2(500);                   break; }
      break;

      case 3:
        A1();
        A3();
        if (e1)                       { z4();                y0=4; break; }
        if (e3)                       { z4();z2(421);z8();   y0=4; break; }
        if (x1==0)                    { z3(250);                   break; }
        if (x1==1)                    { z2(503);                   break; }
        if (x1==2 && y1!=8 && y1!=7) {                       y0=3; break; }
        if (x1==2 && y1==8)           { z6();z3(250);              break; }
        if (x1==2 && y1==7)           { z2(501);                   break; }
        if (x1==3)                    { z4();z3(250);        y0=1; break; }
        if (x1==4)                    { z4();z3(221);z8();   y0=4; break; }
        if (x1==5)                    { z3(250);                   break; }
        if (x1==6 && y3==6)           { z3(250);z7();z4();   y0=1; break; }
        if (x1==6 && y3!=6)           {                      y0=3; break; }
        if (x1==7)                    { z2(500);                   break; }
      break;

      case 4:
        if (e0)                       { z3(220);             y0=1; break; }
      break;
    };
};
```

9.2. Автомат разбора команды (A1)

9.2.1. Словесное описание автомата A1

Автомат предназначен для разбора[8] команд протокола. Автомат A1 вызывается из головного автомата A0, когда необходимо разобрать строку с командой.

Работа автомата начинается в состоянии 1 с проверки имени команды. Если имя верно, то, проигнорировав лишние пробелы, автомат приступает к разбору аргумента. Фактически разбор аргумента заключается в анализе строки, содержащей почтовый адрес. Для разбора непосредственно почтового адреса автомат A1 вызывает вложенный автомат A2. Символу <SP> в графе переходов соответствует символ ‘ ’ в тексте программы.

9.2.2. Схема связей автомата A1

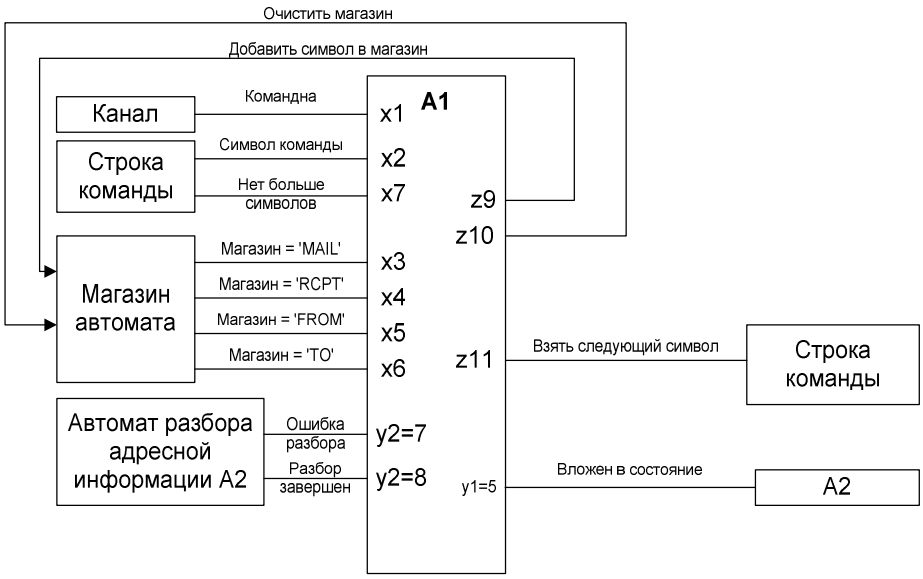
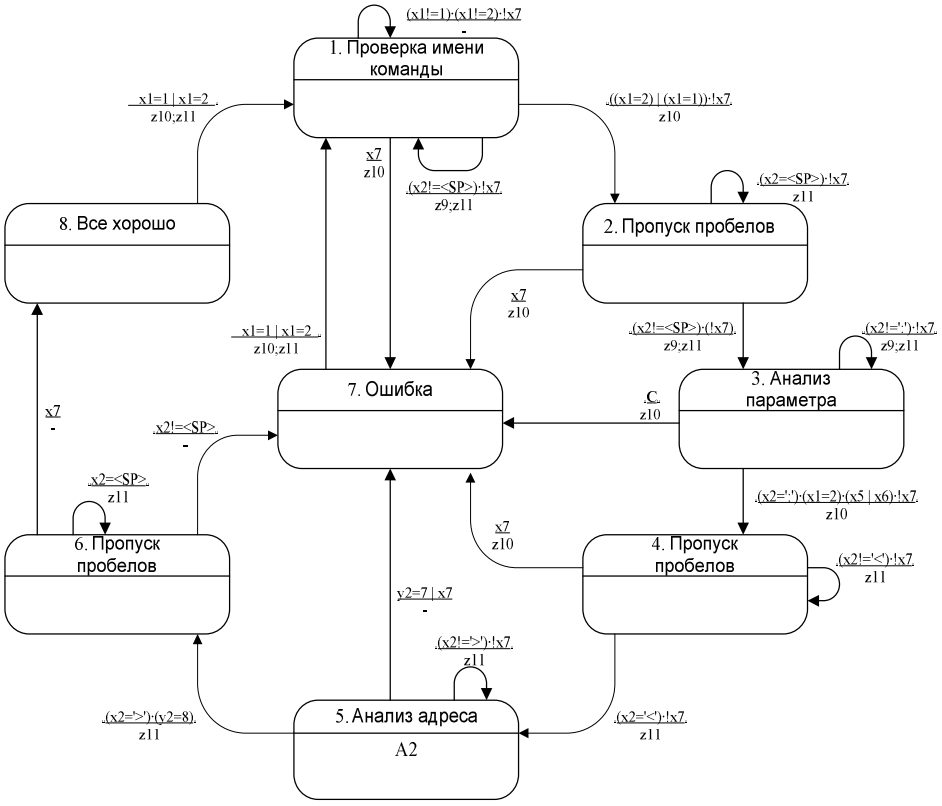


Рис. 5. Схема связей автомата A0

[8] Автомат A1 разбирает только команды, содержащие адресную информацию – MAIL и RCPT. Если автомат A1 будет вызван для другой команды, например DATA, то ничего не произойдет.

9.2.3. Граф переходов автомата A1



C= !((x2=':')·(x1=2)·x6·!x7) | !((x2=':')·(x1=2)·x5·!x7) | x7

Рис. 6. Граф переходов автомата A1

9.2.4. Текст функции, реализующей автомат A1

```
void A1()
{
    switch( y1 )
    {
      case 1:
        if (x2!=' ' && !x7)          { z9();z11();                break; }
        if (x7)                      { z10();              y1=7; break; }
        if (x1!=1 && x1!=2 && !x7)   {                     y1=1; break; }
        if ((x1==1 || x1==2) && !x7 ){ z10();              y1=2; break; }
      break ;

      case 2:
        if (x2==' ' && !x7)          { z11();                     break; }
        if (x2!=' ' && !x7)          { z9();z11();         y1=3; break; }
        if (x7)                      { z10();              y1=7; break; }
      break;

      case 3:
        if (x2!=':' && !x7)          { z9();z11();                break; }
        if (((x2==':' && x1==1 && x5)
         || (x2==':' && x1==2 && x6))
                              && !x7){ z10();              y1=4; break; }
        if ((!((x2==':' && x1==1 && x5)
          || (x2==':' && x1==2 && x6))
                              && !x7)
                              ||(x7)){ z10();              y1=7; break; }
      break;

      case 4:
        if (x2!='<' && !x7)          { z11();                     break; }
        if (x2=='<' && !x7)          {                     y1=5; break; }
        if (x7)                      { z10();              y1=7; break; }
      break;

      case 5:
        A2();
        if (x2!='>' && !x7)          { z11();                     break; }
        if (x2=='>' && y2==8)        { z11();              y1=6; break; }
        if (y2==7 || x7)             {                     y1=7; break; }
      break;

      case 6:
        if (x7)                      {                     y1=8; break; }
        if (x2!=' ')                 {                     y1=7; break; }
        if (x2==' ')                 { z11();                     break; }
      break;

      case 7:
        if (x1==1 || x1==2)          { z10();z11();        y1=1; break; }
      break;

      case 8:
        if (x1==1 || x1==2)          { z10();z11();        y1=1; break; }
      break;
   };
}
```

9.3. Автомат разбора адресной информации (A2)

9.3.1. Словесное описание A2

Автомат предназначен для разбора почтового адреса. Он начинает работу в состоянии 1 с пропуска лишних пробелов, которые могут предшествовать имени. Затем, опустив лишние пробелы, автомат разбирает имя домена и завершает работу. При этом, если в разборе имени (пользователя или домена) встретился символ <SP>, а предыдущий добавленный символ был не ‘.’, то автомат добавит в разбираемое имя (пользователя или домена) точку.

9.3.2. Схема связей автомата A2

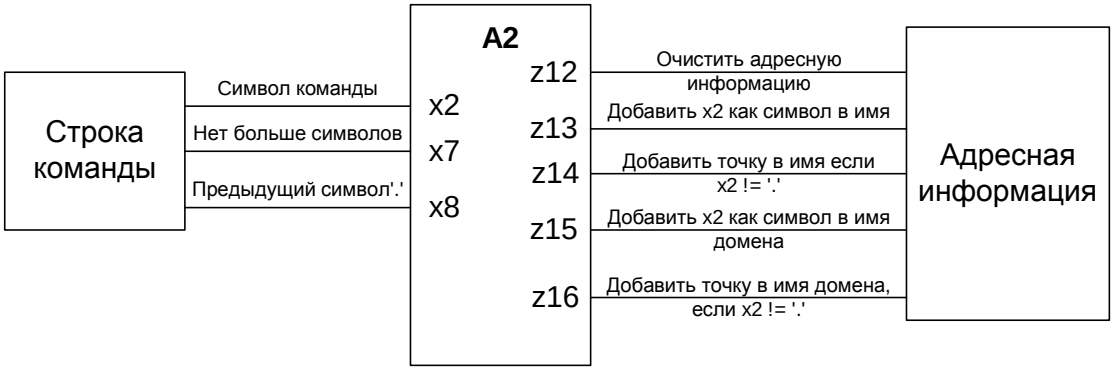


Рис. 7. Схема связей автомата A2

9.3.3. Граф переходов автомата A2

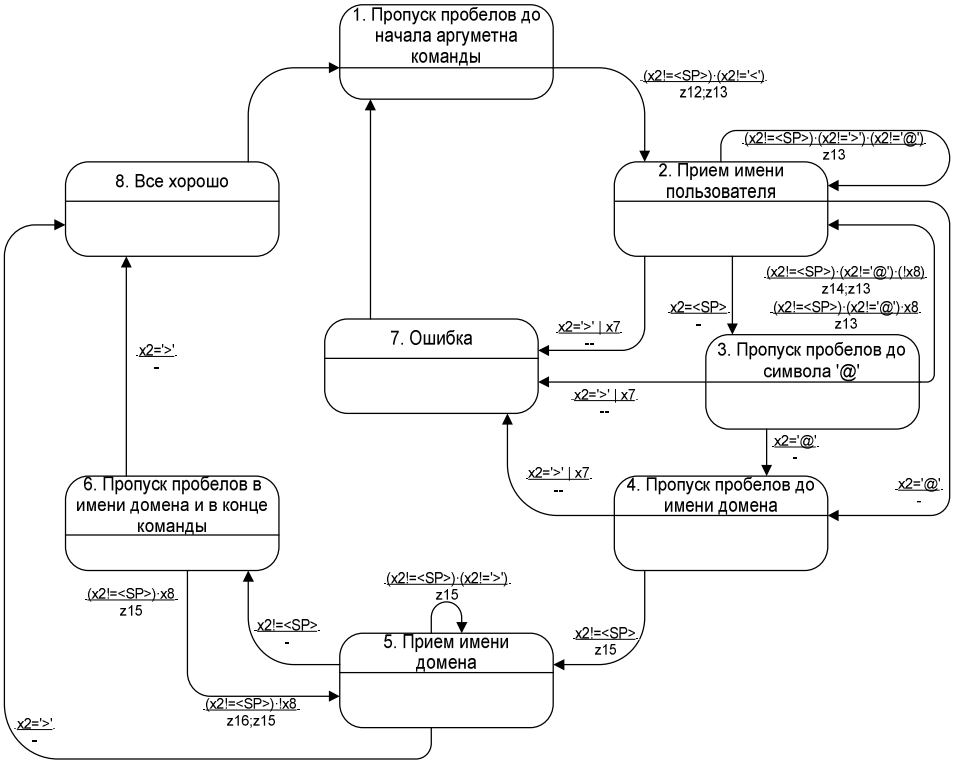


Рис. 8. Граф переходов автомата A2

9.3.4. Текст функции, реализующей автомат A2

```
void A2()
{
    switch( y2 )
    {
      case 1:
        if (x2!=' ' && x2!='<')            { z12();z13();   y2=2; break; }
      break ;

      case 2:
        if (x2!=' ' && x2!='>' && x2!='@') { z13();              break; }
        if (x2=='@')                       {                y2=4; break; }
        if (x2=='>' || x7)                 {                y2=7; break; }
        if (x2==' ')                       {                y2=3; break; }
      break;

      case 3:
        if (x2!=' ' && x2!='@' && !x8)     { z14();z13();   y2=2; break; }
        if (x2!=' ' && x2!='@' && x8)      { z13();         y2=2; break; }
        if (x2=='>' || x7)                 {                y2=7; break; }
        if (x2=='@')                       {                y2=4; break; }
      break;

      case 4:
        if (x2=='>' || x7)                 {                y2=7; break; }
        if (x2!=' ')                       { z15();         y2=5; break; }
      break;

      case 5:
        if (x2==' ')                       {                y2=6; break; }
        if (x2!=' ' && x2!='>')            { z15();              break; }
        if (x2=='>')                       {                y2=8; break; }
      break;

      case 6:
        if (x2=='>')                       {                y2=8; break; }
        if (x2!=' ' && x8)                 { z15();         y2=5; break; }
        if (x2!=' ' && !x8)                { z16();z15();   y2=5; break; }
      break;

      case 7:
                                                          y2=1;
      break;

      case 8:
                                                          y2=1;
      break;
   };
}
```

9.4. Автомат приема текста письма (А3)

9.4.1. Словесное описание автомата А3

Автомат предназначен для приема текста письма. Помимо этого автомат удаляет “лишние” символы ‘.’[9].

9.4.2. Схема связей автомата А3

Рис. 9. Схема связей автомата А3

9.4.3. Граф переходов автомата А3

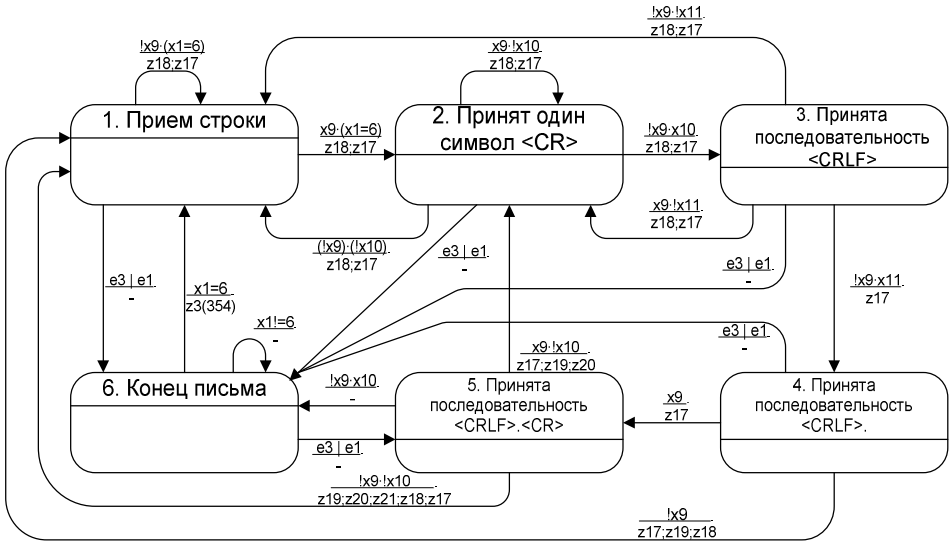


Рис. 10. Граф переходов автомата А3

[9] См. раздел 1.4 “Команда DATA”

9.4.4. Текст функции, реализующей автомат A3

```
void A3()
{
    switch( y3 )
    {
      case 1:
        if (e3 || e1)              {                      y3=6; break; }
        if (!x9 && x1==6)          { z18();z17();               break; }
        if (x9 && x1==6)           { z18();z17();         y3=2; break; }
      break ;

      case 2:
        if (e3||e1)                {                      y3=6; break; }
        if (x9 && !x10)            { z18();z17();               break; }
        if (!x9 && x10)            { z18();z17();         y3=3; break; }
        if (!x9 && !x10)           { z18();z17();         y3=1; break; }
      break ;

      case 3:
        if (e3 || e1)              {                      y3=6; break; }
        if (!x11 && x9)            { z18();z17();         y3=2; break; }
        if (x11 && !x9)            { z17();               y3=4; break; }
        if (!x9 && !x11)           { z18();z17();         y3=1; break; }
      break;

      case 4:
        if (e3 || e1)              {                      y3=6; break; }
        if (!x9)                   { z17();z19();z18();   y3=1; break; }
        if (x9)                    { z17();               y3=5; break; }
      break;

      case 5:
        if (e3 || e1)              {                      y3=6; break; }
        if (x9 && !x10)            { z17();z19();z20();   y3=2; break; }
        if (!x9 && x10)            {                      y3=6; break; }
        if (!x9 && !x10)           { z19();z20();z21();
                                            z18();z17();   y3=1; break; }
      break;

      case 6:
        if (x1==6)                 { z3(354);             y3=1; break; }
        if (x1!=6) ;
      break;
    } ;
}
```

ЗАКЛЮЧЕНИЕ

Рассмотренный пример реализации протокола *SMTP* явно демонстрирует преимущества SWITCH-технологии при реализации протоколов по сравнению со стандартным подходом [4]. Благодаря графам переходов, схемам связи и схеме взаимодействия автоматов значительно упрощаются понимание текста программы, исправление ошибок и внесение изменений.

ИСТОЧНИКИ

1. *Протокол SMTP*. Спецификация *RFC 821*.
2. *Шалыто А.А.* SWITCH-технология. Алгоритмизация программирования задач логического управления. СПб. : Наука, 1998.
3. http://is.ifmo.ru
4. *Зайцев С.С.* Описание и реализация протоколов сетей ЭВМ. М.: Наука, 1989.