Simple Mail Transfer Protocol,
uma implementação simplificada

Vinícius V. da Conceição [1], Leandro de B. Francisco[1], Brivaldo A. S. Junior[1]

[1]Faculdade de Computação– Universidade Federal de Mato Grosso do Sul (UFMS)
Caixa Postal 549 – 79070-900 – Campo Grande – MS – Brasil

{vinicius.victorio.conceicao,leandro.barros.francisco}@aluno.ufms.br,
brivaldo@facom.ufms.br

Abstract. *Simple Mail Transfer Protocol was developed to standardize from sending and receiving of emails, among diferent plataforms, through Internet. This work has as the goal to present a simple implementation of SMTP protocol developed from its initials stages. The implementation was realized using the C programming language and developed on linux computational enviroment.*

Resumo. *O Protocolo de Transferência de Correio Simples (do termo em inglês Simple Mail Transfer Protocol, ou SMTP), foi desenvolvido para padronização do envio e recebimento de emails, entre diferentes plataformas, por meio da Internet. Este trabalho tem como objetivo apresentar uma implementação simples do protocolo SMTP, desenvolvida desde suas etapas iniciais. A implementação foi realizada utilizando-se a linguagem de programação C e desenvolvida em ambiente computacional Linux.*

1. Introdução

O principal objetivo de um protocolo simples de transferência de email (SMTP [Klensin 2008]) é transferir emails com segurança e eficiência. Para isso, conta com diversas especificações, que se seguidas, criam uma poderosa ferramenta de entrega e recebimento de emails.

Um servidor SMTP é independente dos subsistemas de trasmissão e requer apenas uma conexão confiável e que entregue as mensagens em ordem. Geralmente é implementado com o protocolo TCP [Postel 1981], mas pode ser implementado por meio de outros protocolos de transporte.

Uma característica importante do SMTP é sua capacidade de transportar emails por meio de várias redes diferentes, também conhecido como *SMTP Mail Relaying*. Deste modo, um processo pode transferir um email na mesma rede local ou para outro servidor do outro lado do mundo.

2. Estrutura do SMTP

Um cenário normal de uso do protocolo SMTP envolve um cliente SMTP e um servidor. O cliente estabelece uma conexão de duas vias com o servidor e inicia-se uma "conversa"entre eles. O cliente é responsável por transferir a mensagem para um ou mais servidores e caso ocorra falhas, os erros são reportados.

Clientes de email geralmente utilizam protocolos como POP [Gellens 1998] e IMAP [Crispin 2003] para leitura dos emails. Um servidor SMTP pode ser o destino final

de uma mensagem ou ser usado como *relay*, isto é, em algum momento pode assumir o papel de cliente e conectar-se com outro servidor.

A conversa entre um cliente SMTP e um servidor da-se por meio de comandos predefinidos, tais como: `EHLO`, `MAIL`, `QUIT`, etc. Quem envia os comandos é o cliente e quem gera as respostas é o servidor. Deste modo, uma transferência de email pode ser dada por uma única conexão entre o remetente e o destinatário ou pode ocorrer através de uma séria de "saltos"de servidor para servidor até o destino final. Em qualquer um dos casos, quando o cliente termina de enviar a mensagem para o servidor, sendo este o destino final ou não, o servidor deve se responsabilizar por entregar o email com segurança ou informar ao cliente que ocorreu um erro.

Os clientes de email encontram os servidores por meio de *DNS MX Records* (Mail Exchanger Record), que são resolvidos em endereços IP.

O protocolo SMTP foi modificado de modo que o cliente e o servidor aceitem dividir funcionalidades além das já existentes no protocolo original. Essas extensões permitem que versões extendidas de clientes e servidores possam se reconhecer e o servidor comunicar o cliente sobre quais serviços ele suporta.

Graças a essas mudanças, implementações antigas devem suportar até os mecanismos básicos de extensões. Por exemplo, servidores SMTP devem suportar o comando `EHLO` mesmo se não implementam nenhuma funcionalidade extendida e clientes devem usar o comando `EHLO` ao invés de `HELO` (Por motivos de compatibilidade com implementações mais antigas, ambos devem aceitar o comando `HELO` caso o `EHLO` não seja aceito). Estas extensões adicionadas ao protocolo SMTP existem pois alguns serviços, que agora são importantes, não existiam quando o mesmo foi criado.

O servidor SMTP obtém o conteúdo do email pelo comando `DATA`, que tem de estar de acordo com as especificações da RFC 5322 [P. Resnick 2008].

Durante uma transação entre o cliente e o servidor, este mantém três *buffers* de dados. O primeiro armazena o endereço de email do remetente, o segundo armazena todos os destinatários para qual o email deve ser entregue e o terceiro armazena o conteúdo do email.

3. Procedimentos e Especificações

Esta seção descreve as principais especificações e procedimentos do protocolo SMTP, descritos: comandos, transações entre cliente e servidor e responsabilidades do servidor para o cliente.

3.1. Comandos do SMTP

Para atender as especificações da RFC 5321 [Klensin 2008], um servidor SMTP deve implementar no mínimo os seguintes comandos:

- `EHLO` (Extended Hello) ou `HELO` (Hello)
- `MAIL` (Mail)
- `RCPT` (Recipient)
- `DATA` (Data)
- `RSET` (Reset)

- HELP (Help)
- QUIT(Quit)
- NOOP (Noop)
- VRFY (Verify)

3.1.1. EHLO ou HELO

Este comando é usado para identificar um cliente SMTP para o servidor. Tem como argumento um dominio válido do cliente, se o mesmo possuir um. Caso não possua, pode-se usar o endereço IP do cliente. O cliente SMTP identifica um servidor que implementa serviços de extensões por meio do comando EHLO. Caso o servidor aceite o comando, envia para o cliente uma mensagem de boas vindas e uma lista com todos os serviços que implementa. Se o servidor não aceitar o comando, o cliente deve mandar o HELO por questões de compatibilidade

Com o EHLO (ou HELO), o cliente informa ao servidor que uma transação pode ser iniciada. Isto significa que deste ponto em diante, o cliente pode enviar comandos que, se forem na ordem correta, permitem o envio de um email. Caso o EHLO seja enviado mais de uma vez, todas as tabelas, *buffers* e estados da transação tem de ser zerados.

3.1.2. MAIL

Comando inicial de uma transação de envio de email entre o cliente e o servidor. Toma como argumento o endereço de email do remetente e argumentos adicionais podem ser passados caso o servidor aceite serviços de extensão. O MAIL somente pode ser enviado caso nenhuma transação esteja em progresso.

O envio do mesmo limpa os três *buffers* que o servidor armazena (*buffer* do endereço de email do remetente, *buffer* que armazena todos os endereços de email dos destinatários e o que armazena o conteúdo do email). O valor do endereço de email do remetente é armazenado em um dos *buffers*.

3.1.3. RCPT

Vem após o MAIL e tem como argumento o endereço de email do destinatário. Pode-se enviar mais de um RCPT durante a transação entre o cliente e o servidor. Cada uso especifica um destinatário diferente para o email. Assim como o MAIL, este pode ter argumentos adicionais se o servidor suportar serviços de extensão. Os valores aqui passados também são armazenados no *buffer* de destinatários do servidor.

3.1.4. DATA

O DATA é o comando final em uma transação de email. Quando o cliente envia o DATA, o servidor responde com o código 354, o que permite ele enviar qualquer caracter ASCII, e somente os caracteres de comandos: SP(*Space*), HT(*Halt*), CR(*Carriage Return*) e LF(*Line Feed*). Tudo o que o cliente enviou após o código 354, compõe o conteúdo do

email a ser enviado. O final da mensagem é indicado por: "<CR><LF>.<CR><LF>". O conteúdo do email, isto é, o que vem após o comando `DATA`, tem de estar de acordo com a RFC 5322 [P. Resnick 2008].

Assim que o servidor detecta que a mensagem terminou (Logo que recebe <CR><LF>.<CR><LF>), o mesmo armazena o conteúdo da mensagem no *buffers* de data. Feito isso, ele salva o objeto email localmente em um arquivo independente para ser enviado e zera os *buffers* que foram preenchidos com os comandos `MAIL`, `RCPT` e `DATA`.

A partir do momento que o servidor aceita a mensagem do cliente, o mesmo se torna responável por tudo o que possa acontecer com o email recebido. Se ocorrer uma falha durante o envio do email, o servidor deve informar o cliente sobre a falha.

3.1.5. RSET

É resposável por reiniciar toda transação de email que esteja em andamento entre o cliente e o servidor, assim como todos os *buffers* que o servidor mantém. Pode ser enviado em qualquer momento da transação.

3.1.6. HELP

Este comando serve para o servidor enviar algumas informações de ajuda para o cliente. Não é obrigatório possuir argumentos e não deve interferir nos *buffers* do servidor.

3.1.7. QUIT

Comando responsável por finalizar a conexão entre o cliente e o servidor. O cliente não encerra a conexão imediatamente, ele somente envia o `QUIT` avisando que irá encerrar e espera o servidor responder com alguma mensagem. Pode ser enviado em qualquer momento da transação.

3.1.8. NOOP

Não toma ação nenhuma, somente pede para o servidor enviar resposta com código 250. O comando `NOOP` não deve interferir nos *buffers* do servidor.

3.1.9. VRFY

Tem a função de perguntar ao servidor se o argumento enviado é identificado como um usuário local ou endereço de email. Caso o argumento seja identificado como um usuário local, informações sobre o usuário são enviadas para o cliente. O comando `VRFY` não deve interferir nos *buffers* do servidor.

3.2. Procedimentos

O processo de envio de um email do cliente para o servidor, é feito por meio de uma sequência bem definida de comandos enviados do cliente para o servidor. Se a ordem dos comandos não for seguida, o email não será aceito. Uma transação que segue a ordem esperada pelo servidor pode ser vista na **Figura 1**.

Observe que para cada comando enviado pelo cliente, o servidor responde com um código seguido de uma mensagem. O código é obrigatório e tem um significado predefinido pelo protocolo, mas a mensagem que o segue pode variar de servidor para servidor.

```
S: 220 foo.com Simple Mail Transfer Service Ready
C: EHLO bar.com
S: 250-foo.com greets bar.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<Smith@bar.com>
S: 250 OK
C: RCPT TO:<Jones@foo.com>
S: 250 OK
C: RCPT TO:<Green@foo.com>
S: 550 No such user here
C: RCPT TO:<Brown@foo.com>
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: .
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```

Figura 1. Exemplo de transação entre o servidor (S) e o cliente (C).

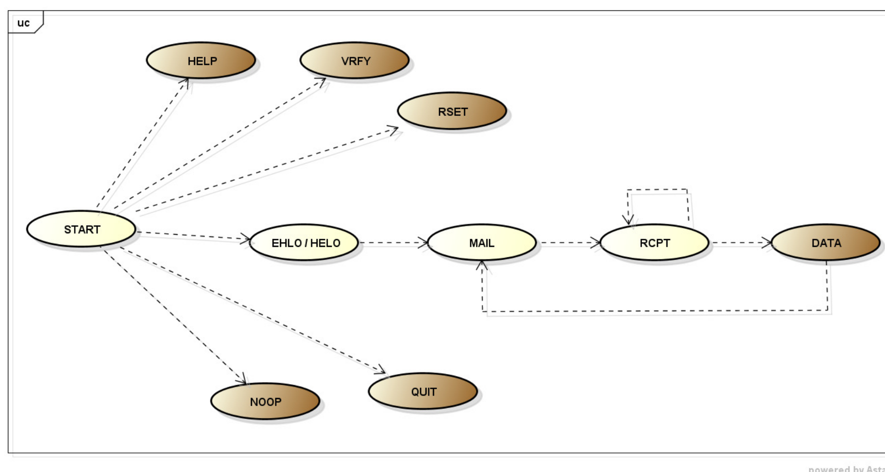

Figura 2. Sequência de comandos do protocolo SMTP retirada da RFC 5321.

Como dito anteriormente, após o servidor aceitar a mensagem do cliente, o mesmo é responsável por entregar o email para o destinatário ou informar o cliente sobre qualquer erro que possa ocorrer.

O email recebido é então colocado na fila de envio de emails que o servidor gerencia e então é entregado localmente (Caso o destinatário seja um endereço local) ou para algum outro servidor.

4. Implementação

O funcionamento do servidor SMTP consiste de quatro funções distintas, sendo uma delas o servidor SMTP propriamente dito e as outras três destinadas ao controle da fila de envio de emails. As mesmas se encontram nos arquivos:

- `smtp.h` - Implementa a função `smtp`
- `sender.h` - Implementa a função `sender`
- `senderH.h` - Implementa a função `senderH`
- `senderW.h` - Implementa a função `senderW`

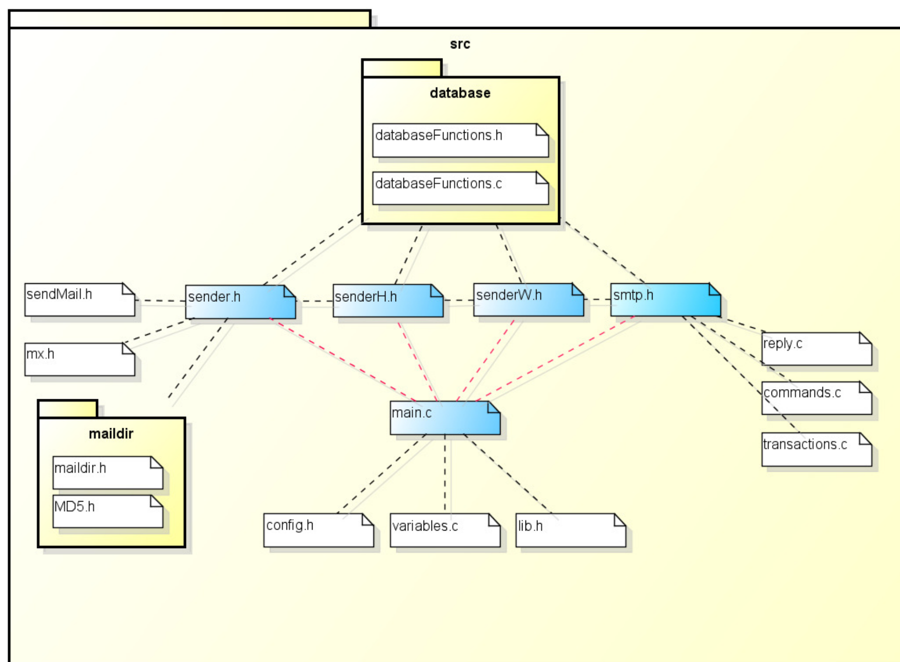

Figura 3. Estrutura de arquivos da implementação.

A estrutura da implementação possui algumas funções e módulos importantes, como: o banco de dados responsável por armazenar informações sobre os emails e usuários, funções de envio, gerenciamento dos emails, respostas do servidor, transação do email e armazenamento local.

4.1. Spool (Fila de envio de email)

É a estrutura onde são armazenados todos os emails que serão enviados pelo processo de envio do servidor SMTP. Armazena tanto emails que serão enviados localmente quanto os que serão enviados para outros servidores.

O `spool` foi implementado utilizando a biblioteca `sqlite3` [Hipp], que implementa um banco de dados localmente e utiliza-se da linguagem padrão *SQL*.

O banco de dados utilizado no controle do `spool` chama-se `local.sqlite` e possui três tabelas principais: `users`, `spool` e `rcpt`.

A tabela `spool`, armazena os seguintes valores:

- `name`: armazena o `local_part` do endereço de email do remetente, que é todo o nome que vem antes do caracter "@"em um endereço de email.
- `domain`: armazena o domínio do endereço de email do remetente.
- `data`: armazena o conteúdo do email.
- `new`: flag responsável por indicar se o email é novo ou não. O valor 1 indifica que um email é novo.
- `first_attempt`: armazena a data e hora da primeira tentativa de envio do email.

Junto a tabela `spool`, a tabela `rcpt` compõe o armazenamento do email como um todo. Esta tabela é responsável por guardar informações como:

- `spool_id`: armazena a chave primária para o spool ao qual esta entrada pertence.
- `name`: armazena o `local_part` do endereço de email de um dos destinatários.
- `domain`: armazena o domínio do endereço de email de um dos destinatários.
- `last_attempt`: armazena a data e hora da última tentativa de envio do email.

4.2. Sender

O processo responsável pelo envio dos emails reside no arquivo `sender.h`, mas ele é auxiliado por dois outros processos: um pertencente ao arquivo `senderH.h` (`sender`) e outro em `senderW.h` (`senderW`)

O funcionamento da função `sender`, que tem como parâmetro um ponteiro para o banco de dados `local.sqlite` e é aberto na função `main`, depende de uma variável de controle chamada `SIGNALFLAG`. Quando `SIGNALFLAG` está em 0, a função espera por uma alteração na *flag*. Quando a *flag* está em 1, a função sender executa uma consulta SQL no banco de dados que retorna todas as entradas na tabela `spool`, em que a coluna `new` tenha valor igual a 1. Feito isso, para cada entrada da tabela `spool` retornada, pega-se seu `id` e é realizado outra consulta SQL, mas desta vez retornando todas as entradas da tabela `rcpt`, em que a coluna `spool_id` tenha valor igual ao `id` do `spool`.

Após todas as consultas SQL, é possível obter os valores das colunas: `name`, `domain` e `data` da tabela `spool` e `name` e `domain` da tabela `rcpt`. Com estes valores, agora é possível enviar o email para o endereço de email obtido com as informações da tabela `rcpt`. O remetente do email é a combinação dos valores da tabela `spool` e o conteúdo do email é o campo `data` da tabela `spool`.

Caso aconteça alguma falha no envio do email, é enviado um email para o remetente sobre o erro ocorrido.

A *flag* de controle da função `sender`, `SIGNALFLAG`, somente é alterada quando um sinal `SIGTERM`, do sistema operacional, é lançado. A função o trata, alterando o valor da *flag* de 0 para 1.

A função `senderH`, presente no arquivo `senderH.h`, tem como parâmetros um ponteiro para o banco de dados `local.sqlite` e o pid da thread que executa a função `sender`. Este processo fica "dormindo"e "acorda"de hora em hora. O objetivo desta

função é realizar uma consulta SQL no banco de dados, retornando todas as entradas da tabela `spool` onde a coluna `new` possui valor igual a 0. Logo em seguida, altera o valor de `new` de 0 para 1, para todas as entradas retornadas pela consulta.

Feito todas as modificações, a função `senderH` envia um sinal `SIGTERM` para a thread executando a função `sender`, avisando-a para alterar a flag `SIGNALFLAG` de 0 para 1.

A função `senderW`, presente no arquivo `senderW.h`, tem um comportamento similar com a função `senderH`, exceto que além de retornar somente as entradas da tabela spool onde a coluna new possui valor igual a 0, também é realizado a verificação se a data e hora, presente na coluna `first_attempt` da tabela `spool`, é igual ou superior a um valor máximo de dias. Todas as entradas retornadas pela consulta são deletadas do banco e envia-se um email para o remetente original do email, avisando-o sobre a deleção.

4.3. Banco de dados

O arquivo `databaseFunctions.h` define múltiplas funções que são usadas para interação com o banco de dados `local.sqlite`, como:

- `saveSpool`
- `deleteSpool`
- `verifyValues`
- `erroHandler`

Todas são implementadas no arquivo `databaseFunctions.c`.

A função `saveSpool` é utilizada pelo arquivo `smtp.h`. Tem como sua principal função salvar um email no banco de dados. Para tal, manipula as tabelas `spool` e `rcpt` e retorna 1 para sucesso e 0 caso ocorra algum erro.

Outra função, é a `deleteSpool`, que é utilizada pela função `sender` e tem como principal objetivo deletar uma entrada na tabela `rcpt` ou na tabela `spool`, dependendo dos parâmetros passados à ela. Assim como `deleteSpool`, a função `erroHandler` também é utilizada pelo arquivo `sender.h`. Assim como o nome sugere, tem como objetivo tratar os erros ocorridos durante o envio de um email. Ela realiza duas operações: alterar o valor da coluna `last_attempt` para a data e a hora no momento da falha e alterar o valor do campo `new` de 1 para 0.

Por último, a função `verifyValues` recebe dois argumentos como parâmetros: uma string contendo o valor do *local_part* de um endereço de email e o valor do dominio. Com isso, realiza-se uma consulta SQL no banco de dados verificando na tabela `users`, se o endereço de email é local ou não. Verifica-se também se apenas o dominio é local, mesmo se o usuário não existir na tabela. Caso o email não for local, é enviado uma mensagem de erro para o cliente informando que o endereço de email não pode ser aceito.

4.4. Maildir (emails locais)

Para os emails de usuários locais, usava-se um padrão de estrutura de arquivo chamado **Mbox** (Mailbox) [Hall 2005], onde eram armazenados todos os emails do usuários em um mesmo arquivo. Por questões de eficiência e mais segurança, o servidor SMTP **QMAIL** [Bernstein 1998], criou um novo padrão chamado Maildir.

O padrão Maildir é uma estrutura de diretório que tem o objetivo de armazenar os emails como arquivos independentes. Foi primeiramente implementado pelo servidor SMTP chamado **QMAIL** [Bernstein 1998].

A estrutura do Maildir segue da seguinte forma:

- Tem como pasta inicial uma pasta chamada `maildir`;
- Dentro de `maildir` há três subpastas chamadas `tmp`, `cur` e `new`;
- Cada usuário local possui uma pasta chamada `maildir`.

Quando o servidor precisa enviar um email destinado a um usuário local, utiliza-se funções presentes no arquivo `Maildir.h`, que são chamadas pela função `sender`.

Assim que um email é passado para a estrutura `Maildir`, verifica-se se existe uma pasta local que armazena os emails do usuário destino. Se não existir, é criado uma pasta com o nome do usuário e dentro dela há a pasta `maildir` e suas subpastas `tmp`, `cur` e `new`. O email é primeiramente colocado na pasta `tmp` e em seguida movido para a pasta `new`.

Os emails salvos no Maildir, são armazenados em arquivos independentes que devem possuir um nome único em todo o sistema. O nome do email é formado pela junção dos seguintes valores:

- A data e hora do recebimento (Ano, Mês, Dia, Hora, Minuto e Segundo);
- Quantidade de segundos do sistema;
- E uma hash MD5 do conteúdo do arquivo.

A estrutura de diretórios Maildir, é lida e gerenciada por um cliente de email que implementa POP [Gellens 1998] ou IMAP [Crispin 2003]. O servidor SMTP é somente responsável por armazenar os emails na estrutura.

4.5. SendMail

A função `sendMail`, presente no arquivo `sendMail.h`, é a função usada pelo arquivo `sender.h` para enviar os emails. Ela conecta com o servidor destino e faz o papel de cliente SMTP, enviando comandos do protocolo (`MAIL`, `RCPT` e `DATA`, por exemplo) e recebendo códigos de respostas. Em caso de sucesso, isto é, o servidor aceita a mensagem enviada pela função, a mesma retorna 1, caso o contrário retorna 0.

4.6. Replies (Respostas do servidor)

Todas as possíveis respostas do servidor SMTP são originadas do arquivo `reply.c`. Recebe como parâmetro um número, que será o código da mensagem desejada, e uma *string* que é onde a mensagem será armazenada.

4.7. Commands (Comandos do servidor)

O arquivo `commands.c`, implementa uma função para cada comando do protocolo SMTP. Nele há funções para os comandos:

- `NOOP`;
- `QUIT`;
- `RSET`;
- `HELP`;

- VRFY;
- EHLO;
- MAIL;
- RCPT;
- DATA.

Atenção especial para as funções que implementam os comandos MAIL e RCPT. Ambas recebem como parâmetro um ponteiro para a variável que trata a transação entre o cliente e o servidor e um ponteiro para o banco de dados local.sqlite.

Existe uma *flag* de controle, mail_allowed, que tem o objetivo de verificar se o endereço de email do remetente é um endereço local ou de fora. Caso seja local, a *flag* mail_allowed é mudada para o valor 1, se não 0. A função Mail, modifica mail_allowed verificando no banco de dados, na tabela users, se o endereço é local ou não.

Quando o usuário envia o comando RCPT para o servidor, é chamada a função Rcpt que faz a mesma verificação da função Mail, mas desta vez para o endereço destino e altera a *flag* rcpt_allowed de acordo. Nesse momento, é feito uma operação OU lógico entre mail_allowed e rcpt_allowed, que verifica as seguintes informações: Se o resultado for verdade, isto é, mail_allowed ou rcpt_allowed com valor 1, a transação pode continuar sem problemas. Caso contrário, é enviado para o cliente uma mensagem com código 550, que significa que o servidor não aceitou os endereços de email passados.

A função Data, que implementa o comando DATA, usa a função saveSpool, responsável por salvar o email no banco de dados. Caso saveSpool retorne 1, é enviado para o cliente um código 250, informando que tudo occoreu bem. Caso contrário, é enviado um código 550 informando erro. Além disso, quando a função saveSpool retorna 1, é enviado para a *thread* que executa a função sender, um sinal SIGTERM para avisá-lo sobre o recebimento de um novo email.

Data também é reponsável por limpar todos os *buffers* usados na transação entre o cliente e o servidor.

4.8. Transactions (Transação entre o cliente e o servidor)

A função transactionSMTP, pertencente ao arquivo transactions.c, é a responsável por tratar toda a transação entre o cliente e o servidor.

Todo comando enviado por um cliente é tratado nesta função. Aqui é verificado a ordem correta dos comandos e é enviado códigos de respostas dependendo da ordem seguida pelo cliente. Além disso, todo comando recebido passa por um verificador de sintaxe, presente no arquivo erServ.h, que verifica se o comando foi digitado corretamente. Ela também é reponsável por preencher os três *buffers* do servidor com seus respectivos valores.

4.9. SMTP - Servidor SMTP

Essa é a principal função do servidor, presente no arquivo smtp.h. Tem como objetivo aceitar conexões na porta 25 (por padrão do protocolo) e tratá-las.

Para cada cliente que tenta conectar ao servidor, é criado uma nova *thread* que irá tratá-lo. Cada uma das *threads* criadas executam a função `transactionSMTP` e tomam como parâmetros os valores do *socket* criado para o cliente, o *pid* (*id* que identifica uma *thread* unicamente) da *thread* que executa a função `sender` e o ponteiro para o banco de dados local.

4.10. Configuração do servidor

O servidor SMTP pode ter alguns de seus valores alterados por meio de um arquivo de configuração chamado `configuration.txt`. A função `readConfig`, implementada no arquivo `config.h`, lê os valores definidos no arquivo de configuração e altera-os de acordo. Alguns valores que podem ser alterados, são:

- PORT - Porta de escuta do servidor. Por padrão o valor é 25,
- MAILDIR_PATH - Caminho onde o diretório `maildir` será criado,
- TIMEOUT - *Timeout* máximo que o servidor aguarda entre cada envio de comando do cliente. Por padrão o valor é 5 minutos (3600 segundos),
- SLEEPTIME - Tempo em que as *threads* responsáveis por `senderH` e `senderW` esperam para acessar o banco `local.sqlite`,
- MAX_DAYS - Tempo máximo de permanência de um email na fila de envio (`Spool`).

5. Conclusão

Por meio deste trabalho, desenvolveu-se uma ferramenta capaz de transferir e receber emails que segue todos os requisitos mínimos que a RFC 5321 exige. Testes reais foram realizados e comprovaram o funcionamento do servidor de email implementado, como: múltiplos clientes conectados ao mesmo tempo, emails recebidos e enviados para servidores de email reais (*gmail, hotmail, etc.*) e emails enviados com anexos.

Contudo, algumas funcionalidades ainda podem ser melhoradas, como a fila de envio de emails, que foi implementada utlizando-se a biblioteca `sqlite3`, mas poderia ser implementada usando uma estrutura de dados que gerencie arquivos e pastas de forma eficiente, armazenando os emails em arquivos independentes. Outra melhoria no servidor, seria extendê-lo adicionando funções de segurança, como SSL e TLS.

Referências

Bernstein, D. J. (1998). Qmail Maildir. Disponível em: http://www.qmail.org/man/man5/maildir.html. Acesso em 30 nov. 2014.

Crispin, M. (2003). INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. Disponível em: https://tools.ietf.org/html/rfc3501. Acesso em 30 nov. 2014.

Gellens, R. (1998). POP3 Extension Mechanism. Disponível em: http://www.ietf.org/rfc/rfc2449.txt. Acesso em 30 nov. 2014.

Hall, E. (2005). The application/mbox Media Type. Disponível em: https://tools.ietf.org/html/rfc4155. Acesso em 30 nov. 2014.

Hipp, D. R. Sqlite3. Disponível em: http://www.sqlite.org/about.html. Acesso em 30 nov. 2014.

Klensin, J. (2008). Simple Mail Transfer Protocol. Disponível em: http://tools.ietf.org/pdf/rfc5321.pdf. Acesso em 30 nov. 2014.

P. Resnick, E. (2008). Internet Message Format. Disponível em: http://tools.ietf.org/pdf/rfc5322.pdf. Acesso em 30 nov. 2014.

Postel, J. (1981). TRANSMISSION CONTROL PROTOCOL. Disponível em: https://www.ietf.org/rfc/rfc793.txt. Acesso em 30 nov. 2014.