REDES DE COMPUTADORES - TP2

Augusto Paes e Thalia Campolina

3 de dezembro de 2012

1 Introdução

No trabalho em questão foi implementada um protocolo de recebimento de e-mails POP, para o qual foi criada uma conexão entre cliente e servidor, sendo criado um programa para cada um deles. Foi implementada utilizando a biblioteca "STREAMS"de socket, e na versão ipv4. Não foi criada a versão ipv6.

Eles suportam sete (07) comandos definidos pela RFC: USER, PASS, STAT, LIST, RETR, DELE e QUIT. Além desses comandos, o usuário pode optar por ler a próxima mensagem com o comando *p*, a mensagem anterior com o comando *a*, excluir uma mensagem com o comando *e*, e sair do programa com o comando *s*.

2 Resumo do Projeto

O projeto foi realizado através de dois arquivos:

1. **cliente.c**

   Possui as funções necessárias para o funcionamento de um cliente, e realiza sua conexão com o servidor. Após o início da conexão, através da biblioteca sockets, é requisitado que o usuário digite um ou mais dos 07 comandos suportados pela implementação POP. Para isso o programa entra em loop e espera que o usuário digite esses comandos, e então os envia para o servidor.

2. **servidor.c**

   Possui as operações necessárias para o funcionamento de um servidor, e realiza sua conexão com o cliente. Após receber, enviado pelo cliente, os comandos que o usuário digitou na tela, o servidor realiza as operações necessárias e dá o retorno relativo a cada um desses comandos aos clientes.

   As funções auxiliares para o programa são a *conta_e-mails*, que realiza a contagem do número de e-mails que o diretório corrente possui. A função *tamanho_emails* calcula o tamanho de cada arquivo. A função *tamanho_total* calula a soma do tamanho de todos os arquivos.

3 Implementação

A implementação foi realizada com a biblioteca STREAMS da socket. O comando realizado para iniciar a conexão foi o socket(AF_INET, SOCK_STREAM, 0), suportando entradas variadas de IP. A família sin_family utilizada foi a AF_NET relativa a arpa net. A porta foi definida, como deveria, sendo a 1234.

O tamanho maximo das mensagens é de 10Mb, e o número máximo de arquivos(e-mails) que suporta é 99.

Foi feito um switch/case dentro de **servidor.c** para os comandos que o POP suporta, que é passado pelo cliente e interpretado pelo servidor:

1. **USER**

   O nome de usuário é salvo em uma variável de string.

2. **PASS**

   Nada é feito à respeito do password, que não será verificado. Porém, nessa etapa foi criada uma função *conta_emails* que possui um loop que verifica os arquivos na pasta atual. Os arquivos com o nome dentro do padrão passado na especificação são iterados em uma variável, retornando assim o número de arquivos na mesma. Apenas arquivos que possuem o nome de usuário (passado pelo comando USER) são considerados.

3. **STAT**

   Retorna o numero de mensagens, que também é contada com a função *conta_emails* que foi utilizada em PASS, e retorna também o tamanho total das mensagens. Esse tamanho, por sua vez, é calculado pela função *tamanho_total* que faz a soma dos tamanhos individuais dos arquivos, que foram calculados pela função *tamanho_emails*. Essa última utiliza as funções ftell, que faz a contagem de bytes até encontrar o ponteiro no final do arquivo, posicionado pela função fseek. O valor do tamanho de cada arquivo é salvo em um vetor declarado como static, chamado *vetor_tamanhos*, para que possa ser armazenado e somado futuramente.

   Esse comando não está lendo a mensagem, a não ser que o usuário digite outro enter. Aí funciona como deveria. Portanto foi incluída uma mensagem para que o usuário digite "enter"novamente.

4. **LIST**

   Retorna, assim como a STAT, o tamanho de e-mails dado pela função *tamanho_emails*. O valor total não precisa ser calculado como foi para o comando acima.

   Esse comando não está lendo a mensagem, a não ser que o usuário digite outro enter. Aí funciona como deveria. Portanto foi incluída uma mensagem para que o usuário digite "enter"novamente.

5. **RETR**

   Comando que possibilita a leitura do e-mail. Ele lê o conteúdo e-mail, salva em uma variável que guarda todo o texto, e imprime na tela do cliente.

   Esse comando não está lendo a mensagem, a não ser que o usuário digite outro enter. Aí funciona como deveria. Portanto foi incluída uma mensagem para que o usuário digite "enter"novamente.

6. **DELE**

   Deleta uma mensagem. A mensagem só deceria ser deletada depois que o comando QUIT abaixo é dado. Porém, esta deletando imediatamente.

   Esse comando não está lendo a mensagem, a não ser que o usuário digite outro enter. Aí funciona como deveria. Portanto foi incluída uma mensagem para que o usuário digite "enter"novamente.

7. **QUIT**

Termina a sessão, e deleta as mensagens que foram marcadas pela flag do comando DELE.

As seguintes funções referentes a comandos não foram implementadas:

1. **p**

   Esse comando lê a próxima mensagem.

2. **a**

   Esse comando lê a mensagem anterior.

3. **e**

   Esse comando exclui a última mensagem lida na tela.

4. **s**

   Esse comando sai do programa e fecha a conexão.

4 Conclusão

Com o trabalho em questão foi possível aprender a trabalhar com a biblioteca *sockets*, necessária para estabelecimento de uma conexão. Com essa biblioteca, e seus comandos, é possível enviar mensagens de um cliente para um servidor e vice-versa. Também aprendemos a trabalhar com IPv4 e com IPv6, de forma que foi possível abranger nosso conhecimento sobre o assunto.