DCC031 – Redes de computadores

Trabalho prático 1 – Servidor POP

Professor: Daniel Fernandes Macedo

Data de entrega: a ser definida

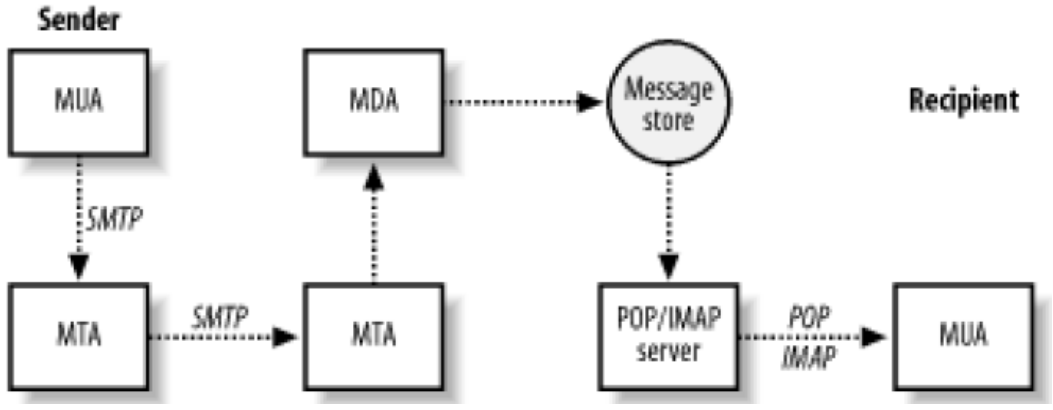
Trabalho em grupos de dois alunos.

Valor do trabalho: 20 pontos

1. Introdução

Neste trabalho iremos implementar o protocolo Post Office Protocol (POP), um protocolo de recebimento de e-mails.

O envio de e-mails emprega um conjunto de protocolos, sendo o POP a última parte. O SMTP cuida da parte de envio de e-mails de um cliente de e-mail para um MTA, ou de um MTA para outro MTA. Um MTA (Mail Transfer Agent) é uma entidade que se encarrega de repassar o e-mail até que este seja enviado ao servidor responsável pela caixa de correio do usuário, o MDA (Message Delivery Agent). A caixa de correio, que em geral também implementa um MTA, irá armazenar em disco as mensagens do usuário até que este venha recolhe-las. O usuário, então, irá conectar-se ao servidor empregando protocolos para acesso de mensagens (por exemplo o POP e o IMAP, definidos respectivamente nas RFCs 918 e 3501), utilizando um MUA (Mail User Agent, por exemplo o Outlook). A estrutura de um serviço de e-mail é mostrada na Figura abaixo.



Vocês irão desenvolver um cliente e um servidor POP, que devem suportar os comandos a seguir, definidos na RFC:

- USER
- PASS
- STAT
- LIST
- RETR
- DELE
- QUIT

Um exemplo de um diálogo entre um cliente e um servidor pode ser visto no site http://en.wikipedia.org/wiki/Post_Office_Protocol.

2. Programas a serem desenvolvidos

Iremos desenvolver quatro programas, em C, chamados *cliente4, servidor4, cliente6* e *servidor6*. Todos devem ser construídos empregando "STREAMS" na biblioteca sockets, ou seja, TCP. A diferença entre eles é que os programas "4" serão implementados utilizando IPv4, enquanto os programas "6" serão implementados utilizando o IPv6.

Como dica, é mais fácil implementar primeiro a versão IPv4, e depois simplesmente copiar a versão IPv4 e adaptar as chamadas da biblioteca socket para o IPv6. As funções são quase todas as mesmas, mas o processo (socket, bind, accept, ...) é o mesmo.

a. Programa Servidor

O programa servidor, que deverá se chamar *servidor*, deverá ser executado sem nenhum parâmetro na linha de comando. Ele irá receber uma conexão por vez, que irá sempre começar com o fornecimento de um nome de usuário e senha. Não é necessário verificar se a senha fornecida pelo cliente está correta, mas é importante guardar o nome do usuário para poder realizar os passos seguintes, que envolvem a obtenção das mensagens.

Cada usuário deverá ver **somente as mensagens destinadas para ele** no servidor. Para simplificar o julgamento do trabalho, e como não temos a parte do MTA, iremos empregar um formato que vai ser o seguinte:

- Cada e-mail vai ser um arquivo separado, que será armazenado *no mesmo diretório do executável*.
- Os arquivos terão uma estrutura de nomes bem regular, para facilitar o desenvolvimento do servidor: "usuario.numero.txt", onde:
  - **Usuario** identifica o usuário que é o destinatário da mensagem;
  - **Número** é um número único, em decimal. Ele será usado no programa servidor para atribuir um número de mensagem a um arquivo; Esse número será empregado para identificar as mensagens no comando LIST. No exemplo abaixo, devem existir os arquivos mrose.1.txt e mrose.2.txt, cada um com 120 e 200 bytes, respectivamente.

```
S:    +OK POP3 server ready 1896.697170952@dbc.mtview.ca.us
C:    USER mrose
C:    PASS senhamuitodificil
S:    +OK mrose's maildrop has 2 messages (320 octets)
C:    STAT
S:    +OK 2 320
C:    LIST
```

```
S:    +OK 2 messages (320 octets)
S:    1 120
S:    2 200
S:    .
```

IMPORTANTE: O programa servidor deverá escutar em uma porta padrão diferente da porta do POP3, que é uma porta abaixo de 1024. No Linux vocês não poderão abrir portas abaixo de 1024 sem serem administradores (root), desta forma iremos padronizar para os nossos servidores e clientes o uso da porta 1234.

b. Programa Cliente

O programa cliente, que deverá se chamar *cliente*, irá receber três parâmetros, como segue:

```
./cliente login senha servidor
```

O usuário irá fornecer o nome de login e senha na linha de comando diretamente, bem como o endereço IP do servidor ao qual se deseja conectar. Vocês irão construir um cliente de e-mail, com menus, onde o usuário é informado no início do programa quantas mensagens existem para ele. Em seguida, o usuário terá um prompt de comandos, que terá as seguintes opções. O prompt não precisa de interface gráfica, tudo pode ser feito via printf.

- Opção 1: Ler próxima mensagem (atalho 'p'). Exibe na tela a próxima mensagem. Se não houver nenhuma mensagem exibida anteriormente, deve-se exibir a primeira mensagem.
- Opção 2: Ler mensagem anterior (atalho 'a'). Exibe a mensagem anterior na tela. Caso não houver anterior, sugiro que se exiba um erro, para simplificar a programação.
- Opção 3: Excluir mensagem (atalho 'e'). Apaga a última mensagem vista na tela. Novamente, para simplificar a implementação, sugiro que a mensagem seja marcada para exclusão, e que a exclusão ocorra **somente** no momento da desconexão.
- Opção 4: Sair (atalho 's'). Sai do programa, fechando a conexão. Espera-se que, nesse momento, o cliente irá enviar um conjunto de comandos DELE para o servidor, que irá apagar as mensagens correspondentes.

Reparem que a opção por apagar as mensagens somente no final da execução é para simplificar o uso dos comandos anterior e próximo. Iremos julgar se as mensagens foram apagadas conectando novamente no servidor e listando as mensagens existentes.

3. Entrega do código

O código e a documentação devem ser entregues em um arquivo Zip (não pode ser RAR nem .tgz nem .tar.gz) no Moodle, contendo um arquivo **readme.txt** com o nome dos integrantes, e um arquivo PDF da documentação. Incluam todos os arquivos (.c, .h, makefile, não incluam executáveis ou arquivos objeto) EM UM ÚNICO DIRETÓRIO. Um makefile deve ser fornecido para a compilação do código. Parte desse trabalho envolve o aprendizado de como construir um makefile e utilizar a ferramenta Make. Este makefile, quando executado sem parâmetros, irá gerar os programas *cliente e servidor*, EXATAMENTE com esses nomes. Submissões onde os programas não seguem as especificações de parâmetros e nomes, makefiles não funcionando ou arquivos necessários faltando não serão corrigidos. Programas que não compilarem também não

serão corrigidos. O julgamento do trabalho será feito em um computador rodando o SO Linux.

Os programas desenvolvidos deverão rodar em um computador **Linux**.

4. Documentação

A documentação, além de ser entregue com o Zip junto ao código, deve ser entregue impressa ao professor até o dia seguinte à entrega, na sala de aula. Caso não tenhamos aula no dia seguinte ao prazo para envio, iremos combinar antecipadamente qual é a forma mais conveniente de entrega da documentação. Trabalhos que forem entregues no Moodle mas sem a documentação impressa **não serão** corrigidos.

O texto da documentação deve ser breve, de forma que o corretor possa entender o que foi feito no código sem ter que entender linha a linha dos arquivos. Implementações modularizadas deverão mencionar quais funções são implementadas em cada módulo ou classe. A documentação deve conter os seguintes itens:

- Sumário do problema a ser tratado
- Uma descrição sucinta dos algoritmos e dos tipos abstratos de dados, das principais funções, e procedimentos e as decisões de implementação
- Decisões de implementação que porventura estejam omissos na especificação
- Testes, mostrando que o programa está funcionando de acordo com a especificação, seguidos da sua análise
- Conclusão e referências bibliográficas

Para este trabalho, a documentação deve conter exemplos de execução do programa, mostrando a linha de comando a ser executada, um arquivo de entrada simples e a sua saída.

5. Avaliação

A avaliação do trabalho será composta pela execução dos programas desenvolvidos e pela análise da documentação. Os seguintes itens serão avaliados:

- A qualidade do código (código bem organizado, com comentários explicativos, variáveis com nomes intuitivos, modularidade, etc).
- Se o código executa as funções esperadas e da forma definida na especificação do trabalho.
- Execução correta do código em entradas de testes, a serem definidas no momento da avaliação. As entradas de teste irão exercitar a funcionalidade completa do código e testar casos especiais ou de maior dificuldade de implementação, mas que devem ser tratados por um programa correto.
- Conteúdo da documentação, que deve conter os itens mencionados anteriormente.
- Coerência e coesão da documentação (apresentação visual e organização, uso correto da língua, qualidade textual e facilidade de compreensão).

Como mencionado anteriormente, trabalhos fora da especificação (por exemplo, sem makefile, que não gerem programas com os nomes especificados, que não compilem ou não possuam os parâmetros esperados) não serão corrigidos. Trabalhos fora da especificação tomam muito trabalho do corretor, que deve entender como compilar e rodar **cada programa avaliado**, tempo esse que deveria ser empregado para avaliar a execução e corretude do código.

6. Pontos extras

Este trabalho possui cinco pontos extras, que serão dados aos grupos que implementarem um servidor que aceite mais de uma conexão ao mesmo tempo. Caso vocês implementem tal funcionalidade, vocês devem marcar **de forma clara** na documentação, para que o professor e o monitor saibam o que deve ser corrigido. Naturalmente, os cinco pontos só serão dados se a implementação for funcional e operar corretamente.

7. Desconto de nota por atraso

Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:01 do dia seguinte à entrega no relógio do Moodle, os trabalhos já estarão sujeitos a penalidades. O valor do trabalho irá decrementar 5% a cada hora de atraso. Para um aluno que entregou o trabalho à 1:38, ou seja, com 1:38 de atraso, iremos descontar 10%, empregando a fórmula a seguir:

```
Nota = valor_correção * (1-0.05*horas_atraso)
```

8. Referências

Programação em C:

- http://www.dcc.ufla.br/~giacomin/Textos/tutorialc.pdf
- ftp://ftp.unicamp.br/pub/apoio/treinamentos/linguagens/**c**.pdf
- http://www.cs.cf.ac.uk/Dave/C/CE.html
- http://www2.its.strath.ac.uk/courses/c/
- http://www.lysator.liu.se/c/bwk-tutor.html

Uso do programa make e escrita de Makefiles:

- http://comp.ist.utl.pt/ec-aed/PDFs/make.pdf
- http://haxent.com.br/people/ruda/make.html
- http://informatica.hsw.uol.com.br/programacao-em-c16.htm
- http://www.gnu.org/software/make/manual/make.html
- http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

Protocolo POP:

- http://en.wikipedia.org/wiki/Post_Office_Protocol
- RFC 918 – Post Office Protocol. http://tools.ietf.org/html/rfc918