DCC023 – Redes de Computadores

Trabalho Prático 3 – Cloud Text

1o. Semestre de 2018

Professor: Luiz Filipe M. Vieira, PhD

Data de entrega: 21/06/2018

Trabalho individual.

Valor do trabalho: 15 pontos

1. Introdução

O principal objetivo do trabalho é praticar a programação com a biblioteca **Socket** e utilizando o **protocolo TCP** com servidor tratando múltiplas conexões.

O trabalho consiste em desenvolver um sistema que permita o usuário a enviar arquivos de texto para um "pasta pessoal" e solicitar arquivos quando desejar. Também será possível registrar novos usuários.

Os programas deverão funcionar utilizando o TCP e com IPv4 e IPv6.

2. Programas a serem desenvolvidos

O trabalho prático consistirá na implementação de dois programas, o cliente e o servidor. Ambos vão receber parâmetros de entrada como explicado a seguir:

```
cliente <ip/nome> <porta>
```

```
servidor <porta>
```

O primeiro programa, o cliente, irá conectar no servidor definido pelo endereço IP (ou nome, por exemplo *login.dcc.ufmg.br*) e porta passados por parâmetro.

Já o servidor executará um serviço, que responderá paralelamente a todos os clientes que se conectarem permitidinho o acesso ao mesmo tempo. A porta a ser escutada é dada pelo parâmetro único do programa.

3. Protocolo Cloud

O cliente poderá enviar quatro tipos de mensagens ao servidor: **registro (N) e envio (S), recebimento (R) e lista (L)**. O cliente obrigatoriamente deverá ficar preso em um loop infinito, recebendo as mensagens, imprimindo seus resultados e estar pronto novamente para uma nova mensagem.

O modo como o cliente e o servidor se comunicam (se utiliza string ou struct) não faz diferença, porém a saída no programa cliente deve seguir exatamente as

mensagens das tabelas a seguir e nada a mais, pois **o trabalho será lido por um programa de correção automática**.

A mensagem de registro (N) deverá ser lida da seguinte maneira:

N [nome_do_usuário] [senha]

Exemplo: **N gabriel 12345**

Ao receber uma mensagem (N), o servidor precisará verificar se já não existe um usuário com esse identificador. O servidor deverá responder o cliente informando o sucesso ou não da operação.

O programa do cliente deverá imprimir na tela **APENAS** as seguintes mensagens após a tentativa do comando de registro:

Mensagem na Tela	Explicação
N 0	Sucesso, usuário criado.
N -1	Erro, usuário já existe.

A mensagem de envio (S) deverá ser lida da seguinte maneira:

S [nome_do_usuário] [senha] [nome_do_arquivo] [conteudo_do_arquivo]

Exemplo: **S gabriel 12345 meu_diario Esta será uma mensagem importante**

Ao receber uma mensagem (S), o servidor precisará verificar se já existe uma mensagem com este mesmo nome (e mesmo assim substituir seu conteúdo), verificar se existe o usuário e verificar a senha.

O programa do cliente deverá imprimir na tela **APENAS** as seguintes mensagens após a tentativa do comando de envio:

Mensagem na Tela	Explicação
S 1	Sucesso, mensagem foi salva porém substituiu um conteúdo já existente.
S 0	Sucesso, mensagem salva.
S -1	Erro, usuário não existe.
S -2	Erro, senha incorreta.

A mensagem de recebimento (R) deverá ser lida da seguinte maneira:

R [nome_do_usuário] [senha] [nome_do_arquivo]

Exemplo: **R gabriel 12345 meu_diario**

Ao receber uma mensagem (R), o servidor precisará verificar se a mensagem existe, se o usuário existe e se a senha confere e enviar para o cliente o resultado do comando e o conteúdo da mensagem.

O programa do cliente deverá imprimir na tela **APENAS** as seguintes mensagens após a tentativa do comando de recebimento:

Mensagem na Tela	Explicação
R 0 [mensagem]	Sucesso, no lugar de [mensagem], coloque o conteúdo da mensagem.
R -1	Erro, usuário não existe.
R -2	Erro, senha incorreta.
R -3	Erro, mensagem não encontrada.

A mensagem de lista (L) deverá ser lida da seguinte maneira:

L [nome_do_usuário] [senha]

Exemplo: **R gabriel 12345**

Ao receber uma mensagem (L), o servidor precisará verificar se o usuário existe e se a senha confere.

O programa do cliente deverá imprimir na tela **APENAS** as seguintes mensagens após a tentativa do comando de lista:

Mensagem na Tela	Explicação
L 0 [arquivo1] [arquivo 2] [...]	Sucesso, no lugar de [arquivo1], coloque o nome dos arquivos. Se não houver nenhum, deixe apenas "L 0".
L -1	Erro, usuário não existe.
L -2	Erro, senha incorreta.

ATENÇÃO: A maneira como as mensagens serão gerenciadas pelo seu programa não fará parte da avaliação. Não é necessário que se guarde os dados em arquivo. Sendo permitida a perda de todos os dados ao fechar o servidor.

ATENÇÃO: Não colocar nenhum outro tipo de saída na tela, como mensagens de boas vindas ou "esperando comando". Ao iniciar o cliente, não imprima nada e apenas espere a entrada de dados do usuário e imprima o resultado de acordo com as tabelas apresentadas.

ATENÇÃO: Qualquer ambiguidade que esta especificação possa ter, reporte imediatamente no Moodle da turma para que as dúvidas possam ser sanadas. As decisões tomadas no Moodle serão as consideradas decisões finais, podendo alterar itens nesta especificação. Portanto, fique de olho no Moodle.

ATENÇÃO: Testes de entrada serão publicados no moodle posteriormente e os mesmos serão utilizados para a avaliação.

4. 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 integrante e o comando para execução do código, e um arquivo PDF da documentação. Incluam todos os arquivos fontes (.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 dois 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.

5. Documentação

A documentação deve ser entregue com o zip junto ao código.

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
- Como foi tratado o IPv4 e o IPv6. Trabalhos que não funcionam com IPv6 perderão metade da nota.
- Testes, mostrando que o programa está funcionando de acordo com a especificação, seguidos da sua análise. Print screens mostrando o correto funcionando do cliente e servidor e exemplos de testes executados.
- Conclusão e referências bibliográficas

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).
- 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.
- Aderência ao protocolo especificado. Iremos usar ferramentas de captura do tráfego da rede (*tcpdump*, *wireshark*) e iremos analisar o código para verificar se a comunicação está implementada da forma definida na documentação.
- 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 executar **cada programa avaliado**, tempo esse que deveria ser empregado para avaliar a execução e corretude do código.

Casos de cópia não serão tolerados. Os códigos poderão ser comparados via ferramenta MOSS (http://theory.stanford.edu/~aiken/moss/).

Qualquer elemento que não esteja especificado neste documento, mas que tenha que ser inserido para que o protocolo funcione, deve ser descrito na documentação de forma explícita.

6. Desconto de nota por atraso

Não serão aceitos trabalhos atrasados.

Não há tempo para corrigi-los.

7. 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/

Material do monitor:

- http://homepages.dcc.ufmg.br/~biasi/teaching/computer-networks.html