DCC023 – Redes de Computadores

Trabalho prático 1 – Batalha Naval

1o. Semestre de 2018

Professor: Luiz Filipe M. Vieira, PhD

Monitor: Andre Gomes

Data de entrega: 30/04/2018

Trabalho individual.

Valor do trabalho: 15 pontos

1. Introdução

O principal objetivo do trabalho é praticar a programação com a biblioteca Socket e usando o protocolo TCP.

O trabalho consiste em desenvolver o jogo batalha naval para o paradigma cliente-servidor. Esta versão pode funcionar com apenas um cliente conectado. Você deverá criar um programa cliente e outro programa servidor.

Batalha naval é um jogo de tabuleiro no qual o objetivo é afundar a frota de navios do inimigo. Inicialmente é definido um tabuleiro (matriz) de 10 x 10. Os quadrados do tabuleiro são identificados na horizontal por números e na vertical por letras. Os tabuleiros não são visíveis entre os jogadores. Os tipos de navios são, ocupando quadrados adjacentes na horizontal ou vertical: porta-aviões (cinco quadrados), navios-tanque (quatro quadrados), contratorpedeiros (três quadrados) e submarinos (dois quadrados). O número de navios são: 1, 2, 3 e 4 respectivamente. O jogo consiste em escolher um espaço do tabuleiro oponente e tentar acertar um dos navios da frota. O navio afunda quando todos quadrados forem adivinhados pelo oponente, quem perder todos os navios primeiro perde o jogo.

Os programas irão funcionar utilizando o TCP e devem funcionar 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 irá executar um serviço, que irá tratar uma conexão por vez. A porta a ser escutada é dada pelo parâmetro único do programa.

3. Protocolo

O cliente irá se conectar ao servidor e o jogo irá se iniciar. O cliente deve posicionar sua frota. Isso pode ser feito lendo um arquivo de entrada. O servidor pode posicionar sua frota escolhendo aleatoriamente a posição inicial e a orientação (vertical, horizontal) dos navios.

O cliente irá enviar a posição escolhida para o tiro, que pode ser lida do teclado. O servidor irá responder se acertou ou não e também a posição do seu tiro, e assim por diante até o jogo terminar.

O servidor pode implementar a escolha da posição do tiro aleatoriamente, e se acertar, ele deve escolher um quadrado vizinho.

No cliente, se pressionado a letra P, ele deve imprimir (em ASCII) seu tabuleiro e o tabuleiro com o que ele já aprendeu do servidor.

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

6. 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.

7. Desconto de nota por atraso

Os trabalhos poderão ser entregues até as 23:55 do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle.

A fórmula para desconto por atraso na entrega do trabalho prático é:

Desconto = 2^{d-1}/0.32 %

onde *d* é o atraso em dias úteis. Note que após 5 dias úteis, o trabalho não pode ser mais entregue.

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/