DCC023 – Redes de Computadores

Trabalho prático 2 – Protocolo Olímpico com UDP

2o. Semestre de 2016

Professores: Ítalo Cunha e Luiz Filipe M. Vieira

Data de entrega: 22/10/2016

Trabalho individual.

Valor do trabalho: 15 pontos

1. Introdução

O objetivo do trabalho prático é aprender **programação socket com UDP**.

As Olimpíadas foram no Brasil e o Comitê Olímpico Internacional (COI) precisa da sua ajuda. O COI precisa coletar os dados das avaliações dos atletas e classificar os atletas. Baseado nessa ideia, você deverá criar um programa cliente que envia os dados dos desempenhos dos atletas para um servidor e requisições da ordem dos tempos e um programa servidor que guarda as informações e processa as requisições recebidas naquela conexão.

Os programas irão funcionar utilizando o UDP e devem ter pelo menos uma retransmissão para garantir a entrega de mensagens, além de 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 Olímpico

O cliente irá se conectar ao servidor e poderá enviar dois tipos de mensagens: dados e controle.

Como o UDP não garante a entrega de mensagens, **o cliente deve implementar pelo menos uma retransmissão**, caso a mensagem não seja recebida a primeira vez, para tentar garantir a entrega de mensagens

As mensagens de dados começam com a letra D seguido de um valor de tempo. O valor de tempo segue o formato onde haverá um número seguido do indicador de tempo (se este for maior que zero), onde **h** indicará horas, **m** indicará minutos, **s** segundos e **ms** milisegundos. Não haverá espaços entre o valor numérico e a letra que indica a unidade de tempo.

Todos os valores são terminados por um "\n". Pode ocorrer mais de um caractere de espaço, mas outros caracteres de separação tais como tab ou '\r' não deverão ser considerados.

As mensagens de controle que começam com a letra C pedem o valor de tempo de uma certa posição. Um número indica a posição desejada. Por exemplo, C 1 pede o valor da primeira posição.

O servidor deverá responder, para cada mensagem de controle, o valor do tempo da classificação requisitada , considerando todos os dados recebidos naquela conexão até aquele momento.

A mensagem de controle que começa com a letra O deve ser respondida pelo servidor enviando todos os valores recebidos em ordem.

A mensagem de controle que começa com a letra Z fecha a conexão.

O cliente e o servidor devem imprimir na tela o conteúdo das mensagens que eles receberem.

Exemplo de execução:

```
cliente:> D 1m 30s
cliente:> D 1h 20m
cliente:> D 58ms
cliente:> C 1
servidor:> 58 ms
cliente:> C 2
servidor:> 1m 30s
cliente:> C 3
servidor:> 1h 20m
cliente:> O
```

```
servidor:> 58 ms
servidor:> 1m 30s
servidor:> 1h 20m
cliente:> Z
```

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.
- Como foi tratado a retransmissão de mensagens.
- 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 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.

A fórmula para desconto por atraso na entrega do trabalho prático ou resenha 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/
- http://mrbook.org/tutorials/make/