DCC031 – Redes de computadores

Trabalho prático 0 – Cálculo de CRC de um arquivo

Professor: Daniel Fernandes Macedo

Data de entrega: a ser definida

Trabalho em grupos de dois alunos.

Valor do trabalho: 5 pontos

1. Introdução

Neste trabalho iremos implementar o mecanismo de detecção de erros conhecido como Cyclic Redundancy Check (CRC). O CRC é empregado nas redes de computadores para verificar, de forma probabilística, se uma mensagem foi ou não corrompida durante a sua transmissão.

O funcionamento do algoritmo de CRC é descrito no livro texto da disciplina na seção 3.2.1. Por ser um algoritmo conhecido, todos os livros de redes de computadores possuem a descrição do método.

2. Programa a ser desenvolvido

Iremos desenvolver um programa, em C, chamado "CRC". O *crc* receberá dois parâmetros **pela linha de comando**: primeiro, o nome do arquivo (binário ou texto) a ser codificado, e em seguida o índice do polinômio gerador, que pode ser 0 ou 1. No caso do valor zero, iremos empregar um polinômio de 8 bits, enquanto para o valor 1 iremos empregar um polinômio de 16 bits. O programa irá gerar, ao final da execução, o resto da divisão da mensagem (os dados do arquivo) quando divididos pelo polinômio selecionado, em HEXADECIMAL, com todas as letras em maiúsculo.

As duas possibilidades de polinômio são as seguintes:

0. $\mathbf{x^8 + x^2 + x + 1}$
1. $\mathbf{x^{16} + x^{15} + x^2 + 1}$

Abaixo seguem duas execuções exemplo.

```
damacedo% crc arquivo.bin 0
0xF0
damacedo% crc arquivo.bin 1
0x4F0A
```

Algumas dicas sobre a correção. O programa deve funcionar para arquivos texto (.txt) e para arquivos binários (doc, pdf, xls, ...), de tamanhos variados. Poderemos testar arquivos de alguns bytes, ou mesmo arquivos de alguns Megabytes. **Atenção:** alguns editores de texto inserem caracteres "a mais" no arquivo, que podem influenciar no resultado final da codificação. Isso pode ser verificado com o comando *ls*:

```
damacedo% echo "Cat" > arquivo.txt
damacedo% ls -l arquivo.txt -rw-r--r-- 1 damacedo staff 4 Mar
6 18:33 arquivo.txt
```

Uma forma simples de verificar o que exatamente o seu programa irá codificar é abrir o arquivo em um editor de texto que exibe o arquivo em hexadecimal (existe uma lista destes programas em http://en.wikipedia.org/wiki/Comparison_of_hex_editors).

2. 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 [3]. Este makefile, quando executado sem parâmetros, irá gerar o programa *crc*, EXATAMENTE com esse nome. 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**. Neste primeiro trabalho isso **não deve ser** um problema, mas sugiro fortemente a execução em um computador Linux mesmo para o TP0 (os trabalhos subsequentes DEVERÃO executar em Linux, e ocorrem diferenças em plataformas diferentes quando executamos programas de rede).

3. 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

Atenção: Como o CRC é muito empregado nas redes de computadores, existem diversas implementações prontas do mesmo na Internet. Por isso, a documentação deve mostrar da melhor forma possível que foram os alunos que escreveram o código, que ele não foi baixado da Internet. Tal tipo de cópia é fácil de ser identificado por ferramentas automáticas (eu vou rodá-las!), e porque os códigos da Internet são, em geral, altamente otimizados, o que os tornam visivelmente diferentes do código de um aluno de graduação típico). No caso de entregas de códigos que levantem suspeita, os membros do grupo poderão ser chamados para uma entrevista para explicar o código desenvolvido.

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.

4. 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.

5. 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)
```

6. 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/