REDES DE COMPUTADORES
TRABALHO PRÁTICO 0:
Cálculo de CRC de um arquivo

Sandro Miccoli - 2009052409 - smiccoli@dcc.ufmg.br
Leandro Duarte - 2009052271 - leandro.assis@dcc.ufmg.br

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

14 de outubro de 2012

Resumo. *Esse relatório descreve como foi implementado o algoritmo de detecção de erros conhecido como Cyclic Redundancy Check (CRC). Neste trabalho específico iremos utilizar polinômios geradores de 8 e 16 bits.*

1. INTRODUÇÃO

O CRC é uma técnica utilizada para detecção de erros de transmissão de dados digitais. As mensagens a serem transmitidas são tratadas como códigos polinomiais, sendo representada por uma série de binários.

A cada bloco de dado transmitido é anexada um valor de checagem (*check value*), que é baseado no resto de uma divisão polinomial entre o conteúdo dos dados e um polinômio gerador. Quandos esses dados são recebidos, é feito o mesmo cálculo e, caso os valores de checagem não coincidam, pode-se inferir que ocorreu um erro na propagação desses dados [Wikipedia].

2. MODELAGEM

O programa foi dividido em dois módulos, o primeiro (hex_bin) responsável pela manipulação dos dados e conversão para binário e hexadecimal. O segundo (crc) para realizar o XOR entre dois bits e efetivamente calcular o CRC.

A seguir descreveremos os algoritmos e as principais funções e procedimentos implementados para o trabalho:

2.1. Funções implementadas

- *char xor (char a, char b)*

  Descrição: Calcula a operação OU exclusivo entre os caracteres A e B recebidos por parametro.
  Parâmetros: Dois caracteres (a e b).
  Complexidade: $O(1)$, pois são realizadas apenas atribuições.

- *void CalculaCRC(char* bin, char* polinomio)*

  Descrição: Calcula o CRC a partir de uma sequência de binários e do polinômio gerador.

Parâmetros: Duas sequências de binários.
Complexidade: $O(m * n)$, sendo m o tamanho do arquivo e n o tamanho do polinômio, pois possui dois laços aninhados, o externo que percorre toda a cadeia de bits do arquivo, e o interno que faz a operação bit a bit dos restos das divisões sucessivas com o polinômio. .

- *char* ReadFile(char *name)*

  Descrição: Lê todos os dados do arquivo e retorna seu conteúdo em um buffer.
  Parâmetros: Nome do arquivo.
  Complexidade: $O(n)$, sendo n o tamanho do arquivo.

- *void BinToHex(char* bin, char *hex)*

  Descrição: Converte binário para hexadecimal.
  Parâmetros: Sequência de bits e outro vetor para armazenar o resultado em hexadecimal.
  Complexidade: $O(n)$, sendo n o tamanho da sequência de binários.

- *char * HexToBin(unsigned char c)*

  Descrição: Converte hexadecimal para binário.
  Parâmetros: Caractere em hexadecimal.
  Complexidade: $O(1)$, pois é realizada apenas atribuições.

- *void ArquivoToBin(char * bin, char * arquivo))*

  Descrição: Converte o conteúdo do arquivo para um vetor de binários.
  Parâmetros: Vetor de binário e vetor que contém o arquivo.
  Complexidade: $O(n)$, sendo n o tamanho do arquivo.

3. SOLUÇÃO PROPOSTA

Para solucionar o problema do CRC, primeiro convertemos o arquivo de entrada em um array de binários, para podermos trabalhar com ele como vimos em sala de aula.

3.1. DECISÕES DE IMPLEMENTAÇÃO

Para fins de cálculo do CRC, os caracteres de quebra de linha e fim de arquivo foram considerados como parte integrante do arquivo, por entendermos que os mesmos serão transmitidos conjuntamente ao conteúdo.

4. IMPLEMENTAÇÃO

4.1. Código

4.1.1. Arquivos .c

- **main.c** Arquivo principal que aloca memória para os binários e faz as chamadas das funções de leitura do arquivo e cálculo do CRC.
- **crc.c** Contém implementação da operação XOR e do CRC.
- **hex_bin.c** Contém funções de manipulação de sequências binárias, hexadecimais e conversão do arquivo para binário.

4.1.2. Arquivos .h

- **crc.h** Contém o cabeçalho da operação XOR e do CRC.
- **hex_bin.h** Contém o cabeçalho das funções de sequências binárias, hexadecimais e conversão do arquivo para binário.

4.2. Compilação

O programa deve ser compilado através do compilador GCC através de um makefile. Os modulos possiveis são:

- ./make
  Com este comando, o programa será compilado.
- ./make run
  Com este comando, o programa irá rodar da seguinte maneira: ./crc arquivo.txt 0.
- ./make clean
  Com este comando o executavel e outros arquivos desnecessários serão apagados.

4.3. Execução

A execução do programa tem como parâmetros:

- Um arquivo de entrada (texto ou binário)
- Um índice do polinômio gerador, que pode ser 0 ou 1, para polinômios de 8 e 16 bits, respectivamente.

  O comando para a execução do programa é da forma:

```
./crc <arquivo binario de entrada> <índice do polinômio - 0 ou 1>
```

5. AVALIAÇÃO EXPERIMENTAL

Fizemos testes com arquivos no formato texto e binário. Um teste com arquivos pequenos e outro com arquivos razoávelmente grandes (alguns mb).

Teste realizado: arquivo .txt com os caracteres 'tp', sem aspas, gerando 2 bytes + o byte de fim de arquivo.
Binário de entrada: 011101000111000000001010000000000
Polinomio 0: 100000111
Resultado do CRC: 01011000 = 0x58 em hex

Teste realizado: arquivo .txt com milhares de caracteres, gerando 907349 bytes + o byte de fim de arquivo.
Binário de entrada: não o colocaremos aqui por causa do seu tamanho
Polinomio 1: 0101010110110111
Resultado do CRC: 0111011000110110 = 0x7636 em hex

Teste realizado: arquivo binário executável, com 74062 bytes.
Binário de entrada: não o colocaremos aqui por causa do seu tamanho
Polinomio 1: 0101010110110111
Resultado do CRC: 01011000 = 0x55B7 em hex

Teste realizado: arquivo binário pdf, com 103763 bytes.
Binário de entrada: não o colocaremos aqui por causa do seu tamanho
Polinomio 1: 0101010110110111
Resultado do CRC: 01101000 = 0x68 em hex

6. CONCLUSÃO

Apesar de encontrarmos diversas implementações do CRC na internet, optamos por construir uma que se adequasse ao conteúdo que vimos em sala de aula e o que está detalhado no livro "Redes de Computadores"[Tanenbaum].

O algoritmo do CRC se mostrou eficiente para detecção de erros de trasmissão de dados, pois permite que sejam detectados 1 ou mais erros, inclusive em rajada, o que não é possível com outros algoritmos. Além disso, ele se mostrou simples, o que reduz a complexidade de detecção de erros.

Referências

Tanenbaum, A. S. *Computer Networks*.

Wikipedia. Cyclic redundancy check. `http://http://en.wikipedia.org/wiki/Cyclic_redundancy_check`.