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 efeticamente 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: $O(n)$, pois ??????????????????????.

- *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 uma 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

4. IMPLEMENTAÇÃO

4.1. Código

4.1.1. Arquivos .c

- **main.c** Arquivo principal que aloca memória para os binários do arquivo e calcula o 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, através do seguinte comando:

```
make
```

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

5.1. Resultado

6. CONCLUSÃO

Referências

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