

UNIVERSIDADE FEDERAL DE MINAS GERAIS
Redes de Computadores -- Turma Z

TP0

Cyclic Redundancy Check

Alunos: Augusto Oliveira Paes
Thalia de Almeida Magalhães Campolina
e-mail: guttolipaes@gmail.com
thaliacampolina@gmail.com

Belo Horizonte, Minas Gerais. Outubro de 2012.

CYCLIC REDUNDANCY CHECK

1. INTRODUÇÃO

O problema apresentado é o *Cyclic Redundancy Check*, conhecido como CRC. Ele consiste em detectar erros de transmissão, que podem ser detectados isoladamente ou em rajada.

O programa proposto realiza a leitura de um arquivo de forma binária e retorna o resto da divisão por um polinômio pré-definido, de forma que esse resto deve manter-se igual na recepção desse arquivo pelo outro lado da comunicação.

2. MODELAGEM E SOLUÇÃO

Para a resolução desse problema foi proposta a criação de uma função principal – denominada “ReadFromFile” – e de funções auxiliares. Na função citada, foi realizada a leitura do arquivo em binário, e cada número binário lido foi guardado em uma variável do tipo *char*. Essa variável é a representação em binário relativa à tabela ASCII e possui 8 bits, portanto pode ser representada também por um inteiro (*int*), uma vez que este possui 16 bits.

Foram utilizadas duas heurísticas: uma para polinômios geradores de 8 bits e uma para polinômios de 16 bits.

2.1 Polinômios de 8 bits

Nessa heurística foi realizada uma concatenação dos 8 bits lidos, com outros 8 bits de valor “0”. Após essa concatenação, é realizada uma divisão desses 16 bits obtidos pelo polinômio gerador de 8 bits, neste caso $x^8 + x^2 + x + 1$. Esse polinômio ser representado por um valor binário de 100000111, ou um valor decimal de 263. Então o resto dessa divisão é retornado pelo programa. Para realizar essa divisão, utilizamos o valor inteiro relativo aos 16 bits obtidos, com o valor inteiro relativo ao polinômio gerador.

2.2 Polinômios de 16 bits

Essa heurística é bem parecida com a de 8 bits, porém a leitura inicial realizada é de dois bytes, ou seja, 16 bits. Portanto são lidos dois caracteres binários, que são concatenados, e depois o resultado da concatenação é mais uma vez concatenado com os 16 bits de valor “0”, numa variável do tipo *long int*. Só então é feita a divisão pelo polinômio gerador de 16 bits, que neste caso é $x^{16} + x^{15} + x^2 + 1$, representado pelo valor binário 11000000000000101 equivalente ao decimal de 98309.

2.3 Módulos

Módulo crc.h

Esse arquivo possui a chamada das funções que estão no módulo *crc.c*, assim como sua descrição que está como comentário acima de cada uma.

Módulo crc.c

Possui as funções auxiliares que são chamadas pela função principal *ReadFromFile*. São elas as funções que deslocam à esquerda os bits de 8 ou 16 posições e as funções que calculam o módulo das divisões de acordo com cada heurística.


Módulo main.c

É o módulo que possui o programa. É onde ocorre a leitura dos argumentos que são passados no terminal pelo usuário, junto ao comando que executa o programa compilado. Também é realizada a abertura do arquivo que foi passado como parâmetro, a interpretação de qual polinômio gerador será utilizado, a chamada da função *ReadFromFile* e o fechamento do arquivo.

4. RESULTADOS

O programa foi executado em um ambiente de Linux. Os testes a seguir foram feitos em um computador com o sistema operacional Ubuntu 11.04, em um computador HP com processador Intel Core i2 Duo.

A compilação, a chamada e a saída do programa podem ser vistas de acordo com essa tela apresentada, mostrando o terminal do ambiente de trabalho:



```
Arquivo Editar Ver Pesquisar Terminal Ajuda
thalia@thatha:~/Workspace/BSI/5o_p/Redes/TP0$ make
--- COMPILANDO OBJETO "crc.o"
--- COMPILANDO PROGRAMA ---
thalia@thatha:~/Workspace/BSI/5o_p/Redes/TP0$ cat example.txt
1010101010101
thalia@thatha:~/Workspace/BSI/5o_p/Redes/TP0$ ./crc example.txt 1
0x5C0E5C0E5C0E5C0E5C0E5C0E15C90
thalia@thatha:~/Workspace/BSI/5o_p/Redes/TP0$ ./crc example.txt 0
0xB7BEB7BEB7BEB7BEB7BEB7BEB7C1
thalia@thatha:~/Workspace/BSI/5o_p/Redes/TP0$
thalia@thatha:~/Workspace/BSI/5o_p/Redes/TP0$
```

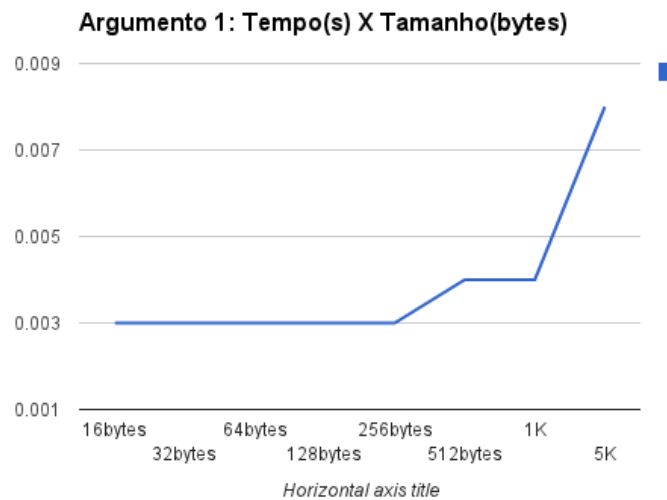
Para testar o programa foram gerados arquivos de tamanhos diferenciados, conforme observado nos gráficos apresentados a seguir. Para simplificar o resultado e padronizar as entradas, com o fim de analisar o tempo de execução do programa crc em função das mesmas, foram gerados apenas arquivos do tipo “*.txt”.

Outras extensões foram geradas para garantir o funcionamento do programa, sendo apresentadas separadamente.

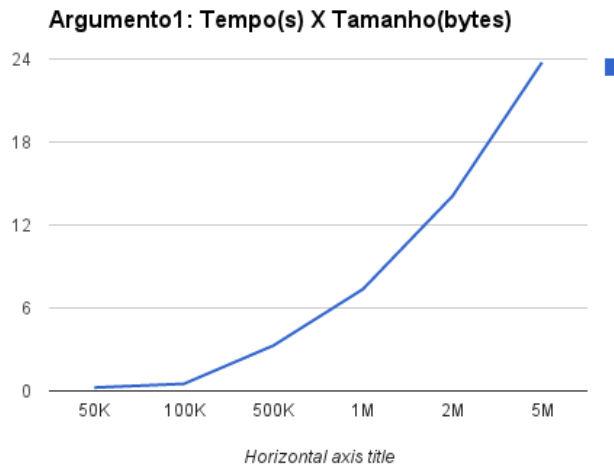
4.1 Análises de resultado para polinômio de 16 bits

4.1.1 Gráfico: tempo de execução X tamanho do arquivo

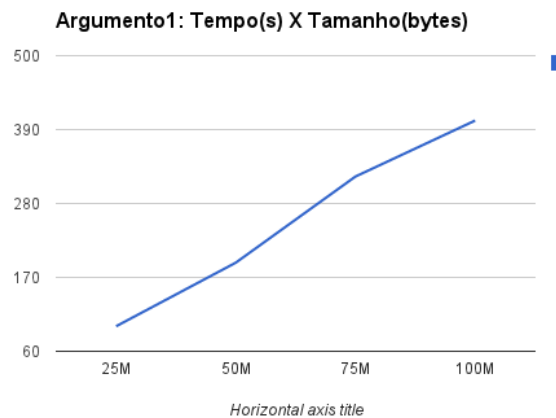
Com uma entrada de 16 bytes até os 256 bytes não houve um aumento significativo do tempo, mantendo em 0,003 segundos. A mudança pode ser vista a partir de 512 Kbytes, porém o tempo não pode ser considerado significativo, sendo que a execução do programa foi praticamente instantânea. Isso pode ser observado no gráfico abaixo:



A partir dos 50 Kbytes a mudança já foi mais significativa, porém ainda não foi atingida a casa dos segundos. Isso acontece a partir dos 500 Kbytes, que foi processado em aproximadamente 3,2 segundos. Desse tamanho para frente, o tempo foi aumentando de forma exponencial até os 23,7 segundos.

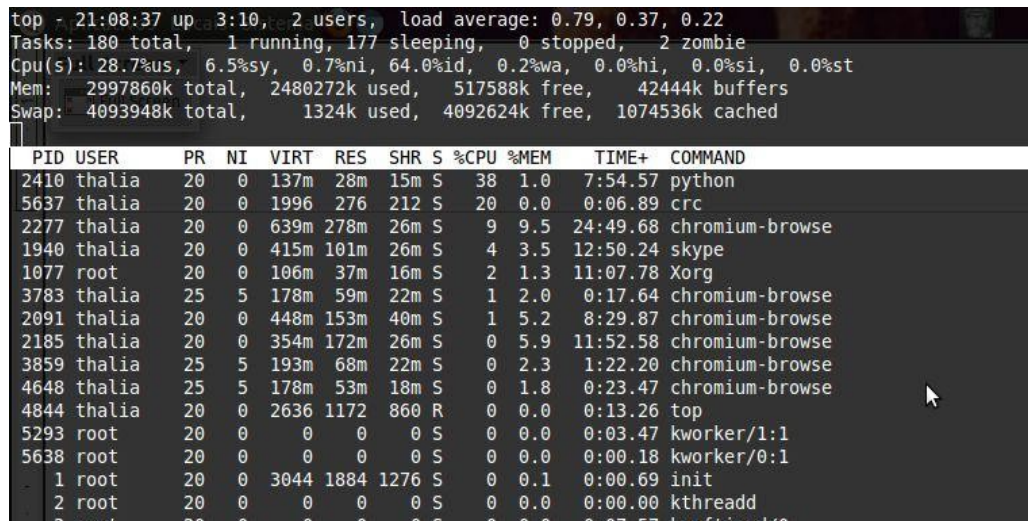


O tempo aumenta consideravelmente ao aumentarmos muito o tamanho da entrada, como podemos ver no gráfico abaixo. Demonstra-se que, a partir dos 25 Mbytes, processados em aproximadamente 97,3s, a função é praticamente linear com o aumento até os 100M, atingindo mais de 6 minutos.



4.1.2 Comando TOP durante a execução do programa crc com entrada de 100M:

Durante a execução do programa com o comando ./crc, o gasto de memória e ocupação da CPU do computador foram observados, e abaixo encontra-se a imagem da tela com o comando “top” que foi dado na execução da entrada de 100M. O gasto de memória pelo programa é insignificante, e a cpu necessária é apenas em 20% na imagem, variando entre 20 e 40%.



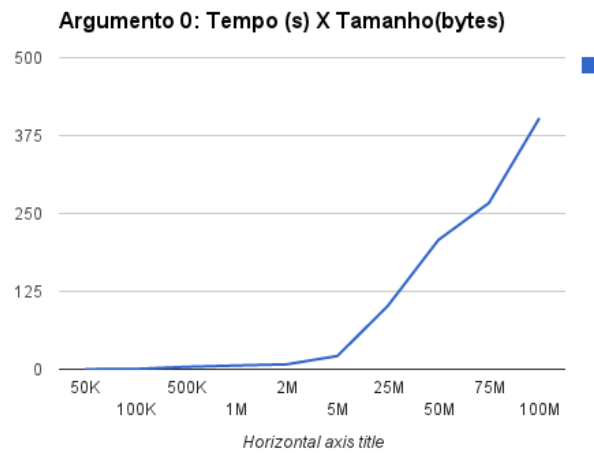
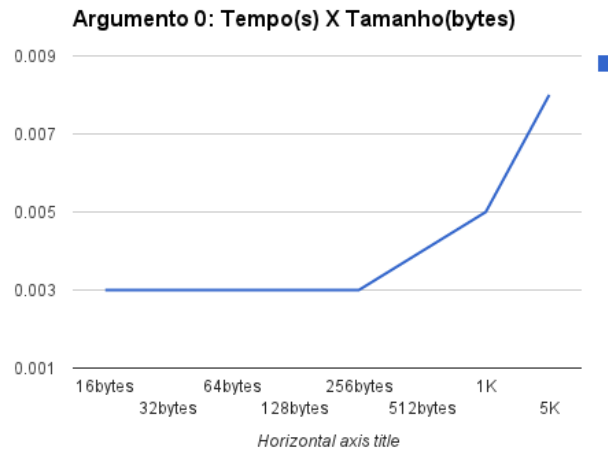
```
top - 21:08:37 up 3:10, 2 users, load average: 0.79, 0.37, 0.22
Tasks: 180 total, 1 running, 177 sleeping, 0 stopped, 2 zombie
Cpu(s): 28.7%us, 6.5%sy, 0.7%ni, 64.0%id, 0.2%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2997860k total, 2480272k used, 517588k free, 42444k buffers
Swap: 4093948k total, 1324k used, 4092624k free, 1074536k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2410	thalia	20	0	137m	28m	15m	S	38	1.0	7:54.57	python
5637	thalia	20	0	1996	276	212	S	20	0.0	0:06.89	crc
2277	thalia	20	0	639m	278m	26m	S	9	9.5	24:49.68	chromium-browse
1940	thalia	20	0	415m	101m	26m	S	4	3.5	12:50.24	skype
1077	root	20	0	106m	37m	16m	S	2	1.3	11:07.78	Xorg
3783	thalia	25	5	178m	59m	22m	S	1	2.0	0:17.64	chromium-browse
2091	thalia	20	0	448m	153m	40m	S	1	5.2	8:29.87	chromium-browse
2185	thalia	20	0	354m	172m	26m	S	0	5.9	11:52.58	chromium-browse
3859	thalia	25	5	193m	68m	22m	S	0	2.3	1:22.20	chromium-browse
4648	thalia	25	5	178m	53m	18m	S	0	1.8	0:23.47	chromium-browse
4844	thalia	20	0	2636	1172	860	R	0	0.0	0:13.26	top
5293	root	20	0	0	0	0	S	0	0.0	0:03.47	kworker/1:1
5638	root	20	0	0	0	0	S	0	0.0	0:00.18	kworker/0:1
1	root	20	0	3044	1884	1276	S	0	0.1	0:00.69	init
2	root	20	0	0	0	0	S	0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0	0.0	0:00.00	ksoftirqd/0

4.2 Análises de resultado para polinômio de 8 bits

4.2.1 Gráfico: tamanho da entrada X tempo

O comportamento é extremamente similar ao comportamento observado para o polinômio de 16 bits, o tempo de execução é muito próximo, sendo em determinadas entradas iguais, em outras menores e em outras ainda maiores, mesmo que em proporções mínimas. Portanto é possível concluir que o tamanho do polinômio não modifica significativamente o tempo de execução.



4.2.2 Comando TOP durante a execução do programa crc com entrada de 100M:

O comando “top” também foi realizado para a monitoração do programa crc com entrada de 100M para polinômio de 8 bits, tendo o desempenho muito parecido com o polinômio de 16 bits. Não existe nenhuma diferença aparente quanto ao uso da CPU e da memória com a mudança do polinômio gerador.

5. CONCLUSÃO

É um método eficiente de detecção de erros, pois detecta erros em até 1bit que esteja diferente da mensagem original.

Por ser um método simples, a verificação pode ser feita em arquivos pesados sem ocupar muita memória e CPU do computador, sendo eficiente e eficaz. A utilização de um polinômio gerador de 16 bits ou de 8 bits não altera significativamente nessas ocupações, e também não modifica muito o tempo de execução do programa.

A execução pode ser feita em arquivos de diferentes extensões, retornando portanto um número variável de caracteres pois cada extensão possui seus adicionais de formatação e outros detalhes. Portanto não é possível comparar entradas de extensões diferentes esperando obter-se o mesmo resultado.

6. BIBLIOGRAFIA

http://en.wikipedia.org/wiki/Cyclic_redundancy_check

Detecção de Erros. FULHO, Constantino Seixas. UFMG, Departamento de Engenharia Eletrônica.

Redes de Computadores. TANEMBAUM, Andrew S., 5a edição.