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

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 polinônios 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 Corel 2 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:

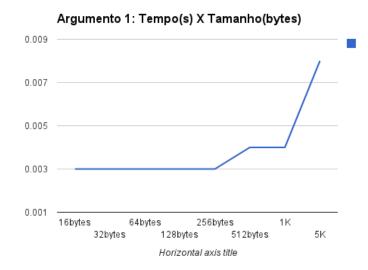
Para testar o programa foram gerados arquivos de tamanhos diferenciados, conforme observado nos gráficos apresentados a seguir. Para simplificar o resultado e padronizar as entradas, com o fim de analisar o tempo de execução do programa crc em função das mesmas, foram gerados apenas arquivos do tipo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

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

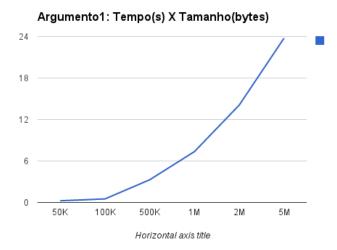
4.1 Análises de resultado para polinômio de 16 bits

4.1.1 Gráfico: tempo de execução X tamanho do arquivo

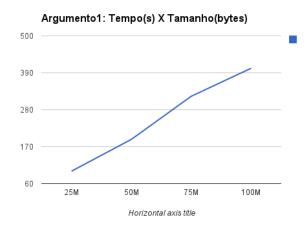
Com uma entrada de 16 bytes até os 256 bytes não houve um aumento significativo do tempo, mantendo em 0,003 segundos. A mudança pode ser vista a partir de 512 Kbytes, porém o tempo não pode ser considerado significativo, sendo que a execução do programa foi praticamente instantânea. Isso pode ser observado 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:

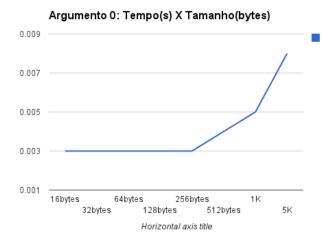
Durante a execução do programa com o comando ./crc, o gasto de memória e ocupação da CPU do computador foram observados, e abaixo encontra-se a imagem da tela com o comando "top" que foi dado na execução da entrada de 100M. O gasto de memória pelo programa é insignificativo, e a cpu necessária é apenas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

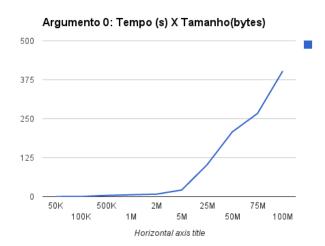
					100					opped, : , 0.0%hi	, 0.0%si, 0.0%st	
em:	2997860	k tot	al,	24802	272k i	used,		51758	88k fr	ee, 42	444k buffers	
wap:	4093948	k tot	al,	13	324k (used,	40	9262	24k fr	ee, 1074	536k cached	
PID	USER	PR	NI	VIRT	RES	SHR	S	℃PU	%MEM	TIME+	COMMAND	
2410	thalia	20	Θ	137m	28m	15m	S	38	1.0	7:54.57	python	
637	thalia	20	0	1996	276	212	S	20	0.0	0:06.89	crc	
277	thalia	20	0	639m	278m	26m	S	9	9.5	24:49.68	chromium-browse	
940	thalia	20	Θ	415m	101m	26m	S	4	3.5	12:50.24	skype	
077	root	20	Θ	106m	37m	16m	S	2	1.3	11:07.78	Xorg	
783	thalia	25	5	178m	59m	22m	S	1	2.0	0:17.64	chromium-browse	
091	thalia	20	0	448m	153m	40m	S	1	5.2	8:29.87	chromium-browse	
185	thalia	20	0	354m	172m	26m	S	0	5.9	11:52.58	chromium-browse	
859	thalia	25	5	193m	68m	22m	S	Θ	2.3	1:22.20	chromium-browse	
648	thalia	25	5	178m	53m	18m	S		1.8		chromium-browse	K .
844	thalia	20	0	2636	1172	860	R		0.0		top	R
	root	20	Θ		Θ		S		0.0		kworker/1:1	
355	root	20	Θ	Θ	0		S		0.0		kworker/0:1	
202	root	20	Θ	3044	1884	1276			0.1			
100	root	20	Õ	0	0	0	Š	ō			kthreadd	
5		20	Õ	ŏ	ő	ő	-	ő	0.0	0.07.57	keofticad (A	

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.

4.3 Testando outras entradas

4.3.1 extensões txt e doc

thalia@thatha:~/Workspace/BSI/5o p/Redes/TPO\$./crc testes/example 16by.txt 1 0x5C0E5C0E5C0E5C0E5C0E5C0E5C0E15C90 thalia@thatha:~/Workspace/BSI/5o p/Redes/TP0\$./crc testes/teste.doc 1

4.3.2 extensão xls

thalia@thatha:~/Workspace/BSI/5o p/Redes/TP0\$./crc testes/teste.xls 1 $\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FFF}0000\mathsf{FF}$

4.3.3 extensão pdf

thalia@thatha:~/Workspace/BSI/5o p/Redes/TP0\$./crc testes/teste.pdf 1 0x103A59C716960166045E31FFFF8131FFFF80E1FFFF80F5FFFF014015E0914B713E537F9DDE61376A815F1003FC0314AD14B711449E271209DFC0D836A15463B2D12E7BAE1210CBB12E8FB13917F3 282B6BB46DD1EFFFECCA3FFFF1F5116783FFFF52C66765F4BCFFFE80FF218FFFF8023FFFE825EEC5DFFFF80236D16FFFE883AFFFF80CDFFFFF9BAFFFEAE3B10333FFFE8163FFFE8055FFFF80CD2DF FFFE8163D8A3148AE173D8BA0FFFF9E70D4ED5F566E95FFFF80FF5C0E520EFFFF017CFFFF3E76FFFE80EB4C7BFFFFFFFFFF6C9F13A05FFFE8177B23015D5F100D17F3282B6BB4615D5F100D8CC09 E90DE0414B713E537F915DF51489012DEF1314137F95E8B4EEE15F9C8CC09E90B73DE25D12DEF272D124F314B9C15F9C6E1315509684ADE0E271116C5FC0D1C083493315FE25D12DEF272D24B794B2 5F564EB2308A5D2D7BE02E179490FFFF014AFFFEBDB3E46BFFFF008CFFFF0000FFFF0050FFFEACDAFFFF00B4FFFE8D4FFFF00D2FFFFC03FFFF0064FFFF7179FFFFE71BFFFE8131FFFF017211EFFF F9C62FFFF81A9FFFE80F5FE58AB88FFFF80D21075BFFFEACDA15B7816BCCFFFF008C16B6FFFFF015E9D46FFFF1364FFFF80911361DFFFF81A9FFFE81A9FFFF81BFFFF80F5FFFE80D7FFFF9F2EFFF003CFFFFA2C6FFFF8091FFFF800FFFFF012C5FEFFFF80C3FFFF8037FFFE8019FFFF0168FFFE813BFFFE8019FFFF0000FFFE9051308AFFFF8109FFFE8017FFFF003CFFFF808714519FFFE80F5FFF1 07D72881FFFFE9DBFFFE805FFFFEBBF5FFFEFDB91589EFFFF00DC11D733D6443C9FFFF0172FFFF0EA7FFFE800FFFFE80AF17533FFFF80A516A61FFFF80B9FFFE8145FFFF90<u>7</u>BFFFF8037FFFFBE17FF FE802DFFFF0136FFFE8109C3B0FFFF8073FFFE80CDFFFF003CFFFFEC61FFFF8163FA3BFFFF125DFFFF8113FFFE8131FFFE80B9F35BFFFF818BFFFE809BF8221688A15A0914AAC1654FFFF0122FFF8 12714714DF07FFFF014023F6FFFEF115768AFFFFE818BFFFF818B134A4D1EBD3C8FFFE8109E7BDFFFF0172FFFF00D2FFFF00TAFFFFE3D7FFFF8087FFFE816DFFFF3750FFFF0D2AFFFFBE03F FFF012CFFFF81A912DC4175A0FFFF014AFFFF392DFFFE8019FFFE8019FFFF00D214311FFFF00DCFFFF001E4D51FFFE818BFFFE805FFFF80B9B588FFFF015EFFFF0000FFFF8177FFFE8091FFFF8023 329FFFFF816314CF765A0FFFF8C03FFFF0118FFFF8087FFFE8087FFFF44092E75FFFF708E116B1FFFF014A37D8FFFFAD20FFFF4BD2FFFF8127FFF804BDFBFFFFE9E66FFFF019AFFFF8131FFFF00FA FFFE8037FFFE812FFFF80FAFFFF80A5FFFF9934FFFF8127FFFF001E755BFFFF017CFFFE804BFFFE80FF5574FFF8091FFF80FFFFE816DFFFE8127FFFF009679C4FFF8087FFFFCFA1FF FF81317FC2FFFEF0A7FFFE8177FFF6848FFFF0168FFFE805FFFFED7BEFFFF811AFFFF00BEFFFE8195FFFF804B247FFFFF0014FFFE8430F3FBFFFEDB10F2B1FFFF8159FFFF1612FFFF002A FFFF0136FFFE802DFFFF80C3FFFF4262FFFEC501FFFF27A6FFFE818BFFFE80912817FFFE825DFFFF00FAFFFF001EFFFF80D7FFFF81A9FFFF8096FFFF805FFFE813B12205FFFF00961749DFFFF800F FFE8055171F11EB716CB2FFFF80C3FFFF809BFFFE8023FFFE8005FFFF4C9EFFFF0078FFFF819FFFF0050FFFFE80919C950FFFFE80A53803FFFF6360FFFFC300FFFE8069CB65FFFF2DD2FFFF80D7

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.