Relatório do EP2 de MAC 0216

Nome: Odair Gonçalves de Oliveira
Número USP: 13671581

Resumo

Este relatório refere-se ao Exercício Programa 3 (EP3) da disciplina Técnicas de Programação I (MAC 0216), orientada pelo professor Daniel Macedo Batista, do curso de Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo (IME - USP). O EP3 envolveu a implementação na linguagem C das mesmas operações realizadas no EP1, ou seja, foi implementado um programa que cria hashings em código hexadecimal para strings dadas como entrada, porém utilizando bibliotecas estáticas e dinâmicas.

Conteúdo

1 Introdução e Objetivos

O EP3 consistiu na implementação de um programa que converte strings dadas como entrada em código de 32 caracteres utilizando os simbolos do código hexadecimal, assim como já feito no EP1. Entretanto, dessa vez, utilizamos a linguagem C e implementamos as funções utilizando bibliotecas estáticas e dinâmicas, que são um dos tópicos finais do conteúdo da disciplina de Técnicas de Programação I. Além dessas funções, houveram funções novas a serem implementadas e utilizamos scripts em bashscrit para facilitar a compilação dos códigos e a execução dos testes, recuperando o conteúdo principal do EP anterior.

Os objetivos desse EP eram aproximar o aluno das utilidades organizacionais de utilizar bibliotecas de funções para construir um programa, permitindo maior organização no projeto, o que é bem importante em projetos muito grandes. Além de promover uma maior familiaridade com a linguagem C de programação e tudo que ela pode oferecer. Além disso, também utilizar os conhecimentos em bash para gerar um script para facilitar a compilação.

2 Testes e Resultados

As configurações do computador em que os testes foram rodados estão expostas a seguir:

- Processador: 12º Geração - IntelCore i5-12500H;
- Memória RAM: 8 GB;
- Sistema Operacional: Ubuntu 22.04.3 LTS;
- Arquitetura: x86_64;

Os testes desse EP envolviam avaliar o tempo gasto por cada uma das 7 funções implementadas. Assim sendo, o código implementado no arquivo 'teste.c' inicialmente gerava strings aleatórias a partir de uma função que utilizada o 'rand()' e 'srand()'. Essa função não gerava valores '0', pois a função 'strlen()' utilizada para vetores de char não funcionava bem quando a string possuia um '0' no meio, pois isso era compreendido como fim da string, assim, o tamanho retornado ficava incorreto. Por esse motivo foi necessário alterar o protótipo e parâmetros de várias das funções para que o tamanho real das strings passadas também fosse um dos parâmetros. Entretanto, na função de entropia essa alteração não foi realizada, assim sendo, decide apenas gerar strings sem o '0' para os testes do arquivo 'teste.c' ao invés de ajustar novamente e incluir um novo parâmetro na função.

Após gerar as strings que iam aumentando de tamanho gradativamente, o teste é rodado 10 vezes consecutivas para cada string gerada e ao final, antes de seguir para a próxima string, é printado na tela os resultados. Confira abaixo dois resultados, um para uma string de tamanho 10.000 caracteres e outra de tamanho 100.000:

Testes - String com 10.000 caracteres:

Função	**Média**	**Mínimo**	**Máximo**
ep3CriaVetorMagico	0.000005s	0.000000s	0.000048s
ep3CalculaEntropiaShannon	0.000020s	0.000000s	0.000204s
ep1Passo1Preenche	0.000001s	0.000000s	0.000006s
ep1Passo2XOR	0.000015s	0.000002s	0.000118s
ep1Passo3Comprime	0.001149s	0.000059s	0.010911s
ep1Passo4Hash	0.000000s	0.000000s	0.000001s
ep1Passo4HashEmHexa	0.000001s	0.000001s	0.000001s
Tempo TOTAL: 0.011916s			

Testes - String com 100.000 caracteres:

Função	**Média**	**Mínimo**	**Máximo**
ep3CriaVetorMagico	0.000006s	0.000000s	0.000064s
ep3CalculaEntropiaShannon	0.000245s	0.000000s	0.002446s
ep1Passo1Preenche	0.000008s	0.000000s	0.000077s
ep1Passo2XOR	0.000148s	0.000001s	0.001467s
ep1Passo3Comprime	0.010379s	0.000031s	0.103455s
ep1Passo4Hash	0.000000s	0.000000s	0.000001s
ep1Passo4HashEmHexa	0.000000s	0.000000s	0.000001s
Tempo TOTAL: 0.107869s			

3 Conclusão

Observando os resultados expostos e também de outros testes que realizei pude notar que não há muita variação nas funções 'ep3CriaVetorMagico', 'ep1Passo4Hash' e 'ep1Passo4HashEmHexa', o que faz sentido considerando que todas trabalham com valores fixos nos loops, 256 e 16 respectivamente, não sendo influenciadas pelo tamanho da entrada. Em relação a função 'ep3CalculaEntropiaShannon' podemos ver que seu tempo é afetado linearmente pelo tamanho da entrada, o desempenho foi aproximadamente multiplicado por 10 assim como o tamanho da entrada, o que faz todo sentido, já que a função tem desempenho linear de acordo com a entrada enquanto que o resto do processo roda no máximo 255 vezes.

Já função 'ep1Passo1Preenche' começa apresentar, bem sutilmente, algum sinal de ter seu desempenho afetado pelo tamanho da entrada, porém, isso não é tão visível já que essa função apenas realiza uma cópia usando funções prontas da biblioteca pronta da linguagem C para lidar com strings. Já a função 'ep1Passo2XOR' sofre um pouco mais no seu desempenho, mas ainda não muito pois o loop principal do programa é o teto da divisão do tamanho da entrada pelo valor do tamanho do bloco (que no caso é 16).

Chegando por fim na função 'ep1Passo3Comprime' que é a função mais afetada pelo tamanho da entrada, já que há múltiplos loops dentro de loops e operações sendo realizadas. Como podemos notar o tempo também está sendo multiplicado por 10 praticamente. Essa é a função mais custosa em termos de desempenho.Vale destacar que as últimas duas citadas tem que verificar a validade do vetor mágico passado como parâmetro, o que não influencia muito no tempo, pois acaba sendo uma avaliação constante para qualquer vetor. E para concluir, em relação ao valor da entropia de Shannon calculada, é possível ver que enquanto maior a string gerada, mais o valor aproxima-se de 1 pois a distribuição dos caracteres começa tornar-se mais uniforme.