Relatório do EP3 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 uma biblioteca estática e outra dinâmica, que são um dos tópicos finais do conteúdo da disciplina de Técnicas de Programação I. Além dessas funções, foram também implementadas funções novas, uma para produzir um 'vetor mágico', uma lista de inteiros entre 0 e 255 sem repetição, e uma para calcular a entropia de Shannon de uma string de acordo com as instruções disponíveis aqui. Também utilizamos scripts em bashscript para facilitar a compilação dos códigos e a execução dos testes, recuperando o conteúdo principal do EP2, feito anteriormente.

Os objetivos desse EP foram 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, principalmente na implementação das novas funções. Além disso, outro objetivo foi retomar os conhecimentos em bash para gerar um script para facilitar a compilação e testes que seriam executados.

2 Detalhes da Implementação

Como a grande parte das funções já haviam sido implementadas anteriormente, o maior trabalho foi de reescrever elas do Python para o C, o que permitiu fazer uma análise comparativa entre as duas linguagens. Um dos grandes problemas presentes na implementação em C foi a questão dos valores dentro das strings, que muitas vezes eram interpretados como valores inteiros negativos o que gerava problemas em momentos de acessar espaços no vetor, já que não dá para acessar índices negativos nos vetores em C como no Python. Para contornar essa situação, quando notava que o valor estava negativo (valores inteiros maiores que 127 ficavam com seu valor negativo), atualizava seu valor para 256 (que era o valor máximo inteiro dos caracteres) acrescido desse valor negativo, pois assim conseguia obter o valor original.

Tive alguns problemas para a compilação, principalmente com a biblioteca dinâmica, mas com ajuda de alguns tutoriais e do monitor, acredito que deu tudo certo no final. A compilação também é responsável por gerar uma documentação Doxygen, que é possível graças ao modo que os comentários das funções implementadas foram feitas. Essa ferramenta permite gerar uma página web (em html) com as informações das suas funções. Admito que não consegui acessar as informações perfeitamente nessa página, mas ela estava sendo gerada corretamente.

Em relação a novas funções, não tive muito problema em escrever, já tinha experiência anterior com a linguagem C e C++ utilizadas em outras disciplinas, então foi muito mais fácil do que traduzir as outras funções do Python para a linguagem C. Usei o meu código em Python do EP1 como referência principal para testar e verificar se as minhs funções em C estavam fazendo o que era esperado. No fim, os testes com as strings do EP1 devolveram o mesmo código hash quando utilizada o vetor mágico fornecido, então acabou dando tudo certo.

3 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			

Valores da entropia de Shannon para strings de diferentes tamanhos:

Tamanho	**Valor da Entropia**
10	0.415241
100	0.786095
1000	0.976125
10000	0.997369
100000	0.999055
1000000	0.999274

4 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, 16 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 também, bem sutilmente se comparada a outras, algum sinal de ter seu desempenho afetado pelo tamanho da entrada, porém, isso não é tão intenso 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 nas últimas duas citadas, ocorre a verificação da validade do vetor mágico passado como parâmetro, o que não influencia muito no tempo, pois acaba sendo uma avaliação constante para qualquer vetor dado, mas é importante destacar isso. E para concluir, em relação ao valor da entropia de Shannon calculada, foi possível notar que enquanto maior a string gerada, mais o valor aproxima-se de 1 pois a distribuição dos caracteres começa tornar-se mais uniforme.