Relatório do EP1 de MAC 0216

Nome: Odair Gonçalves de Oliveira
Número USP: 13671581

11 de setembro de 2023

Resumo

Este relatório refere-se ao Exercício Programa 1 (EP1) 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 EP1 envolveu a implementação de programas em Python e Assembly com o objetivo de promover a aprendizagem por meio da comparação entre essas implementações. Este documento tem como objetivo descrever a abordagem utilizada, os desafios enfrentados e as lições aprendidas durante o desenvolvimento do projeto.

Conteúdo

1 Introdução e Objetivos

O EP1 consistiu na implementação de um mesmo programa em duas linguagens diferentes para realizar uma comparação principalmente em termos de tempo de execução e facilidade de escrita. As linguagens utilizadas foram Python e Assembly para uma arquitetura X86_64.

Em relação ao programa implementado, a ideia era explorar o conceito e a utilidade das funções de hash, que mapeiam entradas de tamanho variável para valores de tamanho fixo, com aplicações em áreas como criptografia e indexação de tabelas hash. No caso, as entradas fornecidas eram strings que seriam mapeadas para um código de 32 caracteres usando os dígitos do sistema hexadecimal. Nesse contexto, o exercício consistia em desenvolver um código em Assembly e Python que realizasse essa mesma tarefa, ou seja, gerar um código a partir de um texto de entrada, com o propósito de realizar uma análise comparativa de desempenho entre os programas.

2 Detalhes da Implementação

Sobre a implementação realizada: Inicialmente, comecei a desenvolver o programa em Python, seguindo as instruções fornecidas no enunciado. Para garantir uma melhor organização do código, optei por dividir cada etapa dos 4 passos em diferentes funções, que corresponderiam às 4 sub-rotinas distintas do programa em Assembly. Meu objetivo era evitar o uso de funções predefinidas em Python, como a função hex(), que converte um número para seu valor hexadecimal, a fim de criar algo semelhante no código Assembly. Isso se deve ao fato de que o Assembly não possui funções prontas para esse propósito. Portanto, desenvolvi meu próprio conversor de decimal para hexadecimal em Python para poder replicar essa estratégia no código em Assembly.

A parte mais desafiadora da implementação em Assembly foi depurar o código usando o GDB. Isso ocorreu porque, ao contrário do Python, onde é possível imprimir o resultado após cada passo para verificar o sucesso ou fracasso, em Assembly, imprimir resultados é um processo um pouco mais complexo, tornando essa estratégia de depuração mais trabalhosa. No entanto, após compreender como verificar os valores das variáveis e registradores, a implementação se tornou mais fluída.

No código Assembly, dividi os 4 passos em sub-rotinas e também criei outras sub-rotinas para tarefas como impressão e leitura, com o objetivo de manter o código organizado como sugeriu o professor. Encontrei alguns desafios relacionados ao gerenciamento dos tamanhos das variáveis e a suas relações com os registradores, bem como problemas de falhas de segmentação por tal fato. No entanto, estou confiante de que consegui corrigir todos esses problemas antes da entrega.

Em relação ao comentários presentes no código: No Python, evitei colocar muitos comentários já que o código ficou bem semelhante ao pseudocódigo do enunciado, o que deixou bem evidente o que estava sendo feito em cada parte. Em Assembly, como não é sempre óbvio o que está acontecendo em cada seção, decidi investir mais nos comentários, explicando previamente o que seria feito e o que cada registrador estava guardando. No fim, fique satisfeito com o resultado e ao revisitar o código, não fiquei tão perdido graças aos comentários.

Em relação ao tamanho do código, o código em Python ficou bem mais conciso do que o código em Assembly. Abaixo é possível comparar o tamanho de cada um dos arquivos em que os programas foram escritos:

Tamanho Arquivo Python	Tamanho Arquivo Assembly
4.7 kB	11.0 kB

O arquivo em Assembly é quase o triplo do arquivo em Python, o que faz sentido considerando que uma simples operação como uma impressão na tela em Python custa uma linha e em Assembly pelo menos 5 linhas, o que exemplifica bem essa relação de tamanho entre ambos. Além disso, o arquivo em Assembly está repleto de comentários que tornam o programa ainda mais extenso para garantia de sua legibilidade, o que não foi muito necessário em Python.

3 Testes e Resultados

Os testes foram feitos utilizando o '<' para usar como entrada o arquivo contendo os textos que seriam utilizados. Ao usar o comando '*cat*' obtive resultados diferentes para alguns testes com o programa em Assembly, provavelmente algo relacionado ao 'enter' extra que estava sendo lido quando isso ocorria. Por tal motivo, continuei usando o '<' para os testes feitos.

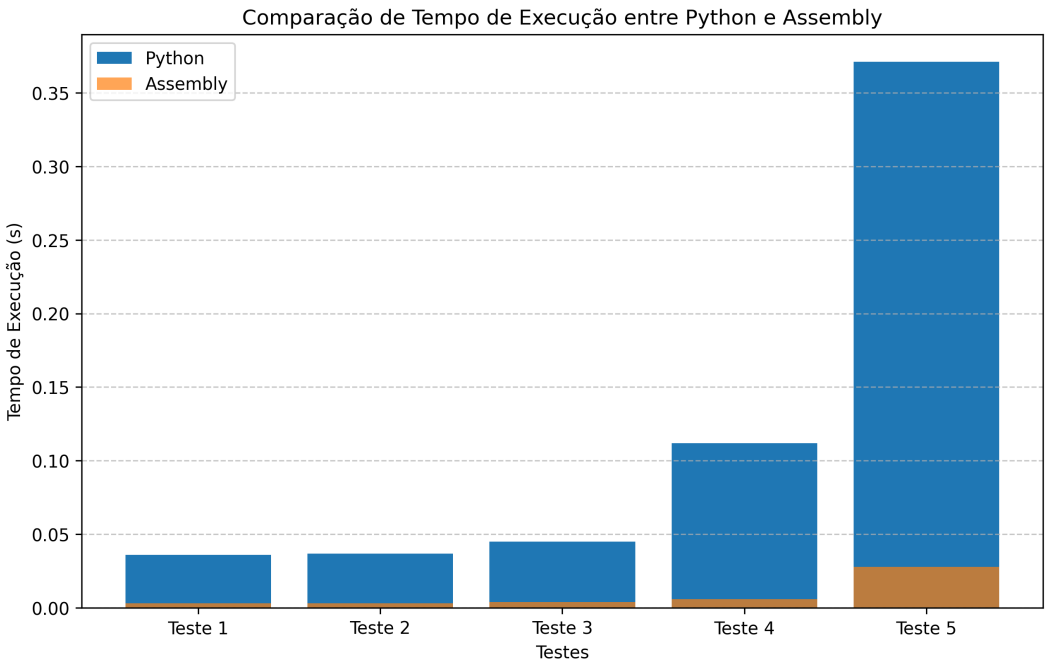
As configurações do computador em que os testes foram rodados estão expostas a seguir:

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

Para a análise comparativa foram realizados testes utilizando 5 arquivos de tamanhos diferentes passados como entrada para os programas. Cada um desses arquivos foram testados 10 vezes em cada programa, ou seja, o programa em Python foi testado 50 vezes e o programa em Assembly 50 vezes também. Em cada um dos testes, foi registrado, fazendo uso do comando **time**, o tempo de execução do programa e ao final do testes, foi calculada a média aritmética para avaliar o desempenho médio de tempo dos programas com cada arquivo. Ao rodar o comando **time** na execução do programa a saída é dividida em 3 partes: *user*, *sys* e *real*. O tempo registrado na tabela seguir foi o *real*. Confira os resultados na tabela a seguir:

Entrada	Média (Python)	Média (Assembly)
texto10.txt	0.036s	0.003s
texto100.txt	0.037s	0.003s
texto1000.txt	0.045s	0.004s
texto10000.txt	0.112s	0.006s
texto100000.txt	0.371s	0.028s

Para uma melhor visualização dos resultados, veja o gráfico a seguir:



4 Análise dos Resultados

Como podemos analisar a partir dos resultados expostos anteriormente, o programa em Assembly teve um desempenho extremamente melhor que o programa em Python em todos os testes, sendo aproximadamente 90% mais eficiente nos testes. Tal resultado era esperado, considerando que a função de utilizar uma linguagem de montagem é essencialmente essa, ter uma eficiência melhor no desempenho já que é uma linguagem de baixissimo nível que permite interagir diretamente com os registradores, a memória mais rápida disponível.

Em mais detalhes, para testes pequenos, até de 10 até 1000 caracteres, não é possivel observar diferenças significativas, tanto no programa em Python como no programa em Assembly.. A partir do teste de 10.000 palavras já é possível ver uma variação maior em ambos, principalmente no programa em Python, que chega a quase triplicar o tempo de execução. No Assembly, apesar de ainda manter um tempo pequeno, em comparação com os testes anteriores, observamos o tempo praticamente duplicar.

Para o último teste, com 100.000 caracteres, o tempo de execução deu um salto bem maior em ambos programas, porém, ainda sim, o programa em Assembly rodando o teste com 100.000 caracteres foi melhor que o programa em Python para apenas 10 caracteres. A razão por trás desses resultados estão em algumas características que diferenciam o Python do Assembly, sendo algumas delas:

1. Abstração da Linguagem - Isso refere-se ao fato de que Pyhton é considerada uma linguagem de alto nível, enquanto Assembly se aproxima diretamente da linguagem de máquina do computador, o que permite que haja um controle mais preciso das operações lógicas, aritméticas e que envolvem a memória, o que torna o processo mais eficiente.

2. Compilação - O código em Assembly pode ser compilado diretamente em código para a máquina, em instruções, enquanto o Python, por ser de mais alto nível, necessita de uma 'camada extra' nessa compilação, responsável por interpretar o que está sendo realizado antes de isso ser traduzido para o código da máquina, o que gasta mais tempo.

Por outro lado, o Assembly não apresenta a mesma portabilidade do Python, que pode ser rodado em mais dispositivos, com o interpretador adequado, o que não ocorre em Assembly por se comunicar quase que diretamente com a arquitetura do computador que está sendo utilizado. Além disso, toda a questão da extensão do código, da dificuldade de escrita e leitura também são pontos importantes ao se pensar na elaboração de um projeto. Ambas linguagens tem vantagens e desvantagens: Assembly apresenta um desempenho muito melhor no geral, teoricamente é o esperado, e isso foi comprovado empiricamente nos testes, e Python permite escrever um código muito mais légivel, com portabilidade e conciso, porém com menos desempenho.

5 Conclusão

Em conclusão, achei muito interessante ver como o desempenho do programa em Assembly foi extremamente melhor do que o desempenho do programa em Python, comprovando que era esperado. Pude também analisar contrastando a dificuldade da escrita em Assembly, considerando a difícil legilibidade do código, mesmo com os comentários, com a facilidade proporcionada pelo Python, com inúmeras funções já implementadas.

Apesar de Pyhton proporcionar uma escrita e leitura bem mais tranquila de se lidar, o desempenho acaba custando muito em comparação a implementação feita em Assembly. O Assembly é complexo de se escrever e ler, e acaba tornando o código bem mais extenso, porém, em compensação, o desempenho é significativamente melhor, ou seja, tudo acaba dependendo do projeto que está se desenvolvendo. Python, e outras linguagens de alto nível, pode ser adequado para projetos em que o desempenho pode ser um pouco sacrificado pela questão da portabilidade, da manutenção do código e da legibilidade (o que é importante, por exemplo, em um projeto em grupo). Já o Assembly tem como principal vantagem o desempenho, e se não for um código que precise de constantes revisões e que necessite de um comunicação mais direta com a arquitetura do computador, talvez essa linguagem seja a melhor opção.

Concluindo, foi enriquecedor poder visualizar esses resultados de forma empírica, o que permitiu absorver ainda mais os conhecimentos teóricos passados em aula. Gostei do EP1 e espero ter uma experiência tão boa de implementação como essa nos próximos.