UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO

ARQUITETURA DE COMPUTADORES

Relatório de Desenvolvimento do Projeto da 3ª Unidade

Simulador de Memória Virtual

Pedro Avelino Ferreira Nogueira - **20180062339**

Tiago Onofre Araújo- **20180144855**

Natal, Dezembro de 2019

Índice

1 Introdução

O estudo da arquitetura de computadores é essencial para que se possa utilizar os recursos computacionais de maneira otimizada e eficiente, utilizando as técnicas que permitem os melhores desempenhos [3]. A definição de arquitetura de computadores comumente é relacionada como o conjunto de atributos de um computador que um programador deve compreender para que possa escrever instruções lógicas que resolvam algum tipo de problema pela máquina, ou seja, compreender em detalhes o que o programa irá fazer no momento da execução. De maneira mais técnica, a Arquitetura de Computadores se preocupa com os projetos conceituais e fundamentais da estrutura operacional de um sistema computacional.

Interpretando o conceito de memória em Arquitetura de computadores, compreendemos que memória é um componente capaz de recuperar ou armazenar um conjunto de bits, que chamamos de dados. Pode-se afirmar que as memórias são os componentes com uma das maiores diversidades de implementação, visto que há várias características que procuramos para atender determinadas tarefas, como velocidade, capacidade, modelo, custo, etc. Um tipo de memória existente é a RAM (Random Access Memory), ela é comumente aplicada em computadores no geral, para armazenar as informações dos programas que estão sendo executados no computador. Porém, o espaço de memória da RAM é limitado e quando todo esse espaço é consumido, o processo em andamento passa a utilizar a memória virtual.

A memória virtual representa um espaço que é reservado na instalação do sistema operacional no computador e pode ser considerada como uma reserva da memória RAM, já que o sistema operacional utiliza memória virtual para complementar a memória física. Ela é fundamental porque, é responsável por executar os programas quando a memória RAM está sem espaço. Dessa forma, é estendido a quantidade de memória disponível para os dados temporários, que são utilizados pelos programas em execução. Quando a memória RAM é liberada, o gerenciador de memória move os arquivos da memória virtual para a RAM, que são conhecidos como arquivos de páginas. Assim, com essa realocação vários programas podem ser executados sem ter necessidade de aumentar a memória física.

A principal tecnologia utilizada no desenvolvimento da aplicação foi a linguagem de programação Python, uma linguagem de alto nível, que permite a construção de aplicações com fácil leitura do código e exige poucas linhas de código se comparado ao mesmo programa em outras linguagens. Neste documento, descreveremos a forma de organização e execução do código, detalhes sobre a implementação, análise dos resultados e conclusão acerca dos resultados.

Diante disso, o objetivo deste relatório é relatar o processo de desenvolvimento de um simulador de memória virtual que realiza acessos à memória, tanto leitura quanto escrita, com base no endereço virtual, tabela de páginas e acesso à endereço físico.

2 Desenvolvimento da solução

2.1 Projeto Python

A implementação do domínio da aplicação foi feita seguindo os padrões de orientação a objetos oferecidos pela linguagem de programação Python. A utilização de bibliotecas e o estilo de programação voltado para a legibilidade do código permitiu um desenvolvimento com mais agilidade. Os requisitos disponibilizados pela linguagem foram a fácil leitura de código e a redução do número de linhas de código, isso porque Python combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros.

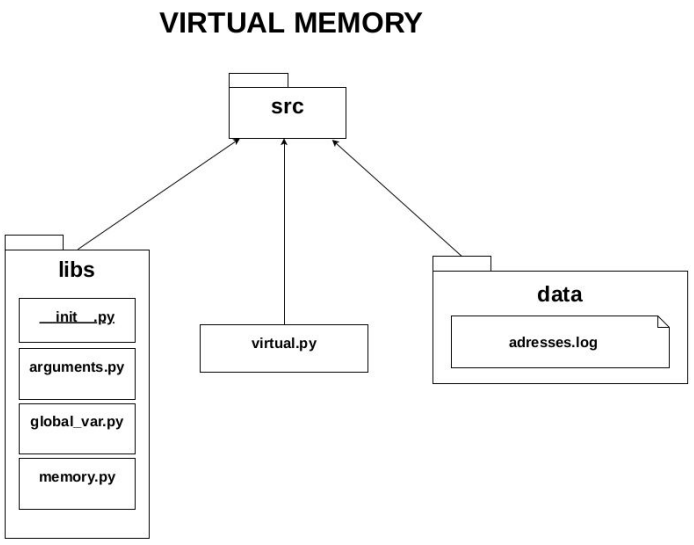
Devido a tipagem de Python ser forte, a construção do projeto teve os seus valores e objetos bem definidos, pois estes possuem tipos estabelecidos e não sofrem coerções como em C ou Perl. A grande gama de tipos de dados nativos disponibilizada pela linguagem permitiu a correta e ágil manipulação dos dados de entrada exigidos pela aplicação. Atrelado aos tipos primitivos e com o objetivo de permitir a definição de novos tipo de dados, existem as classes, que também foram utilizadas [1].

Um recurso de bastante importância oferecido pelo Python é a indentação. Essa foi desenvolvida para ser uma linguagem de fácil leitura, com um visual agradável, frequentemente usando palavras e não pontuações como em outras linguagens, sendo assim a indentação é obrigatória. Para atingir esse requisito a linguagem separa os blocos de código, usando espaços em branco e indentação ao invés de delimitadores visuais como chaves (C, Java) ou palavras (BASIC, Fortran, Pascal). O projeto foi implementado utilizando a biblioteca padrão, geralmente citada como um dos maiores trunfos da linguagem, fornecendo ferramentas para diversas tarefas. Sendo assim, existe uma grande variedade de ferramentas fornecida pela biblioteca padrão, tornando Python poderosa para conectar componentes diversos de software [2].

Na implementação da aplicação utilizamos o paradigma de Python de Orientação a Objetos, visto que com ele temos um melhor controle e estabilidade do nosso projeto, mais especificamente, utilizamos conceitos de classe, objetos, métodos, acessores, modificadores, construtores, entre outros.

3 Descrição de organização

Como foi apresentado, utilizamos a linguagem de programação Python com paradigma de orientação a objetos, por isso, nosso projeto está estruturado com classes, módulos, objetos, construtores, métodos, dentre outros. A seguir, na figura 1, o diagrama de pacote, representa a estrutura do projeto.



Fonte: autores (2019)

Agora detalharemos sobre cada pasta e subpasta do projeto:

data - pasta que guarda os arquivo(s) com os endereços de memória.

virtual.py - classe principal do projeto que instancia as variáveis principais (file_path page_size, memory_size, PRA) e o objeto `Memory` que faz a leitura do arquivo e realiza a simulação.

libs - a pasta que armazena as classes e as funções principais.

- **__init__.py -** identificador para verificar que a pasta é um módulo e irá importar tudo que está na pasta para o arquivo principal.
- **arguments.py -** se encontra a função para processar os argumentos da linha de comando.
- **global_var.py -** se encontra as constantes globais para verificação de qual algoritmo de substituição foi escolhido.
- **memory.py -** arquivo que contém a `class Memory` com os métodos principais da memória virtual:
  - `_search_in_virtual_ :` realiza a busca do endereço na paginação e retorna se encontrou ou não o endereço.
  - `FIFO:` realiza a política de substituição do endereço com base na estrutura de dados fila (first-in first-out) .

- `LRU:` realiza a política de substituição do endereço com base na última página referenciada.
- `RANDOM:` realiza a política de substituição do endereço com base na biblioteca randômica de Python.
- `simulate:` seleciona qual política de substituição será utilizada.
- `report:` responsável por mostrar no log do terminal um pequeno relatório, contendo:
    - a configuração utilizada
    - o número total de acessos à memória contidos no arquivo;
    - o número de page faults;

4 Descrição da execução do código

Como o Python possui um interpretador, não é necessaŕio compilar.

Para executar esse projeto é necessário a versão do python3.7.3.

No terminal deverá ser digitado os seguintes argumentos exigidos:

python3 virtual.py <algoritmo de substituição> .data/<arquivo.txt> <tamanho da página> <tamanho da memória>

<algoritmo de substituição> - “fifo”, “lru” ou “random”

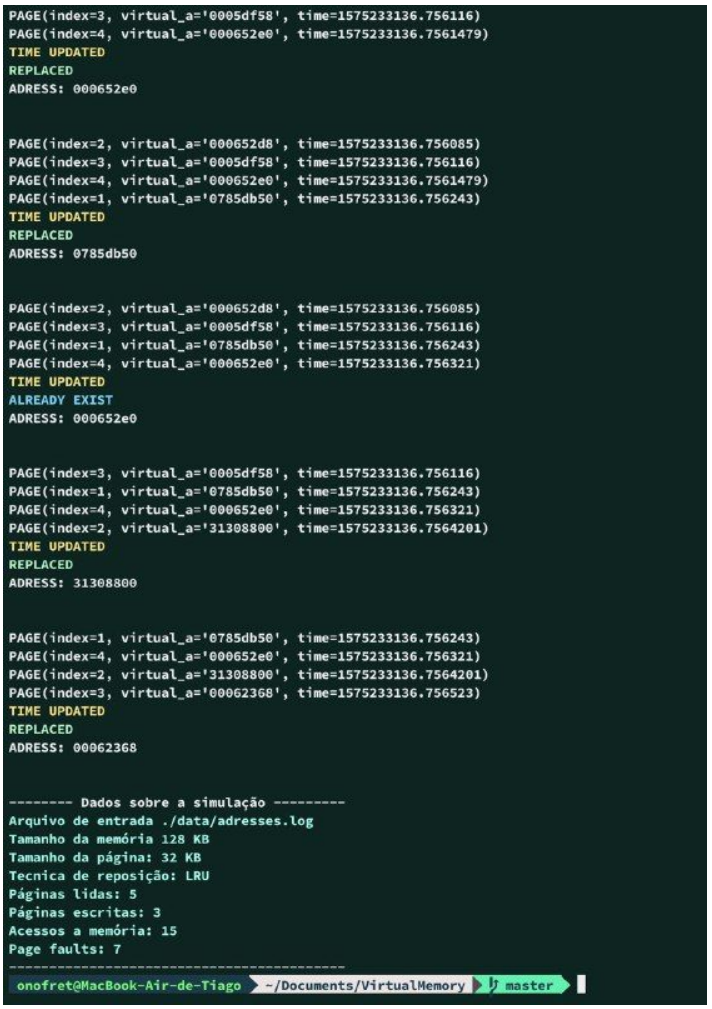
<tamanho da página> - valor entre 2 e 64.

<tamanho da memória> - valor entre 128 e 16.384

obs: a formatação do **<arquivo.txt>** deverá ser da seguinte maneira:

<endereço de memória> <operação de leitura ou escrita: “W” ou “R”>

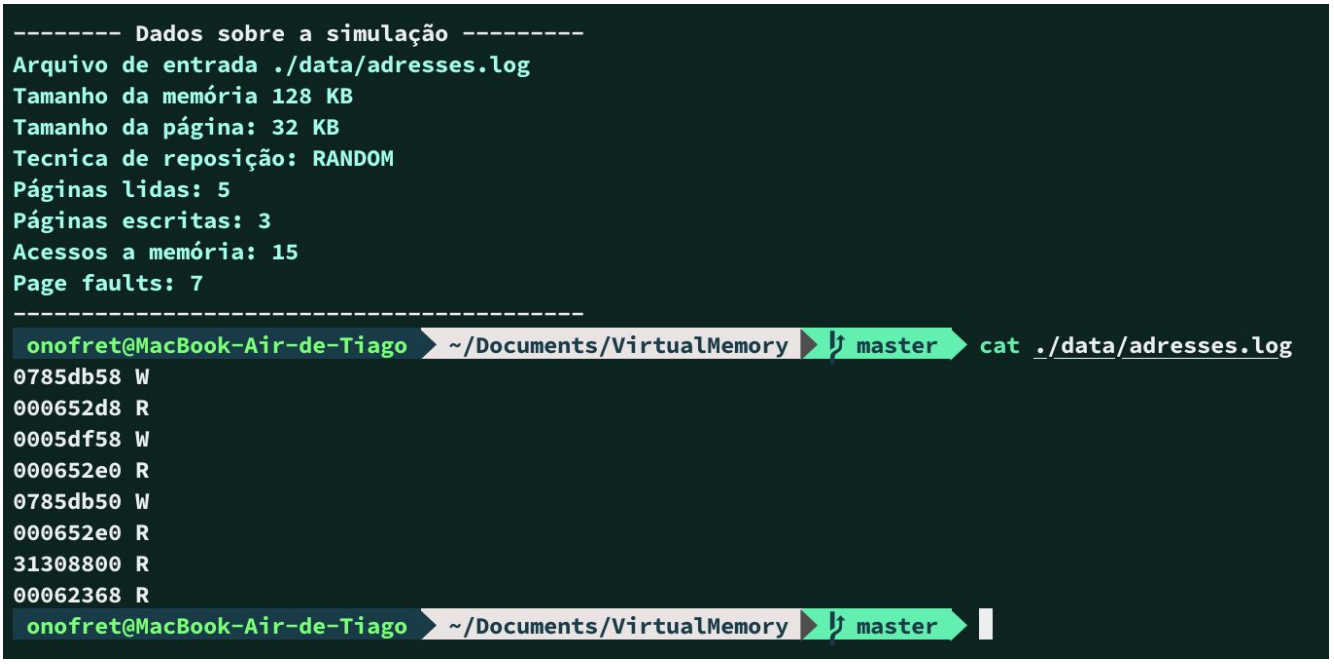
Após todos os argumentos serem inseridos na ordem certa, segue imagem da saída esperada:



Fonte: autores (2019)

5 Análise de resultados

Os resultados obtidos foi o relatório gerado pelo simulador, mostrado a seguir no terminal.



Ao final da execução, o programa gera um pequeno relatório, contendo:

- A configuração utilizada (definida pelos quatro parâmetros),
- O número total de acessos à memória contidos no arquivo
- O número de page faults.

De acordo com o arquivo de entrada, mostrado na imagem, podemos analisar que o programa está funcionando corretamente, visto que a saída está de acordo com a saída esperada. Vejamos:

As quatro primeiras linhas mostram os 4 parâmetros definidos, <arquivo de entrada> <tamanho da memória> <tamanho da página> <técnica de reposição>.

A linha seguinte, mostra as páginas lidas, ou seja, as páginas que tem como operação o “R”, que neste exemplo possui 5 . O mesmo vale para as páginas escritas, que contém o “W” e neste exemplo possui 3.

Criamos uma variável para verificar e contar os acessos que foram feitos a memória, essa variável é incrementada a cada busca e substituição na memória principal. Esse exemplo inclui 8 endereços, deles 6 distintos e 2 que se repetem, gerando um total de 7 page faults. Mostrado na última linha.

6 Conclusão

Durante a execução do projeto como um todo, surgiram alguns desafios, como a implementação de estrutura de dados em python e a utilização da biblioteca time. Contudo, tais desafios foram cumpridos com muita persistência e pesquisa, servindo como experiência para futuros projetos.

Os resultados obtidos com a aplicação supriram os requisitos da descrição do projeto, visto que o simulador de memória virtual comportou-se como previsto. Todos os métodos de substituição foram implementados e estão funcionando de acordo com os testes de saída esperada. Com isso, pode-se concluir que o simulador de memória virtual tratado aqui, opera como uma memória virtual.

7 Referências Bibliográficas

[1] Python.org. 2019.

[2] Przemyslaw Piotrowski. Build a Rapid Web Development Environment for Python Server Pages and Oracle. Oracle. Consultado em 01 de dezembro de 2019

[3] Gerrit A. Blaauw & Frederick P. Brooks (1997). Computer Architecture: Concepts and Evolution. [S.l.]: Addison-Wesley. ISBN 0-201-10557-8.

CAETANO, Prof. Daniel. Arquitetura e Organização de Computadores: Memórias. Disponível em: <http://www.caetano.eng.br/aulas/2012b/files/aoc_ap06.pdf>. Acesso em: 25 nov. 2019.

MILLER, Brad; RANUM, David. **Programação orientada à objetos.** Disponível em: <https://panda.ime.usp.br/pensepy/static/pensepy/13-Classes/classesintro.html>. Acesso em: 28 nov. 2019.

BONTEMPO, Nicolas. **Programando Python Orientado a Objetos.** Disponível em: <https://medium.com/@nicolasbontempo/programando-python-orientado-a-objetos-d0069b2f1eb5>. Acesso em: 26 nov. 2019.

geeksforgeeks.org. 2019.

stackoverflow.com. 2019.