Simulação de Gerência de Memória

Alexandre Araujo[1], **Gabriel Kurtz**[2]

[1]Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

alexandre.henrique@acad.pucrs.br, gabriel.kurtz@acad.pucrs.br

Abstract. *This article proposes a software solution to simulate a memory manager of an operating system running with processes through commands inputted by both randomly and a pre-written file. The article will cover all the memory manager features as well as how it works and what it should looks like.*

Resumo. *Este artigo propõe uma solução de software para simular um gerenciador de memória de um sistema operacional funcionando com processos através de comandos lançados tanto por um arquivo de texto pré definido quanto por um algorítimo aleatório de criação e término de processos. Este artigo cobrirá todas as funcionalidades implementadas no gerenciador de memória assim como seu funcionamento e como deveria se comportar em um cenário real.*

1. Introdução

O gerenciador de memória é uma parte fundamental na organização dos processos para que estes sejam executados da maneira mais performática possível, utilizando algoritimos que contribuem para identificação de quais informações dos processos deve estar em um local de rápido acesso. Para criação do software, foi utilizada a linguagem Python, que, por sua fácil utilização, baixa fidelidade a tipos, bibliotecas nativas e recursos poderosos, contribuiu para um desenvolvimento rápido e de fácil leitura.

2. Comportamento do Gerenciador de Memória

2.1. Leitura de instruções

As instruções pré-estabelecidas definem o comportamento simulado do sistema operacional interagindo com gerenciador de memória. Através delas é possível informar a aplicação quais os parâmetros que descreverão os componentes do sistema, tendo como alterativa as seguintes opções: algoritimo de entrada de dados, algoritimo de troca dos processos, quantidade e tamanho das páginas de memória e quantidade de páginas. O simulador entende a seguinte estrutura de dados de entrada, onde os valores são separados apenas por espaços e quebras de linha:

```
0
0
8
64
16
C  p1  16
C  p2  18
C  p3  10
A  p1  14
A  p1  20
A  p2  17
A  p3  10
M  p1  8
A  p1  20
```

Onde:

- Primeira linha: algoritimo de entrada de comandos.
- Segunda linha: algoritimo de troca de processos.
- Terceira linha: quantidade de páginas em memória.
- Quarta linha: quantidade de endereços físicos na memória.
- Quinta linha: quantidade de páginas em disco.
- Sexta linha em diante: comandos separados por:
  - Identificador do comando.
  - Nome do processo.
  - Parâmetro do comando.

3. Criação de processos

3.1. Comando de execução

A criação de novos processos a serem executados pelo sistema operacional é feita atráves do comando `C P T`, onde `C` é uma constante identificadora deste comando, `P` representa um parâmetro que informa o nome do processo e `T` representando um valor inteiro da quantidade que se deseja ser alocada.

3.2. Comportamento

A criação de processos na memória é dividida em 2 etapas: verificação de espaço em memória e população das páginas. No primeiro, ele percorre toda a memória com a intenção de assegurar de que haverá espaço disponível para os dados de entrada. Feito isso, o sistema segue para a persistencia destes dados em memória. Um algorítimo simples para identificar o endereço mais próximo em memória é executado. Caso não haja espaço suficiente em memória, o sistema executa um algoritimo de swap (que troca informações guardadas em memória pelas informações desejadas que estão em disco) e em seguida passa a popular novamente em memória.

4. Acesso à memória

4.1. Comando de execução

O acesso à memória alocada de um processo é feita atráves do comando `A P E`, onde `A` é uma constante de identidade deste comando, `P` representa um parâmetro string que informa o nome do processo e `E` representando um valor inteiro do endereço que se deseja acessar.

4.2. Comportamento

Inicialmente é feito uma verificação para validar se o endereço fornecido pertence ao intervalo de tamanho do processo. em seguida é feita uma varredura na memória para encontrar o endereço desejado em memória. Caso não obtenha sucesso, o gerenciador de memória executa o algoritmo de swap (o mesmo explicado anteriormente) para puxar para memória os dados em disco.

5. Alocação de memória

5.1. Comando de execução

A alocação de memória é feita atráves do comando `M P T`, onde `M` é uma constante que identifica este comando, `P` representa um parâmetro string que informa o nome do processo e `T` representando um valor inteiro do tamanho que se deseja realocar para o processo escolhido.

5.2. Comportamento

A alocação de memória funciona de maneira complementar ao da criação de novos processos. Logo após a verificação, é recuperado o endereço da page que ainda não tenha sido completamente preenchida e, caso exista, é populada com os dados.

6. Termino de processo

6.1. Comando de execução

O término do processo é feito atráves do comando `T P`, onde `T` é uma constante que identifica este comando e `P` representa um parâmetro string que informa o nome do processo a ser removido da memória.

6.2. Comportamento

A deleção de um processo da memória funciona de maneira simples e intuitiva.

7. Swap de memória

Quando ocorre Page Fault, ou seja, o programa precisa acessar setores de memória que estão em disco, ocorre um Swap de memória. Uma página sai da memória primária e passa para o disco. A seleção de qual página será retirada da memória pode ocorrer, no nosso programa, de duas maneiras: aleatória(qualquer uma das páginas pode ser selecionada aleatoriamente) ou Least Recently Used(a página que foi acessada para escrita ou leitura mais tempo atrás será selecionada).

O swap pode ocorrer tanto quando queremos acessar para leitura um dado que está em um setor que encontra-se em uma página no disco quanto em casos de escrita. Na escrita, pode ocorrer em dois casos: quando não há mais espaço na memória para escrever mas há espaço para alocar novas páginas no disco ou então quando queremos escrever em um processo que já existe e sua página mais recentemente escrita ainda tem espaço vazio e encontra-se em disco.

O processo é relativamente simples. Um algoritmo no nosso programa seleciona a página que vai sair da memória. No caso de tentativa de leitura de dados que estão em disco, a página selecionada sai da memória e entra na memória a página que contém os dados solicitados. No caso de precisarmos alocar uma página nova que está ociosa para escrita, trocamos a primeira página vazia do disco pela página selecionada a sair da memória. E no caso de necessitarmos preencher uma página já alocada de um processo, trocamos a página em questão pela página selecionada pelo algoritmo a sair da memória.

8. Erros de gerenciamento

8.1. Memória insuficiente

Ocorre quando há a requisição de criar um novo processo ou de alocar mais memória para um processo, porém somando o espaço alocável na memória e no disco não há espaço suficiente para a solicitação. Neste caso o sistema avisa que não há memória e não realiza o processo.

8.2. *Page Fault*

São os mesmos casos que originam o Swap de memória. Quando alguma requisição necessita ter na memória algum dado que está em disco para escrita ou para leitura, dizemos que ocorreu um Page Fault.

9. Conclusão

Acreditamos ter conseguido produzir um algoritmo bastante razoável e, dentro dos parâmetros estabelecidos, correto. Durante uma fase bastante avançada da produção, percebemos que teria sido melhor utilizar threads para simular o comportamento aleatório, assim teriamos uma boa oportunidade de colocar em prática os conhecimentos adiquirido durante as aulas da disciplina. No entanto, acreditamos que o trabalho proposto tenha contribuído para compreensão de grande parte dos conteúdos propóstos em aula.

Além disso, Nos exemplos diponibilizados para testes, nosso algoritmo foi capaz de reproduzir os resultados propostos de maneira rápida e efetiva para todos os cenários programados.