Simulador de Gerência de Memória

Leonardo Pellegrini Silva

Av. Independência, 2293 - Universitário, Santa Cruz do Sul - RS, 96816-501

leonardopellegrini@mx2.unisc.br

1. Descrição do Simulador

O *software* desenvolvido é capaz de simular a gerência de memória de duas maneiras distintas, pela minha interpretação dos requisitos do trabalho, sendo que a primeira engloba as primeiras sete opções do menu principal, funcionando somente manualmente, e a segunda está representada pela última opção, que realiza uma simulação totalmente automática, lendo os processos de um arquivo de escolha do usuário. Ambas as maneiras de execução funcionam em ambientes isolados, pois uma necessita de um controle temporal, enquanto a outra não implementa esse tipo de controle. Desenvolvido na linguagem de programação Python 3.x, somente é necessário uma instalação da mesma e da biblioteca tkinter.

2. Requisitos do Sistema

Embora o *software* tenha sido desenvolvido no sistema operacional *Linux*, tendo instalado o interpretador do Python 3 e a biblioteca tkinter, deve ser possível sua execução em qualquer SisOp que suporte os itens descritos anteriormente. Sobre o SisOp utilizado, suas configurações são as seguintes: OS Linux Peppermint 9 Nine x86_64 com *kernel* 4.15.0-51-generic; Python versão 3.6.8.

2.1. Instalação no Linux

A maioria das distribuições de linux derivadas do *Ubuntu*, como é o caso do *Peppermint*, já contém o Python 3 instalado, somente sendo necessário baixar e instalar a biblioteca tkinter com o seguinte comando:

```
sudo apt-get install python3-tk
```

Após a instalação da biblioteca, o simulador pode ser utilizando diretamente do terminal, utilizando o comando:

```
python3 main.py
```

3. Utilização do Simulador

O simulador conta com um menu de fácil uso e entendimento, suas opções serão descritas nessa seção, bem como o que é necessário ser informado em cada uma, caso seja uma inserção, ou seleção, e qual será sua saída. Um fator importante sobre a utilização: a memória inicial do simulador possui 64 bytes, podendo ser alterada facilmente no código fonte, esse valor está guardado na variável TAM, na linha de número 7.

O arquivo que será lido é selecionado por meio gráfico, utilizando a biblioteca tkinter, e o arquivo resultante contendo o *log* será guardado na mesma pasta em que o código fonte se encontra, ou, caso seja executado de outro diretório, será salvo nesse diretório, com o nome de *log.txt*. Foi incluído um arquivo de teste, *processos.txt*, que funciona nessa memória, gerando algumas situações interessantes, como a falta de espaço na memória, necessitando da realização de uma operação de compactação automática.

3.1. Opção – Encerrar o programa

Ao ser selecionada, encerrará a execução do programa.

3.2. Opção – Alterar algoritmo de alocação

Ao ser selecionada, exibirá um sub-menu para o usuário, onde deverá ser selecionado, entre as opções disponíveis, qual será o algoritmo utilizado na inserção de um processo, sendo o algoritmo selecionado exibido no menu principal.

3.3. Opção – Adicionar um processo para execução

Ao ser selecionada, será necessário informar o número de identificação do processo, e em seguida, o tamanho, em bytes. Caso já exista algum processo com o mesmo número de identificação, será informado para que o usuário insira outro valor.

3.4. Opção – Remover um processo da execução

Ao ser selecionada, é necessário que o usuário informe o número de identificação do processo que será removido, podendo ser removidos *n* processos em sequência, e, após que todos os processos desejados sejam removidos, deverá ser inserido o número zero para retornar ao menu anterior.

3.5. Opção – Exibir a lista de espaços livres

Ao ser selecionada, exibe a lista contendo os espaços livres da memória, são informados o começo de cada espaço e seu tamanho em bytes. A lista de espaços livres e atualizada após toda e qualquer operação que efetue mudanças na memória, como inserções, remoções, compactações e formatações (após a leitura do arquivo).

3.6. Opção – Exibir o registrador de realocação

Ao ser selecionada, exibe o registrador que guarda as realocações dos processos, contendo o número de identificação do processo, e sua posição antes e depois de ser realocado.

3.7. Opção – Exibir o mapa de utilização de memória

Ao ser selecionada, exibe um mapa da memória, contendo o número de idenficação de cada processo no(s) byte(s) correspondente(s), e, nos espaços livres, exibe o texto *None*.

3.8. Opção – Compactar a memória

Ao ser selecionada, efetuará a compactação da memória, efetivamente movendo todos os processos atualmente em execução para o início da memória, eliminando possíveis lacunas entre esses. Após essa operação, o registrador de realocação será atualizado.

3.9. Opção – Carregar processos de um arquivo e iniciar a simulação

Ao ser selecionada, será requisitado que o usuário escolha um arquivo contendo a lista de processos para a execução. Esse arquivo deverá ser formatado no mesmo modelo descrito nos requisitos desse trabalho, exemplo:

P1 (itentificação do processo)

5 (tamanho, em bytes, do processo)

0 (unidade de tempo de início do processo)

10 (duração, em unidades de tempo, do processo)

… (seguir o modelo para os processos seguintes, sendo aceitos *n* processos)

Após a seleção do arquivo, os três algoritmos de alocação, First-fit, Best-fit e Worst-fit, serão executados sobre a lista de processos lida, e suas respectivas execuções serão guardadas no arquivo de saída *log.txt*, assim como descrito no item 3.

4. Considerações finais

O desenvolvimento desse trabalho foi tranquilo, sem maiores problemas para o funcionamento dos métodos principais, somente por conta de pequenos *bugs* decorrentes de erros nas condições de alguns *if's* e *else's*, que resultaram em ocorrências cômicas, como um *loop infinito* no funcionamento da leitura do arquivo, pois o método de matar processos não comparava o tempo inicial com a duração ao tempo atual da execução.

Foi uma experiência ótima desenvolver na linguagem Python, pois a mesma permite grandes abstrações no codigo, além de facilitar o uso de estruturas de dados complexas, se comparado com linguagens de mais baixo nível.