T2-SISOP

Alunos: Adilson Medronha e Luís Lima

Instruções para execução

O arquivo de entrada

- O arquivo de entrada para o programa é um **json**, e nele é necessário colocar algumas informações como:
  - ***typee:*** tipo de alocação desejada
    - ***variable:*** para particionamento variável
    - ***fixed:*** para particionamento fixo
  - ***capacity:*** capacidade da pilha de memória
  - ***partition:*** número de posições para cada partição
  - ***alg:*** algoritmo utilizado
    - ***best-fit***
    - ***worst-fit***
    - Se o particionamento for **fixo**, apenas deixe esta configuração em branco e.g., `"alg": ""`
  - ***processes:*** lista contendo os processos a serem modelados. Cada processo contém:
    - ***event*:** tipo de evento do processo
      - ***IN***
      - ***OUT***
    - ***pid:*** ID do processo
    - ***psize:*** tamanho do processo
- Dentro da pasta `./inputs` existem alguns exemplos para teste pré-criados.

Informações da implementação

- O trabalho em geral explora **todas** as abordagens solicitadas no enunciado do trabalho. Entretanto, um dia antes da entrega do trabalho surgiu uma dúvida entre a dupla se deveria ser implementado a **paginação** quando um processo é maior do que uma partição (que não consta no enunciado do trabalho). Sendo assim, um e-mail foi enviado para o professor e este informou que o esperado era que fosse realizado a **paginação** dos processos neste caso, porém, visto que esta informação não constava no trabalho, o professor decidiu que irá aceitar a implementação que **não considera a paginação dos processos neste caso**. Portanto, este trabalho **não aborda a paginação dos processos**.
- O trabalho está separado em quatro classes (arquivos):
  - `alocation.py`
  - `fixed.py`
  - `variable.py`
  - `memory_manager.py`
- A classe `Alocation` é apenas uma **classe abstrata** que contém alguns métodos comuns entre as implementações (fixa e variável), visando **polimorfismo**.
- A classe `Fixed` modela as **partições fixas de mesmo tamanho**.

- A classe `Variable` modela as **partições com tamanho variável**.
- A classe `MemoryManager` modela a *MMU*, e é responsável por executar os processos e alocá-los para o devido sistema de particionamento.

A execução

- Para executar o programa basta digitar o seguinte comando:

```
python3 memory_manager.py <caminho_do_arquivo>
python3 memory_manager.py ./inputs/exemplo1_fixa.json
```

- **Durante a execução, o programa irá solicitar uma interação do usuário (quando aparecer o símbolo `>>`) para que prossiga para o próximo passo.**
- Quando não houver mais processos na lista de processos lidos do arquivo de entrada o programa é terminado.

A saída (resultado)

```
PROCESS ID: A
PROCESS SIZE: 3
PROCESS EVENT: IN
   +---------+
1  |         | <--- Searching...
2  |         |
3  |         |
4  |         |
5  |         |
6  |         |
7  |         |
8  |         |
9  |         |
10 |         |
11 |         |
12 |         |
13 |         |
14 |         |
15 |         |
16 |         |
   +---------+
>>
```

- Durante a execução é a saída informa qual processo está sendo executado, seu tamanho, e o evento (entrada ou saída).
- Além disso, uma pilha de memória é simulada (exemplo acima).
- Na pilha de memória, existe um "ponteiro" que diz onde o programa está **procurando**, **alocando**, **limpando** ou **adicionando uma fragmentação interna** em um espaço de memória.

- **Na partição fixa**, as **fragmentações internas** serão abordadas com a string `///////`.
- Quando um processo não cabe na memória (ou por não ter espaço suficiente ou **por ter um tamanho maior do que a partição, quando é particionamento fixado)** e o programa irá lançar a seguinte mensagem:
    - `INSUFFICIENT MEMORY SPACE!`