

README

Descrição dos arquivos:

- `processos.json`: este arquivo estrutura os dados necessários (tabela de processos) que serão manipulados para a simulação do algoritmo Round Robin. Para cada processo fornecido, há o `Id`, `name` (nome), `tempo de chegada` (`arrival`), `execution` (tempo de execução).
- `main.py`: consiste no algoritmo em si, definição das funções requeridas, e também da importação dos dados e bibliotecas necessários.

Descrição do código principal

O trecho inicial do código refere-se a importação das bibliotecas e configuração inicial dos arquivos

1) Na sequência há a função `load_preocesesses_from_json` onde o arquivo `Json` é lido, e então é extraído a lista de processos e é retornado uma lista de dicionários. A seguir uma breve descrição do funcionamento:

- a) `with open(file_path, 'r') as file::` Abre o arquivo `JSON` em modo de leitura (`'r'`).
- b) `json.load(file):` Converte o conteúdo do arquivo `JSON` em um dicionário `Python`.
- c) `.get('processes', [])`: Pega a lista que está dentro da chave `"processes"` no `JSON`. Se a chave não existir, retorna uma lista vazia `[]`.
- d) `for process in processes::` Itera sobre cada item (processo) da lista que veio do `JSON`.
- e) `processed_process = {...}`: Cria um *novo* dicionário para cada processo. Isso é feito para garantir que estamos trabalhando com dados limpos e que temos apenas as chaves que esperamos (`id`, `name`, `arrival`, `execution`).
- f) `return processed_data`: Retorna a lista de processos formatada.

2) A função `round_Robin_sschedling(process, ...)` se trata da função que executa o algoritmo Round Robin em si. A seguir está uma breve descrição da função:

- a) **Parâmetros:** Aceita a lista de processos, o `quantum`, a `velocidade de simulação`, e os `prints logs`.
- b) **Inicialização:** inicialmente temos a criação do “relógio” global da simulação, e na sequência cria uma cópia da lista de processos que é armazenada em `queue`
- c) **Loop principal:** utilizando o “while `queue`” o algoritmo vai permanecer no loop enquanto houver processos.

- d) `process = queue.pop(0)`: equivale ao "First-In, First-Out" (FIFO). Ele pega o **primeiro** processo da fila para executá-lo.
- e) `if 'start' not in process`:: Se for a primeira vez que este processo está sendo executado, ele grava o tempo atual na chave 'start' que depois será utilizado para o tempo de resposta
- f) Funcionamento do Quantum:
 - i) `if process['execution'] > quantum`:
 - ii) `time += quantum`: O relógio avança o valor total do quantum.
 - iii) `sleep(...)`: Pausa a simulação para "gastar" esse tempo.
 - iv) `process['execution'] -= quantum`: O tempo de execução restante do processo é diminuído.
 - v) `queue.append(process)`: **Isso é o "Round Robin"**. O processo é **preemptado** (interrompido) e colocado de volta no **FIM** da fila.
 - vi) `else`: (Processo vai ser finalizado)
 - vii) `time += process['execution']`: O relógio avança apenas o tempo que faltava.
 - viii) `sleep(...)`: Pausa a simulação por esse tempo restante.
 - ix) `process['execution'] = 0`: O processo é marcado como finalizado.
 - x) `process['completion_time'] = time`: Grava o tempo de finalização.
 - xi) `completed_processes.append(process)`: O processo é movido da fila de prontos para a lista de concluídos.

Por fim, temos a `return completed_processes`: quando o loop while termina (fila vazia), a função retorna a lista de processos concluídos, agora com os dados de 'start' e 'completion_time'.