Simulador de Processos

Autores: Denis do Nascimento Rodrigues,
Ruan Pedro de Araujo Anjos e
Fabrício Fontenele Vieira

Este projeto é um simulador de algoritmos de escalonamento de processos, desenvolvido em Python. Ele permite criar processos com tempos de execução definidos e simular dois algoritmos clássicos de escalonamento: **FIFO** (**First-In-First-Out**) e **SJF** (**Shortest Job First**).

Funcionalidades
Adição e remoção de processos com tempos de execução customizados.
Simulação dos algoritmos FIFO e SJF, exibindo ordem de execução, tempo de espera, turnaround e médias.
Interface interativa via linha de comando para:
Adicionar novos processos.
Remover processos existentes.
Mostrar informações detalhadas dos processos.
Escolher executar os algoritmos FIFO, SJF ou ambos.

Estrutura dos Principais Arquivos
main.py: Ponto de entrada do simulador. Cria processos de exemplo e chama a aplicação principal.
process.py: Classe responsável por gerenciar a lista de processos (adicionar, remover, listar, exibir informações).
execução.py: Função principal de interação, oferecendo ao usuário a escolha do algoritmo de escalonamento.ddsa

Exemplo de Uso
Ao rodar o programa, você poderá:
Adicionar múltiplos processos com nomes e tempos de execução.
Escolher entre FIFO, SJF ou ambos para simulação.
Visualizar tabelas detalhadas com ordem de execução, tempo de espera e turnaround para cada algoritmo.
A qualquer momento, você pode voltar ao menu principal para adicionar/remover processos ou executar os algoritmos novamente.

Por exemplo:
Você pode adicionar os processos A (tempo 5), B (tempo 2) e C (tempo 8).
Depois, escolher executar o algoritmo FIFO para ver os resultados.
Em seguida, voltar ao menu e executar o algoritmo SJF para comparar os resultados.
Finalmente, você pode usar a opção de "Mostrar Informações dos Processos" para revisar os processos que foram adicionados.

Código

```
# Autores: Denis do Nascimento Rodrigues,
# Ruan Pedro de Araujo Anjos e
# Fabrício Fontenele Vieira


class Process:
    def __init__(self):
        self.listProcesses = {}

    def addProcess(self, key, time):
        self.listProcesses[key] = time

    def removeProcess(self, key):
        if key in self.listProcesses:
            del self.listProcesses[key]
            return True
        return False

    def getKey(self):
        return list(self.listProcesses.keys())

    def getProcess(self):
        return list(self.listProcesses)

    def showInfo(self):
        for key, value in self.listProcesses.items():
            print(f'Processo: {key} / Tempo de processo: {value}')
from process import Process



class Fifo:
    def __init__(self, process: Process):
        self.process = process

    def executar(self):
        if not self.process.getKey():
            print("Não há processos. Adicione processos primeiro.")
            return
        print("-"* 50)
        print("\nResultado da simulação (FIFO):")
```

```
        print("-"* 50)
        print("Processo | Ordem | Execução | Espera | Turnaround")
        print("-"* 50)

        total_waiting_time = 0
        temp_turnaround_total = 0
        waiting_time = 0
        order = 1

        for key in self.process.getKey():
            temp_exec = self.process.listProcesses[key]
            turnaround = waiting_time + temp_exec

            print(f"{key:^8} | {order:^5} | {temp_exec:^8} |
{waiting_time:^6} | {turnaround:^10}")
            total_waiting_time += waiting_time
            temp_turnaround_total += turnaround

            waiting_time += temp_exec
            order += 1

        n = len(self.process.getKey())
        print("-"* 50)
        print(f"Média do tempo de espera: {total_waiting_time /
n:.2f}")
        print(f"Média do tempo de turnaround: {temp_turnaround_total /
n:.2f}")

from process import Process

class SJF:
    def __init__(self, processos: Process):
        self.processos = processos

    def executar(self):
        if not self.processos.listProcesses:
            print("\nNão há processos. Adicione processos primeiro.")
            return
        print("-"* 50)
        print("\nResultado da simulação (SJF):")
        print("-"* 50)
        print("Processo | Ordem | Execução | Espera | Turnaround")
        print("-"* 50)
```

```
        ordered_processes =
sorted(self.processos.listProcesses.items(), key=lambda item: item[1])

        total_waiting_time = 0
        total_turnaround_time = 0
        waiting_time = 0
        order = 1

        for key, temp_exec in ordered_processes:
            turnaround = waiting_time + temp_exec

            print(f"{key:^8} | {order:^5} | {temp_exec:^8} |
{waiting_time:^6} | {turnaround:^10}")
            total_waiting_time += waiting_time
            total_turnaround_time += turnaround

            waiting_time += temp_exec
            order += 1

        n = len(ordered_processes)
        print("-"* 50)
        print(f"Média do tempo de espera: {total_waiting_time /
n:.2f}")
        print(f"Média do tempo de turnaround: {total_turnaround_time /
n:.2f}")


from fifo import Fifo
from sjf import SJF


def application(process):

    while True:
        print("="*50)
        print(" "* 16,"Escolha a ação")
        print("1 - Executar Algoritmos\n2 - Adicionar Processo")
        print("3 - Remover Processo\n4 - Mostrar Informações dos
Processos\n5 - Sair")
        print("="*50)

        action = input("Digite sua opção: ")
```

```
        if action == "1":
            run_algorithms(process)

        elif action == "2":
            while True:
                add_process(process)
                continueQuestion = input('Quer adicionar mais
processos? (S/N): ').upper()

                if continueQuestion == 'S' or continueQuestion ==
"SIM":
                    continue
                else:
                    print("-" * 50)
                    break

        elif action == "3":
            remove_process(process)
        elif action == "4":
            show_process_info(process)
        elif action == "5":
            print(" "*17,"FIM DE EXECUÇÃO")
            break
        else:
            print("-" * 50)
            print(" " * 17, "Valor Inválido")
            print("-" * 50)

        contin = input("\nVolta para o menu ? [S/N]: ").upper()

        if contin == "S":
            continue
        elif contin == "N":
            print(f" " * 15, "FIM DE EXECUÇÃO")
            break
        else:
            print("Opção inválida.\n")

def run_algorithms(process):
    if not process.getKey():
        print("\nNão há processos para executar. \nAdicione processos
primeiro.")
```

```
        return

    print("-"*50)
    print("Escolha o algoritmo de escalonamento:")
    print("1 - FIFO\n2 - SJF\n3 - Ambos")
    print("-"*50)

    option = input("Digite sua opção: ")

    if option == "1":
        fifo = Fifo(process)
        fifo.executar()
    elif option == "2":
        sjf = SJF(process)
        sjf.executar()
    elif option == "3":
        fifo = Fifo(process)
        fifo.executar()
        sjf = SJF(process)
        sjf.executar()
    else:
        print("-"*50)
        print(f" "*17,"Valor Invalido")
        print("-"*50)

def add_process(process):
    key = input("\nNome do processo a ser adicionado: ").upper()
    time = int(input("Tempo de execução do processo: "))
    process.addProcess(key, time)
    print(f"Processo ({key}): tempo {time} adicionado.")
    print("-"*50)

def remove_process(process):
    key = input("Digite a chave do processo a ser removido: ").upper()
    if process.removeProcess(key):
        print(f"Processo {key} removido.")
    else:
        print(f"Processo {key} não encontrado.")

def show_process_info(process):
    print("-"*50)
    process.showInfo()
```

```python
from process import Process
from execução import application

def main():

    processo = Process()
    application(processo)

if __name__ == "__main__":
    main()
```