# AppWeb: Simulador de escalonamento de processos

**Informações gerais.**

Centro Universitário: Uniruy Wyden

Aluno: Thiago Rodrigues dos Santos

Disciplina: Algoritmos e Complexidade 2025.2

Projeto: Simulador de escalonamento de processos

Repositório github: https://github.com/ThiagoR17/Simulador-Interativo-de-Algoritmos-de-Escalonamento-de-CPU

Link do Deploy:

## Descrição do projeto

**. Objetivo da aplicação:**

Principal objetivo foi criar uma aplicação voltada para o estudo acadêmico de Sistemas Operacionais. A proposta é transformar conceitos teóricos em uma experiência visual e dinâmica, facilitando a compreensão dos alunos por meio da simulação prática.

. Funcionalidades principais:

1. Simulação de Múltiplos Algoritmos

2. Visualização Dinâmica em Tempo Real

3. Geração e Gerenciamento de Processos

4. Análise Comparativa de Desempenho

. **Tecnologias utilizadas:**

.Frontend:HTML/CSS + Booststrap

.Backend: Python Flask

. Banco de dados: servidor Flask

. Deployment: AWS EC2

## Estrutura de dados utilizadas

| Rotina/Função | Est. Dados | Justificativa do Uso |
|---|---|---|
| Armazenamento da entrada | **Array / Lista** | Estrutura simples para receber e armazenar a lista inicial de processos vinda da interface. |
| Fila de Prontos (FCFS / Round Robin) | **Fila (Queue / deque)** | Garante a propriedade **FIFO (First-In, First-Out)**. Perfeita para a ordem de chegada (FCFS) e para re-enfileirar processos após o quantum (RR). Operações O(1). |
| Fila de Prontos (SJF Preemptivo) | **Fila de Prioridade (Heap)** | Essencial para o SJF. Permite encontrar e extrair o processo com o **menor tempo restante** em tempo logarítmico (***O*** (log ***N***)). |
| Fila de Prontos (Múltiplas Filas) | **Array de Filas (list[deque])** | Mapeia o conceito do algoritmo, onde cada fila no array representa um nível de prioridade (ex: Fila 0 = Alta, Fila 1 = Média). |
| Diagrama de Gantt | **Array / Lista** | Armazena a sequência cronológica de qual processo (ou "Idle") estava na CPU a cada unidade de tempo. |

## Algoritmos Implementados

- **Ordenação:**
- **Timsort** (algoritmo padrão do Python, list.sort()) usado em **todos** os escalonadores para pré-ordenar a lista de processos de entrada pelo **Tempo de Chegada**.
- **Complexidade (Timsort):**
    - Melhor caso: O(n)
    - Caso médio: O(n ***log n***)
    - Pior caso: O(n ***log n***)
- **Busca (Seleção de Processo):**
- **FCFS e Round Robin:** A "busca" pelo próximo processo é uma operação de remoção da fila (deque.popleft()).
    - **Complexidade: O (1)**, pois a fila garante a ordem correta.
- **Múltiplas Filas:** A busca consiste em verificar um pequeno número de filas (3) em ordem de prioridade.
    - **Complexidade: O(1)**, pois o número de filas é uma constante.
- **Outros Algoritmos e Técnicas Relevantes:**
- **Algoritmo Guloso (Greedy):**
    - Usado no **SJF (Shortest Job First) Preemptivo**. A cada passo, o algoritmo faz a escolha "gulosa" de executar o processo com o menor tempo de execução restante.
- **Fila de Prioridade (Min-Heap):**
    - É a estrutura de dados central do **SJF Preemptivo**. Ela permite que o algoritmo encontre e selecione o processo mais curto de forma eficiente.
    - Complexidade: Inserção (heappush) e Remoção (heappop) são **O(*log n*).**

## Recursidade e Equações de Recorrência

| Rotina Recursiva | Objetivo | Equação de Recorrência | Complexidade |
|---|---|---|---|
| **Timsort / Merge Sort** (Usado internamente pelo Python) | Pré-ordenar a lista de processos por tempo de chegada. | $T(n) = 2T(n/2) + O(n)$ | $O(n \log n)$ |

A única recursividade presente no projeto é a utilizada **implicitamente** pelo algoritmo de ordenação Timsort (o list.sort() padrão do Python), que é baseado no **Merge Sort**, um algoritmo de "Divisão e Conquista".

## Técnicas de Programação Aplicadas

**Algoritmos Gulosos (Greedy):**

Utilizado na implementação do **SJF (Shortest Job First) Preemptivo**. A cada unidade de tempo, o algoritmo toma a decisão localmente ótima (gulosa) de escalonar o processo que tiver o **menor tempo de execução restante** disponível na fila de prontos.

**Estruturas Avançadas (Heap):**

Uma **Min-Heap (Fila de Prioridade)** é a estrutura de dados central que permite o funcionamento eficiente do algoritmo **SJF Preemptivo**. Ela é usada para armazenar

fila de prontos, permitindo que a busca pelo processo mais curto seja realizada em tempo $O(\log n)$.

## Complexidade Assintótica

| Rotina/Função | Caso Melhor | Caso Médio | Caso Pior | Notação Θ/Ω |
|---|---|---|---|---|
| **Pré-ordenação** (Timsort) | O(n)$ | O(***n log n***) | O(***n logn***) | θ (***n log n***) |
| **Seleção de Processo** (FCFS, RR, M. Filas) | O (1) | O(1) | O(1) | θ (1) |
| **Seleção de Processo** (SJF Preemptivo) | O (1) | O(l***og n***) | O(l***og n***) | Ω (1)O(***log n***) |
| **Loop da Simulação** (FCFS, RR, M. Filas) | O(T) | O(T) | O(T) | θ (T) |
| **Loop da Simulação** (SJF Preemptivo) | O(T ***log n***) | O(**T *logn***) | O(T ***log n***) | θ (T ***log n***) |

## Observações Finais: Desafios/Melhorias

- O uso de uma **Min-Heap (Fila de Prioridade)** no **SJF Preemptivo** foi crucial. Ela permite encontrar o processo mais curto em O(log n) a cada passo, resultando em uma complexidade total de O(n log n + T log n). Sem ela, o custo seria O(n log n + T*n).

- O **Algoritmo Guloso (Greedy)** é a base do **SJF**, que sempre escolhe a tarefa mais curta disponível, otimizando o tempo médio de espera.

- A **pré-ordenação** O(n log n) dos processos por tempo de chegada é um custo pago uma única vez que simplifica muito a lógica de simulação, permitindo que os algoritmos FCFS, RR e Múltiplas Filas rodem em tempo Theta(T).

- **Futuras melhorias:** A maior melhoria seria converter todos os algoritmos para um modelo **orientado a eventos** (como fizemos no fcfs_scheduler_optimized). Isso eliminaria a dependência de T (o "tick" do relógio) e faria a complexidade depender apenas de n (o número de eventos, ex: chegadas e términos). A complexidade total para FCFS e RR se tornaria O(n log n).