1. Justificativa de Design

A estrutura de dados escolhida foi baseada em listas encadeadas customizadas (ListaDeProcessos e Node). Esse design é eficiente para o escalonador porque permite inserções e remoções em tempo constante O(1), facilitando o gerenciamento dinâmico dos processos de acordo com suas prioridades. Além disso, o uso de diferentes listas para alta, média e baixa prioridade permite separar claramente os processos, reduzindo o tempo de busca e melhorando a organização do escalonamento.

2. Análise de Complexidade

Na implementação, as operações principais apresentam as seguintes complexidades: - Inserção em uma lista de processos: O(1), pois ocorre no final ou início da lista encadeada. - Remoção de processos: O(1), quando ocorre na cabeça da lista. - Busca de processos: O(n), já que pode ser necessário percorrer toda a lista. Assim, o desempenho geral do escalonador é adequado para o problema, com operações críticas de escalonamento (inserção/remoção) em tempo constante.

3. Análise da Anti-Inanição

A lógica de anti-inanição garante que processos de baixa prioridade não fiquem indefinidamente sem execução. Isso é feito por meio de uma política de promoção de prioridade: após certo tempo ou número de ciclos, processos de prioridades mais baixas podem ser promovidos para níveis mais altos. Sem essa regra, processos de baixa prioridade poderiam nunca ser executados caso houvesse constante chegada de processos de alta prioridade, gerando injustiça e ineficiência no sistema.

4. Análise do Bloqueio

Quando um processo necessita acessar o recurso 'DISCO', ele é retirado da lista de pronto e movido para uma lista de bloqueados. Seu ciclo de vida é o seguinte: 1. O processo entra inicialmente na lista correspondente à sua prioridade (alta, média ou baixa). 2. Ao solicitar o 'DISCO', ele é transferido para a lista de bloqueados. 3. Após o término da operação de E/S, o processo retorna à lista de prontos de acordo com sua prioridade. Esse mecanismo garante que o escalonador respeite o tempo de espera pelo recurso, evitando falhas de concorrência.

5. Ponto Fraco

O principal gargalo da implementação está nas operações de busca dentro das listas encadeadas, que apresentam custo O(n). Isso pode ser problemático em cenários com grande número de processos. Uma melhoria teórica seria substituir as listas encadeadas por uma estrutura mais eficiente, como filas de prioridade baseadas em heaps (PriorityQueue). Com isso, operações de inserção e remoção manteriam boa eficiência, enquanto a busca e seleção de processos se tornariam mais rápidas e escaláveis.