1) A estrutura de dados de lista duplamente encadeada é uma escolha eficiente para implementar este escalonador. Isso se deve pela performance de suas operações fundamentais, como adicionar um processo no final da fila(O(1)) e remover um processo do inicio da fila(O(1)). Como o escalonador lida com processos que entram e saem do sistema de forma constante, a eficiência nessas operações de inserir e remover é crucial para manter a performance. O fato de usar quatro listas separadas, sendo uma para cada nível de prioridade e uma para bloqueados, também é eficiente por eliminar a necessidade de procura de um processo por toda a estrutura.

2) A complexidade das principais operações é:
2.1 Adicionar um processo : O(1), pois envolve apenas manipulação de ponteiros.
2.2 Remover um processo : O(1), porque acessa diretamente o head.
2.3 Executar um ciclo de CPU : O(1), pois a lógica central apenas verifica se as listas não estão vazias e executa o Add e Remove, que são operações constantes.

3) A lógica de anti inanição garante a coerência na execução ao impedir que processos de alta prioridade sejam os únicos a serem executados, permitindo assim que os processos de média e baixa prioridade consigam participar do processo.

4) Primeiramente o processo é adi
cionado no fim da sua lista de prioridade original, ao decorrer dos ciclos, quando chega sua vez de ir para a fila de execução o Scheduler verifica se o processo precisa do recurso “DISCO”. Como ele precisa, o processo não é executado e é adicionado no final da lista bloqueados. No ciclo seguinte o processo mais antigo(removido pela cabeça) é desbloqueado e transferido para a lista de sua prioridade (adicionado ao final) enquanto aguarda sua vez de ir para a lista de execução onde vai ser analisado quando chegar a sua vez, mas nessa hora ele ira para a lista de execução.

5) O maior problema identificado no escalonador implementado é o impacto negativo sobre processos longos. Sempre que um processo não consegue ser concluído dentro de um único ciclo, ele é automaticamente reposicionado no final da fila. Com isso, precisa aguardar novamente a execução de todos os demais processos para receber uma nova oportunidade de processamento. Esse comportamento cria um ciclo repetitivo executar parcialmente, aguardar um longo período e retornar para o fim da fila que prolonga de maneira significativa o tempo total necessário para a conclusão de tarefas mais extensas.

Proposta de Melhoria Teórica

Uma alternativa teórica para atenuar esse problema é a adoção do modelo de Filas de Feedback com Múltiplos Níveis (MFQ – Multilevel Feedback Queue). Nesse esquema, o sistema dispõe de diversas filas, cada uma com prioridade distinta e tempos de execução diferenciados:

Filas de prioridade mais alta atendem processos curtos, que tendem a ser finalizados rapidamente, liberando recursos e mantendo o sistema ágil.

Filas de prioridade mais baixa recebem processos mais longos, realocados quando não conseguem ser concluídos nas filas de prioridade superior.

Para evitar que processos permaneçam indefinidamente nas filas inferiores, o modelo prevê um mecanismo de promoção: tarefas que aguardam por muito tempo podem ser movidas para filas de maior prioridade, assegurando sua execução.

Esse mecanismo equilibra eficiência e justiça. Ele favorece a conclusão rápida de processos curtos, o

que melhora o desempenho geral do sistema, mas ao mesmo tempo garante que processos longos não sejam negligenciados, assegurando que todos tenham a chance de serem concluídos em tempo hábil.