1) Justificativa de Design

Neste projeto foi uma lista encadeada simples criada do zero. Essa estrutura é eficiente para o escalonador porque permite a inserção no final em O(1) e remoção no início em O(1), que são as operações mais frequentes. Além disso, não impõe limite de tamanho fixo, adaptando-se ao número de processos, e também mantém a ordem de chegada, a FIFO, essencial para simular filas de prioridade.

2) Análise da complexidade (Big-O) das operações que foram implementadas

Processo (construtor, getters/setters) é O(1)

ListaDeProcessos.adicionar éO(1)

ListaDeProcessos.remover é O(1)

ListaDeProcessos.isEmpty é O(1)

ListaDeProcessos.imprimirLista é O(n)

ListaDeProcessos.contemId é O(n)

Scheduler.adicionarProcesso é O(N)

Scheduler.contemId é O(N)

Scheduler.desbloquearProcesso é O(1)

Scheduler.escolherProcesso é O(1)

Scheduler.executarCicloDeCPU é O(N)

Scheduler.imprimirEstado é O(N)

Main (leitura + adição de processos) é O(P × N)

Main (loop execução de ciclos) é O(C × N)

3) Análise da Anti-Inanição

No meu projeto, o escalonador usa o atributo contador_ciclos_alta_prioridade. Em que após 5 execuções seguidas de alta prioridade, ele força a execução de um processo de prioridade média ou, se não houver, de baixa prioridade. Dessa forma, é garantido que processos de menor prioridade também recebam tempo de CPU, evitando starvation. Se essa regra não existisse, teria o risco de processos de baixa

e média prioridade nunca serem executados, se sempre houvesse processos de alta prioridade na fila.

4) Análise do Bloqueio

Quando o escalonador detecta que um processo precisa de "DISCO", ele não é executado naquele ciclo e é enviado para a lista de bloqueados. No início de cada ciclo, o método desbloquearProcesso() é chamado, removendo o processo mais antigo da lista de bloqueados e o reinserindo na sua fila de prioridade original.

O ciclo de vida de um processo que precisa do “DISCO” é: primeiramente entra normalmente em sua fila de prioridade. Depois, ao solicitar "DISCO", é movido para a lista de bloqueados. Após isso, em um novo ciclo, ele é desbloqueado e retorna ao final da sua lista de prioridade. Por fim, continua executando até consumir todos os ciclos e finalizar.

5) Ponto Fraco

O principal gargalo de desempenho está na verificação de IDs duplicados com o método contemId. Essa operação é O(n) em cada lista e, como existem 4 listas, o pior caso pode chegar a O(4n). Apesar dela ser aceitável para um número reduzido de processos, em cenários maiores poderia se tornar ineficiente. Uma melhoria teórica seria adotar uma tabela hash para buscas em O(1), mas isso iria violar a regra do trabalho que proíbe o uso de estruturas prontas.