Kaik Araújo Cabral - 2ºPeríodo Allen
1. Justificativa de Design:

A estrutura de dados utilizada é uma lista encadeada simples com ponteiro para o final (tail). Ela é eficiente para o escalonador porque permite operações de remoção do início (O(1)) e adição no final (O(1)), o que simula perfeitamente o comportamento de uma fila (queue) para escalonamento round-robin dentro de cada nível de prioridade. Como não usamos estruturas prontas, implementamos manualmente os nós e referências, atendendo à regra fundamental.

2. Análise da complexidade (Big-O):

- Adição no final (addToEnd): O(1), pois usa o tail.
- Remoção do início (removeFromFront): O(1), apenas atualiza head.
- Verificação de vazio (isEmpty): O(1).
- toString (impressão): O(n), onde n é o número de processos na lista, pois percorre todos os nós. No scheduler, as operações principais (seleção, execução, bloqueio) são O(1) por ciclo, exceto impressões que são O(total processos).

3. Análise da Anti-Inanição:

A lógica garante justiça ao forçar a execução de um processo de média ou baixa prioridade após 5 execuções consecutivas de alta prioridade, resetando o contador. Sem essa regra, processos de baixa prioridade poderiam sofrer inanição (starvation) se houver um fluxo constante de processos de alta prioridade, nunca permitindo que os de menor prioridade avancem.

4. Análise do Bloqueio:

Um processo que precisa de "DISCO" inicia na sua lista de prioridade. Quando selecionado para execução, em vez de rodar, é movido para o final da lista de bloqueados (simulando espera por I/O). No início de cada ciclo subsequente, o processo mais antigo nos bloqueados é removido do início e adicionado de volta ao final da sua lista de prioridade original, com o recurso_necessario setado para null (assumindo que o I/O completou). Assim, na próxima seleção, ele executa normalmente sem bloquear novamente. Isso simula o ciclo: pronto -> bloqueado -> pronto.

5. Ponto Fraco:

O principal gargalo de performance é a operação toString para impressão das listas em cada ciclo, que é O(n) e pode ser custosa se houver muitos processos. Uma melhoria teórica seria otimizar a impressão mantendo um contador de tamanho e imprimindo apenas IDs sem percorrer toda lista a cada ciclo, ou usar uma estrutura com acesso mais rápido (mas respeitando as regras), como uma lista duplamente encadeada para outras operações se necessário.