- **Justificativa de design:**

A estrutura de dados principal é uma lista duplamente encadeada, ela é a escolha mais eficiente para isso, pois as operações de adição e remoção, tanto no início quanto no final da fila são extremamente rápidas, com complexidade de tempo constante, ou **O(1)**. Manter ponteiros para a cabeça e a cauda permite adicionar e remover elementos sem a necessidade de percorrer toda a estrutura.

A classe do escalonador adapta a lista encadeada para uma lista duplamente encadeada circular ao fazer com que o último elemento aponte de volta para o primeiro. Esta é a representação de dados perfeita para o algoritmo Round Robin. Ela permite que o escalonador avance do processo atual para o próximo de forma cíclica e infinita com uma única operação **O(1)**, refletindo exatamente como o quantum de tempo da CPU é passado de um processo para o outro em um loop contínuo.

- **Complexidade Big O:**

Análise de complexidade das operações:

Organizar Novos Processos **O(n)**: quando um ou mais processos chegam ao sistema, o método que os distribui para as filas de prioridade corretas precisa iterar sobre cada um deles, formando a complexidade linear

Ler arquivo **O(n)**:a operação lê linha por linha, formando um mecanismo de complexidade linear.

Adicionar um processo na fila **O(1)**: ela apenas atualiza o ponteiro da cauda, mantendo o tempo constante.

Remover um processo da fila **O(1)**: a operação apenas move o ponteiro da cabeça para o próximo nó, o que é uma operação de tempo constante.

Selecionar próximo processo **O(1)**: consiste em várias verificações de if e else, que resulta em uma remoção, que também é constante, mantendo a complexidade da operação.

Executar quantum e avançar **O(1)**: a parte principal de decrementar os ciclos dos processos e mover o ponteiro do atual para o atual.proximo é de tempo constante.

Remover processo do escalonador **O(1)**: graças à implementação da lista duplamente encadeada, a remoção de um processo que concluiu sua execução é feita em tempo constante. O sistema acessa diretamente o nó anterior e o próximo para refazer as conexões, eliminando a necessidade de percorrer a lista. Isso torna o escalonador extremamente eficiente, independentemente do número de processos em execução.

- **Análise da Anti-inanição:**

A lógica do mecanismo de prevenção à inanição é imprescindível para manter a justiça do escalonador, ele previne que processos de alta prioridade monopolizem o escalonador, fazendo com que os de prioridade menores adiem por tempo indefinido sua execução no sistema.

O sistema de anti-inanição é aplicado na classe Espera do escalonador, após os processos serem divididos em suas respectivas prioridades. Ele implementa um contador de processos de alta prioridade consecutivos, após chegar em 5 processos, ele obriga o método de selecionarProximoProcesso a ignorar a fila de alta prioridade e dar uma chance para as prioridades menores. Se essa regra não existisse, o sistema estaria vulnerável ao fenômeno de inanição. Em um cenário com um fluxo constante de novos processos de alta prioridade, as filas de média e baixa prioridade poderiam nunca ser acessadas. Processos menos prioritários ficariam presos em suas filas para sempre, sem nunca receber tempo de CPU para serem executados, o que é inaceitável em um sistema operacional de propósito geral.

- **Análise do Bloqueio:**

Um processo que necessita de acesso ao disco passa por um ciclo de vida específico que o move por vários estados e listas. Primeiramente ele chega ao sistema e é adicionado à lista de espera e aguarda ser organizado em sua devida fila de prioridade, assumindo o estado de pronto. Eventualmente, ele é retirado da fila de prioridade, mas antes de enviá-lo para execução, o método verificarEPrepararProcesso é chamado. Ele detecta que o processo precisa de disco e que o booleano jaUsouDisco é falso. Em vez de ir para o escalonador, o processo é movido para a lista de bloqueados e seu booleano jaUsouDisco marcada como true, assim ele espera para ser chamado novamente. Após alguns ciclos de simulação, o método desbloquearProcessoMaisAntigo remove o processo do início da lista de bloqueados e o devolve para sua fila de prioridade original. Ele volta ao estado de pronto. Na próxima vez que for selecionado, a verificação verificarEPrepararProcesso falhará (pois jaUsouDisco agora é true), e o processo será finalmente enviado ao escalonador para receber seu quantum de CPU e ser executado.

- **Ponto fraco:**

O escalonador não é totalmente preemptivo. Se um processo de baixa prioridade já está no escalonador executando seu quantum, a chegada de um novo processo de altíssima prioridade não o interrompe imediatamente. O processo de alta prioridade precisa esperar o de baixa prioridade terminar seu quantum para poder ser selecionado. Em um sistema operacional real, isso

não aconteceria. Um processo de altíssima prioridade deve poder tomar a CPU de um processo de prioridade mais baixa instantaneamente. O comportamento atual do escalonador pode levar a um atraso na resposta de tarefas importantes.

Otimização de Performance da Lista Duplamente Encadeada

Durante o desenvolvimento do projeto, foi identificado que o uso de uma lista simplesmente encadeada para o escalonador Round Robin gerava um gargalo de performance significativo. A operação de remoção de um processo, necessária quando seus ciclos terminavam, tinha uma complexidade de O(n), pois exigia uma busca linear pelo nó anterior.

Para solucionar este problema, o projeto foi refatorado para utilizar uma lista duplamente encadeada circular. Essa decisão de design eliminou completamente o gargalo, reduzindo a complexidade da remoção para O(1). Com acesso direto aos nós anterior e próximo, o sistema agora mantém uma alta performance de forma consistente, mesmo com um grande volume de processos entrando e saindo da fila de execução. Além disso, foi aplicado o “//” (transformando em comentário) em alguns prints de saída do sistema, tornando-os opcionais, e otimizando a velocidade que o sistema executa os processos, evitando gargalos com saídas exageradas, a aplicação de uma verificação para mostrar o estado das filas a cada 100 quantuns também foi aplicada.