Relatório de Análise e Justificativa de Design

1. Justificativa de Design
Para implementar o escalonador de processos, optamos por utilizar listas encadeadas implementadas do zero, manipulando diretamente nós e referências. Essa decisão garante aderência às regras do projeto, que proíbem estruturas prontas como ArrayList ou LinkedList.

Nota importante: Diferente de uma padronização única, mantivemos duas implementações distintas de listas:
ListaDuplamente para as filas de alta, média e bloqueados. **ListaDeProcessos** (simples encadeada) para a fila de baixa prioridade. Essa escolha foi intencional para exercitar o uso de diferentes estruturas. A lista duplamente encadeada facilita remoções intermediárias e caminhamento em dois sentidos, o que pode ser útil em filas mais dinâmicas (alta, média e bloqueados). Já a lista simplesmente encadeada é suficiente para a fila de baixa, que funciona como FIFO (primeiro a entrar, primeiro a sair).

2. Análise de Complexidade (Big-O)
As principais operações possuem as seguintes complexidades: **Inserção no fim**: O(1), pois mantemos ponteiro para o fim da lista. **Remoção do início**: O(1), apenas atualização de ponteiro. **Busca/impressão**: O(n), percorrendo a lista completa. Portanto, o escalonador é eficiente para inserção e remoção, que são as operações mais frequentes.

3. Análise da Anti-Inanição
A lógica implementada utiliza um contador para controlar execuções consecutivas de processos de alta prioridade. Após 5 execuções seguidas da fila de alta prioridade, o escalonador obriga a execução de um processo da fila de média prioridade, ou da fila de baixa caso a média esteja vazia. Esse mecanismo garante que processos de menor prioridade não fiquem indefinidamente sem CPU.

Sem essa regra, haveria o risco de *starvation*, em que processos de baixa prioridade nunca seriam executados caso houvesse sempre processos de alta prioridade disponíveis.

4. Análise do Bloqueio
Quando um processo precisa do recurso **DISCO** e ainda não foi bloqueado anteriormente, ele não executa no ciclo atual. Em vez disso, é removido de sua fila de prioridade e enviado para a lista de bloqueados. No início de cada ciclo, o escalonador desbloqueia o processo mais antigo da lista de bloqueados, reinserindo-o no final de sua fila de origem. Assim, o ciclo de vida de um processo que precisa de DISCO é:
Fila de prioridade → detecta necessidade de DISCO → movido para lista de bloqueados. Lista de bloqueados → aguardando um ciclo. Retorno para a fila de origem → continua a execução normal até terminar seus ciclos. Esse mecanismo simula de forma simples o gerenciamento de recursos de E/S.

5. Ponto Fraco e Possível Melhoria
O principal gargalo de desempenho do escalonador está na operação de impressão e varredura completa das listas, que possui custo O(n). Embora isso não comprometa a execução em cenários pequenos, em sistemas reais com milhares de processos poderia ser ineficiente.

Uma melhoria teórica seria utilizar uma estrutura mais elaborada, como uma *fila circular* ou até uma *árvore balanceada* para priorização, reduzindo buscas e permitindo maior escalabilidade. No entanto, para o propósito educacional deste projeto, as listas encadeadas implementadas são suficientes e atendem bem às exigências.

Conclusão
O escalonador desenvolvido segue fielmente as regras propostas, utilizando estruturas próprias,

implementando prevenção de inanição, gerenciamento de bloqueios e garantindo execução justa entre processos de diferentes prioridades. A escolha de manter duas implementações distintas de listas foi intencional, como forma de exercitar múltiplos modelos de estruturas de dados, sem comprometer a lógica de escalonamento.