Relatório de Análise

Justificativa de Design: A estrutura de dados escolhida para implementar o Escalonador de processos foi a lista encadeada simples, que se comporta como fila e se organizam em múltiplas filas de prioridade (alta, média, baixa). Essa abordagem se mostrou eficiente por possuir operações O(1) para operações críticas, como o adicionar_final(), remover_inicio() e lista_vazia(), além da simplicidade, com cada nó consumindo apenas a memória necessária para armazenar o processo, sem alocação dinâmica desnecessária, como o que poderia acontecer com Array.

Análise de complexidade (Big-O):

- *adicionar_processo()*: O(1), adiciona o processo diretamente na fila de prioridade correspondente sem necessidade de busca

- *executar()*: O(1) por ciclo

-- Desbloqueio: O(1) (remoção do início da lista de bloqueados).

-- Seleção do processo: O(1) (acesso direto às cabeças das filas).

-- Bloqueio: O(1) (inserção no final da lista de bloqueados).

-- Execução: O(1) (decremento de ciclos e reinserção).

- *status()*: O(n), para exibir o estado de todas as filas, é necessário percorrer cada lista encadeada. No pior caso, onde todos os processos estão em uma mesma fila.

- *main()*: O(n)

-- Leitura do CSV: O(n) “for linha in leitor” percorre cada linha do arquivo uma vez.

-- Conversão de dados: O(1) operações de conversão para int e checagem de strings são constantes.

-- Adição do processo: O(1) cada processo é adicionado ao scheduler com complexidade constante.

Análise da Anti-Inanição: O sistema de anti-inanição impede que processos de baixa prioridade fiquem esperando para sempre. A cada 5 processos de alta prioridade executados, o escalonador é obrigado a executar um processo de média ou baixa prioridade antes de continuar. Isso garante que todos os processos, mesmo os menos importantes, tenham chance de executar. Sem essa regra, processos como backups ou atualizações poderiam nunca rodar se sempre chegassem processos mais urgentes.

Análise do Bloqueio: Quando um processo precisa do recurso "DISCO", ele primeiro é bloqueado, até poder voltar a sua lista de origem e ser escolhido pelo escalonador. Na sua primeira execução, o sistema detecta que ele precisa do DISCO e o move imediatamente para a lista de bloqueados, onde fica esperando. No ciclo seguinte, ele é automaticamente desbloqueado e volta para o final da sua fila de prioridade original. Quando é escolhido novamente, como já solicitou o recurso anteriormente, ele executa normalmente até completar seus ciclos restantes, sem precisar de novo bloqueio.

Ponto Fraco: O principal gargalo de performance está no método status() que exibe o estado das filas. Cada vez que é chamado, ele percorre todas as listas encadeadas para construir as strings de exibição, o que tem complexidade O(n) onde n é o número total de processos. Isso se torna lento quando há milhares de processos.

Melhoria teórica: Implementar um sistema de cache que atualiza a representação das filas apenas quando elas são modificadas. Em vez de percorrer todas as listas a cada exibição, o sistema manteria uma versão atualizada da string de cada fila e só a recalcularia quando processos forem adicionados ou removidos. Isso reduziria a complexidade de O(n) para O(1) na exibição do status.