Relatório de Análise e Justificativa de Design

Escalonador de Processos iCEVOS

Integrantes:

- Igor Pereira Lima
- João Matheus Ramos Araújo
- Erick Rhuan Carvalho de Freitas Pereira

Disciplina: Algoritmos e Estrutura de Dados I

Professor: Dimmy Magalhães

Instituição: ICEV

Setembro 2025

1. Justificativa de Design

Escolha das Estruturas de Dados

Para a ListaDeProcessos optamos por lista encadeada simples para as filas de prioridade pelas seguintes razões: O escalonador precisa constantemente remover processos do início (O(1)) e reinserir no final (O(1) com ponteiro tail), não há limite pré-definido de processos por fila, Ordem FIFO preservada para o funcionamento do algoritmo Round Robin.

Adquirir a eficiência só foi possível pelo Ponteiro Tail que insere diretamente no final da lista com a complexidade O(1), justificando, assim, o Design escolhido. Esta decisão representa um trade-off de 8 bytes de memória por lista em troca de eficiência algorítmica crítica.

Análise do Trade-off:

- Custo: 3 listas × 8 bytes = 24 bytes adicionais
- Benefício: Inserção O(1) vs O(n) - fundamental para escalabilidade
- Justificativa: Com n processos, inserção O(n) resultaria em n^2 operações no pior caso, tornando o sistema inviável

Para a fila de processos bloqueados, escolhemos lista circular pelas características: Rotação natural: Facilita o desbloqueio FIFO (primeiro a entrar, primeiro a sair), Eficiência: Inserção O(1) no final, remoção O(1) da cabeça, Semântica adequada: Lista circular representa bem a natureza cíclica do gerenciamento de recursos

Também optamos por um Node específico para EstruturaProcesso em vez de genérico. Visto que, Elimina necessidade de casting e possíveis ClassCastException, Reduz complexidade do código, adequado ao escopo específico do projeto e Ligeiramente melhor por evitar operações de cast.

2. Complexidade das Operações

As operações críticas do escalonador possuem baixo custo computacional:

- Inserção no final: O(1)
- Remoção do início: O(1)
- Reinserção no final após execução: O(1)
- Mover processos entre listas (ex.: bloqueados): O(1)
- Busca de processo específico (não comum no fluxo): O(n)

Com isso, o sistema se mantém eficiente mesmo com muitos processos.

3. Anti-Inanição

A política de anti-inanição impede que processos de prioridade média ou baixa fiquem indefinidamente sem CPU. O escalonador conta quantos processos de alta prioridade foram executados em sequência e, após cinco execuções, força a execução de um processo de prioridade média. Caso não haja, escolhe-se um de baixa. O contador é então zerado. Essa estratégia garante justiça, pois evita que apenas processos de alta prioridade monopolizem os recursos. Sem essa regra, haveria risco de inanição, com processos de menor prioridade nunca sendo executados.

4. Gerenciamento de Bloqueio

O recurso “DISCO” foi usado para simular bloqueios. Quando um processo precisa desse recurso, ele não executa. Em vez disso, é removido de sua fila de prioridade e colocado no final da lista de bloqueados. No início de cada ciclo, o processo mais antigo dessa lista é desbloqueado e retorna ao final de sua fila de origem. Esse mecanismo mantém a ordem justa de desbloqueio e reintegração, além de ser eficiente, pois as movimentações ocorrem em tempo constante.

5.Identificação do Principal Gargalo

Após análise detalhada do sistema, os principais gargalos identificados são: Processamento sequencial de ciclos, ou seja, não é possível colocar processos paralelos e também o sistema processa um processo por ciclo, ou seja, seu quantum = 1. Isso faz com que cada ciclo de CPU reduza em 1 a quantidade de ciclos necessários em cada processo. Desse modo o comportamento do algoritmo Round Robin se torna mais parecido com uma FIFO, deixando alto o tempo médio de espera de cada processo.

Proposta de Melhoria Teórica

Implementar um aumento do quantum de tempo variável baseado na prioridade, permitindo que processos de alta prioridade executem múltiplos ciclos consecutivos. Ou seja, a cada vez que um processo vai pro CPU, ele reduz a quantidade do quantum no número da quantidade de cpu necessária que cada processo precisa.

Fazendo isso é possível diminuir o tempo médio de espera de cada processo.

Quantum por Prioridade:

- Alta prioridade: Quantum = 3 ciclos
- Média prioridade: Quantum = 2 ciclos

- Baixa prioridade: Quantum = 1 ciclo