
Trabalho Prático – P1

Disciplina: Algoritmos e Estruturas de Dados I
Professor: Dimmy Magalhães
Membros:

[Pedro Henrique Rodrigues Jacques Pinheiro],

[Maria Clara Sousa de Oliveira],

[José Airton Rodrigues Galdino Junior]

Data: [17/09/2025]

Relatório de Análise e Justificativa de Design

1. Justificativa de Design

Para gerenciar os processos do escalonador, utilizamos **listas encadeadas simples** (`ListaDeProcessos`) para cada prioridade (Alta, Média, Baixa) e para os bloqueados.

Essa escolha é eficiente porque:

- Adicionar no final e remover do início é **rápido (O(1))**, perfeito para o comportamento FIFO do escalonador.
- Cada prioridade tem sua própria lista, facilitando a seleção do próximo processo sem misturar tudo.
- A lista de bloqueados permite desbloquear sempre o processo mais antigo, essencial para recursos como “DISCO”.

Em resumo, as listas mantêm o código simples, direto e com operações rápidas na maior parte do tempo.

---

2. Complexidade (Big-O)

Operação	Estrutura	Complexidade
Adicionar no final	ListaDeProcessos	O(1)
Remover do início	ListaDeProcessos	O(1)
Buscar por ID	ListaDeProcessos	O(n)
Remover por ID	ListaDeProcessos	O(n)
Desbloquear primeiro da lista de bloqueados	ListaDeProcessos	O(1)
Selecionar próximo processo a executar	Três listas de prioridade	O(1)

A maior parte das operações críticas é rápida, mas buscas ou remoções por ID podem ficar lentas se houver muitos processos.

---

3. Anti-Inanição

A lógica de anti-inanição funciona assim: a cada **5 ciclos de processos de alta prioridade**, o scheduler força a execução de um processo de **média ou baixa prioridade**.

Isso garante que processos de baixa prioridade **não fiquem esperando para sempre**. Sem essa regra, eles poderiam nunca ser executados, gerando **injustiça no escalonamento**.

---

4. Ciclo de vida de um processo bloqueado (ex.: recurso “DISCO”)

1. O processo é criado e entra na lista de prioridade correspondente (Alta, Média ou Baixa).

2. Quando chega na CPU, se precisa do recurso “DISCO” e ainda não usou, é **movido para a lista de bloqueados**.

3. Ele espera lá até ser o **processo mais antigo** da lista de bloqueados.

4. No início de cada ciclo, o scheduler **remove o primeiro da lista de bloqueados** e o retorna para sua lista de prioridade original.

5. Ele continua executando normalmente, decrementando seus ciclos até finalizar.

Resumo do ciclo:
Prioridade → Bloqueado → Prioridade → Finalizado

---

5. Ponto Fraco e Melhoria

O principal gargalo do escalonador é o uso de **três listas separadas de prioridade**. A cada ciclo, ele precisa verificar cada lista em sequência (Alta → Média → Baixa) para escolher o próximo processo.

Além disso, operações como **buscar ou remover por ID** são O(n), o que pode ficar lento com muitos processos.

Melhoria proposta: usar **uma única fila de prioridade com Heap Binário**:

- O processo de maior prioridade fica sempre no topo, tornando a escolha e remoção **O(log n)**, muito mais rápido do que percorrer listas.

- Uma estrutura única deixa o código mais limpo e facilita aplicar regras de **anti-inanição e bloqueio**.