Relatório do Escalonador de Processos

O relatório a seguir busca responder às seguintes questões:

1. Justificativa de Design: Por que a estrutura de dados utilizada é uma estrutura eficiente para implementar o comportamento do escalonador?
2. Analise a complexidade (Big-O) das operações na sua implementação.
3. Análise da Anti-Inanição: Explique como sua lógica garante a justiça no escalonamento e qual o risco se essa regra não existisse.
4. Análise do Bloqueio: Descreva o ciclo de vida de um processo que precisa do "DISCO", detalhando sua jornada pelas diferentes listas.
5. Ponto Fraco: Qual é o principal gargalo de performance no seu scheduler? Proponha uma melhoria teórica para ele.

1. Justificativa de Design

A estrutura utilizada para o Escalonador de processo foi uma lista simples encadeada. A razão da escolha foi por 3 fatores:

1. **Eficiência nas operações básicas:**
   A lista encadeada simples permite inserções e remoções em tempo constante O(1) quando já se possui a referência do nó. Isso é especialmente útil no contexto de um escalonador, em que processos estão constantemente entrando e saindo das filas de execução.

2. **Simplicidade de implementação:**
   Por se tratar de uma estrutura linear, a lista encadeada simples apresenta baixo custo de implementação e manutenção. Como cada nó contém apenas um ponteiro para o próximo elemento, o gerenciamento da memória é mais enxuto em comparação a estruturas mais complexas.

3. **Adequação ao problema:**
   O comportamento de um escalonador envolve principalmente manipulação sequencial das filas de processos (adição, remoção e movimentação entre listas). Assim, o uso de uma estrutura simples e direta já é suficiente para representar o fluxo dos processos sem introduzir sobrecarga desnecessária.

2. **Analise a complexidade (Big-O)**

A análise a complexidade Big-O, para responder essa pergunta o código será dividido em 8 métodos principais é serão mostrados na tabela a seguir:

Operações	Complexidade
Adicionar	O(1)
Remover	O(1)

Mover para bloqueados	O(1)
Desbloquear	O(1)
Imprimir	O(n)
Leitura de arquivo	O(n)
Loop de Scheuduler	O(n)

As principais operações do escalonador apresentam custo eficiente: inserção, remoção, movimentação para bloqueados e desbloqueio ocorrem em O(1); já operações que percorrem a lista, como impressão, leitura de arquivo e o loop do escalonador, possuem custo O(n). Assim, a complexidade geral do sistema pode ser considerada O(n).

3. Análise da Anti-Inanição

A regra de anti-inanição garante que processos de menor prioridade não fiquem esquecidos. Para isso, a cada cinco execuções de processos da fila de alta prioridade, o escalonador desvia a execução para a fila de prioridade inferior (média ou baixa, caso necessário). Dessa forma, assegura-se justiça no uso da CPU e evita-se o problema de fome (starvation), onde processos de baixa prioridade poderiam nunca ser executados.

4.Análise do Bloqueio

O Processo no método adicionar ele passa por um verificação na variável String Recurso caso a variável seja igual a “DISCO” ele vai para a lista de bloqueados e após um ciclo de execução o algoritmo verifica se o lista de bloqueados é vazia e caso não seja o processo que está na cabeça da lista vai para sua lista de prioridade original. A seguir um exemplo do ciclo de vida de um processo com bloqueado.

Novo processo → entra na fila → solicita DISCO → vai para lista de bloqueados → espera liberação → é desbloqueado → volta para a fila de pronto → executa novamente.

5. Ponto Fraco

O principal ponto fraco identificado na implementação do escalonador está relacionado ao uso da lista encadeada simples como estrutura de dados central. Embora apresente boa performance para inserções e remoções quando a referência do nó já é conhecida, essa estrutura exige tempo de busca linear O(n) para localizar ou percorrer processos específicos. Em cenários com grande número de processos, esse comportamento pode se tornar um gargalo de desempenho.

Outra limitação está na ausência de acesso direto a elementos intermediários, o que torna operações de reordenação ou de verificação de prioridades mais custosas.

Proposta de melhoria teórica:

Uma alternativa seria utilizar uma lista duplamente encadeada, que facilitaria tanto a navegação para frente quanto para trás, otimizando operações de remoção e movimentação entre listas.

Além disso, em sistemas de larga escala, estruturas híbridas, que combinam filas circulares com diferentes níveis de prioridade, poderiam garantir maior eficiência e justiça no escalonamento.