Programação Paralela

Otimização e Desempenho em Processamentos

Alunos: Ana Tereza Guimarães Pereira, Larissa Batista Maciel, Pedro Henrique Ruas Soares, Samuel Gonçalves de Araujo e, Welber Henrique Rodrigues Costa

O QUE É PARALELISMO?

- Técnica que permite executar múltiplas operações ao mesmo tempo.
- Utiliza **threads** para dividir e processar tarefas simultaneamente.
- Aumenta a eficiência e reduz o tempo de execução de programas.

POR QUE O PARALELISMO SE TORNOU OBRIGATÓRIO?

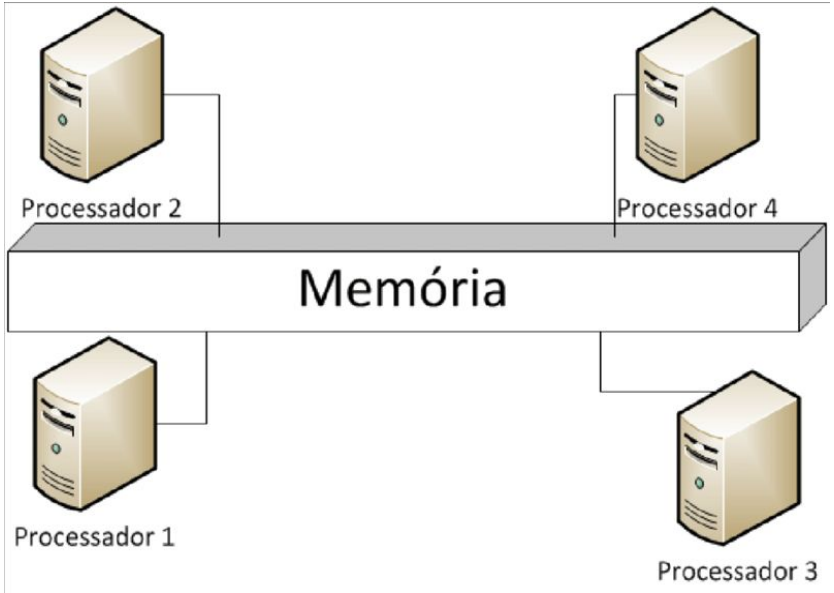
- **Problemas cada vez mais complexos** exigem mais poder computacional.
- **Aumentar o clock não é mais viável** devido ao consumo de energia e aquecimento.
- **Solução:** Aumento do número de núcleos para rodar múltiplas tarefas ao mesmo tempo.

ONDE O PARALELISMO É UTILIZADO

- Usado em **pesquisas científicas** para simulação de moléculas e reações químicas.
- **Previsão do tempo** para análise de trilhões de dados simultaneamente.
- **Renderização de vídeos** para cálculo de bilhões de pixels por segundo.
- Permite resolver vários outros problemas complexos de forma mais eficiente.

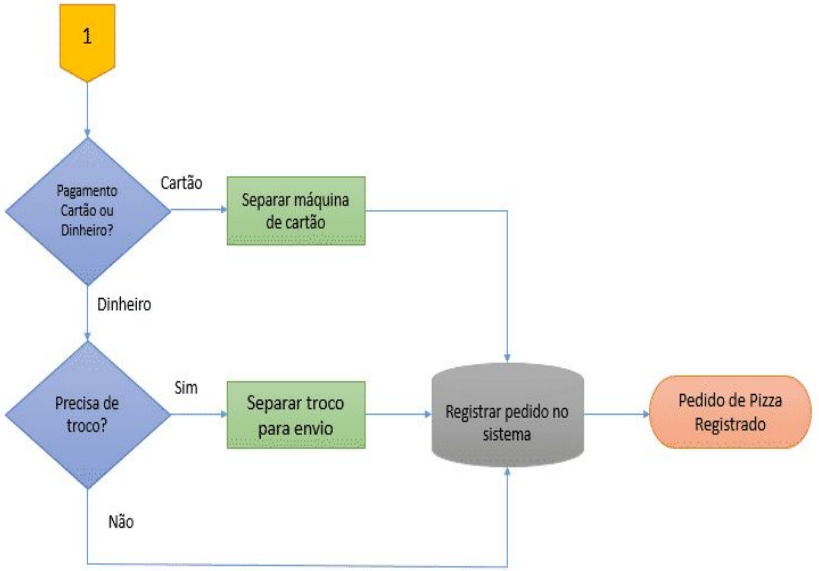
Introdução ao Processo

- Um processo gerencia os recursos necessários para a execução de tarefas.
- **Isolamento entre processos:** Não compartilham memória ou recursos diretamente.
- Garantia de isolamento por mecanismos de proteção de hardware.
- Cada processo opera em diferentes níveis de privilégio.



Estados de um Processo

- **Execução (Running)**: Processo sendo executado pela CPU.
- **Pronto (Ready)**: Processo aguarda para ser executado.
- **Espera (Wait)**: Processo aguarda por um evento ou recurso.



Controle de Processo - PCB

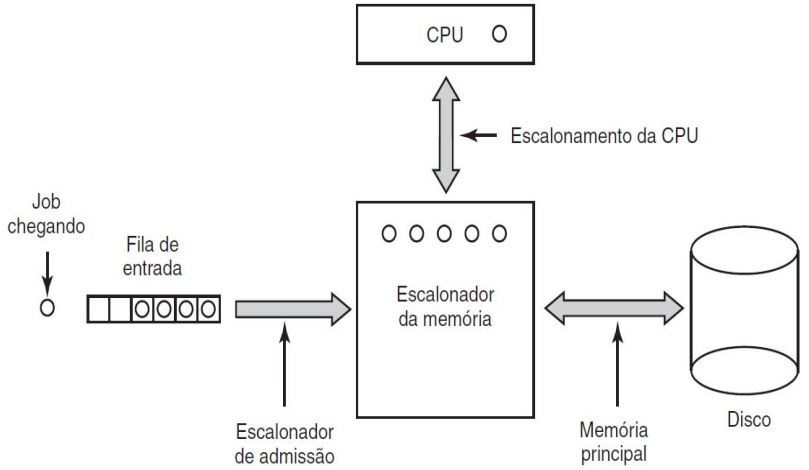
- **Armazena informações sobre cada processo para gerenciamento e controle.**
- **Informações principais**:
- Identificador do Processo (PID)
- Estado atual do processo
- Registradores da CPU
- Informações de escalonamento
- Informações de uso de memória

Escalonamento de Processos

Gerencia a execução dos processos, compartilhando a CPU de maneira eficiente.

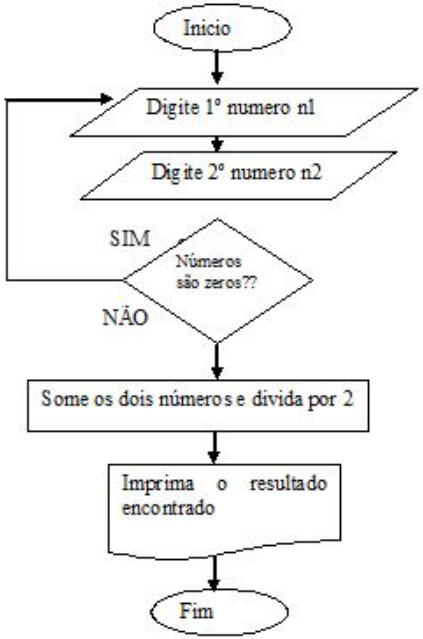
Tipos de escalonadores:

- **Escalonador de Longo Prazo**
- **Escalonador de Curto Prazo**
- **Escalonador de Médio Prazo**



Algoritmos de Escalonamento

- **FIFO (First In, First Out)**: O primeiro a chegar é o primeiro a ser executado.
- **Round Robin**: Tempo fixo de execução antes de ser substituído.
- **Prioridade**: Processos de maior prioridade são executados primeiro.
- **Shortest Job Next (SJN)**: O processo com menor tempo de execução é escolhido primeiro.



Comunicação entre Processos (IPC)

- **Memória Compartilhada**: Acesso comum de memória entre processos.
- **Filas de Mensagens**: Troca de mensagens entre processos via sistema operacional.
- **Pipes**: Comunicação baseada em fluxos de dados.

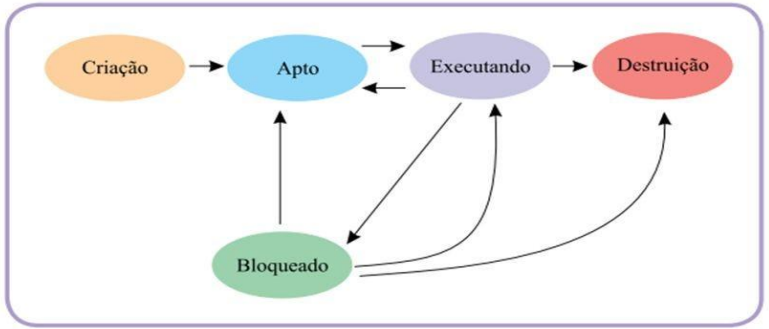
Conclusão

- Resumo dos tópicos abordados: O que são processos, seus estados, controle, escalonamento e IPC.
- Importância do gerenciamento de processos para a eficiência do sistema operacional.

SISTEMAS OPERACIONAIS I

Gerenciamento de processos

Estados do processo:



O QUE SÃO THREADS?

- **Menor unidade** de execução dentro de um processo
- Permitem **execução simultânea** de tarefas
- **Compartilham memória** do processo principal
- Usadas para **melhorar desempenho e resposta** do programa

PARALELISMO DE DADOS

- Quando **múltiplas threads executam a mesma operação** em partes diferentes dos dados.
- Cada thread trabalha em um subconjunto da informação.

X

PARALELISMO DE TAREFAS

- Quando **diferentes threads executam operações distintas ao mesmo tempo**.
- Ideal para aplicações que possuem múltiplas funcionalidades simultâneas.

CONCORRÊNCIA DE THREADS

Concorrência X Paralelismo

Várias threads são alternadas em um único núcleo, criando a sensação de execução simultânea, mas apenas uma thread é executada por vez.

Em sistemas com múltiplos núcleos, as threads podem ser executadas simultaneamente em núcleos diferentes, caracterizando o paralelismo real.

Exemplos de problemas de concorrência:

Condição de Corrida: múltiplas threads acessam e modificam um recurso compartilhado sem controle adequado.

Deadlock: duas ou mais threads ficam presas esperando recursos que nunca serão liberados.

ANALOGIA DO RESTAURANTE

➡ Processo

➡ Thread

Os clientes, representando o usuário, fazem seus pedidos, que simbolizam os processos. Para atender a essas solicitações, existem os cozinheiros, que correspondem aos núcleos do processador. Cada cozinheiro executa diferentes etapas da preparação (threads), usando de estratégias para que os pratos sejam entregues o mais rápido possível, que seriam os algoritmos de escalonamento.

PARALELISMO EM JAVA

ForkJoinPooL

Quando se trata de paralelismo em Java e subdivisão de processos e tarefas, um dos frameworks mais eficientes e utilizados é o **ForkJoinPool**.

PARALELISMO EM JAVA

Ele é particularmente útil quando se trata de dividir uma tarefa grande em sub-tarefas menores, que podem ser executadas em paralelo e depois combinadas.

A lógica funciona da seguinte forma:

PARALELISMO EM JAVA

PASSO A PASSO

- **Tarefa Inicial**: A tarefa começa com um intervalo de números (por exemplo, de 1 a 10). Se o intervalo for grande, a tarefa será dividida.
- **Divisão**: Quando o tamanho da tarefa excede um limite predefinido (neste caso, 1), a tarefa é dividida em duas sub-tarefas.
- **Execução Paralela**: As sub-tarefas são enviadas para execução paralela usando o método fork().

PARALELISMO EM JAVA

PASSO A PASSO

- **Esperando os Resultados**: Após a divisão, o método join() é usado para aguardar o retorno das sub-tarefas.
- **Combinação dos Resultados**: Quando todas as sub-tarefas terminam, os resultados são combinados (no exemplo, somando os valores).
- **Resultado Final**: O resultado final da soma é retornado e impresso.

PARALELISMO EM JAVA

OUTRAS ALTERNATIVAS

1. ExecutorService:

O ExecutorService é um framework mais geral e flexível para a execução de tarefas paralelas

2. CompletableFuture:

É útil quando você tem tarefas assíncronas e precisa de um controle de fluxo

PARALELISMO EM JAVA

EXEMPLO DE CÓDIGO

```java
package paralelismo;

import java.util.concurrent.RecursiveTask;

public class Teste extends RecursiveTask <Integer> {
    private final int inicio;
    private final int fim;

    public Teste(int inicio, int fim) {
        super();
        this.inicio = inicio;
        this.fim = fim;
    }

     protected Integer compute() {
            if (fim - inicio <= 1) {
                return inicio;
            } else {
                int meio = (inicio + fim) / 2;

                Teste subTarefa1 = new Teste(inicio, meio);
                Teste subTarefa2 = new Teste(meio, fim);


                subTarefa1.fork();
                subTarefa2.fork();


                return subTarefa1.join() + subTarefa2.join();
            }
        }
    }
```

FIM