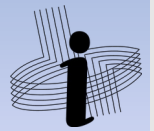
INF 310 – Programação Concorrente e Distribuída

Introdução e Conceitos Básicos

Professor: Vitor Barbosa Souza
vitor.souza@ufv.br

Introdução

- Arquiteturas monoprocessadas
  - Aumento de 50% por ano na performance dos microprocessadores entre 1986 e 2002
  - Aumento da densidade dos transistores → Aumento de temperatura
- Arquiteturas multiprocessadas (*multi-core*)
  - Desde 2005 microprocessadores paralelos são a solução para melhorar performance
  - Permite execução de vários processos de forma simultânea
  - Programas sequenciais precisam ser adaptados
  - Precisamos aprender a aproveitar os vários núcleos de processamento



UFV

Introdução

- Exemplo básico: somando valores de um arranjo de tamanho *n*

```
int a[]={a0,a1,...,an-1};
```

  - Forma sequencial

```
int soma=0;
for(int i=0;i<n;++i)
    soma+=a[i];
```

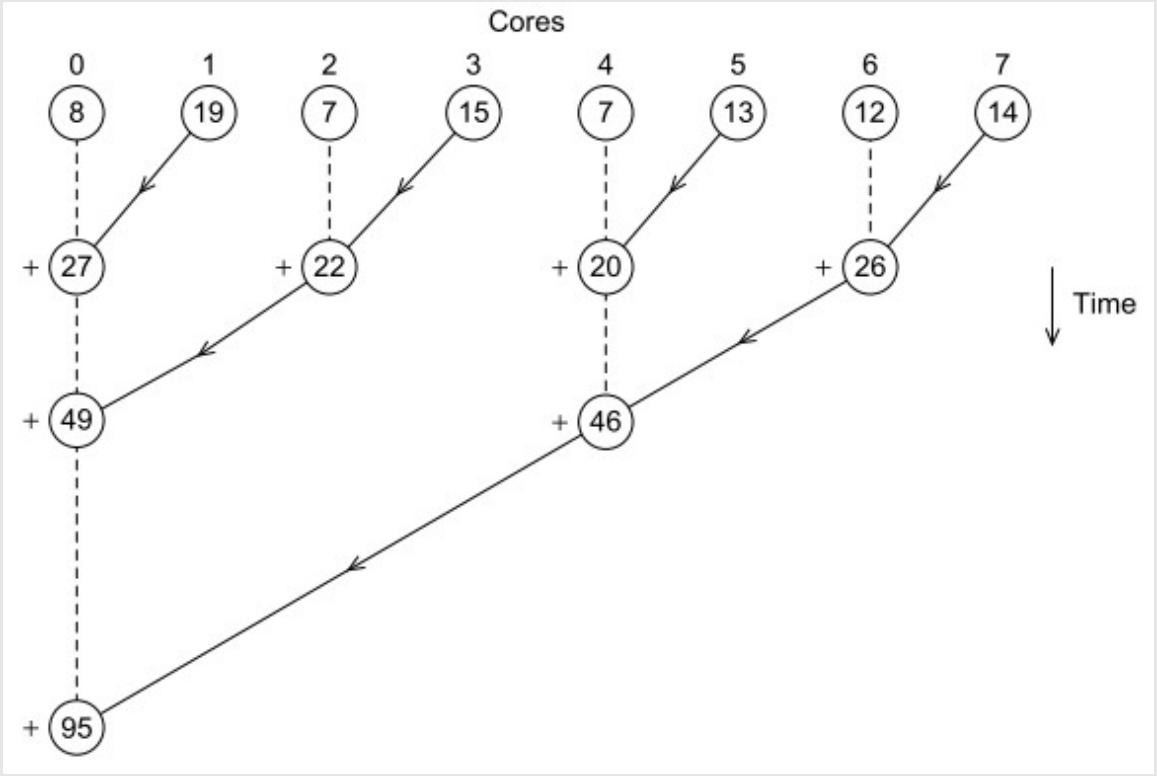
  - Forma paralela (não ótima)

```
int minha_soma=0;
for(int meu_i=meu_inicio; meu_i<meu_fim; ++meu_i)
    minha_soma+=a[meu_i];
if (meu_id == mestre){
    int soma=minha_soma;
    para cada outro core{
        receber valor;
        soma+=valor;
    }
} else {
    enviar minha_soma para mestre;
}
```

Cada valor calculado é recebido iterativamente

Introdução

- Melhor distribuição de carga



Introdução

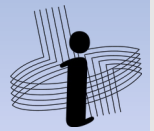
- Paralelismo

Como planejamos

O que pode acontecer

Dando nomes

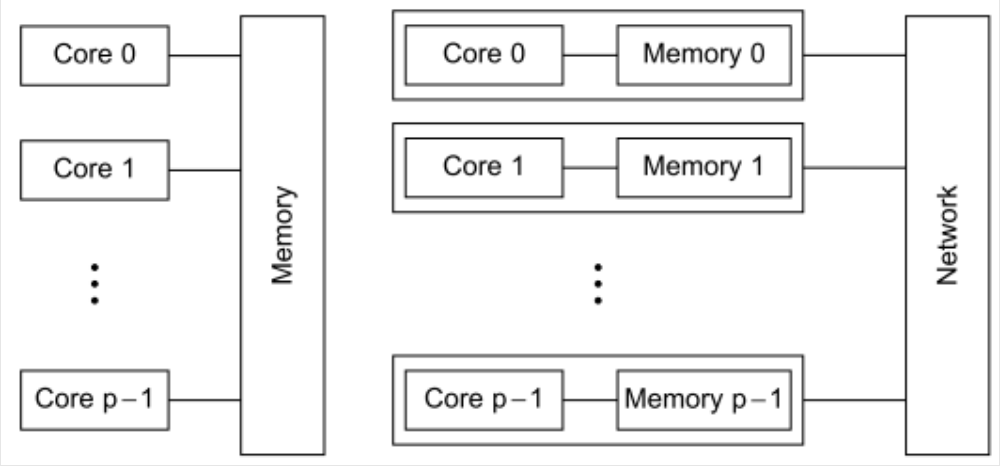
- Principais tipos de paralelismo
  - *Task parallelism*
    - núcleos trabalham com uma certa independência
    - cada *task* pode se assemelhar a um programa sequencial
  - *Data parallelism*
    - cada núcleo se encarrega de processar uma parte dos dados
    - instruções executadas pelos núcleos são similares na maior parte do tempo



UFV

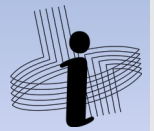
Dando nomes

- Acesso aos dados
  - Memória compartilhada
    - cada núcleo pode ler e escrever em cada posição da memória
  - Memória distribuída
    - cada processo trabalha na sua própria memória
    - comunicação entre processos pode ser realizada através do envio de mensagens, arquivos de saída/entrada, etc.



Dando nomes

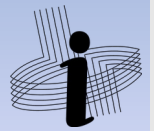
- Programação concorrente
  - várias *tasks* de um programa podem estar executando em um mesmo momento de modo que existe uma concorrência pelo uso dos recursos do sistema
- Programação paralela
  - um problema é dividido em partes menores e cada *task* trabalha de forma colaborativa para resolução do problema
- Programação distribuída
  - processos trabalham de forma colaborativa em um sistema de memória distribuída



UFV

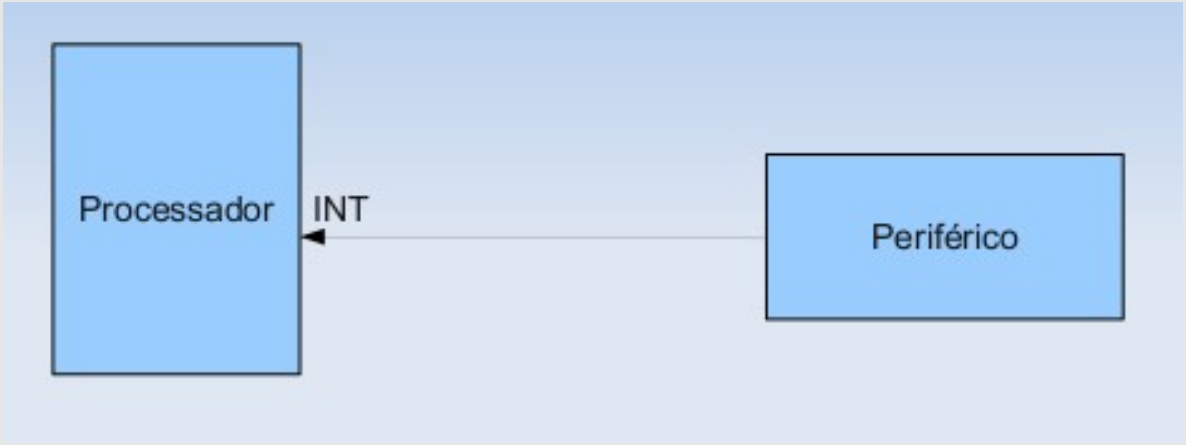
Interrupções

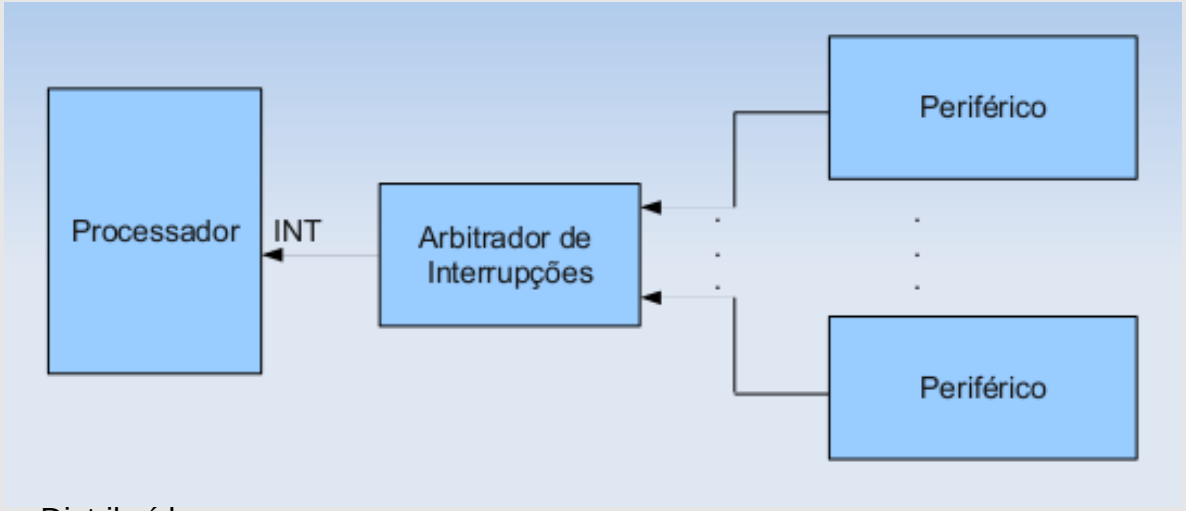
- Como a concorrência acontece no sistema?
- Interrupção de hardware
  - sinal originado por algum dispositivo físico que faz com que o processador suspenda o programa em execução
  - O SO define rotinas de tratamento de interrupção
  - Os processadores possuem instruções para desabilitar (mascarar) interrupções
    - apenas no modo privilegiado uma interrupção pode ser habilitada/desabilitada.



UFV

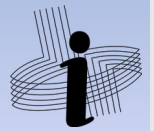
Interrupções





Interrupções

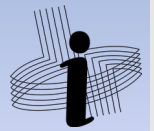
- *Traps* (Interrupções de Software)
  - Gerada pela execução de uma instrução
  - Desencadeia as mesmas ações de uma interrupção de hardware
  - São previsíveis
  - Costumam causar uma interrupção de hardware (instruções de E/S) posterior
  - Chamadas de sistema (*system calls*)
    - Instruções privilegiadas: apenas rotinas do SO podem executá-las
    - Mudança de modo usuário para modo supervisor
    - Volta ao modo usuário após execução



UFV

Interrupções

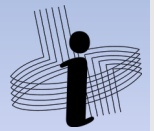
- Um sistema de interrupção típico
  - PC (*Program Counter*)
  - PSW (*Processor Status Word*)
    - 1 bit para indicar modo de execução
      - supervisor ou usuário
    - 3 para mascaramento (*m*)
      - Apenas interrupções com prioridade maior que *m* podem ser executadas
  - SP (*Stack Pointer*)



UFV

Processo

- Processo
  - Todo processo possui área de código e uma área de dados
  - Criação de um processo
    - carga na memória das áreas de dados e código
    - alocação de um conjunto de registradores
    - criação de um Registro Descritor com todas as informações do processo
  - Os processos compartilham o uso do processador
    - ao ser interrompido, as informações do processo são guardadas em seu Registro Descritor
    - quando o processo volta a usar o processador, as informações do Registro Descritor são restauradas
      - Ex: valor do PC



UFV

Processo

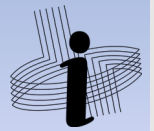
- Exemplo de uma instrução executada no processador
  - Instrução C++ a ser executada

```
res=num1+num2;
```

  - Possível código *Assembly* correspondente

```
mov eax,[num1]   ; carregar num1 para registrador
mov ebx,[num2]   ; carregar num2 para registrador
add eax,ebx      ; somar os valores
mov [res],eax    ; armazenar resultado na memória
```

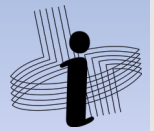
  - O processo pode perder o processador a qualquer momento



UFV

Processo

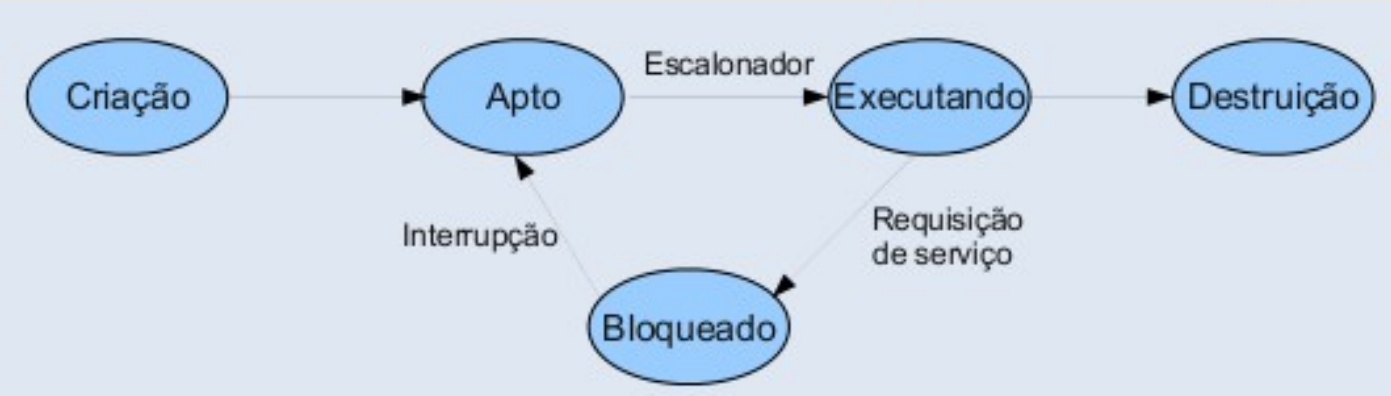
- Em máquinas monoprocessadas apenas 1 processo se encontra em execução a cada instante
  - Processador visto como vários processadores virtuais
  - Ordem de execução geralmente imprevisível
  - Interrupções são transparentes
  - *Traps* bloqueiam o processo
- Em máquinas *multi-core* os mesmos problemas de concorrência são encontrados



UFV

Processo

- Estados de um processo



- Exemplo de execução
  - SO aciona dispositivos de E/S através de instruções protegidas
  - Ao fim interrupções são usadas para notificar a CPU, que pode alocar o dispositivo para outra tarefa
  - Fila de processos aptos (*ready list* ou *ready queue*)

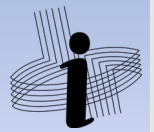
Corrida Crítica (ou condição de corrida)

- A velocidade de execução relativa de processos concorrentes pode gerar resultados diferentes
  - Ex: A = 0; B = 1; C = 2;
  - qual o valor final de A, B e C ?

```
/* código de P1 */
...
B=2*A+C;
...
```

```
/* código de P2 */
...
C=2*A+B;
...
```

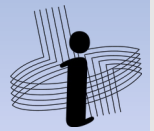
Resposta:
A = 0
B = {1,2}
C = {1,2}



UFV

Recursos Computacionais

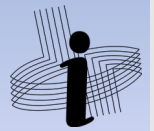
- O SO gerencia diferentes recursos computacionais reais e virtuais
  - Memória, processador, disco, impressora, etc
  - Processos, sistema de arquivos, fila de impressão, etc.
- Os recursos reais implementados diretamente pelo hardware não são muitos
- O software transforma o hardware em um "equipamento virtual"



UFV

Quais as vantagens da multiprogramação?

- Razões para Multiprogramação
  - Respostas rápidas às requisições
  - Utilização eficiente de recursos
  - Tratar processos limitados por entrada e saída (*IO-Bound*) e por processador (*CPU-bound*)
    - De quem deve ser a prioridade?
      - *IO-bound*



UFV