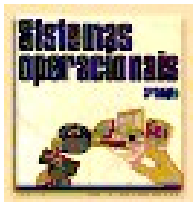
Processos e Threads

Ciclo 2 – AT3

Prof. Hermes Senger



Nota

O presente material foi elaborado com base no material didático do livro *Sistemas Operacionais, 3ª edição, de H.M.Deitel, P.J. Deitel, D.R. Choffnes, Editora Pearson Prentice Hall, São Paulo, 2005*, disponibilizado pela editora.

Objetivos

- **Este capítulo apresenta:**
  - O conceito de processo
  - Estados de processo e transições de estado
  - Blocos de controle de processos (PCBs)/descritores de processos
  - Chaveamento de processos/troca de contexto
  - Como interrupções habilitam o hardware a se comunicar com o software
  - Comunicação interprocessos (IPC)
  - Processos no Unix
  - Threads
  - Semelhanças e diferenças entre processos e threads
  - O ciclo de vida de um thread
  - O básico sobre threads POSIX, Linux, Windows XP e Java

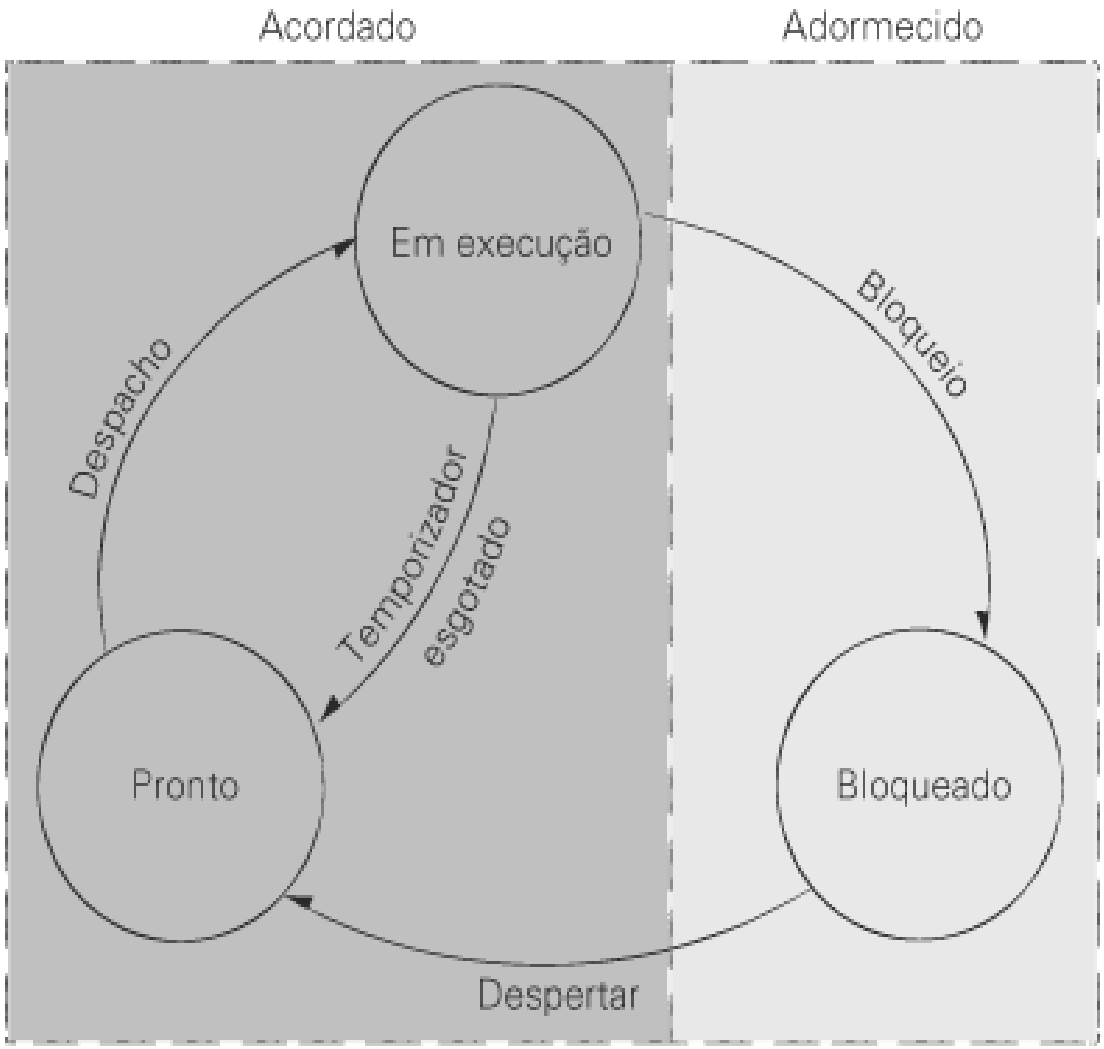
Introdução

- **Os computadores executam várias tarefas (programas, aplicações) ao mesmo tempo.**
  - Ex:
    - Navegar na Internet
    - Compilar um programa
    - Enviar um arquivo para a impressora
    - Reproduzir uma música
    - Etc
- **Processo**
  - Abstração
  - Unidade de execução
  - Processos transitam entre estados (ex: execução, bloqueado, etc).
  - Operações:
    - Criar, destruir, suspender, retomar e acordar.

Definição de processo

- **Um programa em execução**
  - Um processo tem seu próprio espaço de endereço, que consiste em:
    - Região de texto
      - → Armazena o código que o processador executa.
    - Região de dados
      - → Armazena variáveis e memória alocada dinamicamente.
    - Região de pilha
      - → Armazena instruções e variáveis locais para chamadas ativas ao procedimento.

Estados de processo e transições



Gerenciamento de processo

- **Os sistemas operacionais prestam alguns serviços essenciais aos processos. Ex:**
  - Criar processos
  - Destruir
  - Suspender
  - Retomar
  - Mudam a prioridade
  - Bloqueiam
  - Acordar (ativar)
  - Despachar
  - Possibilitar que processos se comuniquem entre si (IPC).

Estruturas de dados internas: Blocos de controle de processo (PCBs) /Descritores de processo

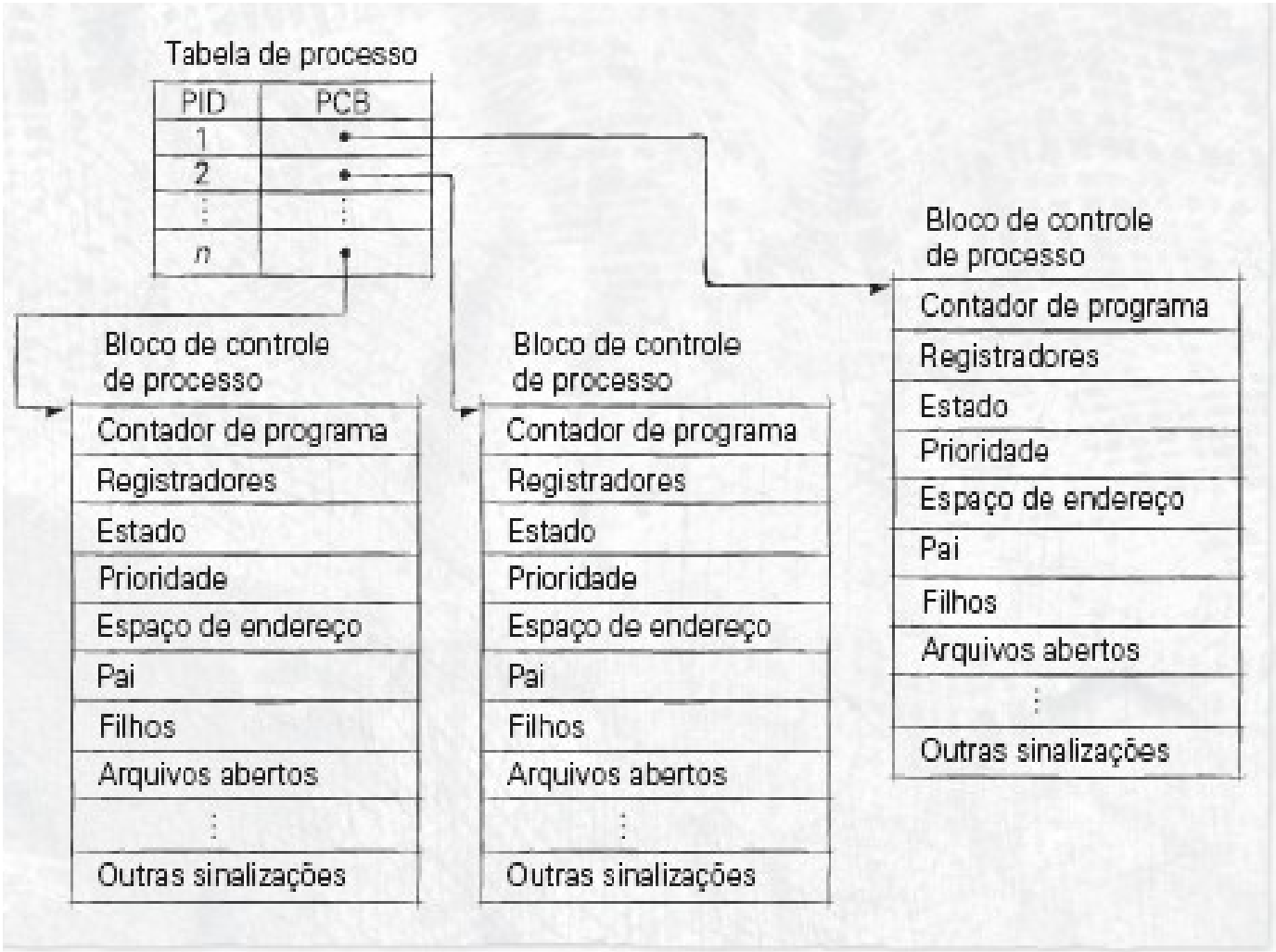
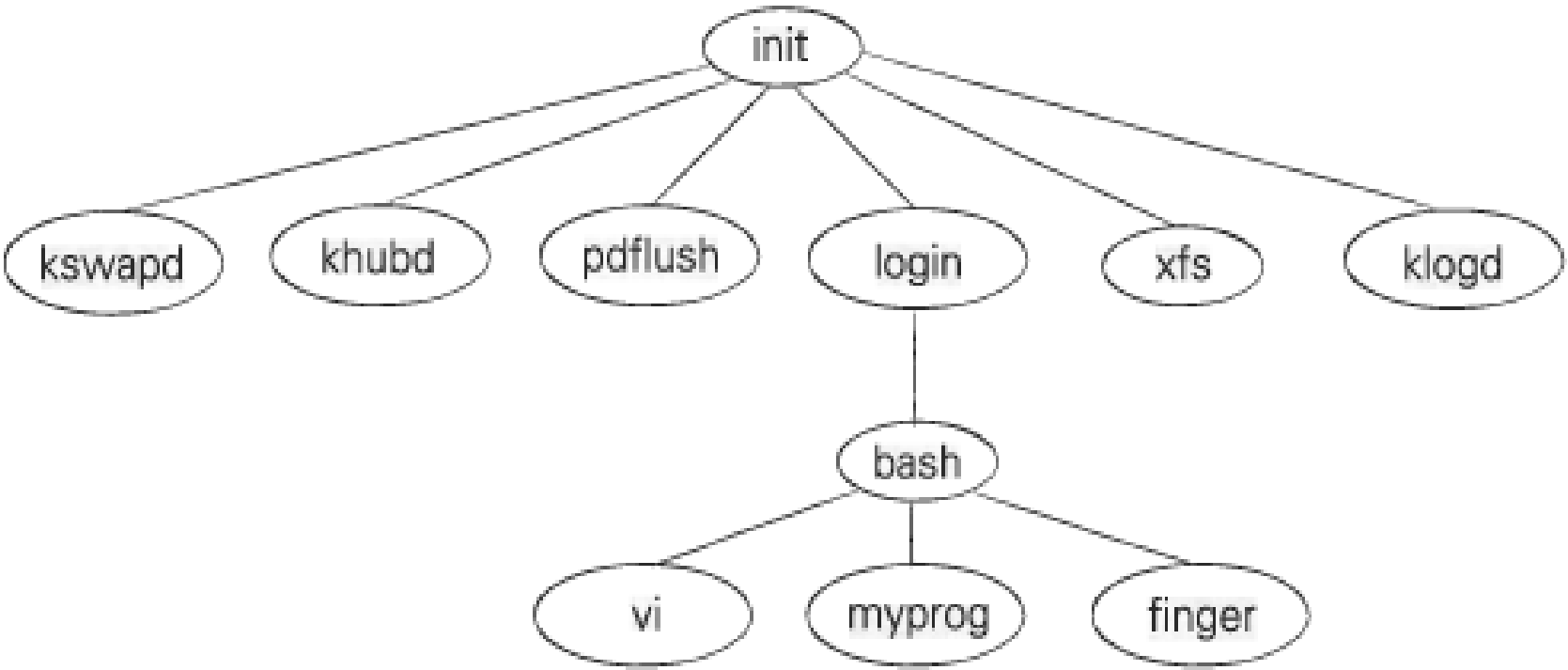


Figura 3.2 **Tabela de processos e blocos de controle de processo.**

Operações de processo

- **Um processo pode gerar um novo processo.**
  - O processo que criou a outro é chamado de **processo-pai**.
  - O processo criado é chamado de processo-filho.
  - Quando um processo-pai é desfeito, os sistemas operacionais em geral podem tomar dois tipos de atitude:
    - **Destruir** todos os processos-filho desse processo-pai.
    - **Permitir** que os processos-filho prossigam independentemente dos processos-pai.

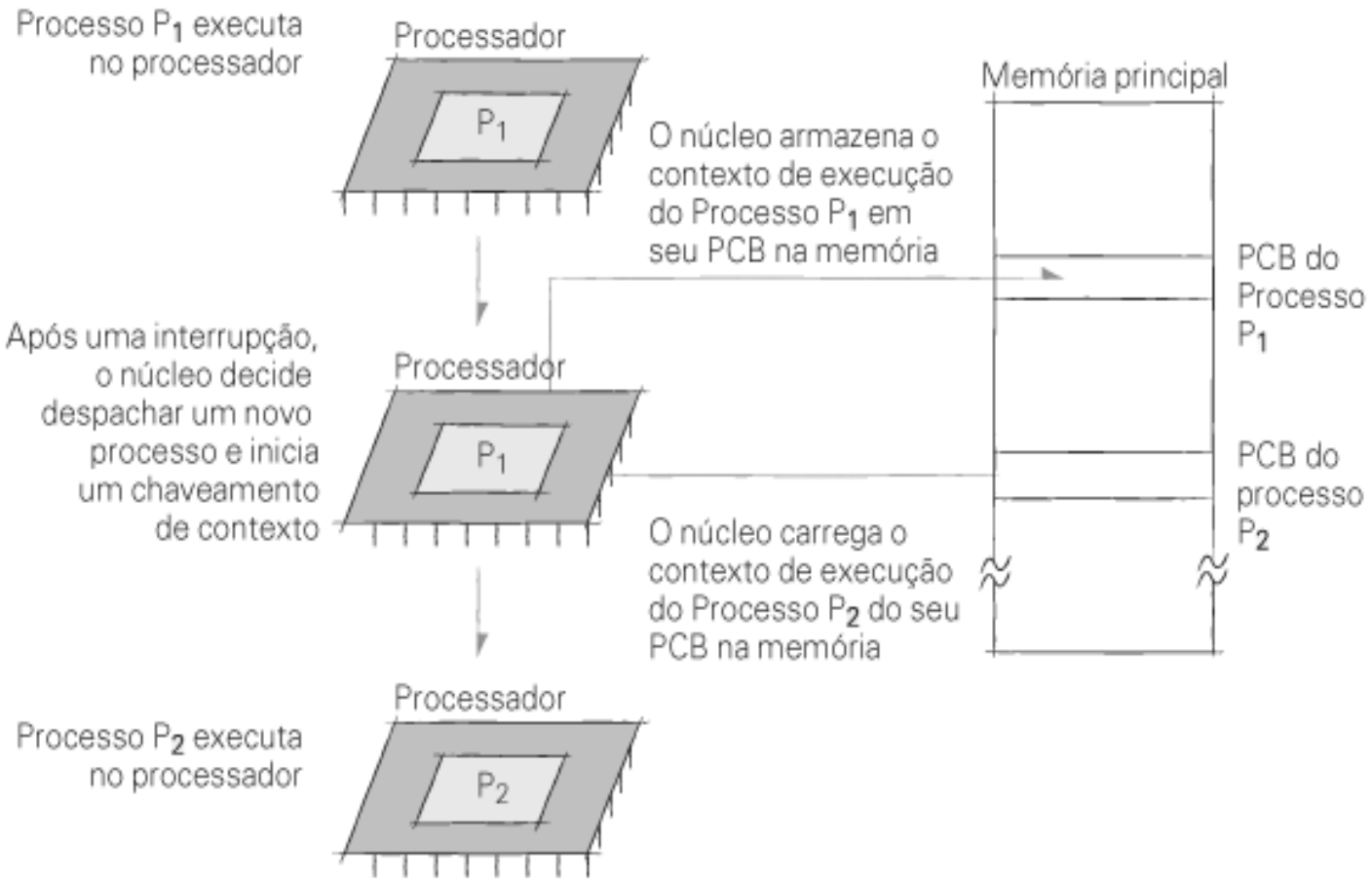
Ex: Hierarquia de processos no Linx



Chaveamento de contexto

- **Chaveamento de contexto**
  - **Interrompe** um processo *em execução* e **começa** a executar um processo previamente *pronto*.
  - **Salva o contexto** de execução do processo *em execução* no PCB (*program control block*) desse processo.
  - **Carrega o contexto** de execução anterior do processo *pronto* no PCB desse último.
  - Não deve ser perceptível aos processos.
  - O processador não pode realizar nenhuma computação "útil" durante o chaveamento.
    - Precisa ser muito **rápido** !
  - É executado no hardware por algumas arquiteturas.

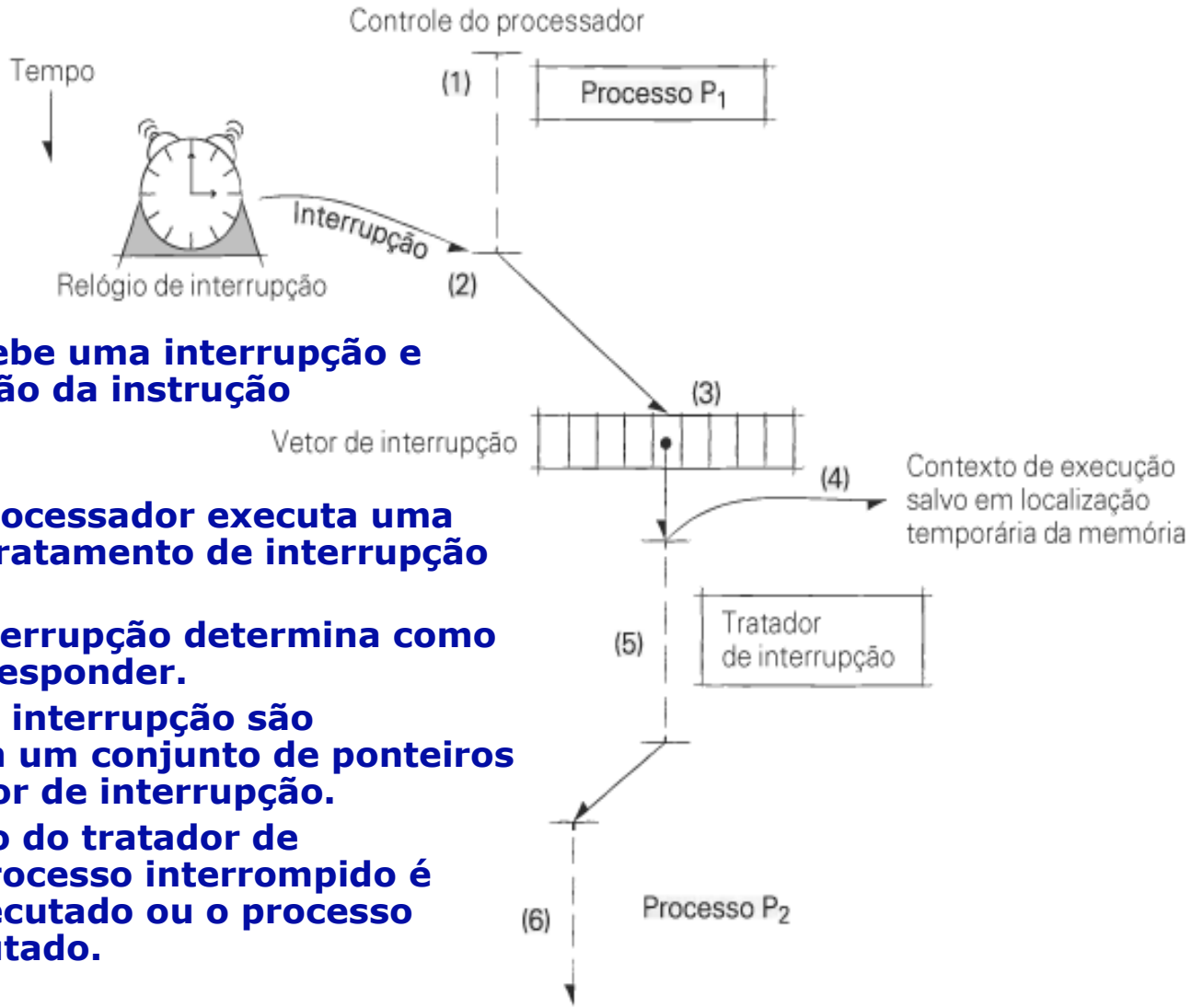
Chaveamento de contexto



Interrupções

- **Habilitam o software a responder a sinais do hardware.**
  - Podem ser iniciadas por um processo em execução.
    - A interrupção é denominada desvio (**trap**).
    - Por exemplo, quando um processo tenta realizar uma **ação ilegal**, como dividir por zero ou referir-se a uma memória protegida.
  - Podem ser iniciadas por algum **evento** que pode ou não estar relacionado ao processo em execução.
    - Por exemplo, quando uma tecla é pressionada no teclado ou o mouse é movido.
  - Exige pouco esforço do processador

Atendimento de interrupções



1. **Processador recebe uma interrupção e conclui a execução da instrução corrente.**

1. **Em seguida, o processador executa uma das funções de tratamento de interrupção do núcleo.**
2. **O tratador de interrupção determina como o sistema deve responder.**
3. **Os tratadores de interrupção são armazenados em um conjunto de ponteiros denominado vetor de interrupção.**
4. **Após a conclusão do tratador de interrupção, o processo interrompido é restaurado e executado ou o processo seguinte é executado.**

Classes de interrupção

- **As interrupções suportadas dependem da arquitetura do sistema.**
  - A especificação IA-32 distingue os dois tipos de sinal que um processador pode receber:
    - **Interrupções**
      - → Notificam o processador de que ocorreu um evento ou que o status de um dispositivo externo mudou.
      - → São geradas por dispositivos **externos** a um processador.
    - **Exceções**
      - → Indicam que ocorreu um **erro**, seja no hardware, seja em decorrência de uma instrução de software.
      - → São classificadas como falhas, desvios ou abortos.

Comunicação interprocessos

- **Muitos sistemas operacionais fornecem mecanismos para comunicações interprocessos (IPC).**
  - Os processos precisam se **comunicar** uns com os outros em ambientes de multiprogramação ou de rede.
    - Ex: um navegador Web pode recuperar dados de um servidor remoto.
  - É essencial para processos que precisam coordenar (**sincronizar**) atividades para alcançar uma meta comum.

Sinais

- **São interrupções de software que notificam o processo de que um evento ocorreu.**
  - Não permitem que os processos troquem dados
  - Processos podem **capturar**, **ignorar** ou **mascarar** um sinal.
    - Um processo captura um sinal especificando uma rotina que o sistema operacional chama quando libera o sinal.
    - Um processo ignora um sinal dependendo da ação-padrão do sistema operacional para tratá-lo.
    - Um processo mascara um sinal instruindo o sistema operacional a não liberar sinais desse tipo até que o processo bloqueie a máscara do sinal.

Estudo de caso: processos no Unix

- Todos os processos têm um conjunto de endereços de memória que é chamado de **espaço de endereço virtual**.
- O núcleo mantém o PCB de um processo em uma **região protegida** da memória que os processos usuários não podem acessar.
- Em sistemas Unix, um PCB armazena:
  - O conteúdo dos **registradores** dos processos.
  - O **identificador** do processo (PID).
  - O **contador de programa**.
  - A **pilha** do sistema.
- Todos os processos são relacionados na **tabela** de processos.
- Processos interagem com o sistema operacional por meio de chamadas ao sistema *(**system calls**).*
- As prioridades de processo são números inteiros entre –20 e 19 (inclusive).
  - Um valor numérico de prioridade mais baixo indica uma prioridade de escalonamento mais alta.
- O Unix fornece vários mecanismos que habilitam os processos a trocar dados, como é o caso dos pipes.

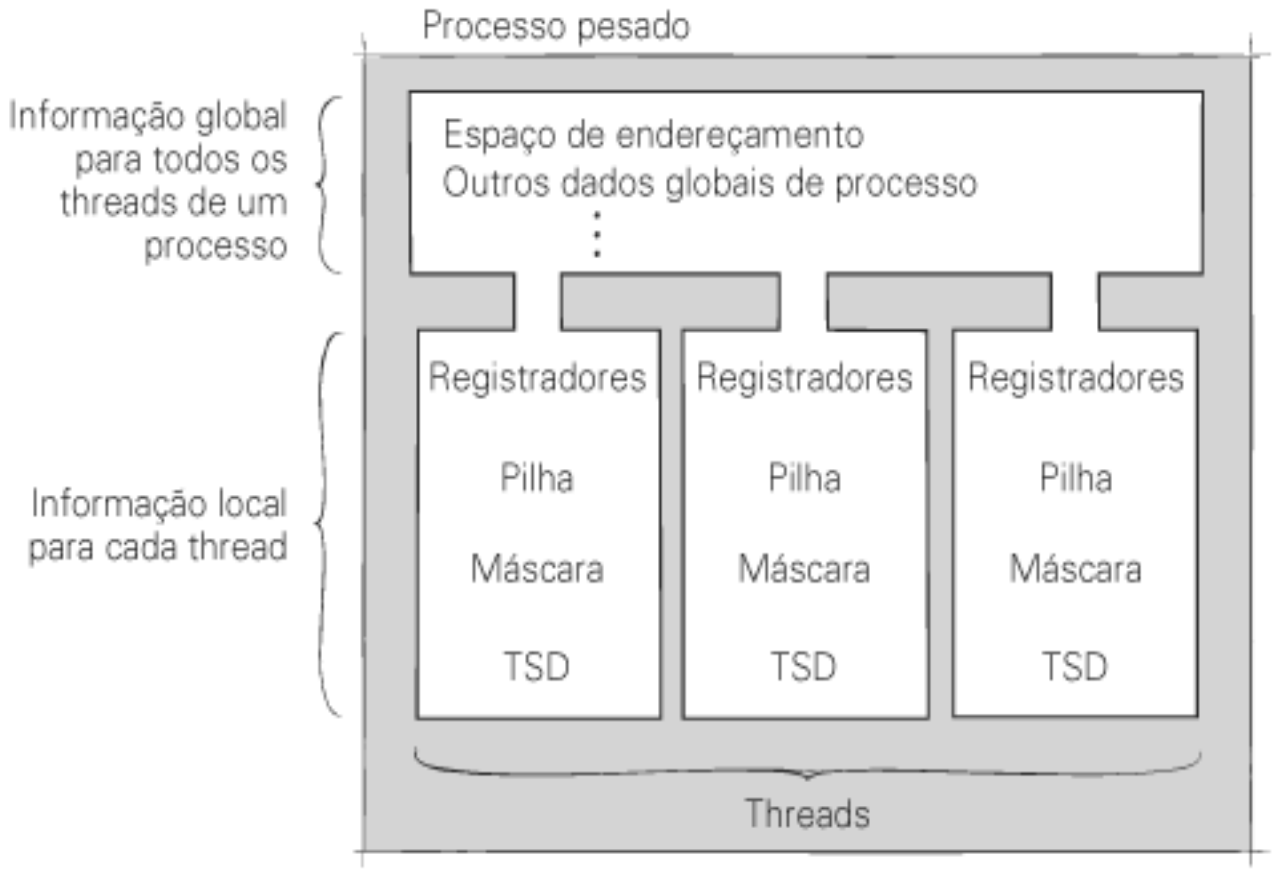
Chamadas ao Sistema no Unix

Chamada ao sistema	*Descrição*
fork	Gera um processo-filho e aloca àquele processo uma cópia dos recursos de seu pai.
exec	Carrega as instruções e dados de um processo no seu espaço de endereço em um arquivo.
wait	Faz com que o processo que está chamando fique bloqueado até que seu processo-filho termine.
signal	Permite que um processo especifique um tratador de sinal para um tipo de sinal particular.
exit	Termina o processo que está chamando.
nice	Modifica a prioridade de escalonamento de um processo.

Threads

- **Várias linguagens modernas disponibilizaram primitivas de concorrência para o programador de aplicações.**
  - **Ex.: Java, C#, Visual C++ .NET, Visual Basic .NET e Python**
  - O programador **escreve** as aplicações contêm threads de execução.
  - Cada thread pode ser uma parte de um programa que pode **executar concorrentemente** com outros threads.
- **Thread**
  - É às vezes chamado de **processo leve** (LWP).
  - Existem threads de instrução ou threads de controle.
  - **Compartilham** espaço de endereço e outras informações do processo
  - Registradores, pilha, máscaras de sinal e outros dados são específicos a cada thread.
- **Os threads devem ser gerenciados pelo sistema operacional ou pela aplicação de usuário.**
  - Exemplos: threads Win32, C-threads, Pthreads.

Thread vs. Processo



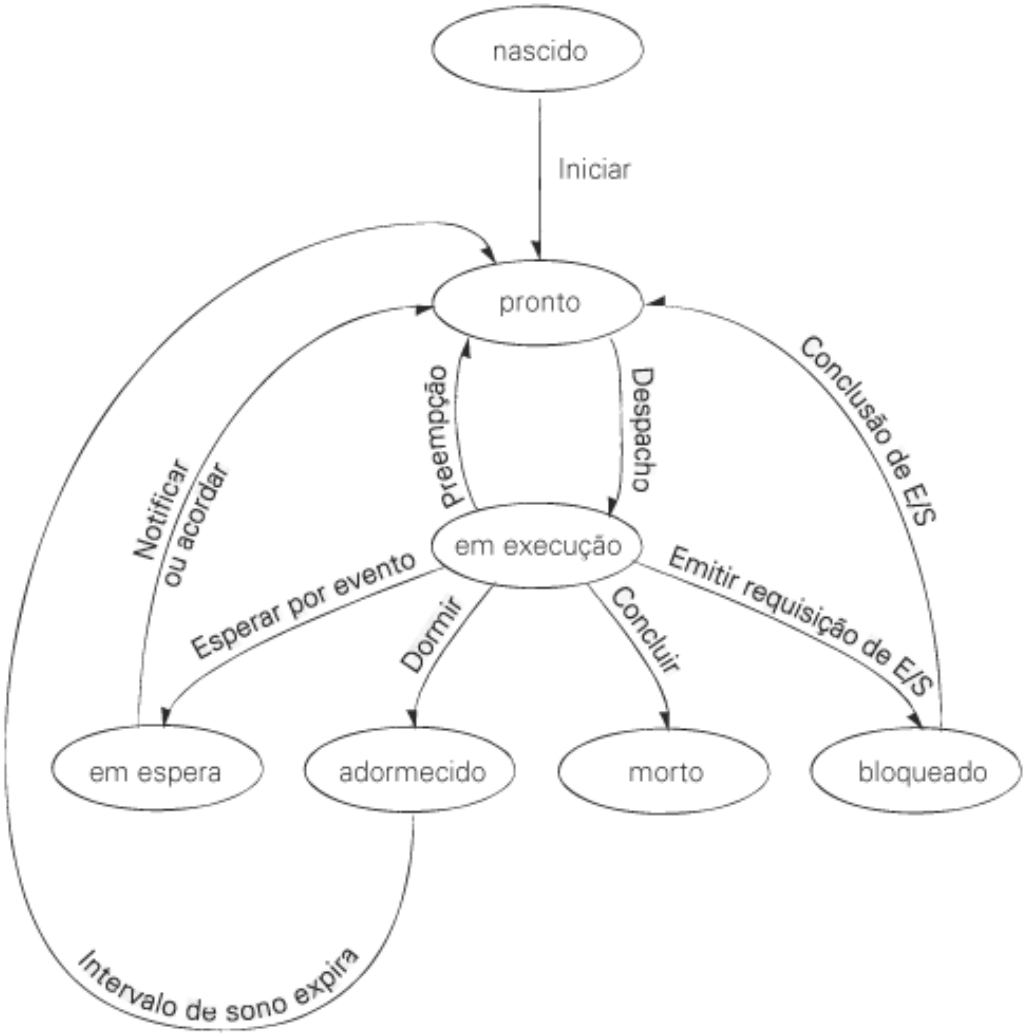
Motivação

- **Atualmente threads são muito utilizados em diversas áreas**
- **Por que criar threads?**
  - Ao projeto de software
    - Maior simplicidade para exprimir tarefas inerentemente paralelas.
  - Ao desempenho
    - Maior escalonamento para sistemas com múltiplos processadores.
  - À cooperação
    - O custo operacional do espaço de endereço compartilhado é menor que o da IPC.

Motivação para criação de threads

- Todo thread transita entre uma série de estados de thread distintos.
- Os threads e os processos têm muitas **operações em comum** (por exemplo, criar, sair, retomar e suspender).
- A criação de thread não requer que o sistema operacional inicialize recursos compartilhados entre os processos-pais e os respectivos threads.
  - Isso **reduz o esforço de criação e término** de threads, em comparação à criação e ao término de processo.
- A troca de contexto também pode ser muito mais rápida (~ 200 vezes em alguns casos)

Estados de thread: ciclo de vida



Modelos de thread

- **Três são os modelos de thread mais conhecidos:**
  - Threads de usuário
  - Threads de núcleo
  - Uma combinação de ambos

POSIX e Pthreads

- **Os threads que usam a API de thread POSIX são chamados de Pthreads.**
  - A especificação POSIX determina que os registradores do processador, a pilha e a máscara de sinal sejam mantidos individualmente para cada thread.
  - A especificação POSIX especifica como os sistemas operacionais devem emitir sinais a Pthreads, além de especificar diversos modos de cancelamento de thread.

Threads Linux

- **O Linux aloca o mesmo tipo de descritor para processos e threads (tarefas).**
- **Para criar tarefas-filha, o Linux usa a chamada `fork`, baseada no Unix.**
- **Para habilitar os threads, o Linux oferece uma versão modificada, denominada `clone`.**
  - `Clone` aceita argumentos que determinam os recursos que devem ser compartilhados com a tarefa-filha.
  - Na prática, programas manipulam threads através de uma API padronizada, chamada pthreads (POSIX threads). Função pthread para criação de thread é mapeada para chamada *clone*.

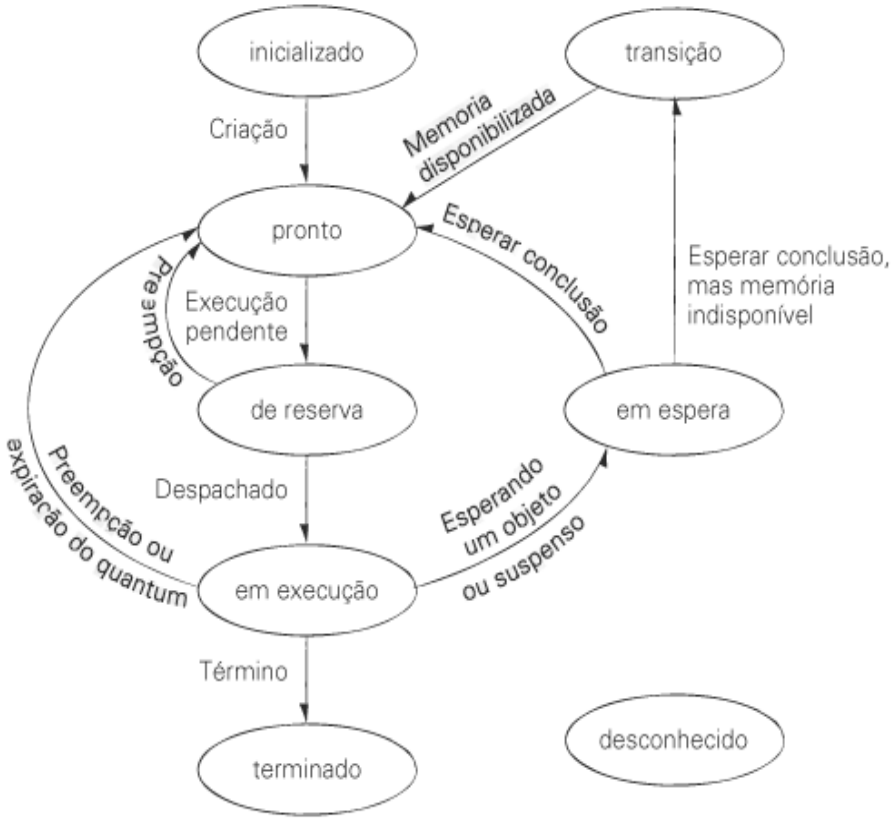
Threads do Windows XP

- **Os threads do Windows XP podem criar fibras.**
  - A execução da fibra é escalonada pelo thread que a cria, e não pelo escalonador.
- O Windows XP fornece a cada processo um reservatório de threads que consiste em inúmeros threads operários, que são threads de núcleo que executam funções especificadas pelos threads de usuário.

Threads do Windows XP

- **Threads**
  - São na verdade a unidade de execução despachada por um processador.
  - Executam uma parte do código do processo, no contexto do processo, usando os recursos do processo.
  - O contexto de execução contém:
    - Pilha de tempo de execução
    - Estado dos registradores da máquina
    - Diversos atributos

Threads do Windows XP



Threads Java

- **A linguagem Java permite que o programador de aplicações crie `threads` portáveis para várias plataformas de computação.**
- **`Threads`**
  - Criados pela classe `Thread`.
  - Executam códigos especificados em um método run de um objeto `Runnable`.
- **A linguagem Java suporta operações como nomeação, ativação e união de threads.**

FIM