CidadesInteligentes.com.br

ProfessorLima.Com

e-Learning

ProfessorLima.Com.Br

Research

Devel-IT.ORG

Instagram

Currículo Lattes

LinkedIn

www.professorlima.com

Aula Anterior

Gerenciamento de Processos - Parte II
- Usando linguagem de Programação
- Simulação dos Processos

Aula de Hoje

Gerenciamento de Processos - Parte III
- Threads
- Modelos de Criação de Threads

Próxima Aula

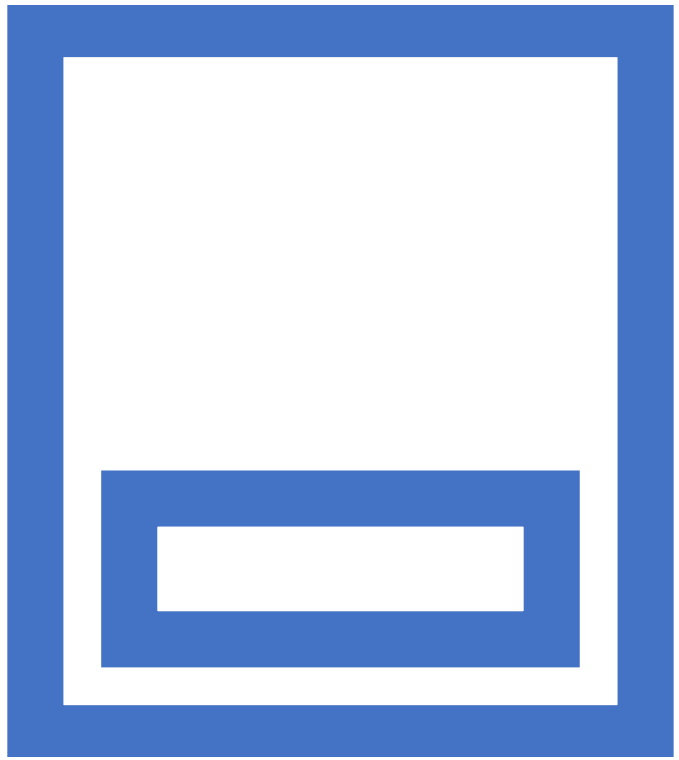
Gerenciamento de Processos
- Escalonamento (FIFO e Prioridade)
- Troca de Contexto

Cronograma

Criação de Processos

Principais eventos que levam à criação de processos

1. Início do sistema
2. Execução de chamada ao sistema de criação de processos
3. Solicitação do usuário para criar um novo processo
4. Início de um job em lote



Término de Processos

Condições que levam ao término de processos

1. Saída normal (voluntária)
2. Saída por erro (voluntária)
3. Erro fatal (involuntário)
4. Cancelamento por um outro processo (involuntário)

Hierarquias de Processos

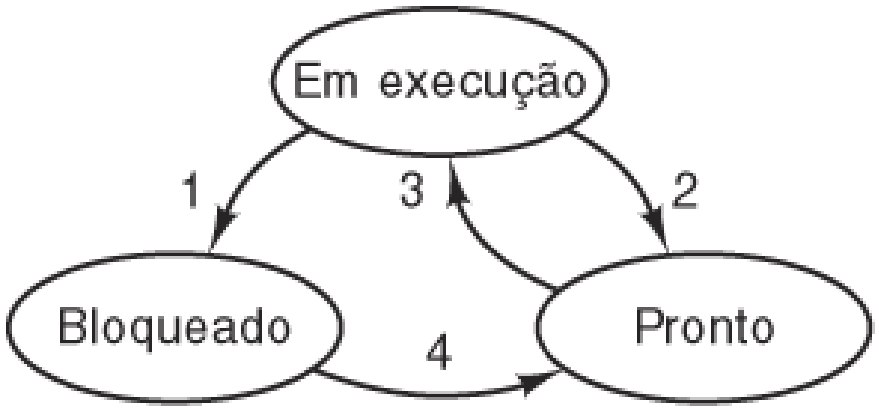
Pai cria um processo filho, processo filho pode criar seu próprio processo

Formam uma hierarquia

- UNIX chama isso de “grupo de processos”

Windows não possui o conceito de hierarquia de processos

- Todos os processos são criados iguais



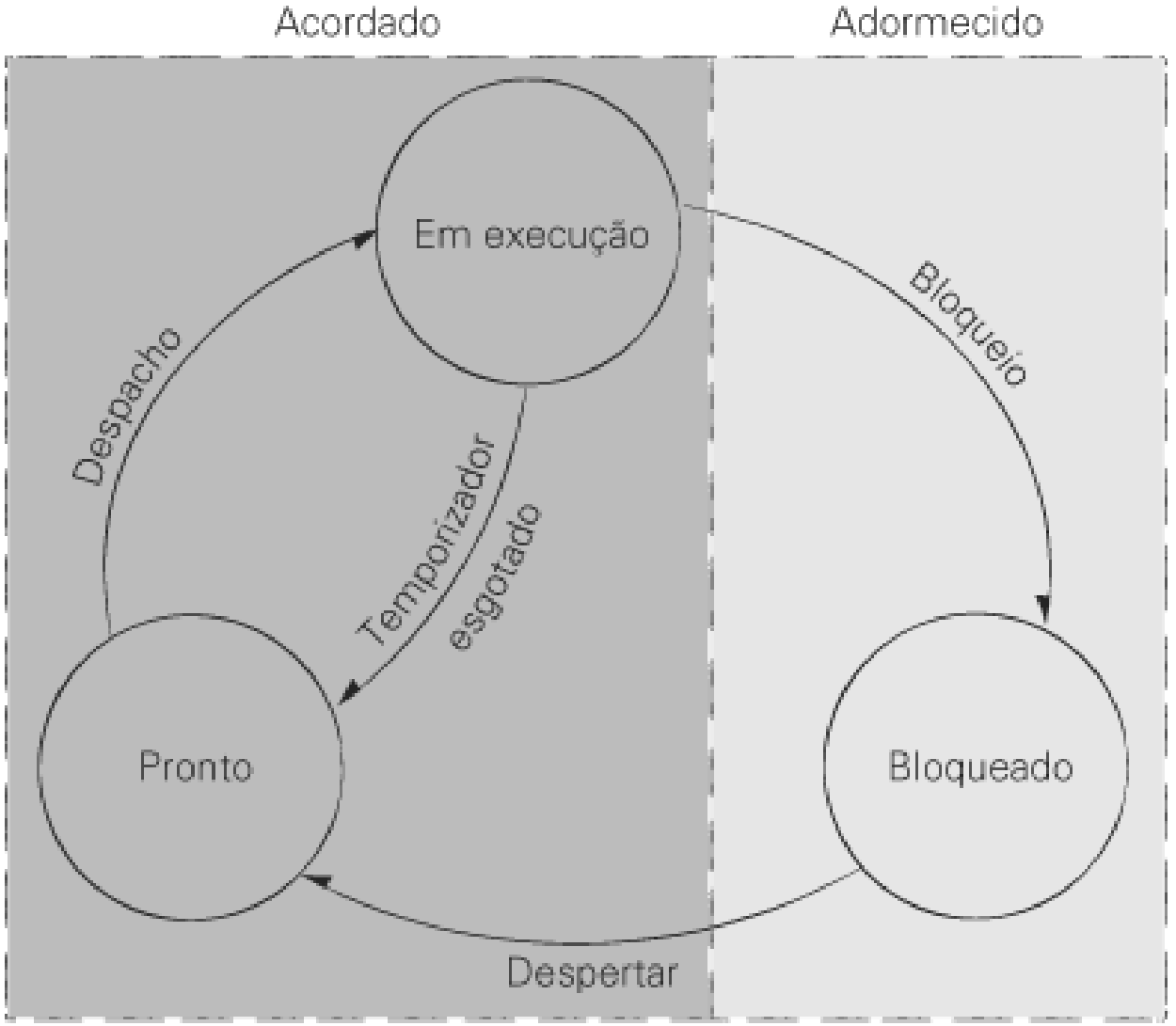
1. O processo bloqueia aguardando uma entrada
2. O escalonador seleciona outro processo
3. O escalonador seleciona esse processo
4. A entrada torna-se disponível

- Possíveis estados de processos
  - em execução
  - bloqueado
  - pronto
- Mostradas as transições entre os estados

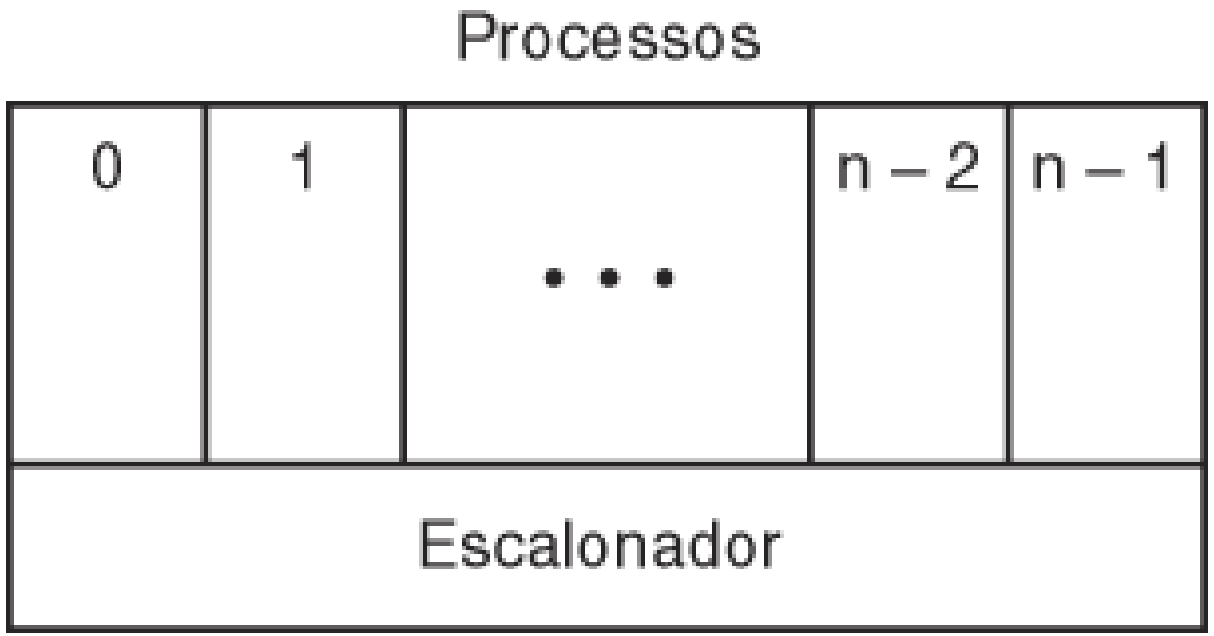
Estados de Processos

Estados de processo e estados de transição

Transições de estado de processo.



Estados de Processos



- Camada mais inferior de um SO estruturado por processos
  - trata interrupções, escalonamento
- Acima daquela camada estão os processos sequenciais

Implementação de Processos

Gerenciamento de processos	Gerenciamento de memória	Gerenciamento de arquivos
Registradores	Ponteiro para o segmento de código	Diretório-raiz
Contador de programa	Ponteiro para o segmento de dados	Diretório de trabalho
Palavra de estado do programa	Ponteiro para o segmento de pilha	Descritores de arquivos
Ponteiro de pilha		Identificador (ID) do usuário
Estado do processo		Identificador (ID) do grupo
Prioridade		
Parâmetros de escalonamento		
Identificador (ID) do processo		
Processo pai		
Grupo do processo		
Sinais		
Momento em que o processo iniciou		
Tempo usado da CPU		
Tempo de CPU do filho		
Momento do próximo alarme		

Campos da entrada de uma tabela de processos

Como vimos, cada processo conta com uma estrutura de controle razoavelmente sofisticada.

Nos casos onde se deseja realizar duas ou mais tarefas simultaneamente, o solução trivial a disposição do programador é dividir as tarefas a serem realizadas em dois ou mais processos.

Isto implica na criação de manutenção de duas estruturas de controle distintas para tais processos, onerando o sistema, e complicando também o compartilhamento de recursos, dados serem processos distintos.

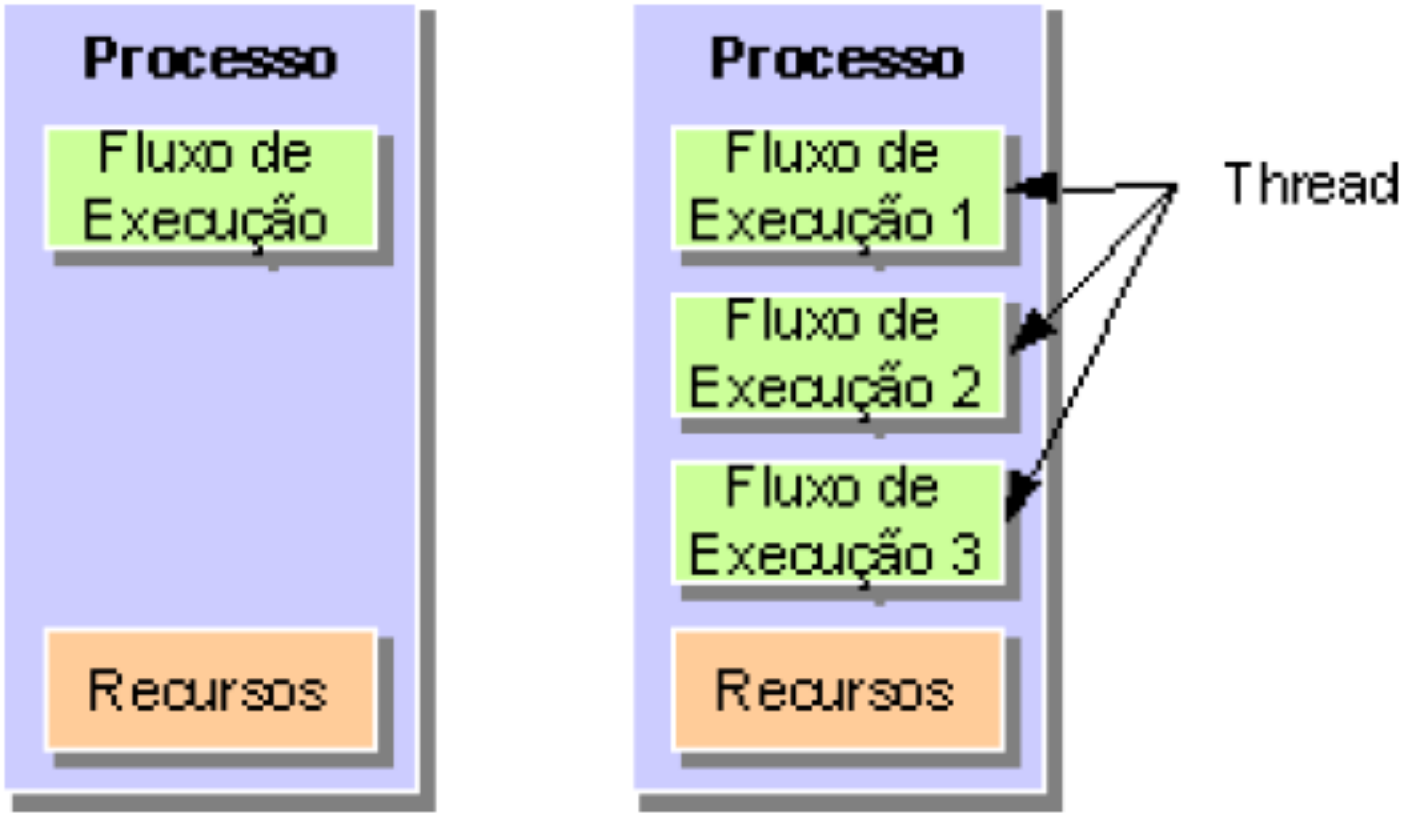
Threads

Threads

- Uma alternativa ao uso de processos comuns é o emprego de threads.
- Enquanto cada processo tem um único fluxo de execução, ou seja, só recebe a atenção do processador de forma individual, quando um processo é divido em threads, cada uma das threads componentes recebe a atenção do processador como um processo comum.
- No entanto, só existe uma estrutura de controle de processo para tal grupo, o espaço de memória é o mesmo e todos os recursos associados ao processo podem ser compartilhados de maneira bastante mais simples entre as suas threads componentes.

Threads

- Desta forma, as threads podem ser entendidas como fluxos independentes de execução pertencentes a um mesmo processo, que requerem menos recursos de controle por parte do sistema operacional.
- Assim, as threads são o que consideramos processos leves (lightweight processes).



Threads

- Sistemas computacionais que oferecem suporte para as threads são usualmente conhecidos como sistemas *multithreading*, os sistemas *multithreading* podem suportar as threads segundo dois modelos diferentes:

- **User threads:** As threads de usuário são aquelas oferecidas através de bibliotecas específicas e adicionais ao sistema operacional, ou seja, são implementadas acima do kernel (núcleo do sistema) utilizando um modelo de controle que pode ser distinto do sistema operacional, pois não são nativas neste sistema.

- **Kernel threads:** As threads do sistema são aquelas suportadas diretamente pelo sistema operacional e, portanto, nativas.

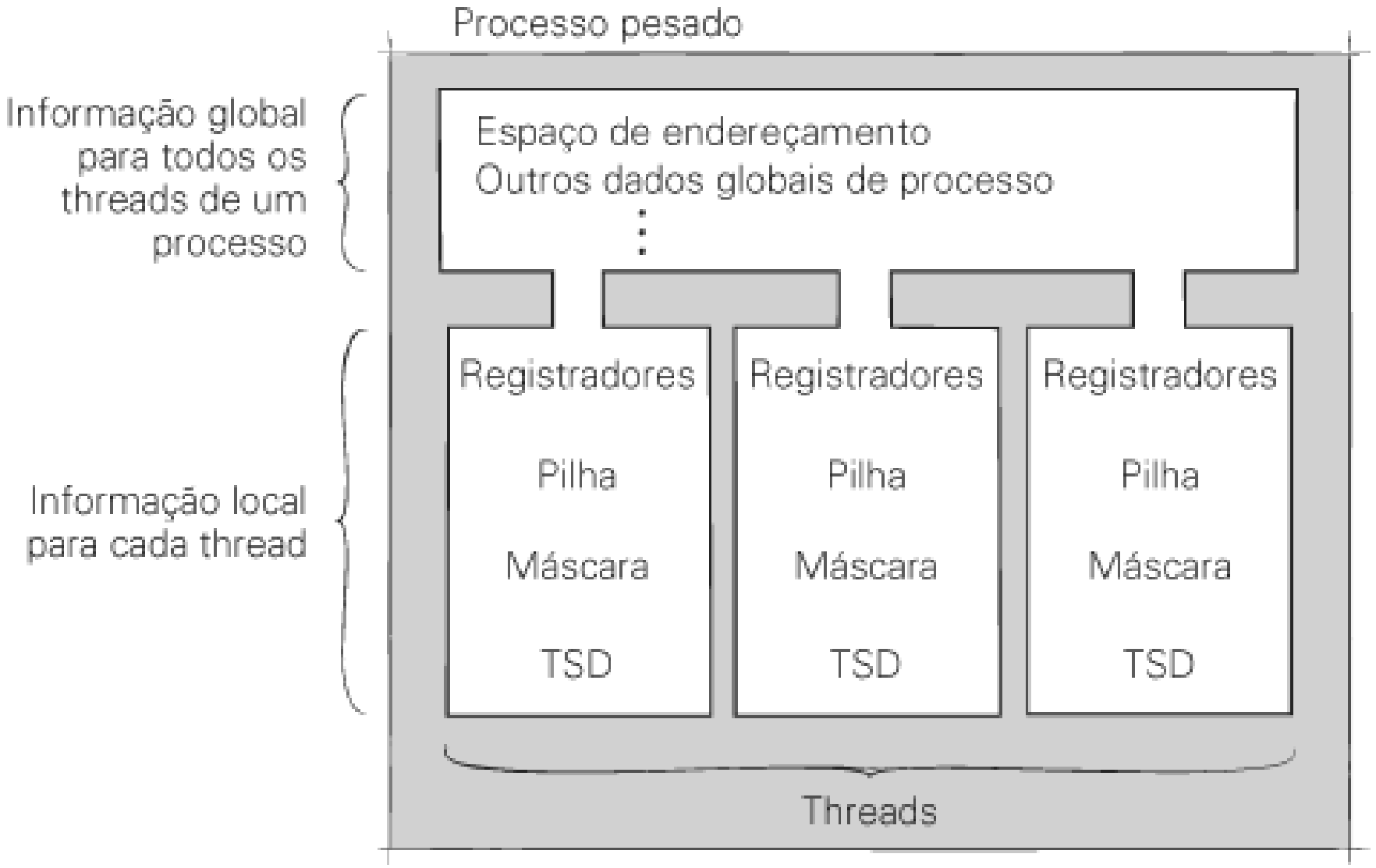
Definição de thread

- **Thread**
  - É às vezes chamado de processo leve (LWP).
  - Existem threads de instrução ou threads de controle.
  - Os threads compartilham espaço de endereço e outras informações globais com seu próprio processo.
  - Registradores, pilha, máscaras de sinal e outros dados específicos de thread são nativos a cada thread.

- **Os threads devem ser gerenciados pelo sistema operacional ou pela aplicação de usuário.**
- **Exemplos: threads Win32, C-threads, Pthreads.**

Definição de thread

Relação entre thread e processo.



Motivação na criação de threads

- **Os threads tornaram-se proeminentes por causa de tendências subseqüentes em relação:**
  - 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 na criação de threads

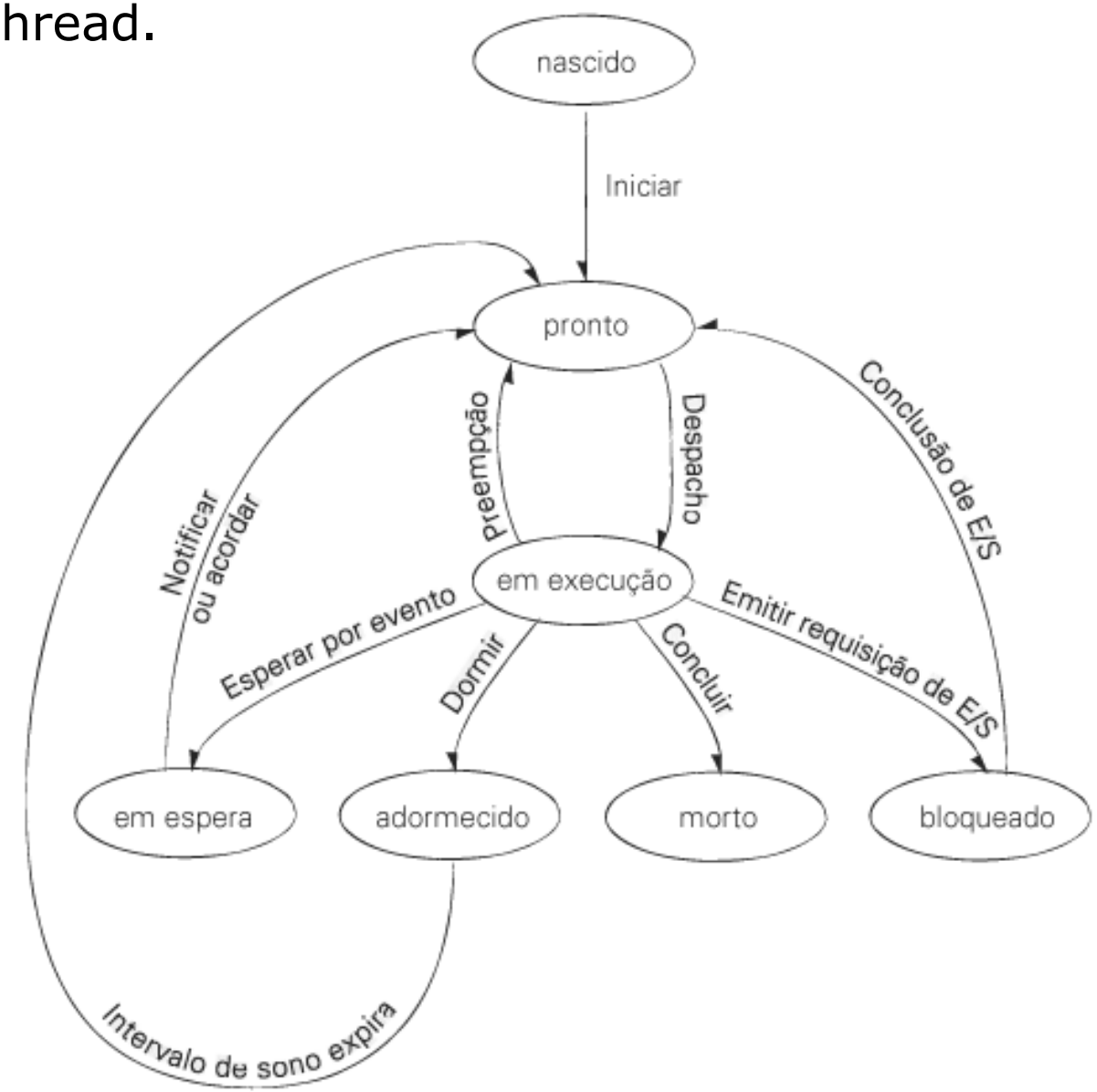
- **Todo thread transita entre uma série de estados de thread distintos.**
- **Os threads e os processos têm muitas operações em comum (por exemplo, criar, sair, retomar e suspender).**
- **A criação de thread não requer que o sistema operacional inicialize recursos compartilhados entre os processos-pai e os respectivos threads.**
  - Isso reduz o esforço de criação e término de threads, em comparação à criação e ao término de processo.

Estados de thread: ciclo de vida de um thread

- **Estados de thread**
  - Estado *nascido*
  - Estado *pronto* (estado *executável*)
  - Estado *em execução*
  - Estado *morto*
  - Estado *bloqueado*
  - Estado de *espera*
  - Estado *adormecido*
    - O período de sono especifica por quanto tempo um thread ficará adormecido.

Estados de thread: ciclo de vida de um thread

Ciclo de vida do thread.



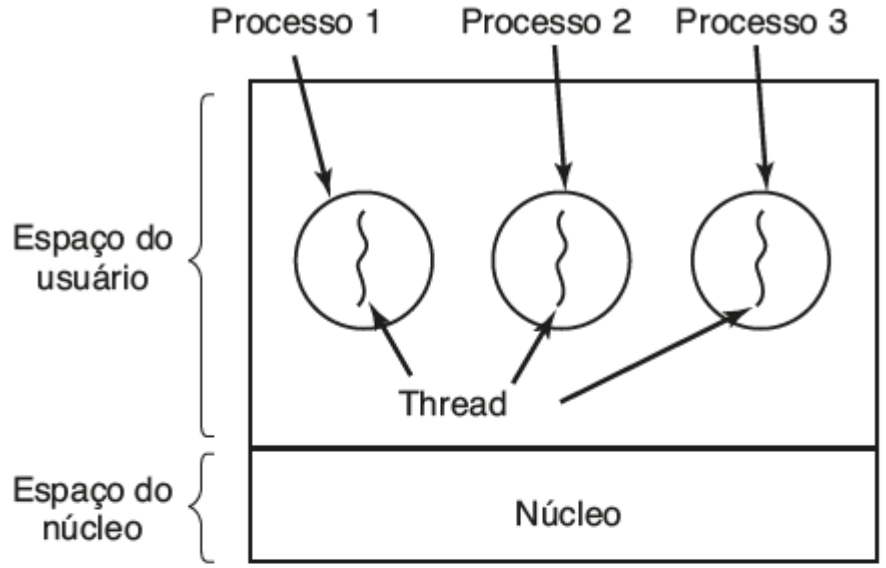
Modelos de thread

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

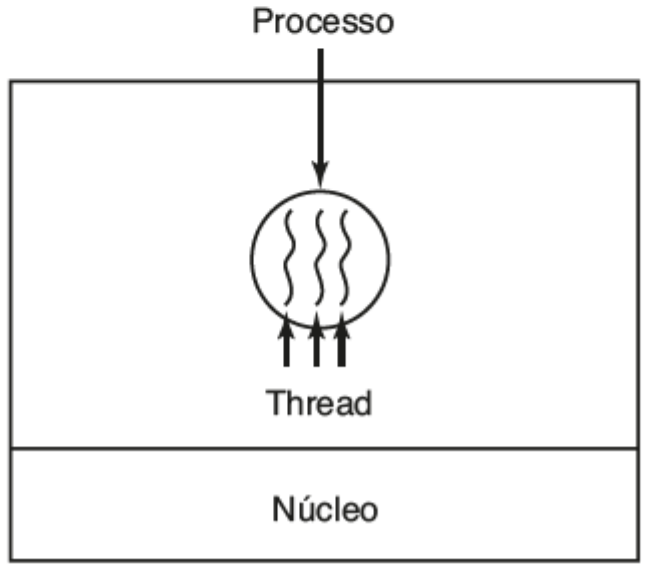
O modelo de thread clássico

- (A) vemos três processos tradicionais. Cada processo tem seu próprio espaço de endereçamento e um único thread de controle. Cada um deles opera em um espaço de endereçamento diferente.

- (B) vemos um único processo com três threads de controle. Embora em ambos os casos tenhamos três threads. Todos os três compartilham o mesmo espaço de endereçamento.



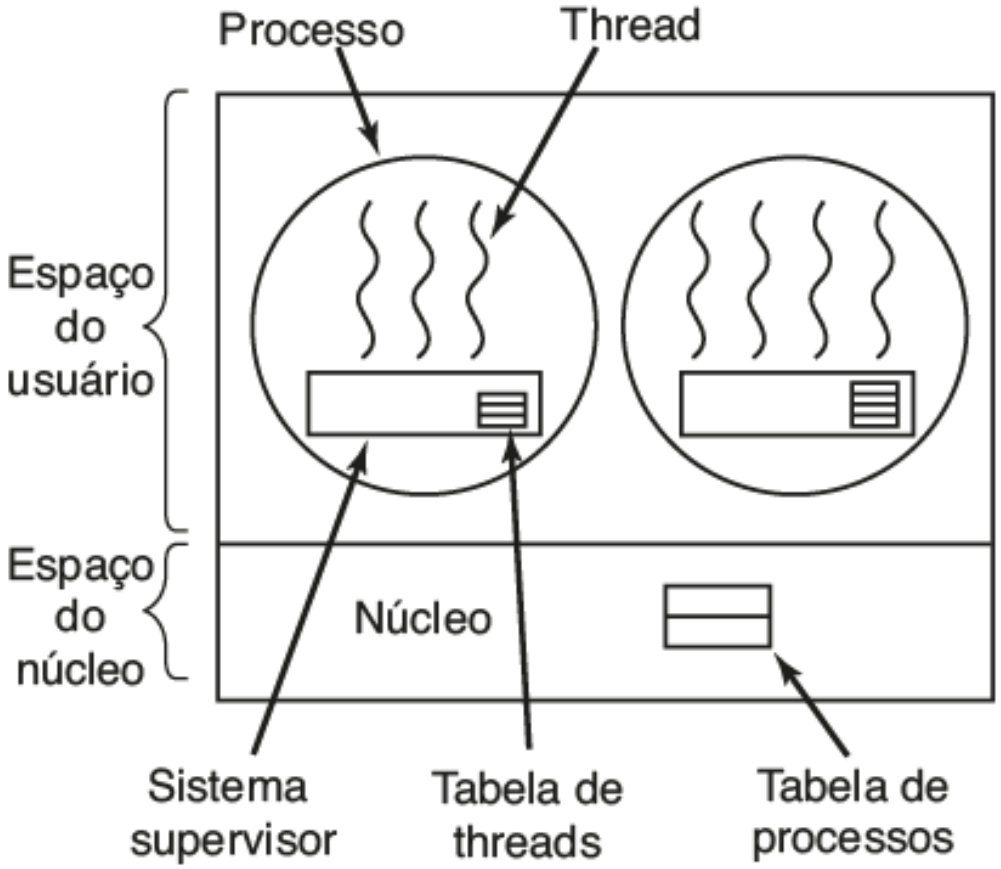
(a)



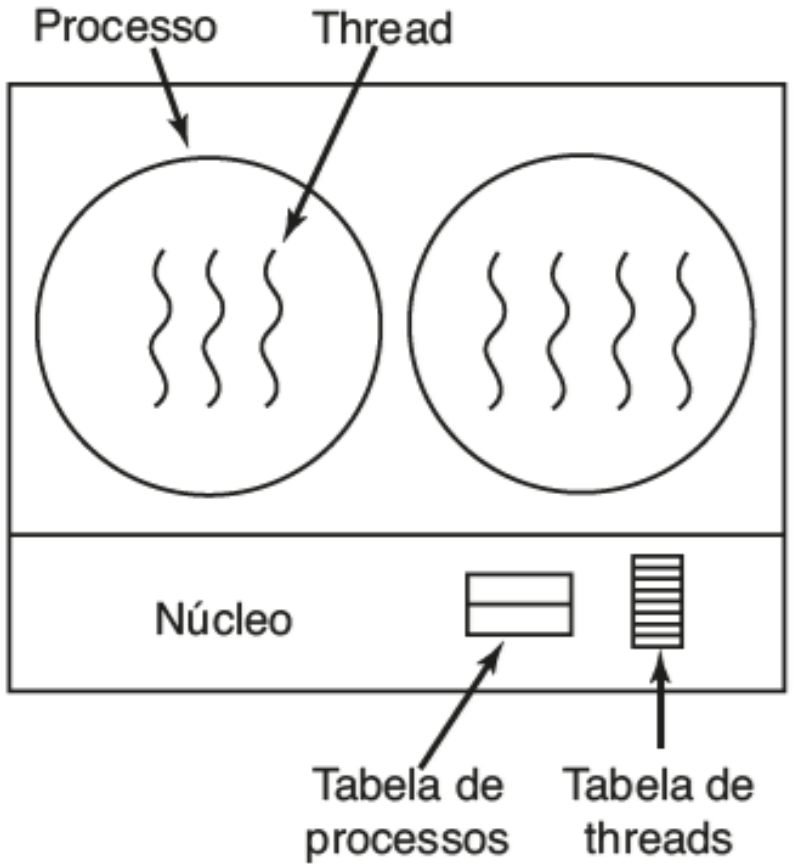
(b)

Implementando threads no espaço e no núcleo

- Há dois lugares principais para implementar threads: no **espaço do usuário (figura a)** e no **núcleo (figura b)**. A escolha é um pouco controversa, e uma implementação híbrida também é possível.



(a)



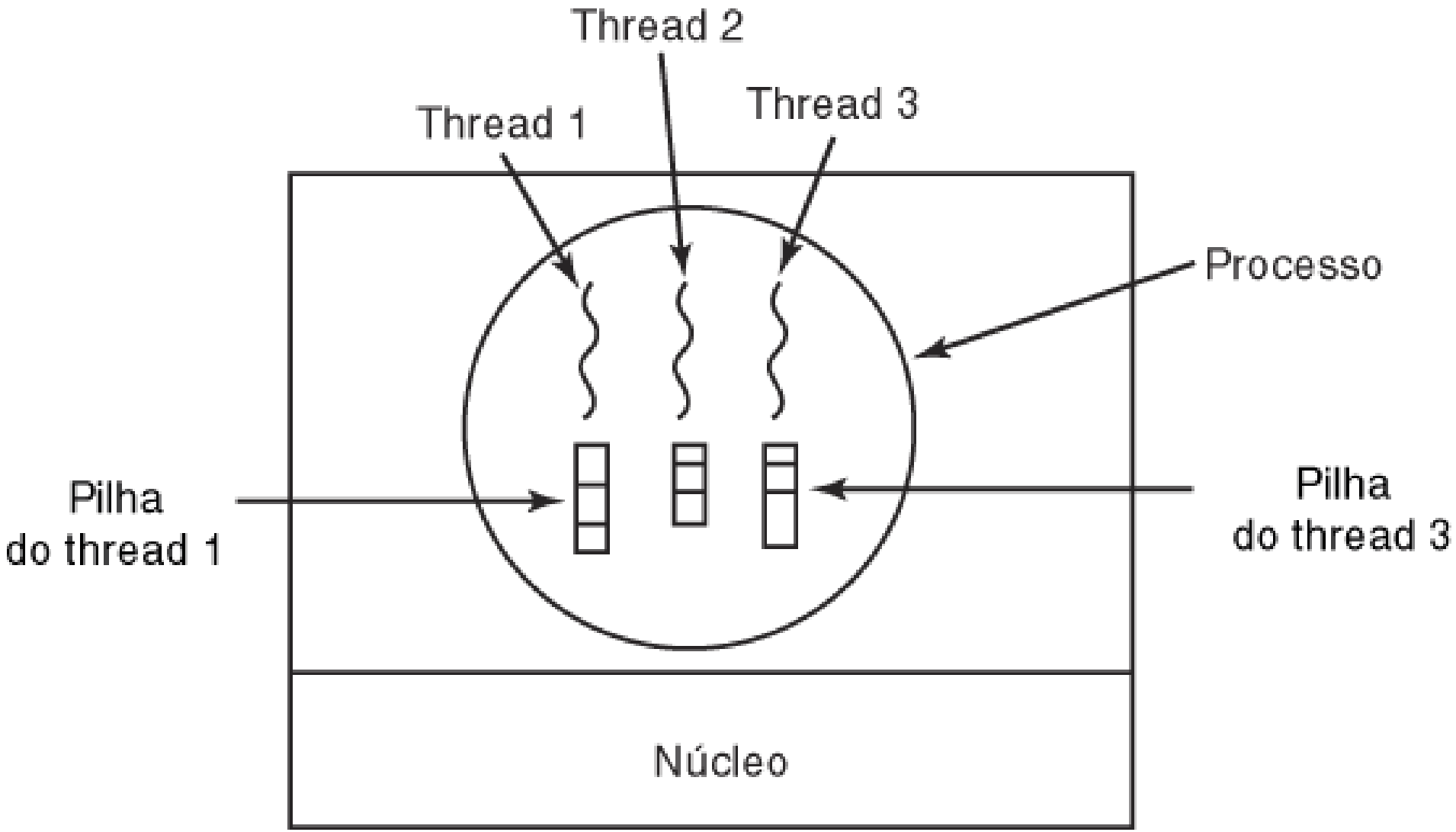
(b)

O Modelo de Thread

Itens por processo	Itens por thread
Espaço de endereçamento	Contador de programa
Variáveis globais	Registradores
Arquivos abertos	Pilha
Processos filhos	Estado
Alarmes pendentes	
Sinais e tratadores de sinais	
Informação de contabilidade	

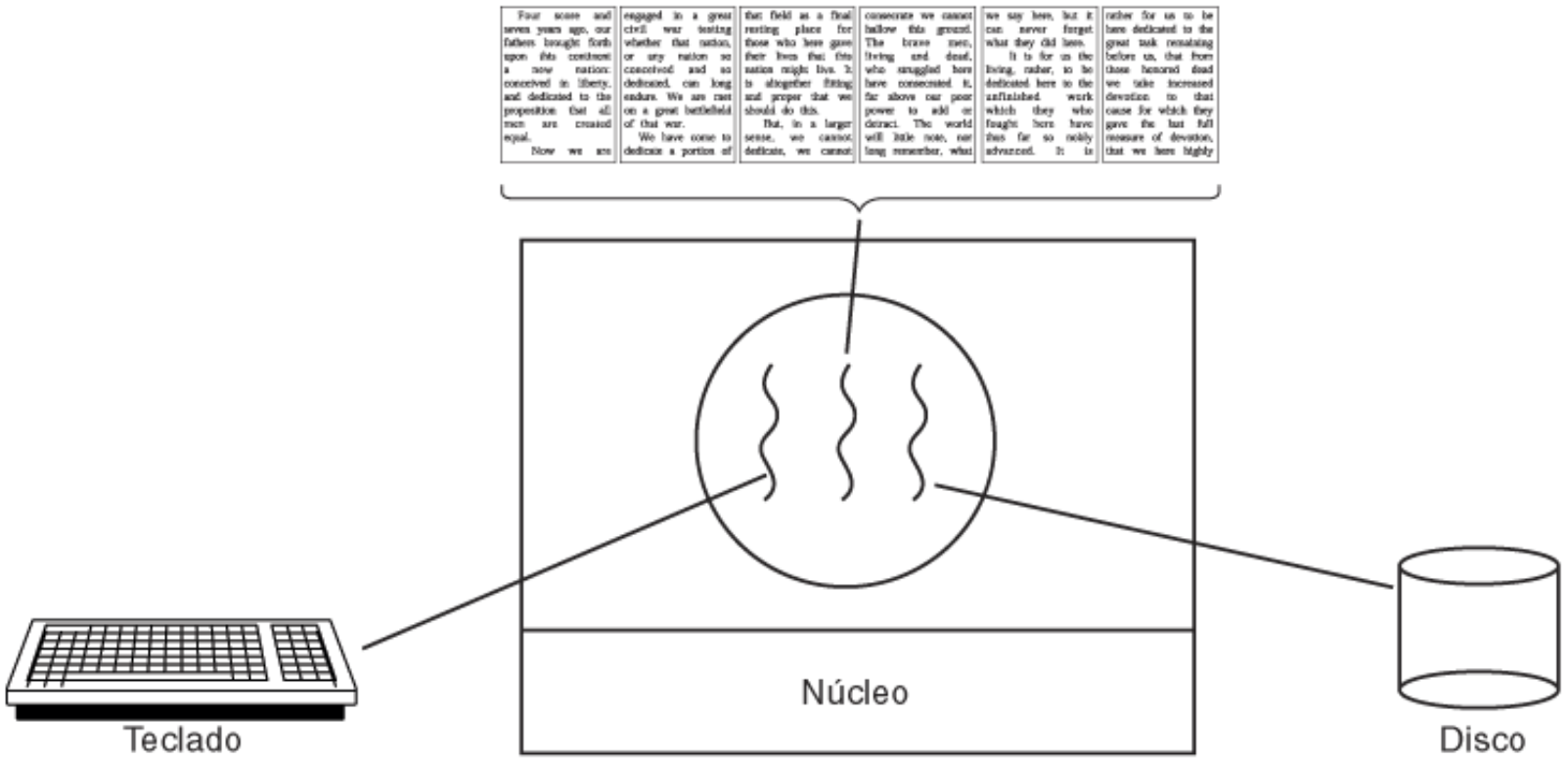
- Items compartilhados por todos os threads em um processo
- Itens privativos de cada thread

O Modelo de Thread



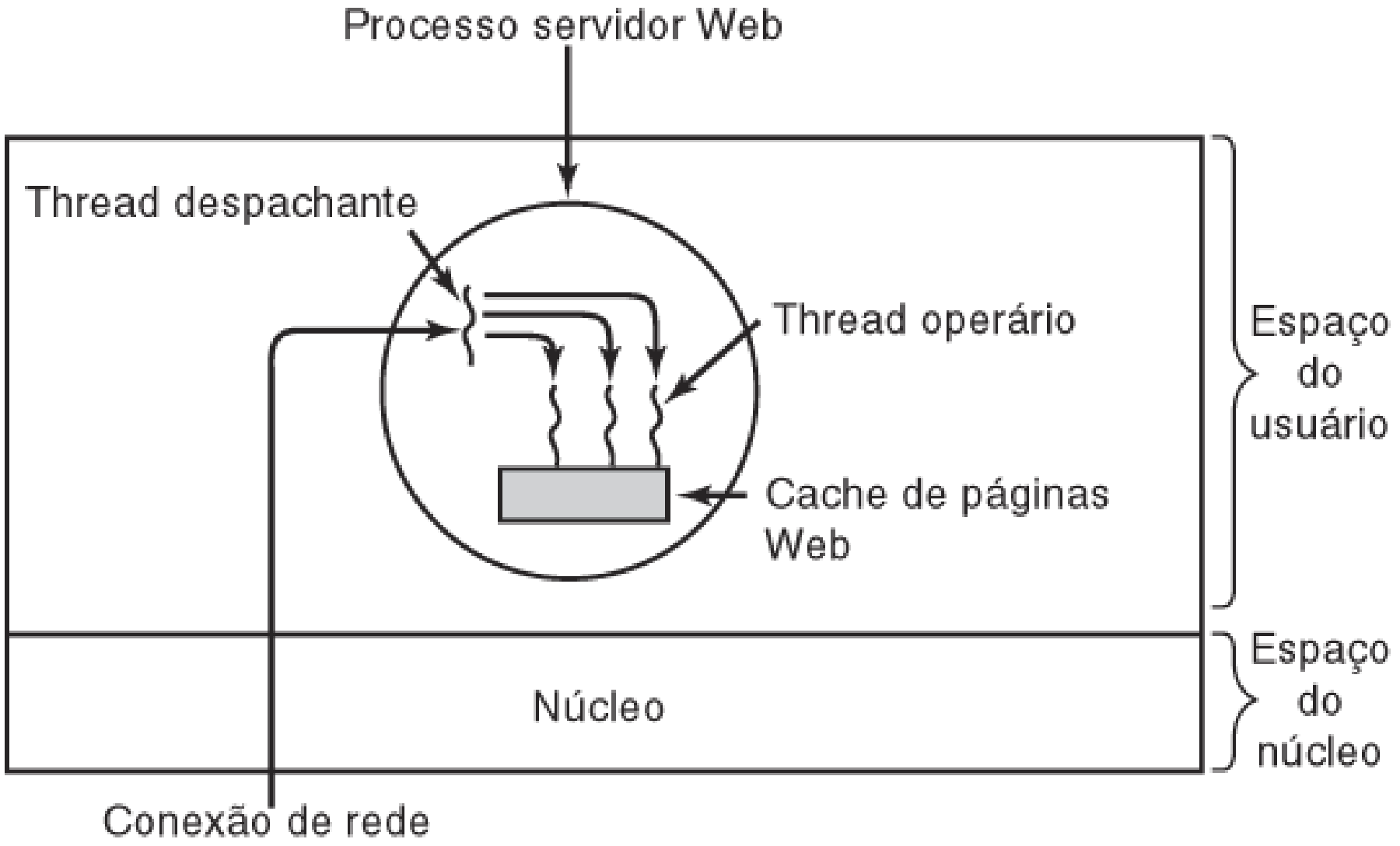
Cada thread tem sua própria pilha

Uso de Thread



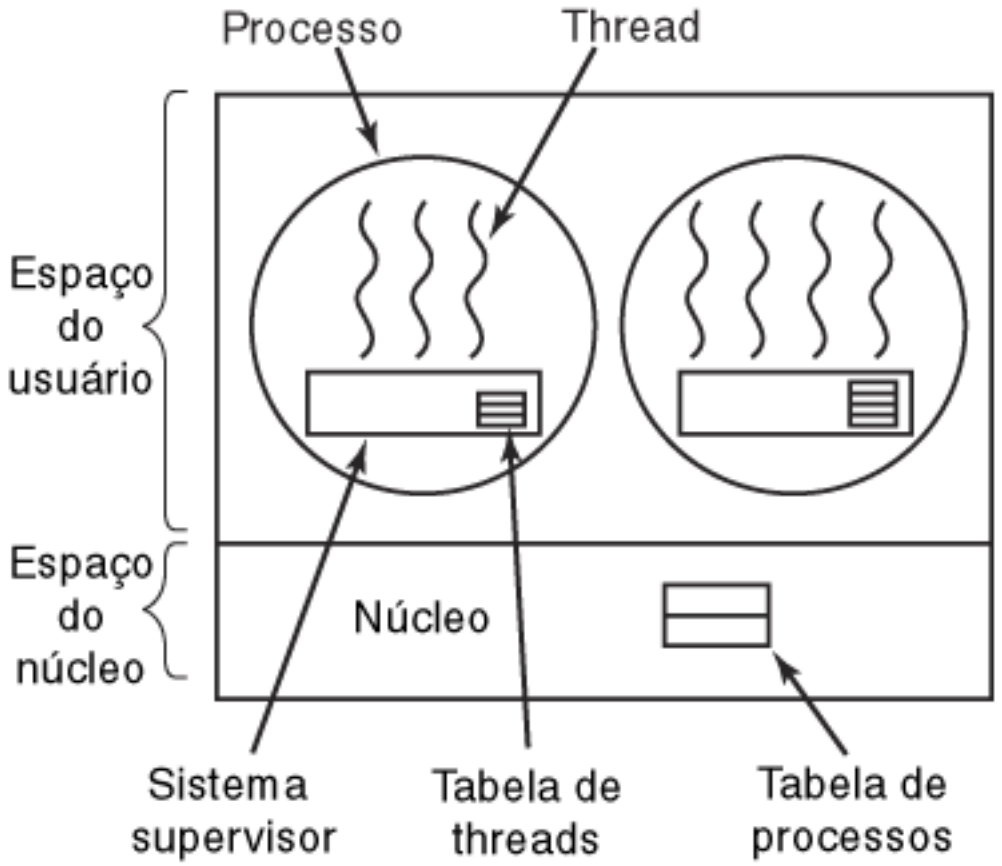
Um processador de texto com três threads

Uso de Thread



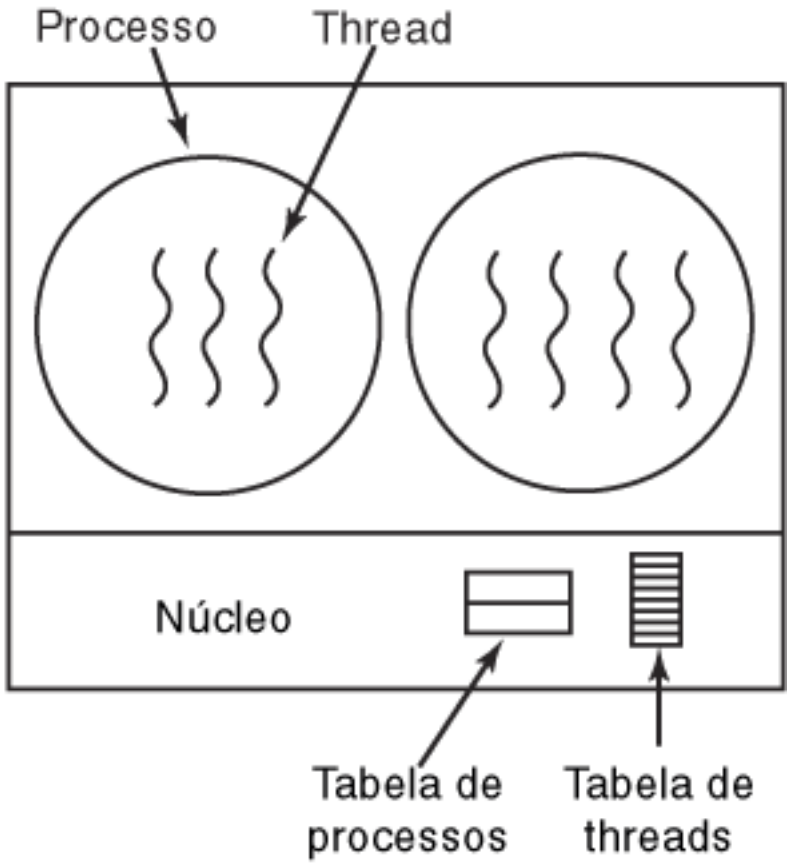
Um servidor web com múltiplos threads

Implementação de Threads de Usuário



Um pacote de threads de usuário

Implementação de Threads de Núcleo



Um pacote de threads gerenciado pelo núcleo

Estudo de caso: processos no Unix

- **Processos no Unix**
  - Todos os processos têm um conjunto de endereços de memória denominado 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 do programa.
    - A pilha do sistema.
  - Todos os processos são relacionados na tabela de processos.

Estudo de caso: processos no Unix

- **Processos no Unix (continuação)**
  - Todos os processos interagem com o sistema operacional por meio de chamadas ao sistema.
  - Um processo pode gerar um processo-filho usando a chamada ao sistema fork, que cria uma cópia do processo-pai.
    - O processo-filho também recebe uma cópia dos recursos do processo-pai.
  - 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.

Estudo de caso: processos no Unix

Chamadas ao sistema do 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.

Estudo de caso do Java Multithread, Parte I: introdução a 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.**

Estudo de caso do Java Multithread, Parte I: introdução a threads Java

Threads Java sendo criados, ativados, dormindo e imprimindo (Parte 1 de 5).

```
1   // Fig. 4.9: ThreadTester.java
2   // Múltiplos threads imprimindo em intervalos diferentes.
3
4   public class ThreadTester {
5
6       public static void main( String [] args )
7       {
8           // criar e dar nome a cada thread
9           PrintThread thread1 = new PrintThread( "thread1" );
10          PrintThread thread2 = new PrintThread( "thread2" );
11          PrintThread thread3 = new PrintThread( "thread3" );
12
13          System.err.println( "Ativando threads" );
14
15          thread1.start(); // ative thread1; coloque-o no estado pronto
16          thread2.start(); // ative thread2; coloque-o no estado pronto
17          thread3.start(); // ative thread3; coloqu-o no estado pronto
18
19          System.err.println( "Threads ativados, main termina\n" );
20
21      } // termine main
22
23  } // temine classe ThreadTester
24
```

Estudo de caso do Java Multithread, Parte I: introdução a threads Java

Threads Java sendo criados, ativados, dormindo e imprimindo (Parte 2 de 5).

```
25    // classe PrintThread controla execução do thread
26    classe PrintThread estende Thread {
27       private int sleepTime;
28
29       // designe nome ao thread chamando construtor superclasse
30       public PrintThread( String name )
31       {
32          super( name );
33
34          // escolha tempo de sono aleatório entre 0 e 5 segundos
35          sleepTime = ( int ) ( Math.random( ) * 5001 );
36       } // termine construtor PrintThread
37
38       // método run é o código a ser executado pelo novo thread
39       public void run( )
40       {
41          // ponha thread para dormir por período de tempo sleepTime
42          try {
43             System.err.println( getName() + "vai dormir por" +
44                sleepTime + " milissegundos" );
45
46          Thread.sleep( sleepTime );
```

• Estudo de caso do Java Multithread, Parte I: introdução a threads Java

Threads Java sendo criados, ativados, dormindo e imprimindo (Parte 3 de 5).

```
47              } // termine try
48
49              // se thread interrompido durante o sono, imprima cópia da pilha ( stack trace )
50              catch ( InterruptedException exception ) {
51                  exception.printStackTrace();
52              } // termine catch
53
54              // imprima nome do thread
55              System.err.println( getName() + "terminou de dormir" );
56
57          } // termine método run
58
59      } // termine classe PrintThread
```

Estudo de caso do Java Multithread, Parte I: introdução a threads Java

Threads Java sendo criados, ativados, dormindo e imprimindo (Parte 4 de 5).

Amostra de resultado 1:

```
Ativando threads
Threads ativados, main termina
thread1 vai dormir por 1217 milissegundos
thread2 vai dormir por 3989 milissegundos
thread3 vai dormir por 662 milissegundos
thread3 terminou de dormir
thread1 terminou de dormir
thread2 terminou de dormir
```

Estudo de caso do Java Multithread, Parte I: introdução a threads Java

Threads Java sendo criados, ativados, dormindo e imprimindo (Parte 5 de 5).

Amostra de resultado 2:

```
Ativando threads
thread1 vai dormir por 314 milissegundos
thread2 vai dormir por 1990 milissegundos
Threads ativados, main termina
thread3 vai dormir por 3016 milissegundos
thread1 terminou de dormir
thread2 terminou de dormir
thread3 terminou de dormir
```

devel-it.slack.com
E-mail: lima@clips.com.br

"Eu atribuo o meu sucesso a isso: eu nunca dei ou tomei qualquer desculpa."

Florence Nightingale

Thank you