Capítulo

4

Conceitos de thread

Sumário

PEARSON Prentice Hall

Objetivos

- **Este capítulo apresenta:**
  - A motivação na criação de threads.
  - Semelhanças e diferenças entre processos e threads.
  - Os vários níveis de suporte a threads.
  - O ciclo de vida de um thread.
  - Sinalização e cancelamento de threads.
  - O básico sobre threads POSIX, Linux, Windows e Java.

PEARSON Prentice Hall

4.1 Introdução

- **As linguagens de propósito geral, como Java, C#, Visual C++ .NET, Visual Basic .NET e Python, disponibilizaram primitivas de concorrência para o programador de aplicações.**
- **Multithreading**
  - O programador especifica que as aplicações contêm threads de execução.
  - Cada thread designa uma parte de um programa que pode executar concorrentemente com outros threads.

PEARSON Prentice Hall

4.1 Introdução

Processos Vs Threads

Processo:

- Um programa em execução
  - Uma unidade de escalonamento
  - Um fluxo de execução
- Um conjunto de recursos gerenciados pelo S.O.
  - Registradores (PC, SP, …)
  - Memória
  - Descritores de arquivos
  - Etc…
- A troca de contexto é uma operação pesada
  - Deve acontecer cada vez que há decisão de escalonamento

4.1 Introdução

Processos Vs Threads

Thread:

- Linha de execução dentro de um processo
  - Sequência independente de comandos de um programa.
    - Fluxo de Execução
- Compartilham recursos do processo
- Ideia simples: associar mais de um fluxo de execução a um processo:
  - Compartilhamento de recursos do processo (PCB - Process Control Block)
  - O PCB deve incluir uma lista de threads!

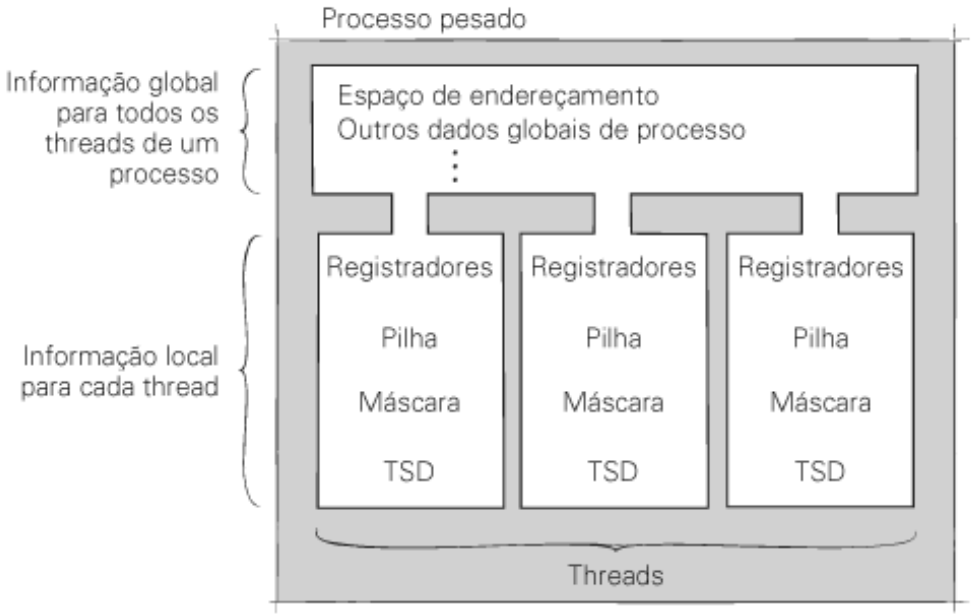
4.2 Definição de thread

- **Thread**
  - É às vezes chamado de processo leve (**LWP - light-weight process**).
  - 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.**

PEARSON Prentice Hall

4.2 Definição de thread

Figura 4.1 Relação entre thread e processo.



4.3 Motivação na criação de threads

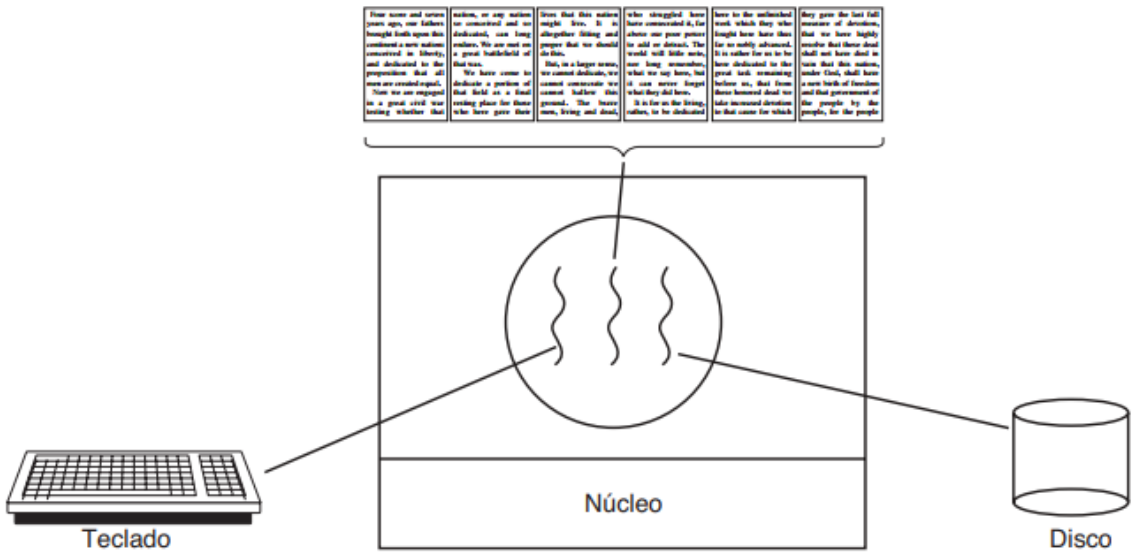
Vantagens:

- São processos “leves”
- Troca de contexto mais rápida;
- São mais rápidas de criar e destruir que processos(algumas vezes 100 vezes mais rápidas);
- Diminui o tempo de resposta do sistema;
- Maior facilidade para mesclar threads I/O-bound com threads CPU-bound.
- Usa eficientemente as arquiteturas multi-processadas/multicores.(paralelismo real).

4.3 Motivação na criação de threads

1) Processador de texto
2) Processos separados não funcionam – o documento tem que estar compartilhado
3) Threads para: parágrafo, fonte, correção, mudança de linha, etc.

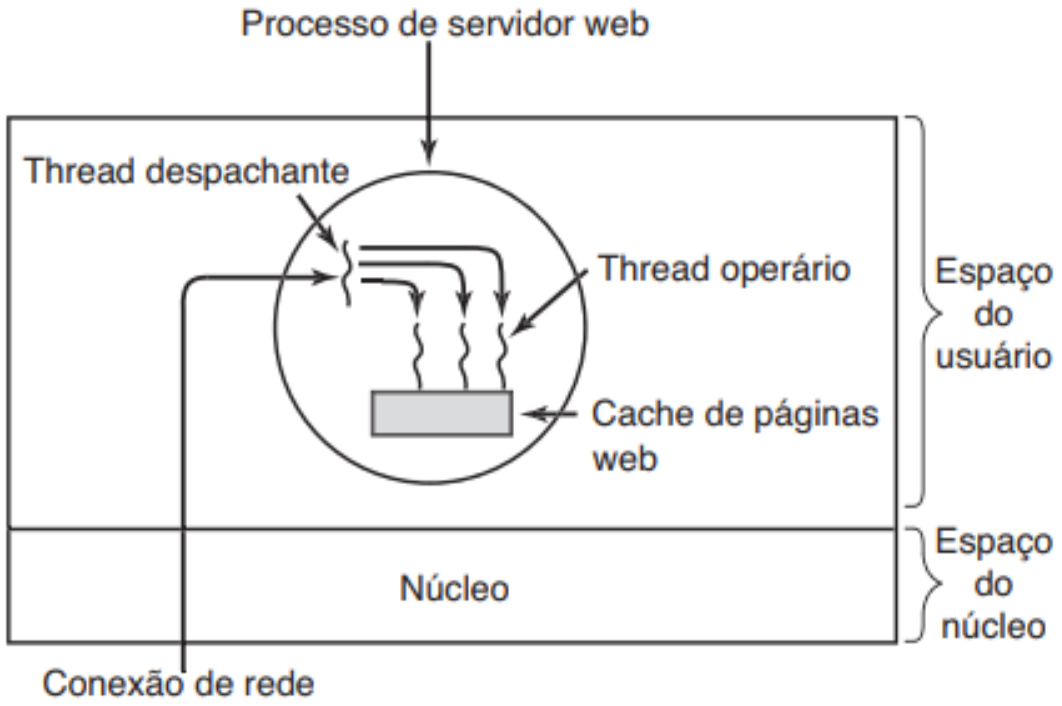
FIGURA 2.7 Um processador de texto com três threads.



4.3.1 Threads (Exemplo: Servidor Web)

1) O despachante (i) lê as requisições de trabalho que chegam, (ii) escolhe uma thread operário ociosa e (iii) entrega a requisição. A thread operário (iv) lê a cache, caso não encontre a informação, (v) inicializa uma leitura de disco.

FIGURA 2.8 Um servidor web multithread.



4.3.1 Threads (Exemplo: Servidor Web)

a) Thread Despachante

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

(a)

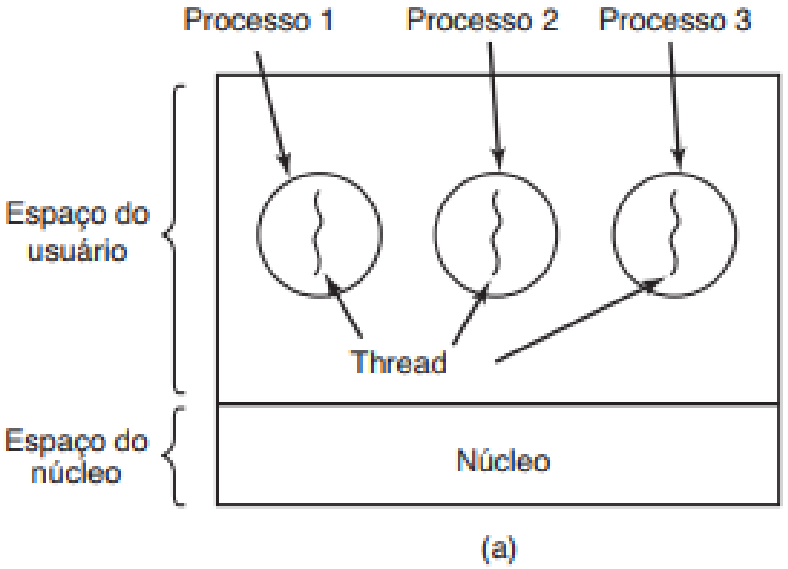
b) Thread Operário

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

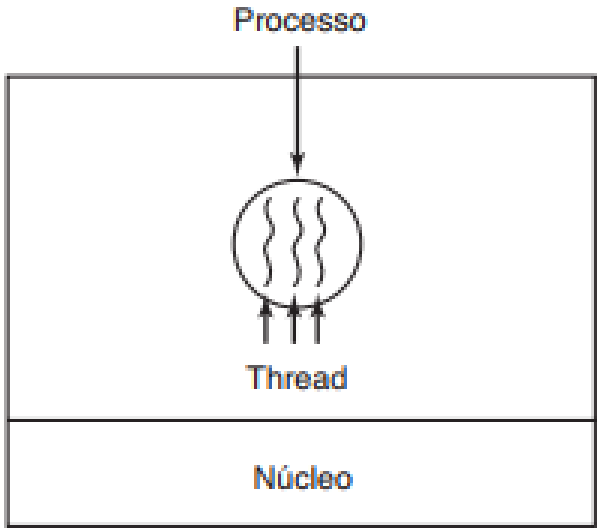
(b)

4.3.2 Processos vs. Threads

a) Três processos, cada um com um Thread
b) Um processo com três Thread



(a)



(b)

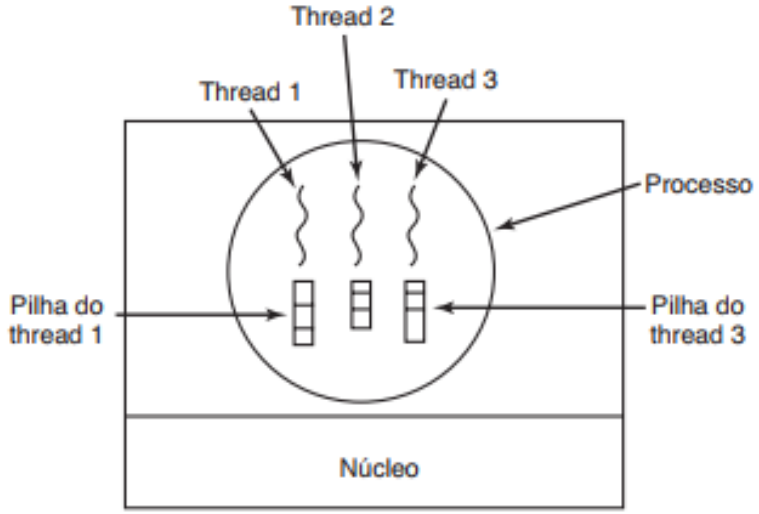
4.3.2 Processos vs. Threads

Cada thread tem sua própria pilha de execução (pois chamam rotinas diferentes), embora **compartilhe o espaço de endereçamento e todos seus dados**

FIGURA 2.12 A primeira coluna lista alguns itens compartilhados por todos os threads em um processo. A segunda lista alguns itens específicos a cada 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	

FIGURA 2.13 Cada thread tem a sua própria pilha.



4.3.3 Problemas com as Threads

1) **Como cada thread pode ter acesso a qualquer endereço de memória dentro do espaço** de endereçamento do processo;

a) uma thread pode ler, escrever ou apagar a pilha ou as variáveis globais de outra thread.

b) Exemplo: **a** = b + c;(instruções) T^1

x = **a** + y;(instruções) T^2

Problema de dependência: Executar primeiro a T^1 para depois a T^2

2) Necessidade de **sincronizar** a execução.

4.4 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.

4.4 Estados de thread: ciclo de vida de um thread

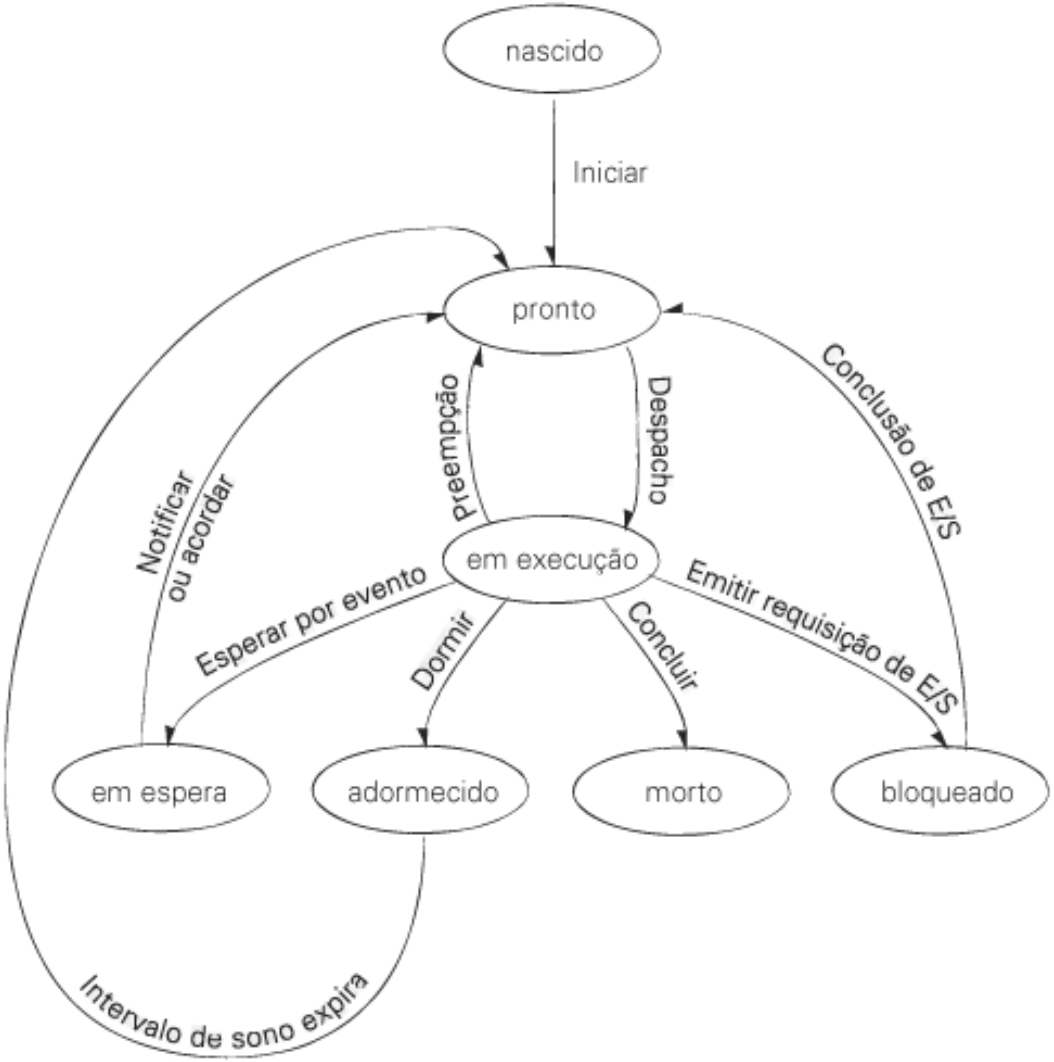


Figura 4.2 Ciclo de vida do thread.

4.5 Operações de thread

- **Os threads e os processos têm operações em comum:**
  - Criar
  - Sair (terminar)
  - Suspender
  - Retomar
  - Dormir
  - Acordar

4.5 Operações de thread

- **As operações de thread não correspondem precisamente às operações de processo.**
  - **Cancelar**
    - Indica que um thread deve ser terminado, mas não garante que o thread será terminado.
    - Os threads podem mascarar o sinal de cancelamento.
  - **Associar**
    - Para que um thread primário aguarde até que todos os outros threads terminem, ele se associa a esses threads.
    - O thread que se associa é bloqueado até que o thread ao qual ele se associou termine.

PEARSON Prentice Hall

4.5.1 Threads – Implementação com Pacote PTreads do POSIX

Todos os threads têm determinadas propriedades.

Cada um tem um identificador, um conjunto de registradores (incluindo o contador de programa), e um conjunto de atributos, que são armazenados em uma estrutura. Os atributos incluem tamanho da pilha, parâmetros de escalonamento e outros itens necessários para usar o thread

FIGURA 2.14 Algumas das chamadas de função do Pthreads.

Chamada de thread	Descrição
Pthread_create	Cria um novo thread
Pthread_exit	Conclui a chamada de thread
Pthread_join	Espera que um thread específico seja abandonado
Pthread_yield	Libera a CPU para que outro thread seja executado
Pthread_attr_init	Cria e inicializa uma estrutura de atributos do thread
Pthread_attr_destroy	Remove uma estrutura de atributos do thread

4.5.2 Um exemplo de programa usando Threads

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMBER_OF_THREADS     10

void *print_hello_world(void *tid)
{
        /* Esta funcao imprime o identificador do thread e sai. */
        printf("Ola mundo. Boas vindas do thread %d\n", tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        /* O programa principal cria 10 threads e sai. */
        pthread_t threads[NUMBER_OF_THREADS];
        int status, i;

        for(i=0; i < NUMBER_OF_THREADS; i++) {
                printf("Metodo Main. Criando thread %d\n", i);
                status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

                if (status != 0) {
                        printf("Oops. pthread_create returned error code %d\n", status);
                        exit(-1);
                }
        }
        exit(NULL);
}
```

4.5.3 Threads – Implementação em Java

Estendendo a **class Thread**

Criar thread: Instanciar classe que herda da classe `Thread'. **class Thread** possui todo o código para criar e executar threads

```
class ThreadSimples extends Thread {

    public void run() {
        System.out.println("Ola de uma
                            nova thread!");
    }

    public static void main(String
                                args[]) {
     Thread thread =
                new ThreadSimples();
     thread.start();

        System.out.println("Ola da
                    thread original!");
    }
}
```


4.5.3 Threads – Implementação em Java

Joining a Thread

1) Permite a uma thread esperar que outra termine;
2) A thread principal esperará thread2 morrer.

```
class ThreadSimples extends Thread {

   public void run() {
      System.out.println("Ola de uma nova thread! "
                         + super.toString() + ".");
   }

   public static void main(String args[]) {
      ThreadSimples thread = new ThreadSimples();
      ThreadSimples thread2 = new ThreadSimples();

      thread.start();
      thread2.start();

      try {
         thread2.join();
      }
      catch (InterruptedException e) {
         e.printStackTrace();
      }

      System.out.println("Ola da thread original!");
   }
}
```


4.5.3 Threads – Implementação em Java

Sleeping a Thread

1) A thread atual fica bloqueada por um número de milisegundos
2) Precisa capturar InterruptedException

```
try {
    Thread.sleep(1);
}
catch (InterruptedException e) {
    e.printStackTrace();
}
```

4.6 Modelos de thread

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

4.6.1 Threads de usuário

- **Os threads de usuário executam operações de suporte a threads no espaço do usuário.**
  - Isso significa que os threads são criados por bibliotecas em tempo de execução que não podem executar instruções privilegiadas nem acessar as primitivas do núcleo diretamente.

PEARSON Prentice Hall

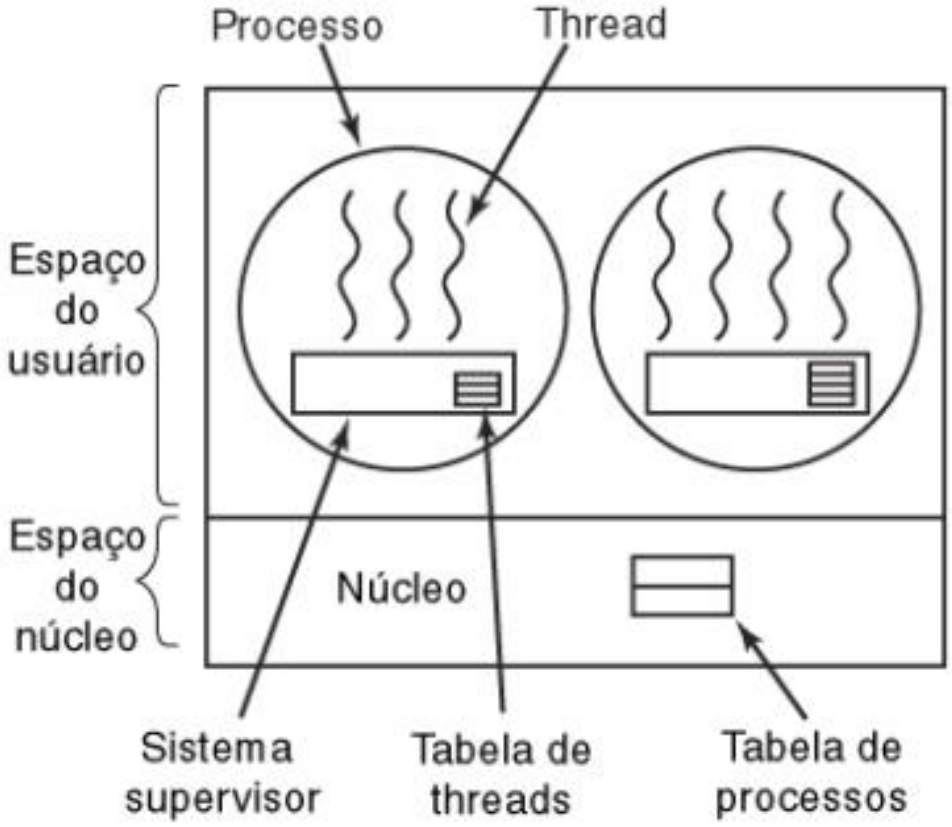
4.6.1 Threads de usuário

- **Implementação de thread de usuário**
  - Mapeamentos de thread muitos-para-um
    - O sistema operacional mapeia todos os threads de um processo multithread para um único contexto de execução.
    - **Vantagens**
      - → As bibliotecas de usuário podem escalonar seus threads para otimizar o desempenho.
      - → A sincronização é realizada fora do núcleo, e isso evita chaveamento de contexto.
      - → É mais portável.
    - **Desvantagens**
      - → O núcleo considera o processo multithread como um único thread de controle.
        - Isso pode fazer com que o desempenho fique abaixo do ideal se um thread requisitar uma operação E/S.
        - Não pode ser escalonado para executar em múltiplos processadores ao mesmo tempo.

PEARSON Prentice Hall

4.6.1 Threads de usuário

Figura 4.3 Threads de usuário.



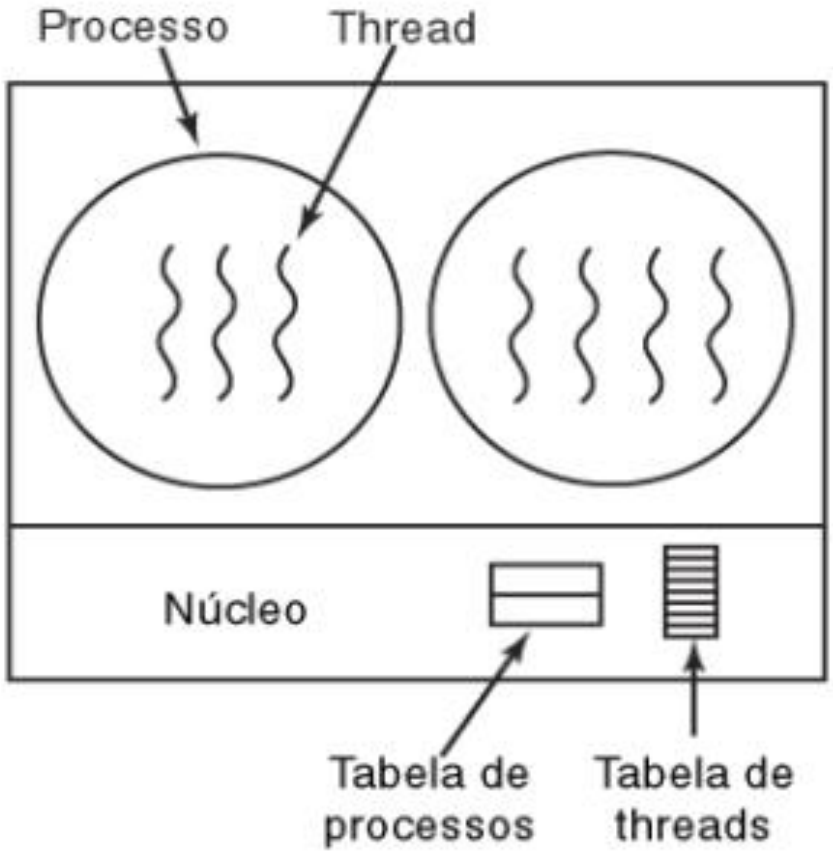
4.6.2 Threads de núcleo

- **Os threads de núcleo tentam resolver as limitações dos threads de usuário mapeando cada thread para o seu próprio contexto de execução.**
  - O thread de núcleo oferece mapeamento de thread um-para-um.
    - Vantagens: maior escalabilidade, interatividade e rendimento.
    - Desvantagens: sobrecarga decorrente do chaveamento de contexto e menor portabilidade em virtude de as APIs(Application Programming Interface) serem específicas ao sistema operacional.
- **Exemplo: Linux, Família Windows, OS/2, Solaris 9**

PEARSON Prentice Hall

4.6.2 Threads de núcleo

Figura 4.4 Threads de núcleo.

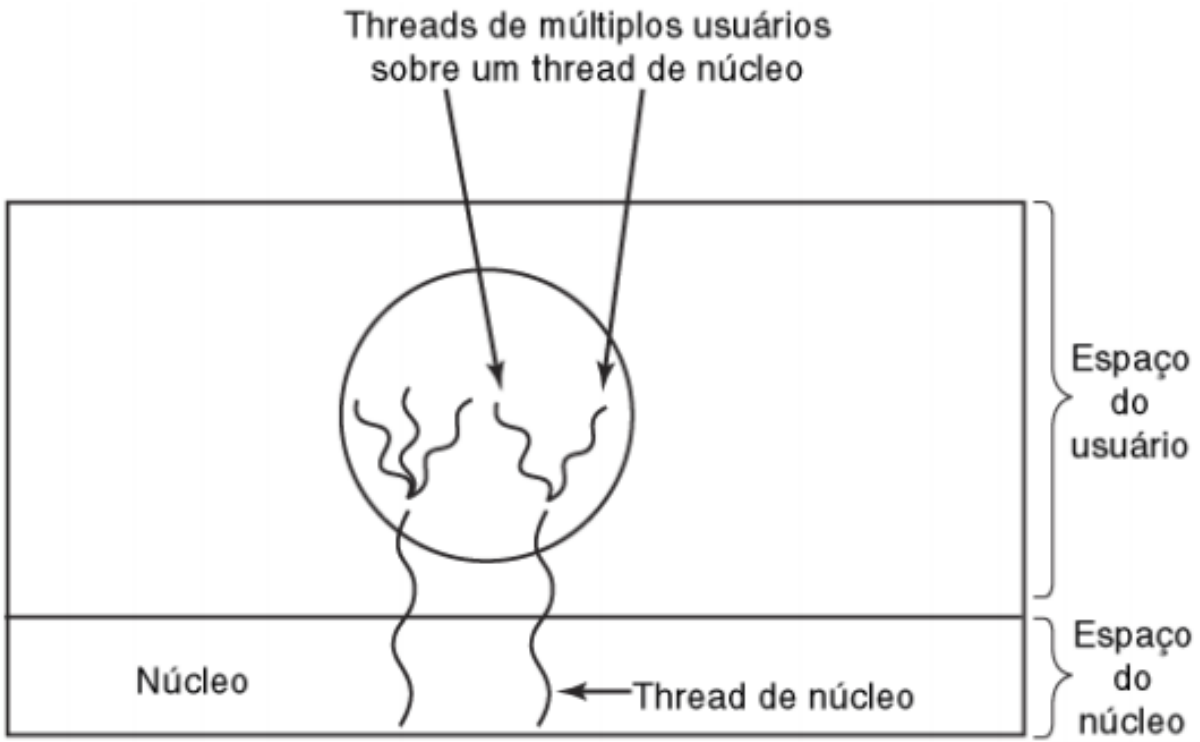


4.6.3 Combinação de threads de usuário e de núcleo

- Mapeamento de threads muitos-para-muitos (mapeamento de threads *m-to-n*)
  - O número de threads de usuário e de núcleo tem de ser o mesmo.
  - Em comparação com os mapeamentos de threads um-para-um, esse mapeamento consegue reduzir a sobrecarga implementando o reservatório de threads.
- **Threads operários**
  - Threads de núcleo persistentes que ocupam o reservatório de threads.
  - Os threads operários melhoram o desempenho em ambientes em que os threads são criados e destruídos com frequência.
  - Cada novo thread é executado por um thread operário.
- **Ativação de escalonador**
  - Técnica que permite que uma biblioteca de usuário escalone seus threads.
  - Ocorre quando o sistema operacional chama uma biblioteca de threads de usuário para determinar se algum de seus threads precisam ser reescalonados.
  - **Exemplos: Solaris até versão 8, HP-UX, Tru64 Unix**

4.6.3 Combinação de threads de usuário e de núcleo

Figura 4.5 Modelo de operação de thread híbrido.



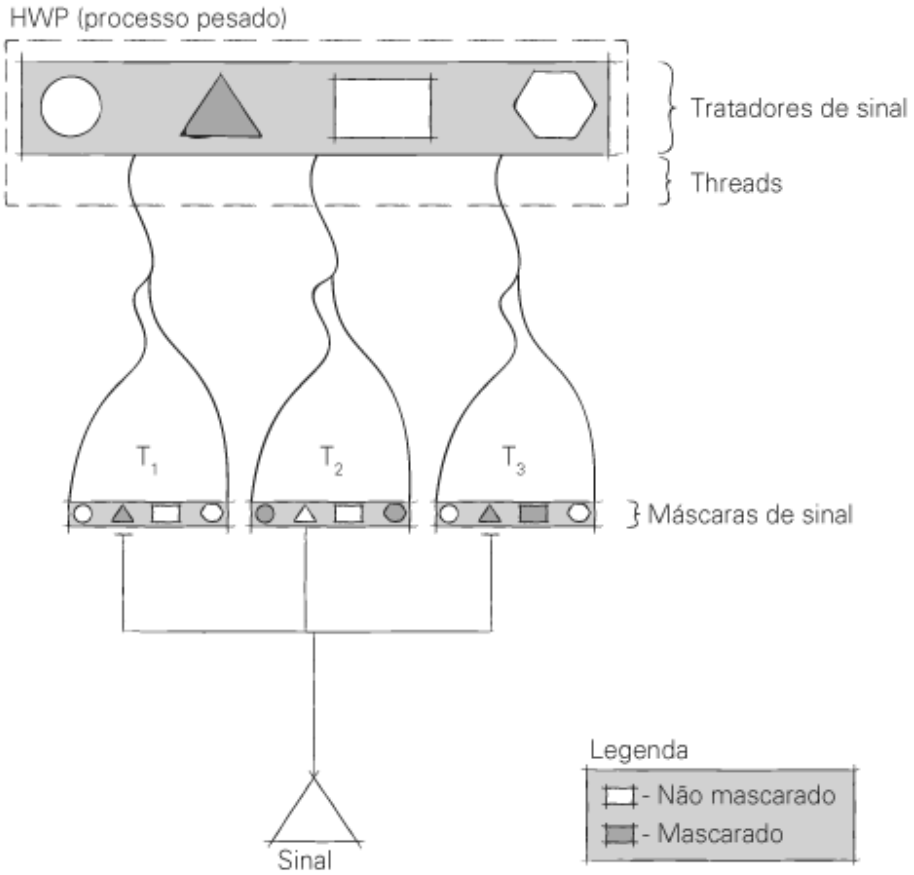
4.7.1 Entrega de sinal de thread

- **Dois tipos de sinal**
  - **Síncrono:**
    - Resulta diretamente da execução de um programa.
    - Pode ser emitido (entregue) para um thread que esteja sendo executado no momento.
  - **Assíncrono**
    - Resulta de um evento em geral não relacionado com a instrução corrente.
    - A biblioteca de threads precisa identificar todo receptor de sinal para que os sinais assíncronos sejam emitidos (entregues) devidamente.
- **Todo thread normalmente está associado a um conjunto de sinais pendentes que são emitidos (entregues) quando ele é executado.**
- **O thread pode mascarar todos os sinais, exceto aqueles que deseja receber.**

PEARSON Prentice Hall

4.7.1 Entrega de sinal de thread

Figura 4.6 Mascaramento de sinal.



4.7.2 Término de threads

- **Término de threads (cancelamento)**
  - É diferente de implementação de threads.
  - Se for terminado prematuramente, o thread pode provocar erros sutis nos processos, porque vários threads compartilham o mesmo espaço de endereço.
  - Determinadas implementações de thread permitem que um thread determine quando ele pode ser terminado, a fim de evitar que o processo entre em um estado inconsistente.

4.8 POSIX e Pthreads

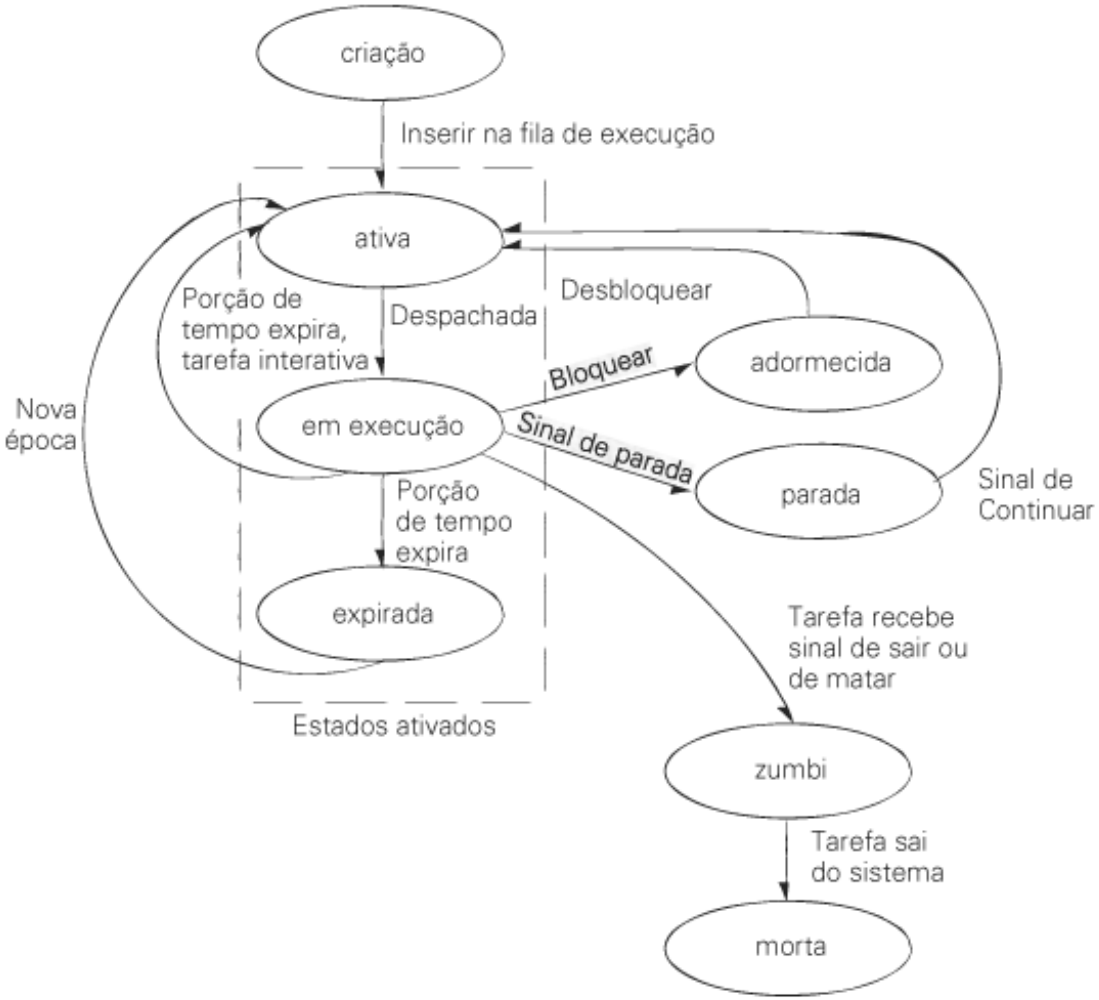
- **Os threads que usam a API de thread POSIX são chamados de Pthreads.**
  - A especificação POSIX determina que os registradores do processador, a pilha e a máscara de sinal sejam mantidos individualmente para cada thread.
  - A especificação POSIX especifica como os sistemas operacionais devem emitir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

4.9 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.

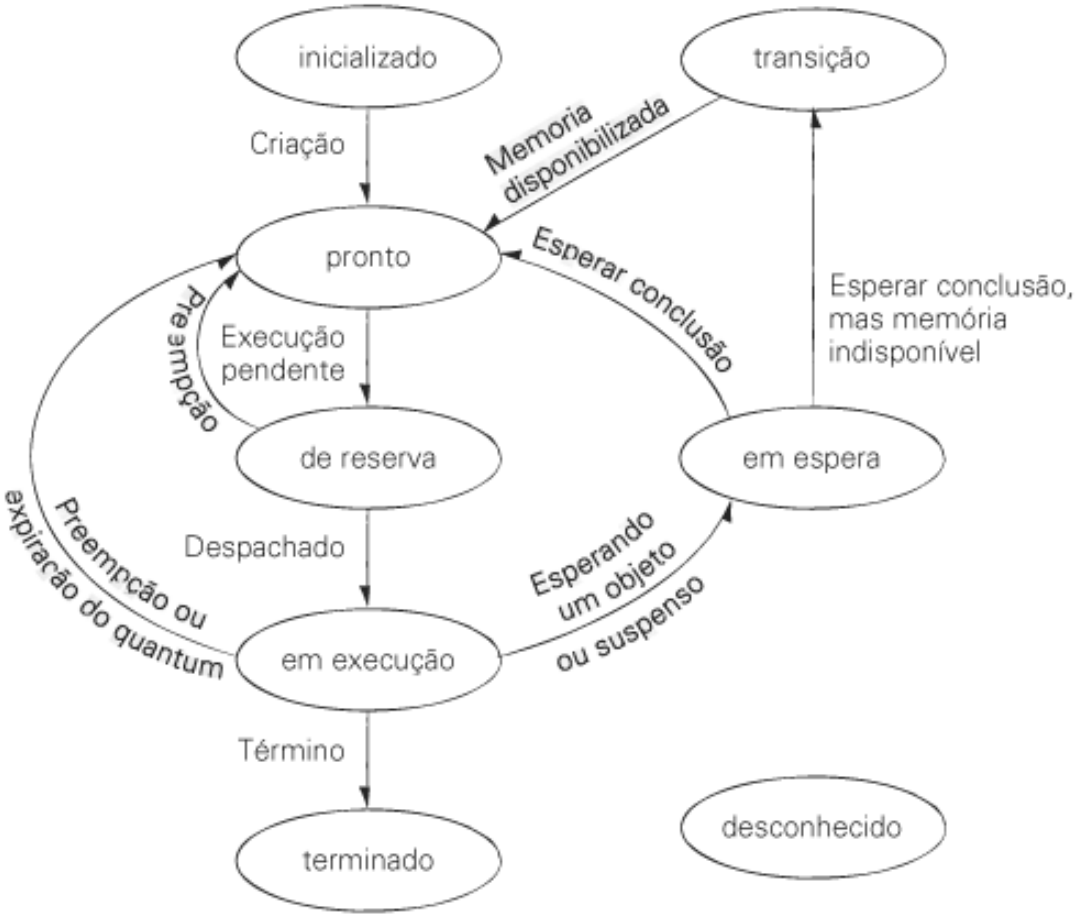
4.9 Threads Linux

Figura 4.7 Diagrama de transição de estado de tarefa do Linux.



4.10 Threads do Windows

Figura 4.8 Diagrama de transição de estado de thread do Windows



4.11 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.**

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

Figura 4.9 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
```


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

Figura 4.9 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 );
```


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

Figura 4.9 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
```

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

Figura 4.9 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
```

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

Figura 4.9 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
```