Aula 10
Sistemas Operacionais I

Threads

Prof. Julio Cezar Estrella
jcezar@icmc.usp.br

Material adaptado de

Sarita Mazzini Bruschi

baseados no livro Sistemas Operacionais Modernos de A. Tanenbaum

Processos

Sistemas Operacionais tradicionais:

Cada processo tem um único espaço de endereçamento e um único fluxo de controle

Existem situações onde é desejável ter múltiplos fluxos de controle compartilhando o mesmo espaço de endereçamento:

Solução: threads

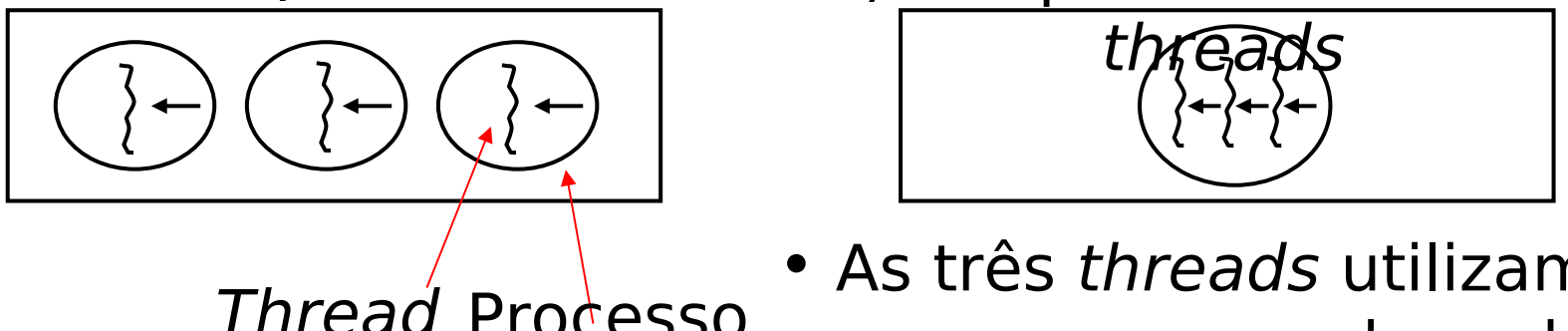
Threads

Um processo tradicional (pesado) possui um contador de programas, um espaço de endereço e apenas uma *thread* de controle (ou fluxo de controle);

***Multithreading**:* Sistemas atuais suportam múltiplas *threads* de controle, ou seja, pode fazer mais de uma tarefa ao mesmo tempo, servindo ao mesmo propósito;

a) Três processos

b) Um processo com três *threads*



- As três *threads* utilizam o mesmo espaço de endereço

Threads

Thread é uma entidade básica de utilização da CPU.

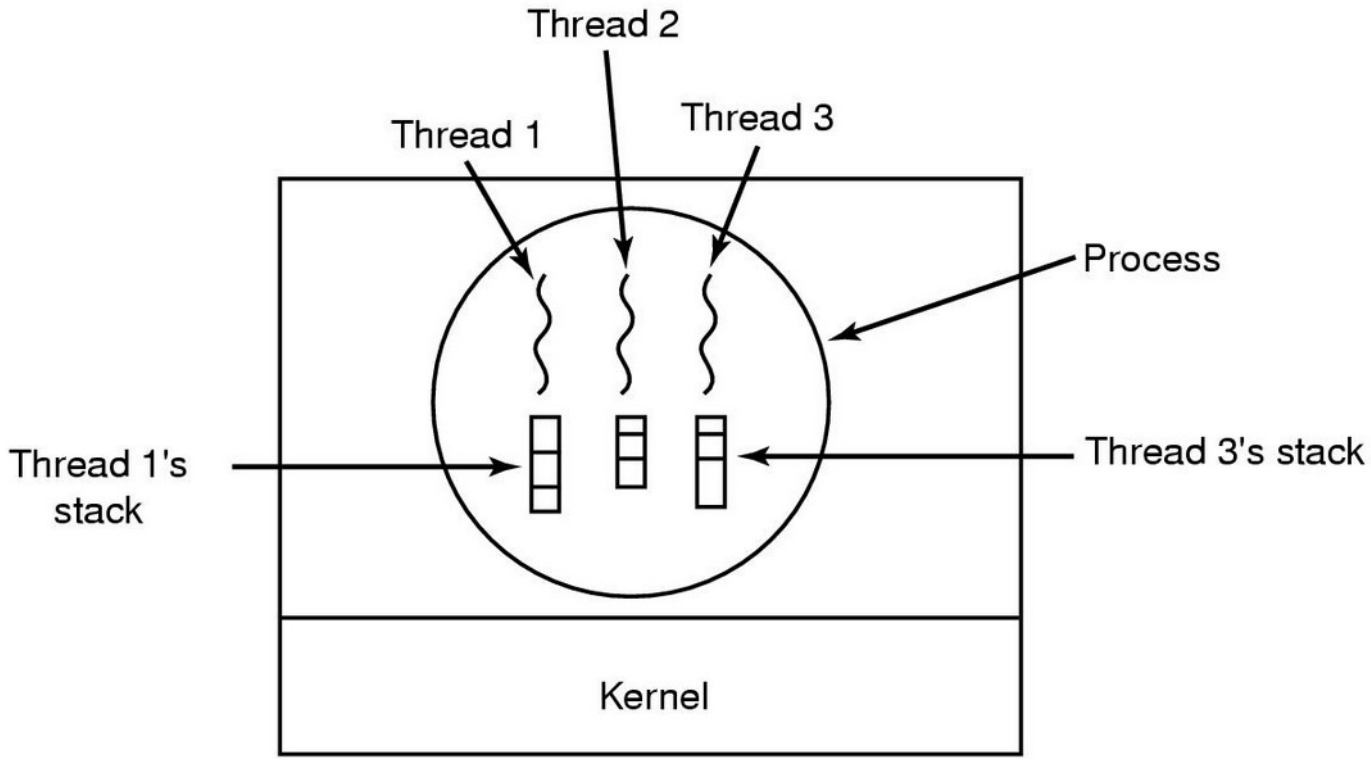
Também conhecidos como processos leves (*lightweight process* ou *LWP*);

Processos com múltiplas *threads* podem realizar mais de uma tarefa de cada vez;

Processos são usados para agrupar recursos; *threads* são as entidades escalonadas para execução na CPU

A CPU alterna entre as *threads* dando a impressão de que elas estão executando em paralelo;

Threads



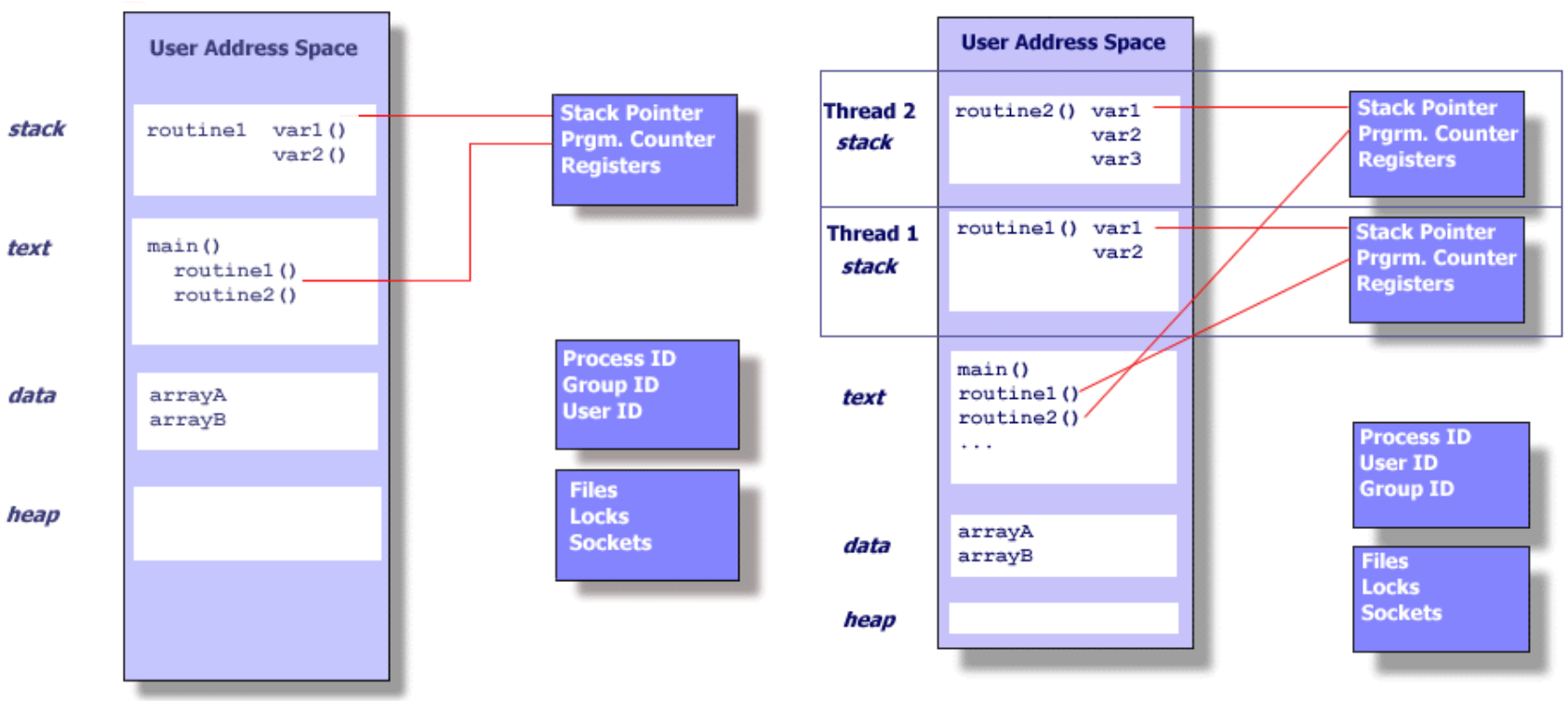
Cada *thread* tem sua pilha de execução

Threads

Itens por Processo	**Itens por *Thread***
☐ Espaço de endereçamento ☐ Variáveis globais ☐ Arquivos abertos ☐ Processos filhos ☐ Alarmes pendentes	☐ Contador de programa ☐ Registradores (contexto) ☐ Pilha ☐ Estado

- Compartilhamento de recursos;
- Cooperação para realização de tarefas;

Threads



Processo Unix

Threads em um processo Unix

Threads

Como cada *thread* pode ter acesso a qualquer endereço de memória dentro do espaço de endereçamento do processo, uma *thread* pode ler, escrever ou apagar a pilha de outra *thread*;

Não existe proteção pois:

É impossível

Não é necessário pois, diferente dos processos que podem pertencer a diferentes usuários, as threads são sempre de um mesmo usuário

Threads

Razões para existência de *threads*:

Em múltiplas aplicações ocorrem múltiplas atividades “ao mesmo tempo”, e algumas dessas atividades podem bloquear de tempos em tempos;

As *threads* são mais fáceis de gerenciar do que processos, pois elas não possuem recursos próprios → o processo é que tem!

Desempenho: quando há grande quantidade de E/S, as threads permitem que essas atividades se sobreponham, acelerando a aplicação;

Paralelismo Real em sistemas com múltiplas CPUs.

Threads

Considere um servidor de arquivos:

Recebe diversas requisições de leitura e escrita em arquivos e envia respostas a essas requisições;

Para melhorar o desempenho, o servidor mantém uma *cache* dos arquivos mais recentes, lendo da *cache* e escrevendo na *cache* quando possível;

Quando uma requisição é feita, uma *thread* é alocada para seu processamento. Suponha que essa *thread* seja bloqueada esperando uma transferência de arquivos. Nesse caso, outras *threads* podem continuar atendendo a outras requisições;

Threads

Considere um navegador WEB:

Muitas páginas WEB contêm muitas figuras que devem ser mostradas assim que a página é carregada;

Para cada figura, o navegador deve estabelecer uma conexão separada com o servidor da página e requisitar a figura → tempo;

Com múltiplas *threads*, muitas imagens podem ser requisitadas ao mesmo tempo melhorando o desempenho;

Threads

Benefícios:

Capacidade de resposta: aplicações interativas; Ex.: servidor WEB;

Compartilhamento de recursos: mesmo endereçamento; memória, recursos;

Economia: criar e realizar chaveamento de *threads* é mais barato;

Utilização de arquiteturas multiprocessador: processamento paralelo;

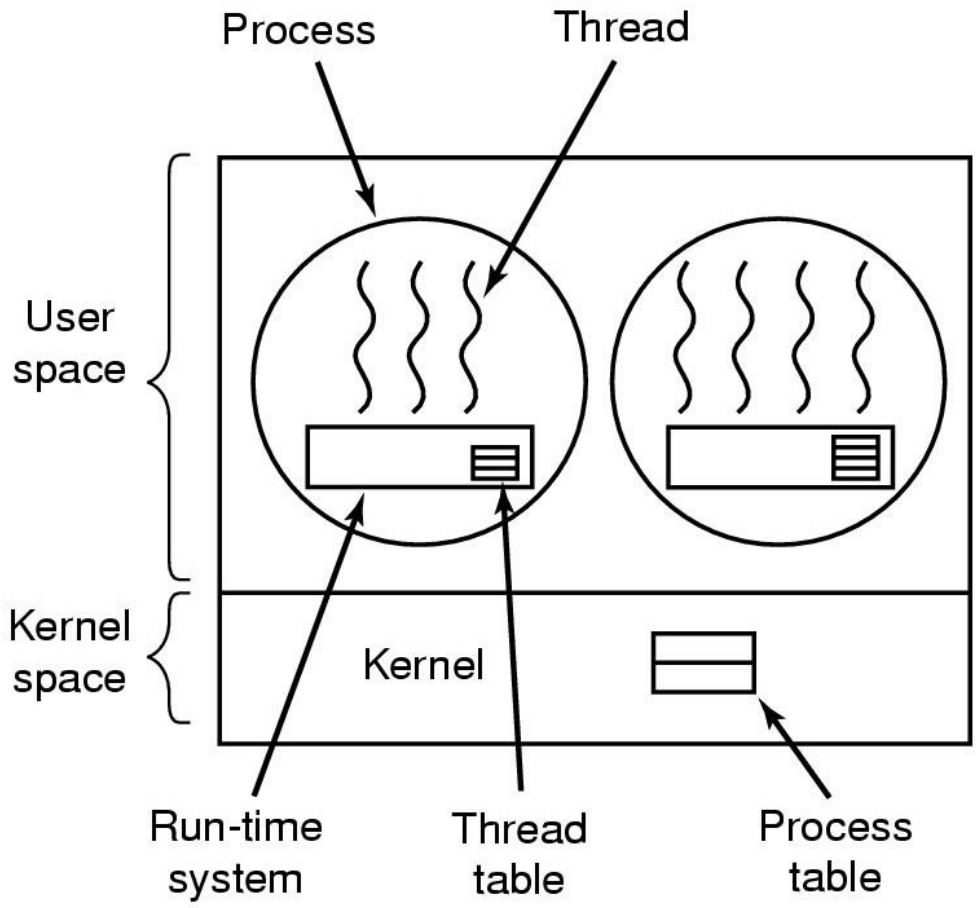
Threads

Tipos de *threads*:

Em modo usuário (espaço do usuário): implementadas por bibliotecas no espaço do usuário;

- Criação e escalonamento são realizados sem o conhecimento do *kernel*;
  - Sistema Supervisor (*run-time system*): coleção de procedimentos que gerenciam as *threads*;
  - Tabela de *threads* para cada processo;
- Cada processo possui sua própria tabela de *threads*, que armazena todas a informações referentes à cada *thread* relacionada àquele processo;

Threads em modo usuário



Threads em modo usuário

Tipos de *threads*: Em modo usuário

Vantagens:

- Alternância de *threads* no nível do usuário é mais rápida do que alternância no *kernel*;
- Menos chamadas ao *kernel* são realizadas;
- Permite que cada processo possa ter seu próprio algoritmo de escalonamento;
- Podem ser implementado em Sistemas Operacionais que não têm threads

Principal desvantagem:

- Processo inteiro é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema;

Implementação de *threads*

Implementação em espaço de usuário:

Problemas:

- Como permitir chamadas bloqueantes se as chamadas ao sistema são bloqueantes e essa chamada irá bloquear todas as threads?
  - Mudar a chamada ao sistema para não bloqueante, mas isso implica em alterar o SO -> não aconselhável
  - Verificar antes se uma determinada chamada irá bloquear a thread e, se for bloquear, não a executar, simplesmente mudando de thread
- *Page fault*
  - Se uma thread causa uma *page fault*, o kernel, não sabendo da existência da thread, bloqueia o processo todo até que a página que está em falta seja buscada
- Se uma thread não liberar a CPU voluntariamente, ela executa o quanto quiser
  - Uma thread pode não permitir que o processo escalonador do processo tenha sua vez

Tipos de *Threads*

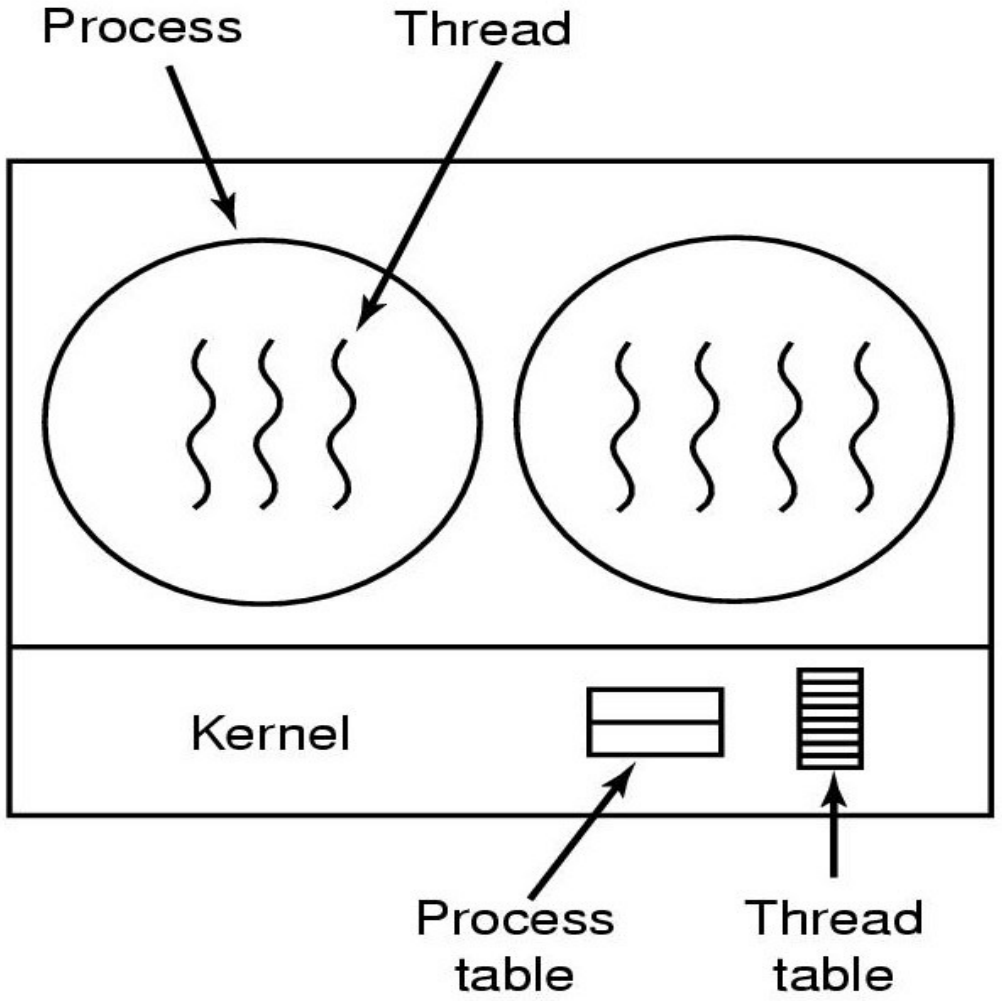
Tipos de *threads*:

Em modo *kernel*: suportadas diretamente pelo SO;

Criação, escalonamento e gerenciamento são feitos pelo *kernel*;

- Tabela de *threads* e tabela de processos separadas;
  - as tabelas de *threads* possuem as mesmas informações que as tabelas de threads em modo usuário, só que agora estão implementadas no *kernel*;

Threads em modo *kernel*



Threads em modo *kernel*

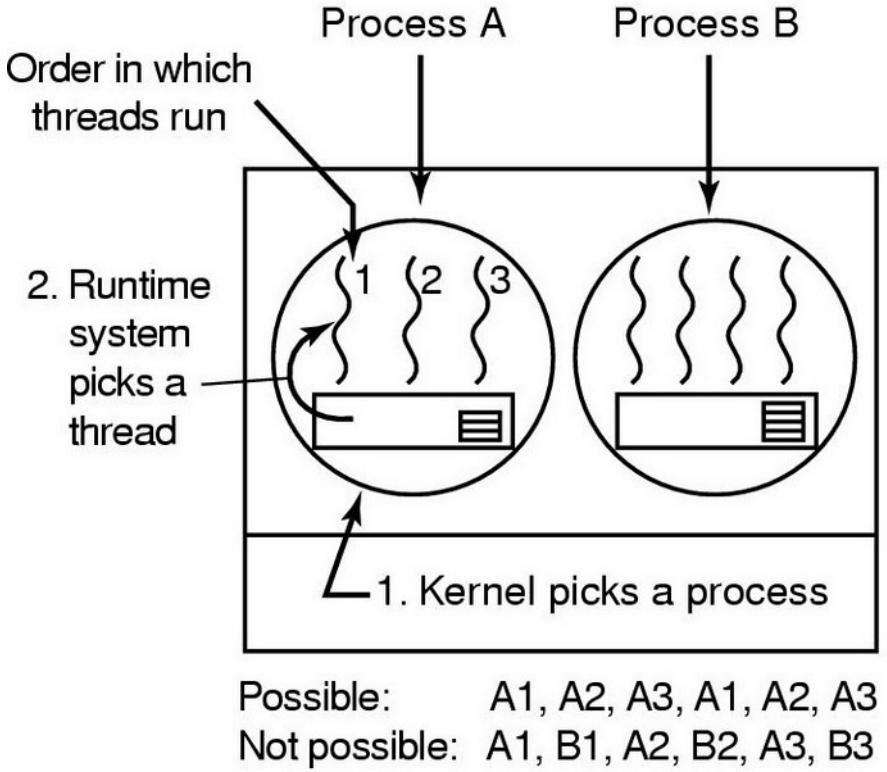
Vantagem:

Processo inteiro não é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema;

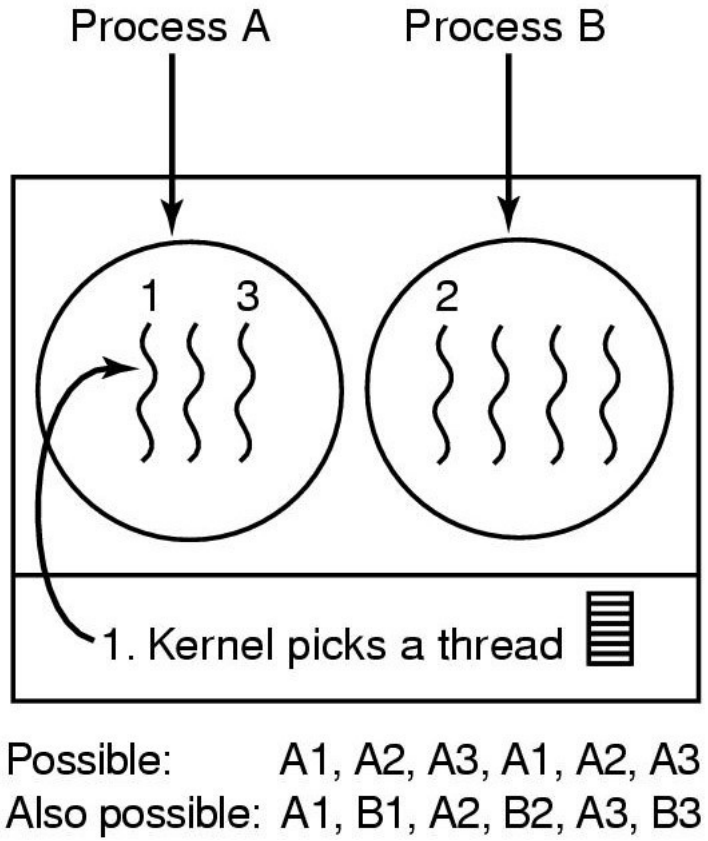
Desvantagem:

Gerenciar threads em modo *kernel* é mais caro devido às chamadas de sistema durante a alternância entre modo usuário e modo *kernel*;

Threads em modo Usuário *x* *Threads* em modo *Kernel*



Threads em modo usuário



Threads em modo *kernel*