SISTEMAS OPERACIONAIS

AULA 7 – THREADS

Prof.ª Sandra Cossul, Ma.

THREADS - INTRODUÇÃO

- À medida que as aplicações se tornaram mais complexas, um único processo por aplicação se tornou inconveniente (não eficiente).
- Uma aplicação geralmente executa tarefas simultâneas sobre os mesmos recursos (dados).
- Demanda de suportar mais de uma tarefa operando sobre os mesmos recursos dentro do mesmo processo → conceito de **thread**

THREADS - INTRODUÇÃO

- Vimos que um **processo** é um **programa em execução**
- As **tarefas** realizadas por um processo são chamadas de **threads**
- Uma **thread** é um **fluxo de execução independente**.

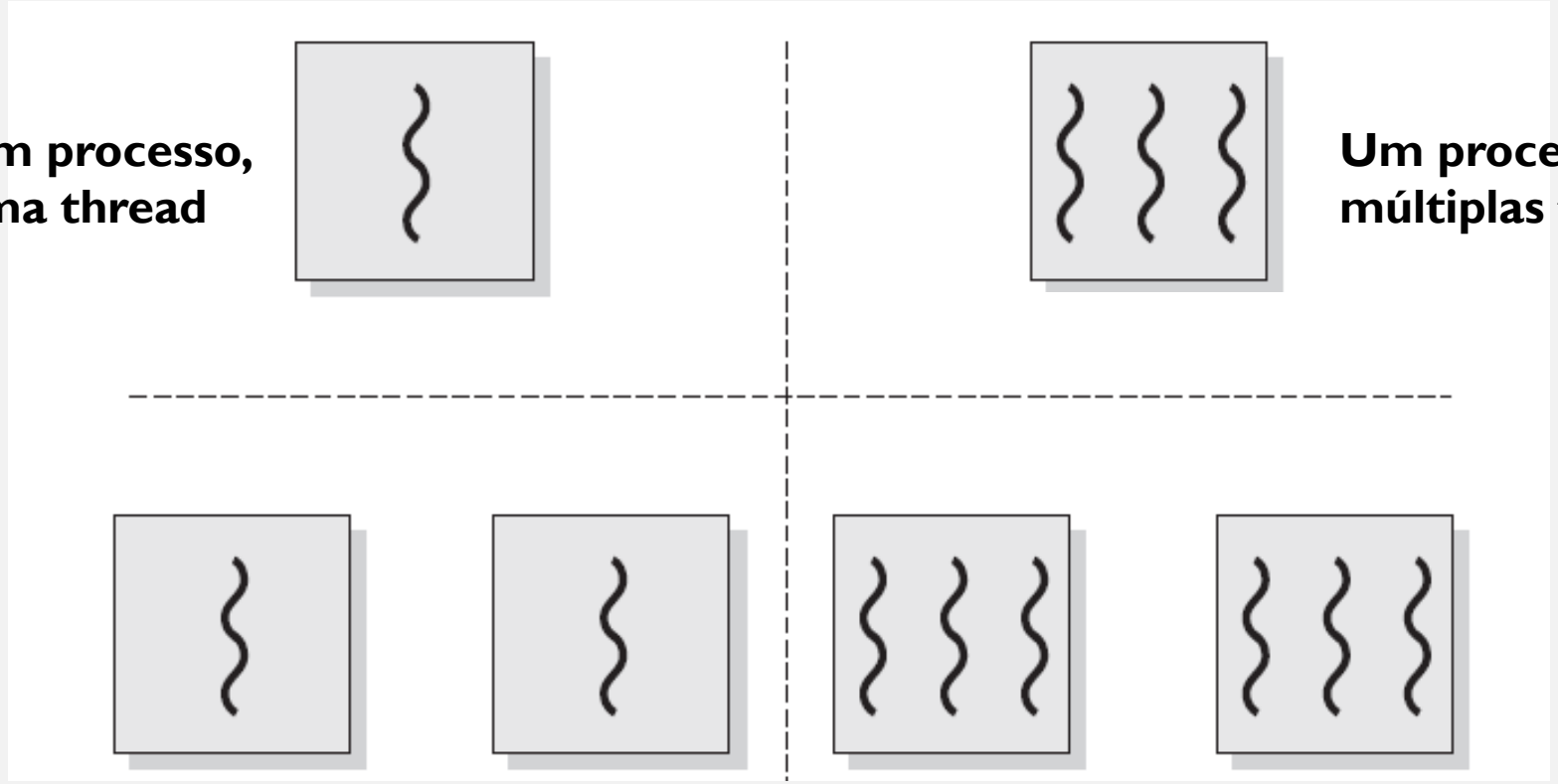
THREADS - INTRODUÇÃO

- Podemos ter mais de um **fluxo de execução** referente ao mesmo processo, o que possibilita realizar mais de uma tarefa ao mesmo tempo.

- A ideia principal é que o processo é decomposto em "**mini-processos**" (os quais chamamos de **threads**).

MULTITHREADING

- Característica do SO em que são suportados **múltiplos** e **concorrentes fluxos de execução** (threads) dentro de um <u>único processo.</u>



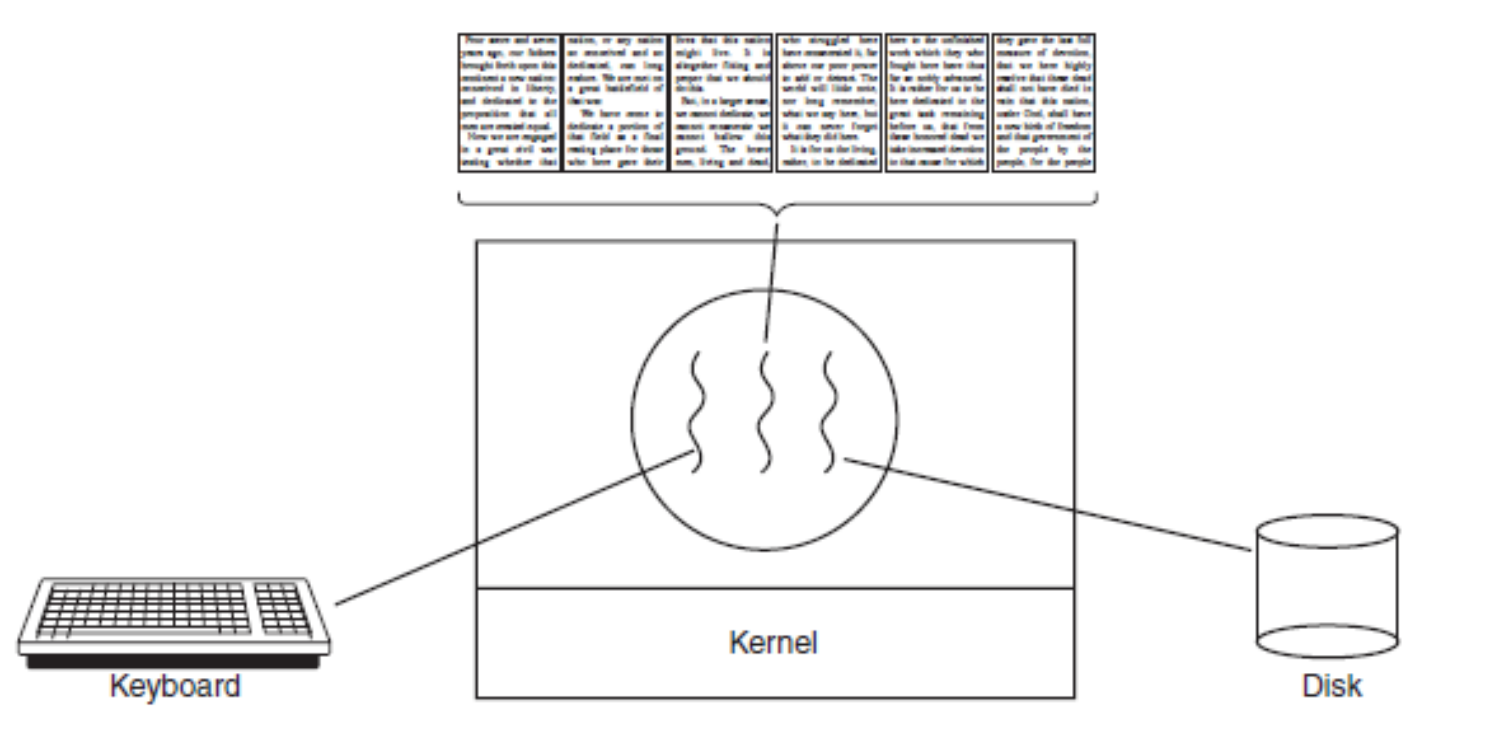
EXEMPLOS SISTEMAS MULTITHREADING

- **Navegador web**
  - Thread 1 → mostrar imagens e textos
  - Thread 2 → adquirir dados da rede

- **Editor de texto (Word)**
  - Thread 1 → mostrar o texto sendo editado
  - Thread 2 → adquirir as informações do teclado do usuário
  - Thread 3 → revisar o texto gramaticalmente (background)
  - Thread 4 → realizar backup

EXEMPLOS SISTEMAS MULTITHREADING

- **Editor de texto (Word)**



THREADS X PROCESSOS

- **Thread** é uma abstração que permite que uma aplicação execute mais de um trecho de código simultaneamente.
- **Processos** permitem ao SO executar mais de uma aplicação ao mesmo tempo.

- Um programa **multithreading** pode continuar executando e respondendo ao usuário mesmo se parte dele está bloqueada ou executando uma tarefa demorada.

THREADS X PROCESSOS

- Existem duas **características** fundamentais que são usualmente tratadas de forma **independente** pelo SO:
  - **Propriedade de recursos** (“resource ownership”)
    - Trata dos recursos alocados aos processos, e que são necessários para a sua execução.
    - **Ex:** memória, arquivos, dispositivos de E/S, etc.
  - **Escalonamento** (“scheduling / dispatching”)
    - Relacionado à unidade de despacho do S.O.
    - Determina o fluxo de execução (trecho de código) que é executado pela CPU.

THREADS X PROCESSOS

- Tradicionalmente o processo está associado a:
  - **um programa em execução**
  - **um conjunto de recursos**

- Em um S.O. que suporta múltiplas threads:
  - **Processos** estão associados somente à **propriedade de recursos**
  - **Threads** estão associadas às atividades de execução (ou seja, threads constituem as unidades de escalonamento em sistemas multithreading).

THREADS X PROCESSOS

- **Primeira coluna**: itens compartilhados por todas as threads em um processo
- **Segunda coluna**: itens privativos de 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	

PORQUE UTILIZAR THREADS?

- **Para dividir as tarefas das várias aplicações**
  - Mesma justificativa de processos?
  - Diferença: threads paralelas podem compartilhar o espaço de endereçamento e os dados entre elas
  - Comunicação e troca de contexto mais rápida !
  - Importante para algumas aplicações (processos são separados)

- **São mais leves em comparação aos processos**
  - Mais fáceis e mais rápidas para criar e terminar, pois elas não tem recursos alocadas a elas
  - Mudanças dinâmicas e rápidas (10 – 100x mais rápido)

PORQUE UTILIZAR THREADS?

- **Ganho de performance**
  - Threads do tipo CPU-bound e I/O bound permitem realizar as tarefas em paralelo tendo um ganho em performance (aplicação mais rápida).

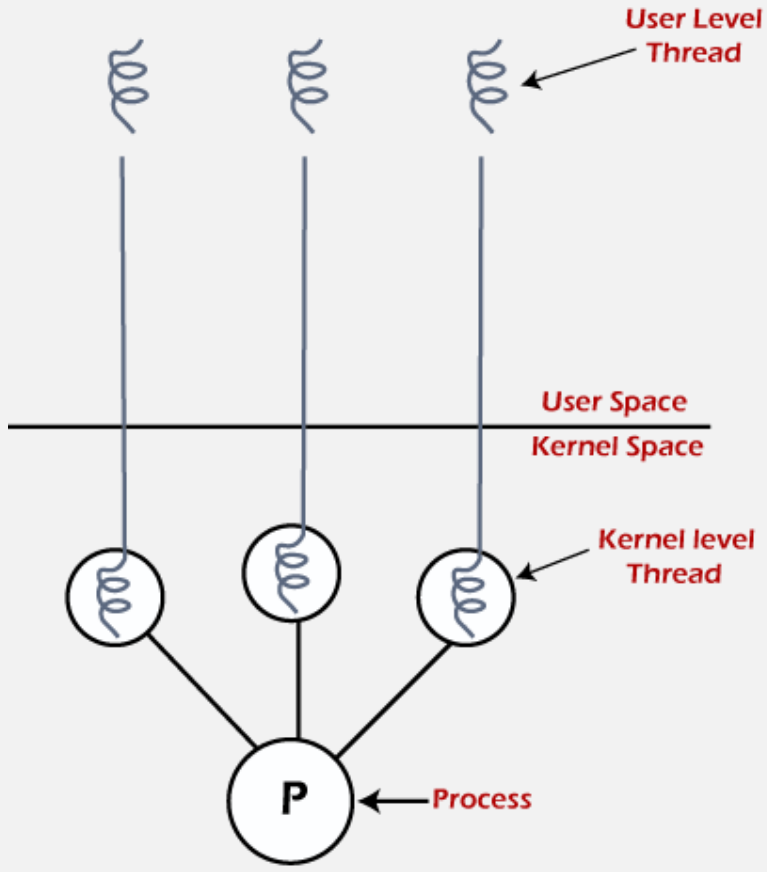
- **Explorar o uso de múltiplos CPUs**
  - paralelismo

ESTADOS DE THREADS

- Assim como os processos, as threads também possuem estados:
  - **Executando**
    - está utilizando o CPU
  - **Bloqueada**
    - esperando por algum evento acontecer
  - **Pronta**
    - escalonada para executar
  - **Terminada**

TIPOS DE THREADS

- Em um SO, existem duas formas de implementação de threads:
  - **Kernel-level:** nível de kernel
  - **User-level:** nível de usuário

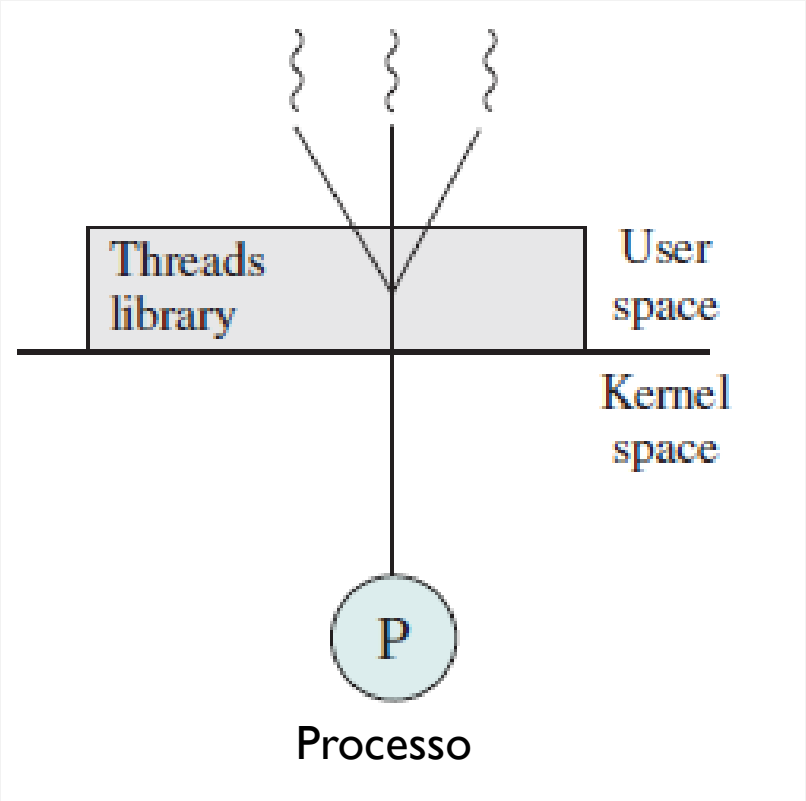


TIPOS DE THREADS
NÍVEL DE USUÁRIO

- O gerenciamento das *threads* é feito no espaço de endereçamento de usuário, por meio de uma biblioteca de threads.
  - A **biblioteca de threads** é um conjunto de funções no nível de aplicação que pode ser compartilhada por todas as aplicações
  - A biblioteca fornece métodos de escalonamento (a aplicação escolhe o melhor algoritmo para ela)

- O kernel desconhece a existência de threads, portanto, o SO não precisa oferecer suporte para *threads*.
  - Mais simples!

TIPOS DE THREADS
NÍVEL DE USUÁRIO



TIPOS DE THREADS
NÍVEL DE USUÁRIO

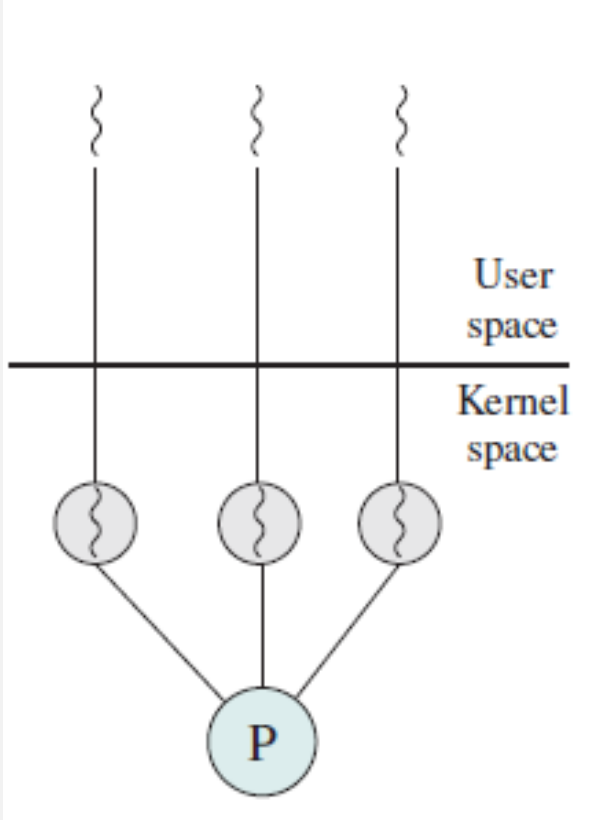
- **Vantagens**:
  - Pode ser implementado em SOs que não tem suporte à threads (e podem executar em qualquer SO)
  - O escalonamento das threads é muito rápido
  - Cada processo pode ter seu próprio algoritmo de escalonamento

- **Desvantagens**:
  - Se uma thread realiza uma system call bloqueante ela bloqueia todas as outras threads também
  - O SO não sabe o que está acontecendo, se o processo pede uma operação de E/S, por exemplo, ele o bloqueia até que a operação termine
  - Uma operação multithreading não pode tirar vantagem do multiprocessamento

TIPOS DE THREADS
NÍVEL DE KERNEL

- O **gerenciamento** das threads é feito pelo **kernel**.
- O kernel pode melhor aproveitar a capacidade de multiprocessamento da máquina, escalonando as várias threads do processo em diferentes processadores.
- O chaveamento das threads é feito pelo núcleo e o escalonamento é "**thread-basis**".
  - O bloqueio de uma thread não implica no bloqueio das outras threads do processo.
- O kernel mantém a informação de contexto para processo e threads.

TIPOS DE THREADS
NÍVEL DE KERNEL



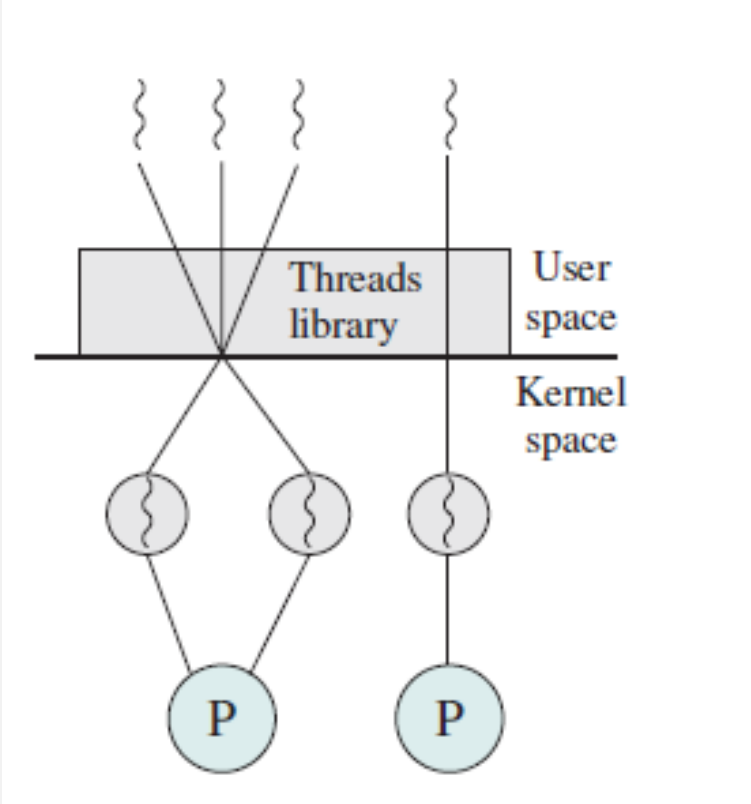
Processo

TIPOS DE THREADS
NÍVEL DE KERNEL

- **Vantagens**
  - O bloqueio de uma thread permite passar o CPU para outra thread
  - O kernel pode escalonar múltiplas threads do mesmo processo nos múltiplos CPUs

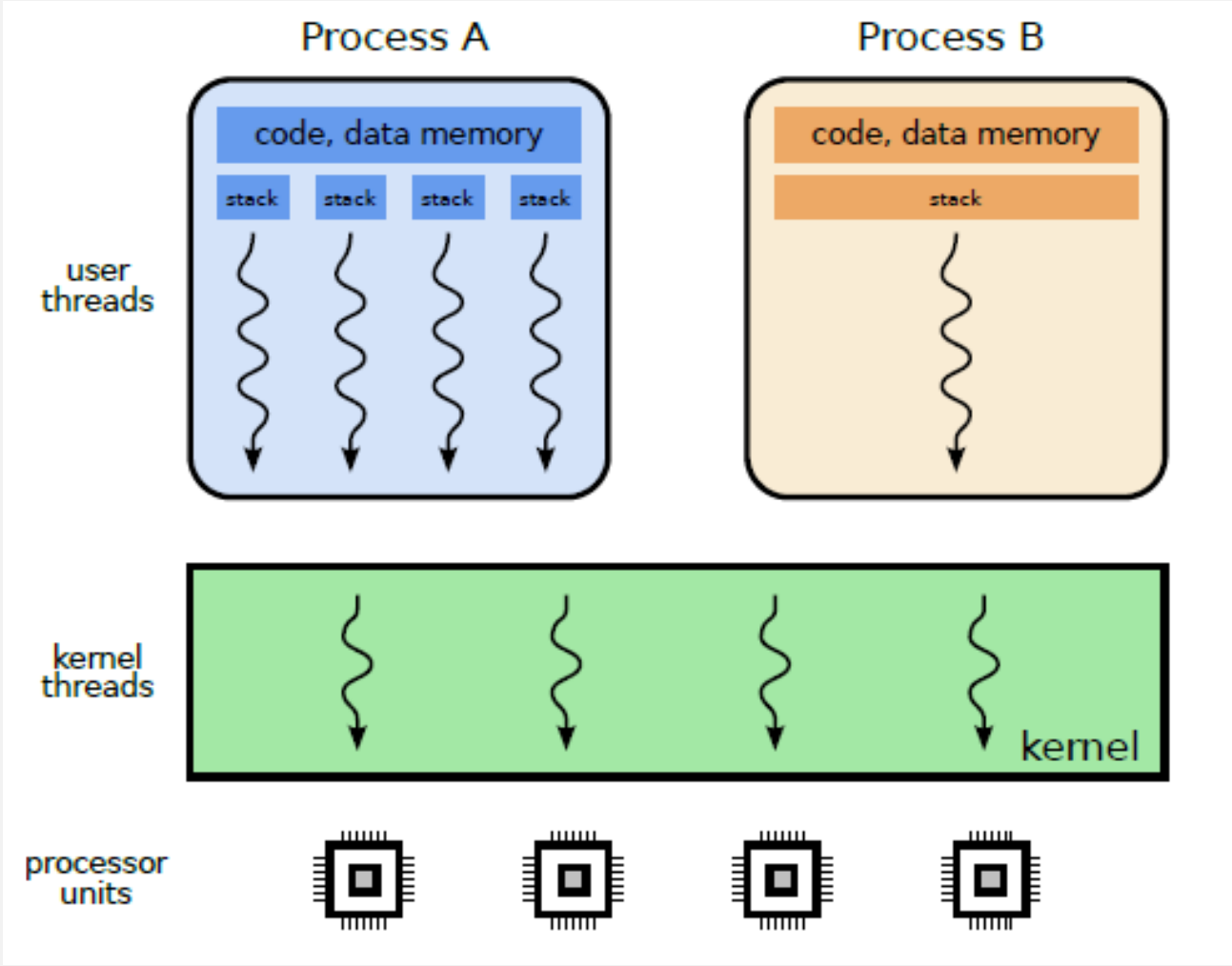
- **Desvantagens**
  - Para a criação de uma thread utiliza-se *system calls* que causam traps (procedimentos lentos)
  - O escalonamento de threads é mais lento
  - O chaveamento de threads de um mesmo processo é mais "custoso" pois depende de uma troca de modo
  - Todas as threads devem possuir o mesmo algoritmo de escalonamento

TIPOS DE THREADS
COMBINANDO NÍVEL DE USUÁRIO E KERNEL



- Se bem implementada, combina as vantagens dos dois modos e minimiza as desvantangens.

TIPOS DE THREADS
COMBINANDO NÍVEL DE USUÁRIO E KERNEL

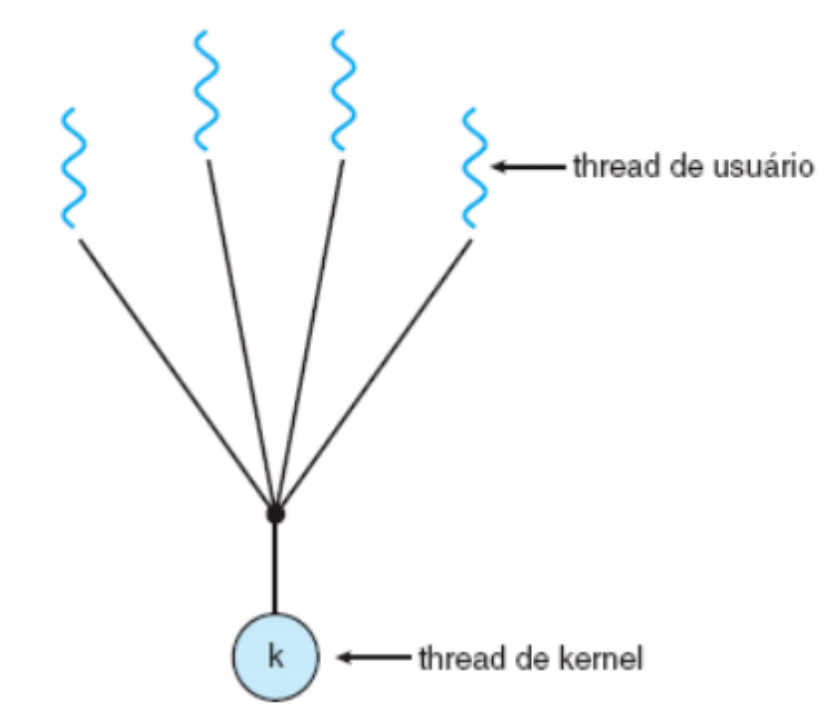


→ O processo A tem várias threads, enquanto o processo B é sequencial (tem uma única thread).

MODELOS DE THREADS

- Deve existir um relacionamento entre as *threads* de usuário e as *threads* de kernel.
- Mapeamento das threads de usuário para as threads de kernel pela biblioteca de threads.

- Essa gerência pode ser feita de diversas formas, conforme os modelos de implementação de threads:
  - **Modelo N:1**
  - **Modelo 1:1**
  - **Modelo N:M**

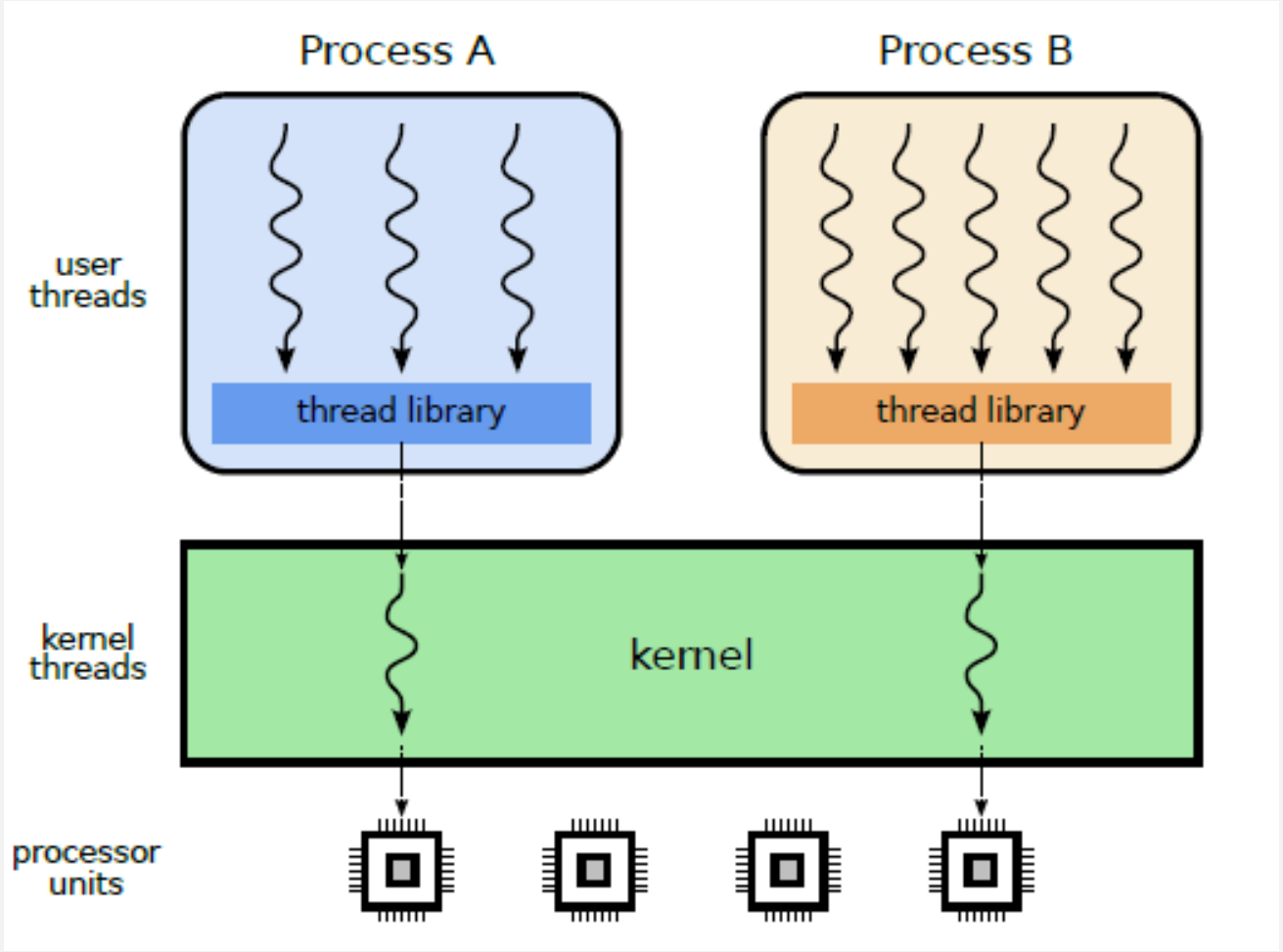


MODELOS DE THREADS – MODELO N:1

- A aplicação pode utilizar várias *threads* conforme sua necessidade, mas o núcleo do SO irá sempre perceber (e gerenciar) apenas um fluxo de execução dentro de cada processo

- **O núcleo mantém apenas uma *thread* de núcleo por processo!**

MODELOS DE THREADS – MODELO N:1



MODELOS DE THREADS – MODELO N:1

- **Vantagens:**
  - Leve e de fácil implementação
  - Como o núcleo vê somente uma *thread*, a carga de gerência imposta ao núcleo é pequena e não depende do número de *threads* da aplicação.
  - Útil em aplicações que necessitam muitas *threads* (jogos, simuladores)

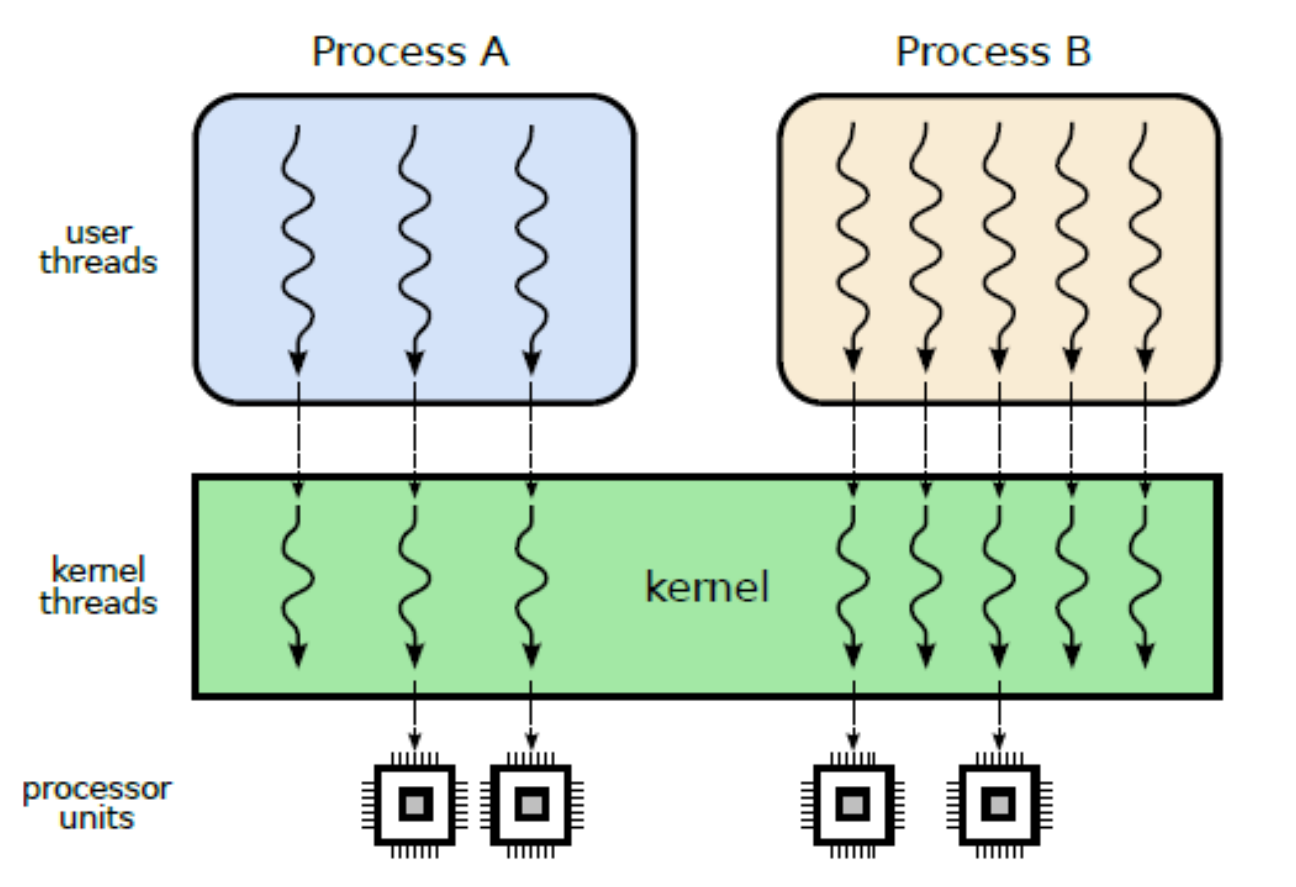
MODELOS DE THREADS – MODELO N:1

- **Problemas:**
  - Operações de E/S realizadas pelo núcleo – se uma thread de usuário solicitar uma leitura do teclado, por exemplo, a thread de núcleo correspondente ficará suspensa até a conclusão da operação, bloqueando todas as outras threads daquele processo.
  - O núcleo do sistema divide o tempo de processamento entre as *threads* de núcleo – um processo com 100 *threads* irá receber o mesmo tempo de CPU do que outro processo com apenas 1 *thread*
  - *Threads* do mesmo processo não podem executar em paralelo – o núcleo somente escalona as *threads* de núcleo nos processadores

MODELOS DE THREADS – MODELO 1:1

- Incorporação da gerência das *threads* dos processos no núcleo do SO
- **Para cada *thread* de usuário é associada uma *thread* correspondente dentro do núcleo!**
- Modelo mais frequente nos SOs atuais.

MODELOS DE THREADS – MODELO 1:1



MODELOS DE THREADS – MODELO 1:1

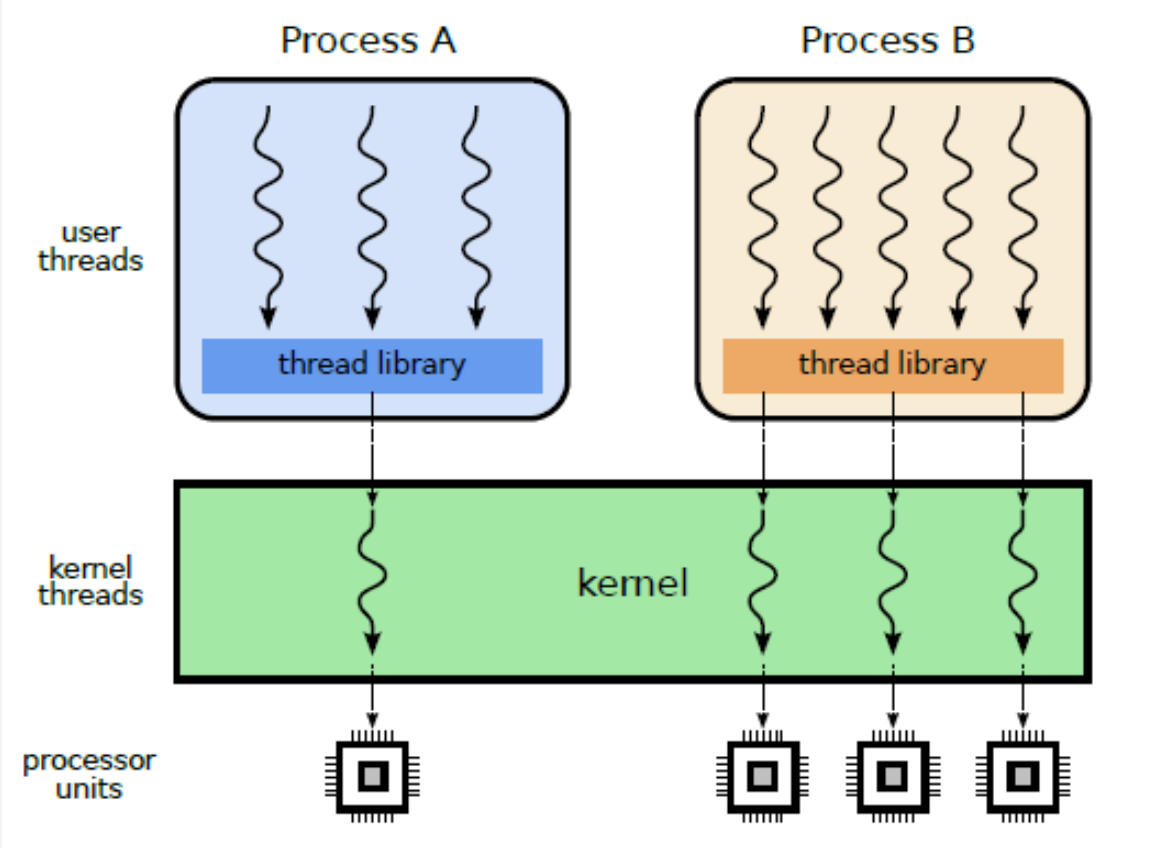
- **Vantagens:**
  - Resolve a questão de *thread* bloqueada (bloqueio de uma thread não afeta as demais em execução)
  - Distribuição de processamento é mais justa (mais threads do mesmo processo podem executar ao mesmo tempo)
- **Problemas:**
  - Escalabilidade (criação de um número muito grande de *threads* impõe uma carga elevada ao núcleo do SO, inviabilizando aplicações com muitos processos)

MODELOS DE THREADS – MODELO N:M

- Modelo híbrido: uma biblioteca gerencia um conjunto de N *threads* de usuário (dentro do processo), que é mapeado em M< N *threads* no núcleo.

- O conjunto de threads de núcleo associadas a um processo, denominado *thread pool*, geralmente contém:
  - **Uma *thread* para cada *thread* de usuário bloqueada**
  - **Uma *thread* para cada processador disponível**
- Esse conjunto é ajustado dinamicamente, conforme necessidade da aplicação.

MODELOS DE THREADS – MODELO N:M



MODELOS DE THREADS – MODELO N:M

- Alia as vantagens de maior interatividade do modelo 1:1 à maior escalabilidade do modelo N:1

- Desvantagem:
  - Complexidade de implementação
  - Maior custo de gerência das *threads* de núcleo, quando comparado aos outros modelos.

COMPARAÇÃO MODELOS DE THREADS

Modelo	**N:1**	**1:1**	**N:M**
Resumo	N *threads* do processo mapeados em uma *thread* de núcleo	Cada *thread* do processo mapeado em uma *thread* de núcleo	N *threads* do processo mapeados em M< N *threads* de núcleo
Implementação	no processo (biblioteca)	no núcleo	em ambos
Complexidade	baixa	média	alta
Custo de gerência	baixo	médio	alto
Escalabilidade	alta	baixa	alta
Paralelismo entre *threads* do mesmo processo	não	sim	sim
Troca de contexto entre *threads* do mesmo processo	rápida	lenta	rápida
Divisão de recursos entre tarefas	injusta	justa	variável, pois o mapeamento *thread*→ processador é dinâmico
Exemplos	GNU Portable Threads, Microsoft UMS	Windows, Linux	Solaris, FreeBSD KSE

USO DE PROCESSOS X THREADS

- **Um processo para cada aplicação**
  - **Robustez** – um erro ocorrido em um programa ficará restrito ao seu espaço de memória
  - **Segurança –** os processos podem ser configurados para executar com usuários (e permissões) distintas, aumentando a segurança
  - **Mecanismos especiais de compartilhamento de dados –** processos não podem acessar áreas de memória dos demais processos (áreas de memória compartilhada)

USO DE PROCESSOS X THREADS

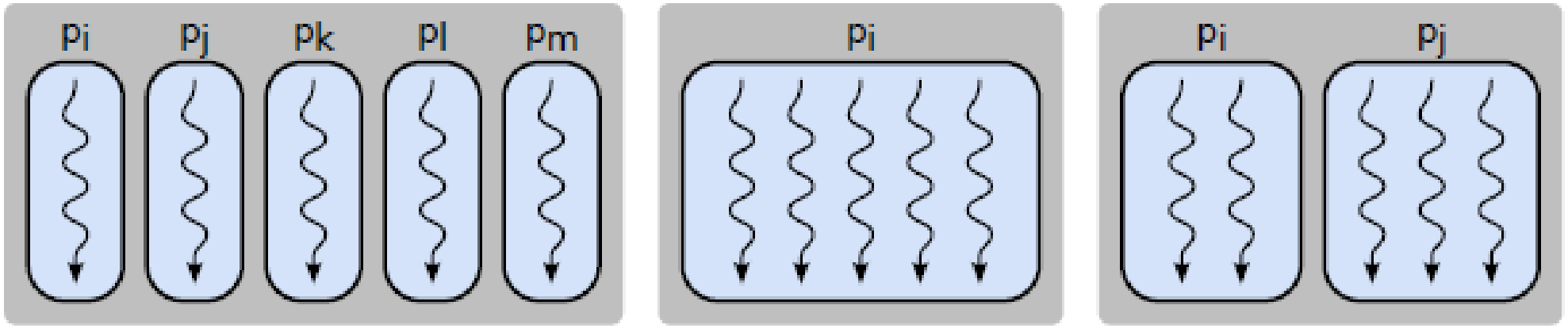
- **Um processo com várias *threads* para cada aplicação**
  - **Simplicidade de interação –** as *threads* compartilham o mesmo espaço de endereçamento e os mesmos recursos (arquivos abertos, conexões de rede, etc.) tornando mais simples a interação entre as *threads*
  - **Execução mais ágil –** é mais rápido criar uma *thread* do que um processo
  - **Risco à robustez –** um erro em um *thread* pode se alastrar às demais, pondo em risco a robustez da aplicação inteira

USO DE PROCESSOS X THREADS

- Sistemas mais recentes e sofisticados usam uma abordagem híbrida, **com o uso de conjunto de processos e *threads*.**

- **A aplicação é composta por vários processos, cada um contendo diversas threads.**
  - Busca-se com isso aliar a robustez proporcionada pelo isolamento entre processos e o desempenho, menor consumo de recursos e facilidade de programação proporcionados pelas *threads*.

USO DE PROCESSOS X THREADS



EXERCÍCIO 1

1. Processo
2. Thread

(1) Agrupa recursos.

(2) Entidade programada para execução na CPU.

(2) Possui um contador de programa que controla qual instrução vai ser executada.

(2) Possui registradores, os quais contêm suas variáveis de trabalho correntes.

EXERCÍCIO 2

- Associe as informações a seguir aos modelos de *threads*:
- ***(1)*** *many-to-one (N:1)* ***(2)*** *one-to-one (1:1)* ***(3)*** *many-to-many (N:M)*

1 A. Tem a implementação mais simples, leve e eficiente.

3 B. Multiplexa as *threads* de usuário em um *pool* de *threads* de núcleo

2 C. Pode impor uma carga muito pesada ao núcleo

1 D. Não permite explorar a presença de várias CPUs pelo mesmo processo

3 E. Permite uma maior concorrência sem impor muita carga ao núcleo

1 F. Se uma *thread* bloquear, todos as demais têm de esperar por ela.

2 G. Cada thread no nível do usuário tem sua correspondente dentro do núcleo.

3 H. É o modelo com implementação mais complexa.

PRÓXIMA AULA

- Comunicação e Sincronização entre Processos

BIBLIOGRAFIA

- Tanenbaum, A. S. **Sistemas Operacionais Modernos.** Pearson Prentice Hall. 3rd Ed., 2009.
- Silberschatz, A; Galvin, P. B.; Gagne G.; **Fundamentos de Sistemas Operacionais.** LTC. 9th Ed., 2015.
- Stallings, W.; **Operating Systems: Internals and Design Principles**. Prentice Hall. 5th Ed., 2005.
- Maziero, C. A.; **Sistemas Operacionais: Conceitos e Mecanismos.** DINF – UFPR, 2019.
- *baseado nos slides da Prof.ª Roberta Gomes (UFES) e do Prof. Felippe Fernandes da Silva