Programação Concorrente

Processos e Threads

Prof. Eduardo Alchieri

Processos

- O conceito mais central em qualquer sistema operacional é o processo
  - Uma abstração de um programa em execução
    - Um **programa** por si só não é um processo. Um programa é uma **entidade passiva**, enquanto o **processo** é uma **entidade ativa**
    - Processos são programas em execução, constituidos por: código executável, pilha de execução, estado do processo, prioridade do processo, valor do contador de programa (registrador PC), valor do apontador de pilha (registrador SP), valores de demais registradores

Processos

- Processo x Programa
  - Exemplo: preparação de bolo de aniversário
    - A receita – programa (algoritmo)
    - Os ingredientes – dados de entrada
    - O cozinheiro – processador
    - Atividade de preparar o bolo (processo)
    - Caso o filho do cozinheiro for picado por uma abelha – interrupção e chaveamento para um novo processo
      - Processo de fornecer cuidados médicos: livro de primeiros socorros (programa), remédios (dados de entrada), o cozinheiro (processador) e atividade de cuidar da picada (processo)
      - Quando terminar, volta para o processo de preparar o bolo

Processos
(modelo de processos)

- Através de processos é possível ter "operações concorrentes"
  - Transformam um única CPU em várias CPUs virtuais
  - Pseudoparalelismo: rápida alternância entre os processos na CPU (multiprogramação)

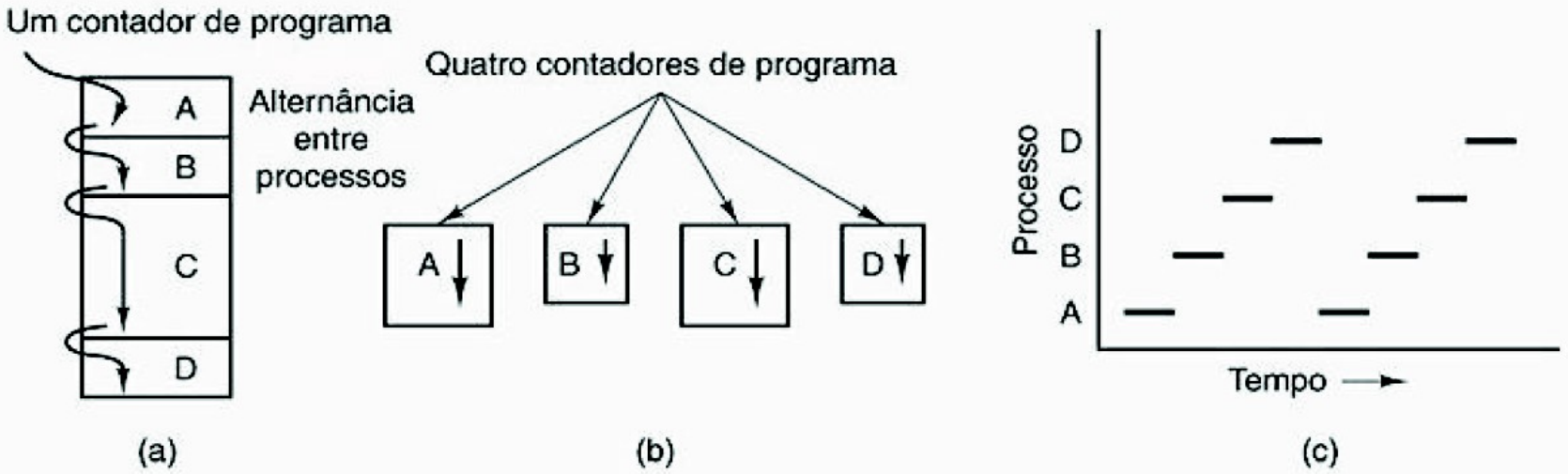


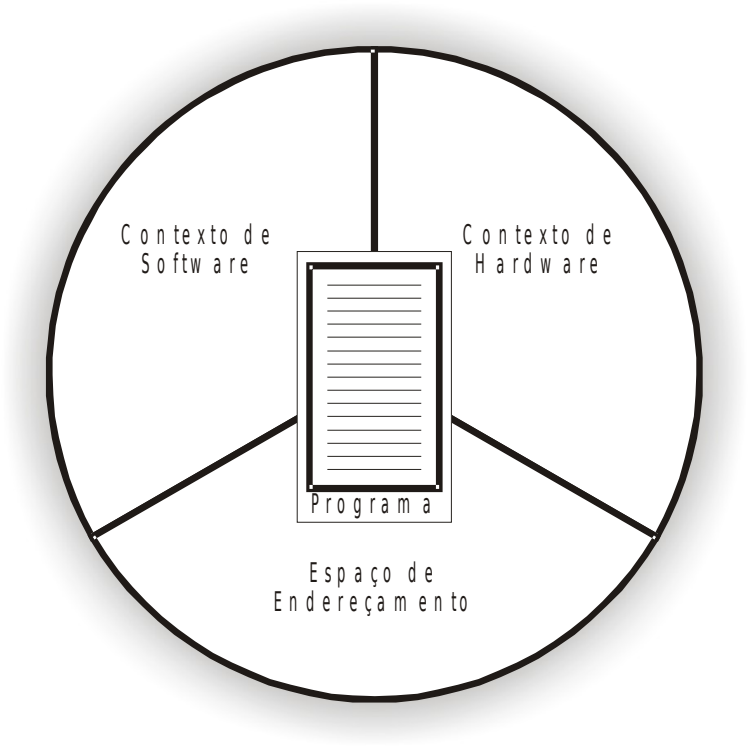
Figura 2.1 (a) Multiprogramação de quatro programas. (b) Modelo conceitual de quatro processos sequenciais independentes. (c) Somente um programa está ativo a cada momento.

Processos
(modelo de processos)

- Recursos utilizados por um processo
  - Memória: um processo precisa de memória para armazenar suas instruções e seus dados
  - CPU: um processo precisa estar de posse da CPU para executar suas instruções
  - Dispositivos: via de regra, um processo precisa realizar alguma tarefa de entrada e saída, em algum dispositivo, como por exemplo receber o valor de uma variável via teclado, ler um disco, etc.
  - Arquivos: um processo, geralmente, precisa gravar ou recuperar alguma informação armazenada em determinado arquivo

Processos
(modelo de processos)

- Um processo é formado por três elementos básicos:
  - Contexto de hardware
  - Contexto de software
  - Espaço de endereçamento

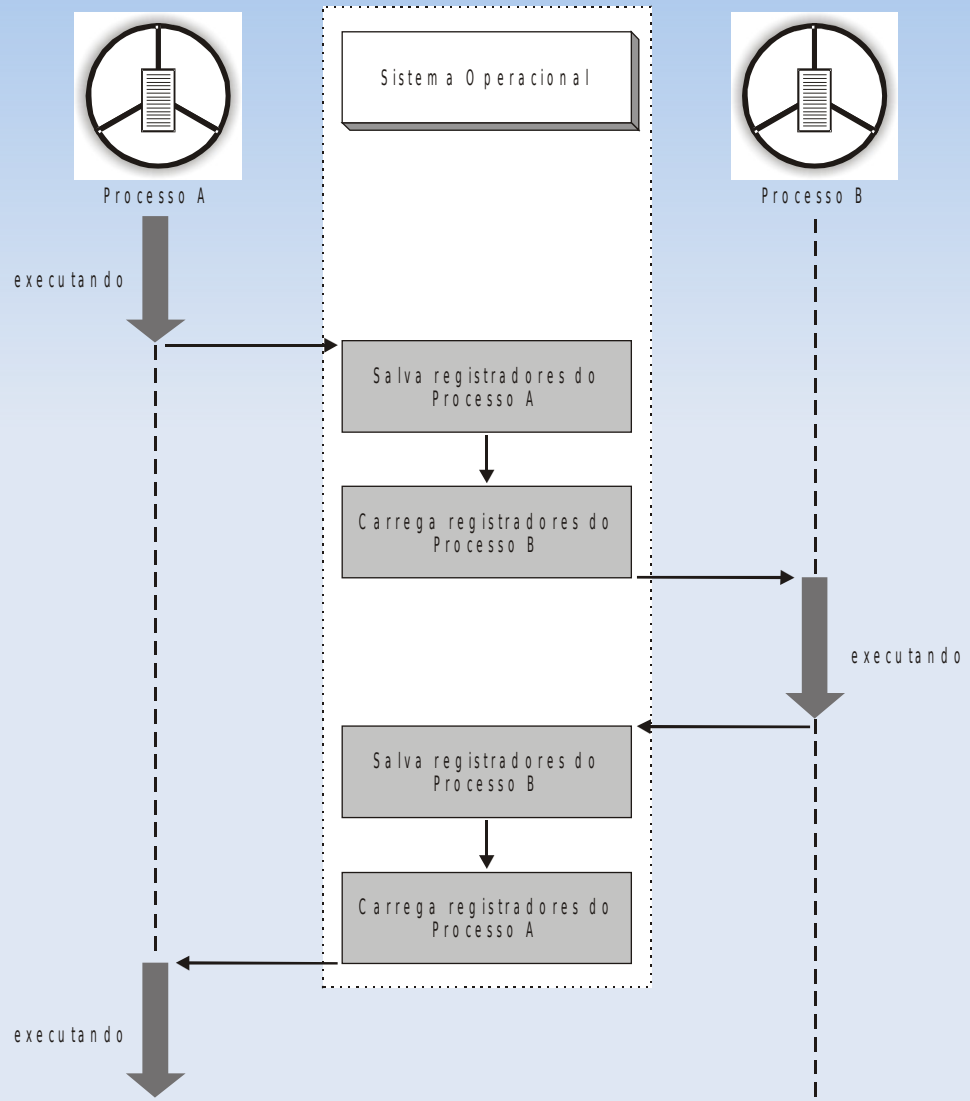


Processos
(modelo de processos)

- Contexto de hardware
  - O contexto de hardware constituí-se, basicamente, do conteúdo de registradores (PC, SP, etc.)
  - Quando um processo está em execução, o seu contexto de hardware está armazenado nos registradores do processador
  - O contexto de hardware é fundamental para a implementação dos sistema de tempo compartilhado (multiprogramados), no qual os processos se revezam na utilização do processador
  - A troca de um processo por outro na CPU, realizado pelo SO, é denominada **Troca de Contexto**

Processos
(modelo de processos)

- Contexto de hardware
  - Na troca de contexto, o SO armazena o conteúdo dos registradores da CPU

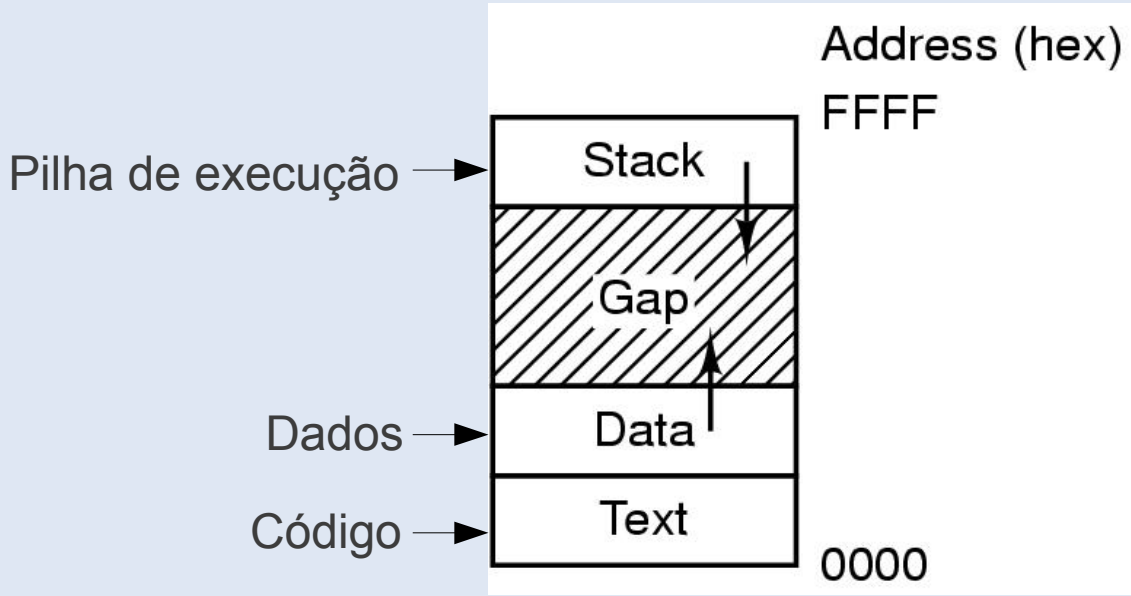


Processos
(modelo de processos)

- Contexto de software
  - O contexto de software especifica características do processo que influênciam na execução de um programa
  - Composto por 3 grupos de informações:
    - Identificação
      - **PID** (*Process Identification*) – cada processo criado pelo sistema recebe uma identificação única. As informações sobre um processo estão armazenadas na tabela de processos, indexada pelo PID
      - **UID** (*User Identification*) – cada processo deve possuir também uma identificação do usuário
    - Quotas
      - São os limites de cada recurso do sistema (arquivos, memória, etc) que um processo pode alocar
    - Privilégios
      - Definem o que o processo pode ou não fazer em relação ao sistema e aos outros processos

Processos
(modelo de processos)

- Espaço de endereçamento
  - Área de memória onde o programa será armazenado, além do espaço para os dados utilizados por ele
  - Cada processo possui seu próprio espaço de endereçamento, que deve ser protegido do acesso dos demais processos



Processos
(criação de processos)

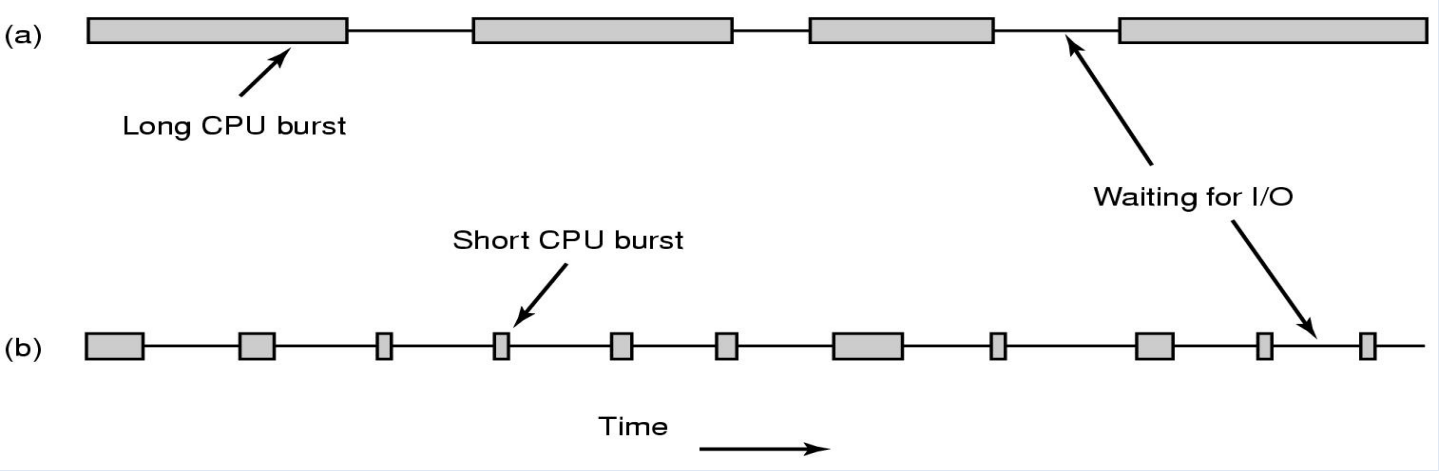
- Principais eventos que causam a criação de um processo
  - Inicialização do sistema
    - Na inicialização vários processos são criados
    - Exemplo: processo para receber requisições de páginas Web
  - Execução de uma chamada de sistema de criação de processo por um processo em execução
    - Processos pais criam processos filhos, que podem criar seus próprios processos filhos e assim por diante
    - Exemplo: no UNIX a chamada fork
  - Uma requisição do usuário para criar um novo processo
    - Em sistemas interativos, digitando um comando ou clicando em um ícone
  - Início de uma tarefa em lote
    - Em computadores de grande porte, quando o mesmo possui recursos suficientes para executar a tarefa

Processos
(término de processos)

- Condições que terminam um processo
  - Saída Normal (voluntária)
    - Quando terminaram seu trabalho
  - Saída por erro (voluntária)
    - Processo descobre um erro fatal
    - Exemplo: compilar um arquivo que não existe
  - Erro fatal (involuntária)
    - Erro causado pelo processo, normalmente erro de programação
    - Exemplo: divisão por zero, referência para memória inexistente
  - Cancelamento por um outro processo (involuntária)
    - Um processo executa uma chamada de sistema dizendo ao SO para cancelar algum outro processo
    - Exemplo: no UNIX a chamada kill

Processos
(tipos de processos)

- Os processos são classificados de acordo com o tipo de processamento que realizam
- Existem dois tipos de processos:
  - CPU-bound: um processo é dito CPU-bound quando passa a maior parte do tempo utilizando o processador
  - I/O-bound:um processo é dito I/O-bound quando passa a maior parte do tempo realizando operações de I/O (E/S)



- Períodos de uso de CPU alternados com períodos de espera por operações de E/S
  - a) Processo CPU-bound
  - b) Processo I/O bound

Processos
(estados de processos)

- Consiste da atividade atual do processo
- Há várias formas de representar os estados de um processo, variando o detalhamento dos estados possíveis

- Segundo Tanembaum

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

- Estados
  - Em execução: o processo tem o controle da CPU (processador)
  - Pronto: aguarda sua vez de usar a CPU
  - Bloqueado: está ocioso e aguarda o ocorrência de algum evento externo

Processos
(implementação de processos)

- Todas as informações sobre um processo são mantidas na tabela de processos
- A tabela de processos tem campos que dizem respeito:
  - à gerência do processo
  - à gerência da memória
  - à gerência de arquivos
- A tabela de processos possui uma entrada por processo e os campos nela contidos variam de sistema operacional para sistema operacional

Processos
(implementação de processos)

- Alguns campos típicos de um processo na tabela

Gerenciamento de processo	Gerenciamento de memória	Gerenciamento de arquivo
Registros	Ponteiro para informações sobre o segmento de código	Diretório-raiz
Contador de programa		Diretório de trabalho
Palavra de estado do programa	Ponteiro para informações sobre o segmento de dados	Descritores de arquivo
Ponteiro da pilha		ID do usuário
Estado do processo	Ponteiro para informações sobre o segmento de pilha	ID do grupo
Prioridade		
Parâmetros de escalonamento		
ID do processo		
Processo pai		
Grupo de processo		
Sinais		
Momento em que um processo foi iniciado		
Tempo de CPU usado		
Tempo de CPU do processo filho		
Tempo do alarme seguinte		

Processos
(modelando a multiprogramação)

- Usando multiprogramação, a utilização da CPU pode ser aumentada
  - Com a multiprogramação mais de um processo compete pela CPU, de forma que quando um processo estiver aguardando por algum evento (ex.: E/S) outro processo pode utilizar a CPU

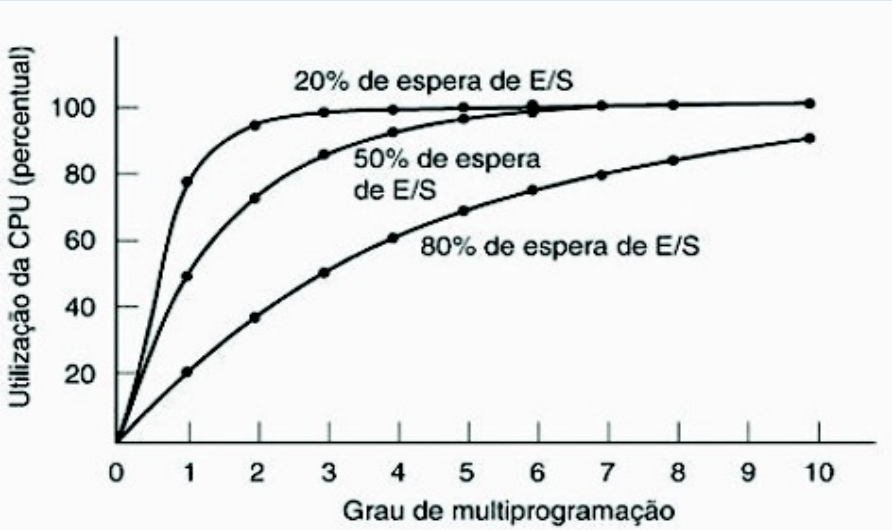


Figura 2.4 Utilização da CPU como função do número de processos na memória.

Processos

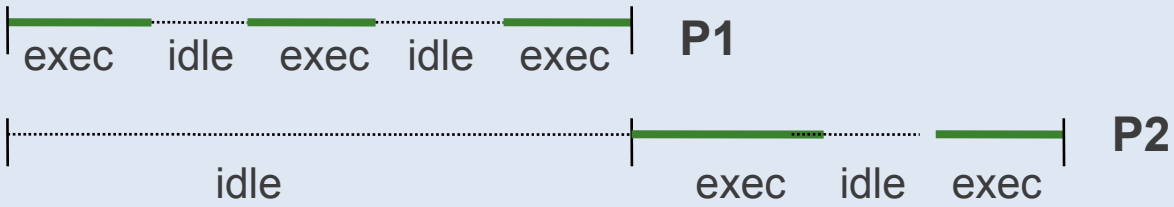
Escalonamento de Processos

Processos
(escalonamento)

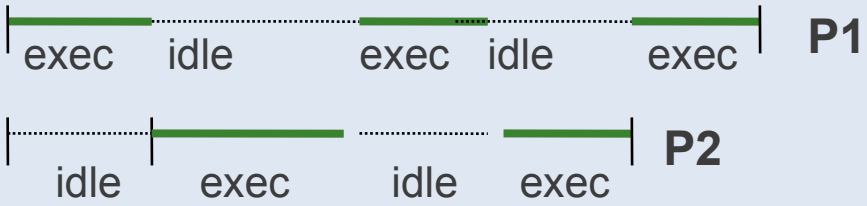
- Em computadores multiprogramados, muitas vezes múltiplos processos (ou threads) competem pelo uso da CPU ao mesmo tempo
  - Isso ocorre sempre que dois ou mais processos estão simultaneamente no estado pronto
- Nestes casos, quando o CPU se encontrar disponível, deverá ser feita uma escolha sobre qual processo executar
  - A parte do sistema operacional que faz esta escolha é o **escalonador** e o algoritmo que ele usa é o **algoritmo de escalonamento**
  - É tarefa do escalonador também determinar quanto tempo o processo poderá utilizar a CPU
  - O algoritmo de escalonamento define, assim, a política de utilização do processador pelos processos

Processos
(escalonamento)

- Quando um processo solicita operações bloqueantes (E/S, por exemplo), sua execução fica suspensa até que o evento solicitado ocorra
  - Se outro processo estiver pronto para execução, o mesmo poderá passar a utilizar a CPU, maximizando a utilização da mesma, melhorando o desempenho percebido do sistema
- Execução de 2 processos sem concorrência



- Execução de 2 processos com concorrência



Threads

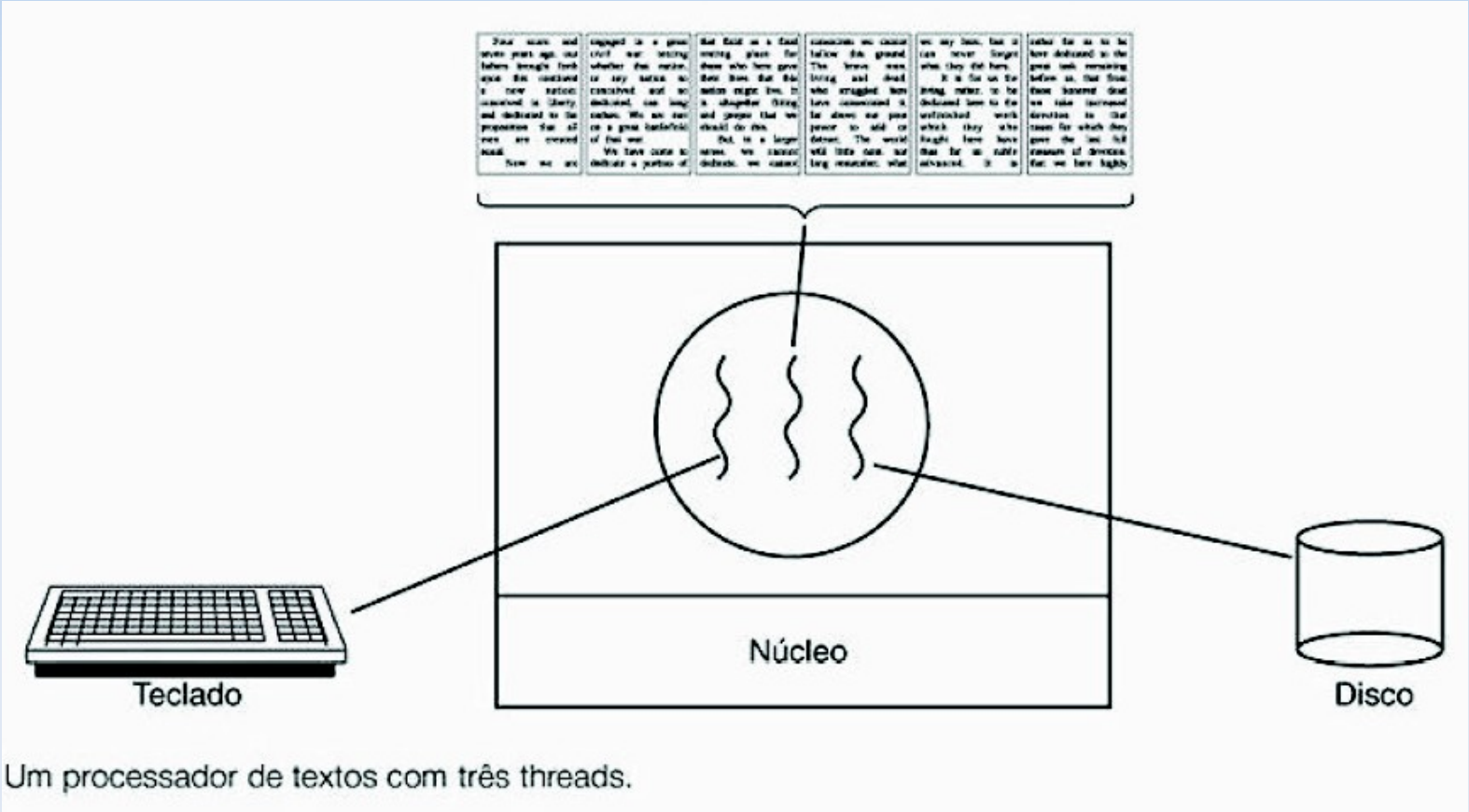
Threads

Threads

- O modelo de processos estudado supõe o uso de apenas uma *thread* por processo
  - Cada processo tem o seu espaço de endereçamento e um único thread de controle
- Contudo, frequentemente há situações em que é desejável ter múltiplos threads de controle no mesmo espaço de endereçamento executando em pseudo paralelo, como se fossem processos separados
  - Exceto pelo espaço de endereçamento compartilhado
- Intuitivamente, threads são como processos dentro de outo processo (são linhas/fluxo de execução)
- Mas, quando é desejável utilizar mais de uma thread?

Threads

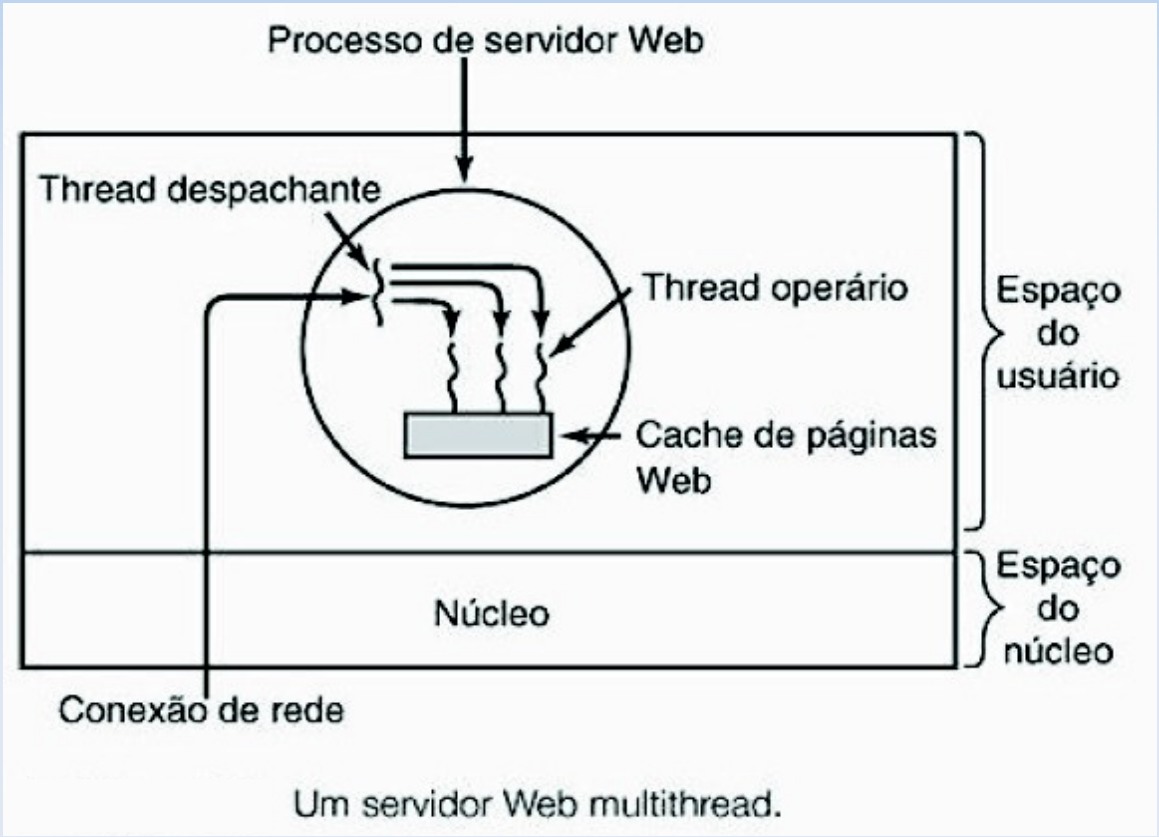
- Exemplo 1: Processador de textos



Um processador de textos com três threads.

Threads

- Exemplo 2: Servidor Web



Um servidor Web multithread.

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

(a)

```
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);
}
```

Threads
(modelo clássico)

- O modelo de processos é baseado em dois conceitos independentes:
  - Agrupamento de recursos: espaço de endereçamento, arquivos abertos, etc. (todos os recursos necessários para realizar alguma tarefa). Aglutinar estes recursos na forma de um processo facilita o gerenciamento destes recursos.
  - Execução: representa o thread de execução do processo, o qual contém:
    - Um contador de programa que aponta para a próxima instrução a ser executada
    - Registradores, que contém as variáveis de trabalho atuais
    - A pilha que traz a história da execução, com uma estrutura para cada rotina chamada mas ainda não terminada
- Apesar de threads serem executadas em processos, ambos são conceitos diferentes e podem ser tratados separadamente
  - Processos são usados para agrupar recursos
  - Threads são as entidades escalonadas para usar a CPU

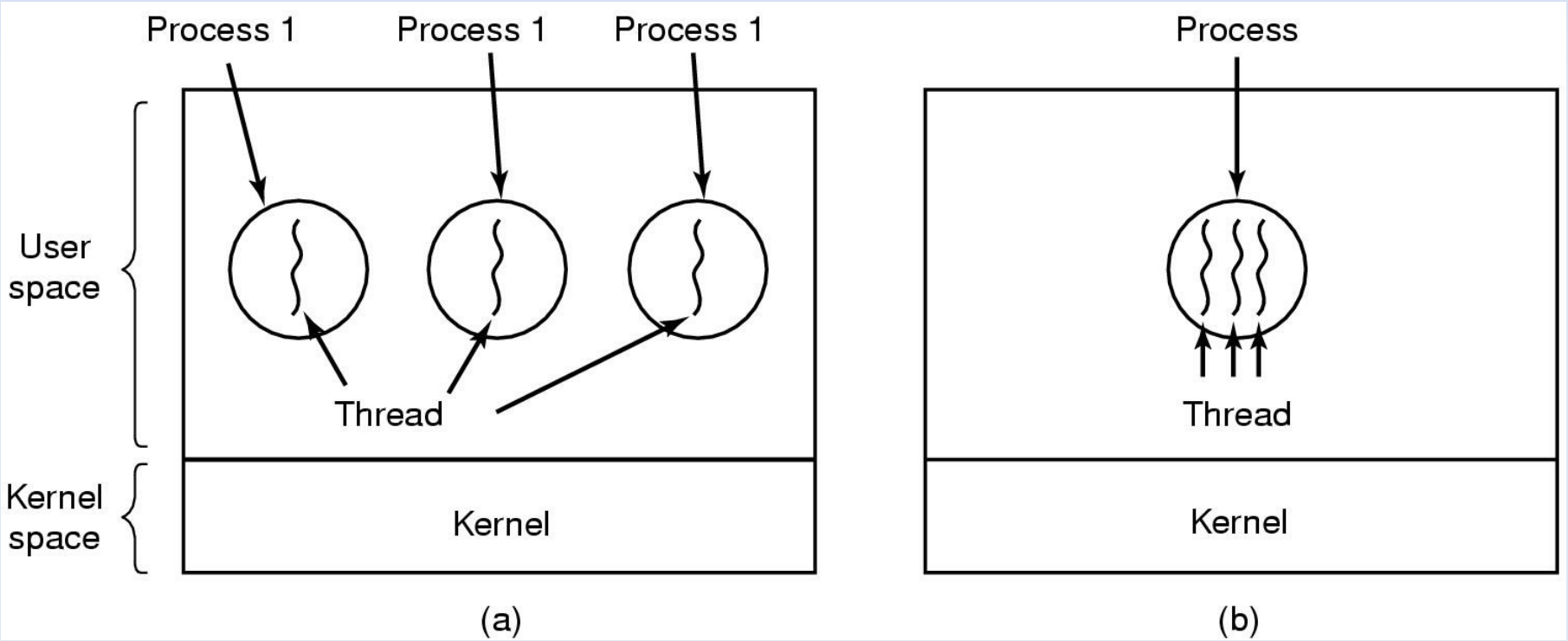
Threads
(modelo clássico)

- O modelo de threads permite que múltiplas execuções ocorram no mesmo ambiente de processo, com um grande grau de independência uma da outra
- Threads (também chamadas de processos leves) são linhas de execução, e compreendem:
  - Id: identificador da thread
  - Endereço da próxima instrução a ser executada
  - Conjunto de registradores em uso
  - Uma pilha de execução
- Threads compartilham com outras threads recursos, como:
  - Trecho de código
  - Dados
  - Arquivos abertos
- **Multithread** é o termo usado para descrever a situação em que é permitido a existência de múltiplos threads em um processo

Threads
(modelo clássico)

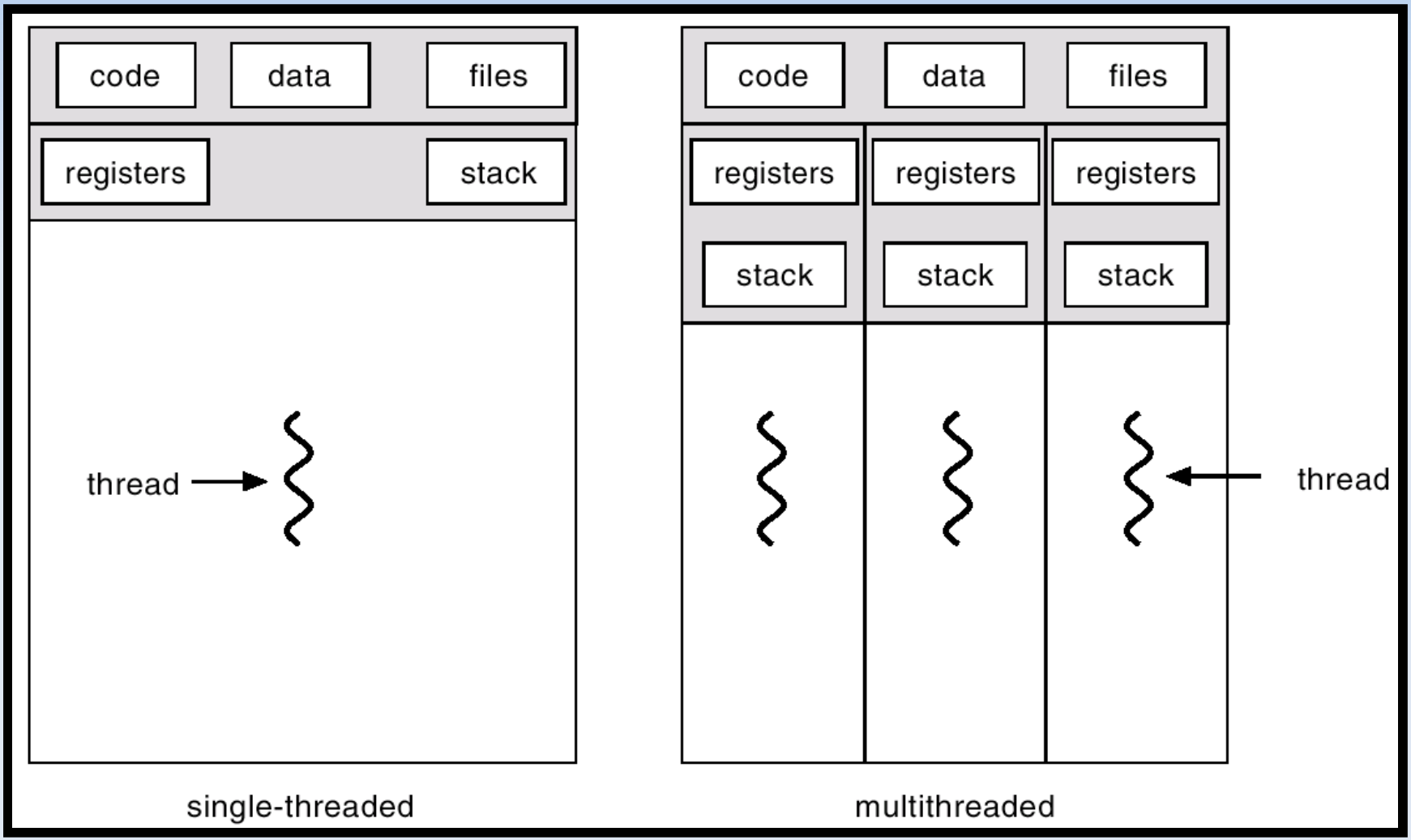
- Uma thread é uma maneira de um programa dividir a si mesmo em duas ou mais tarefas simultâneas

- Processo simples vs. Multithreads



Threads
(modelo clássico)

- Processo simples vs. Multithreads



Threads
(modelo clássico)

- Itens por processo vs. Itens por 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 manipuladores de sinais	
Informação de contabilidade	

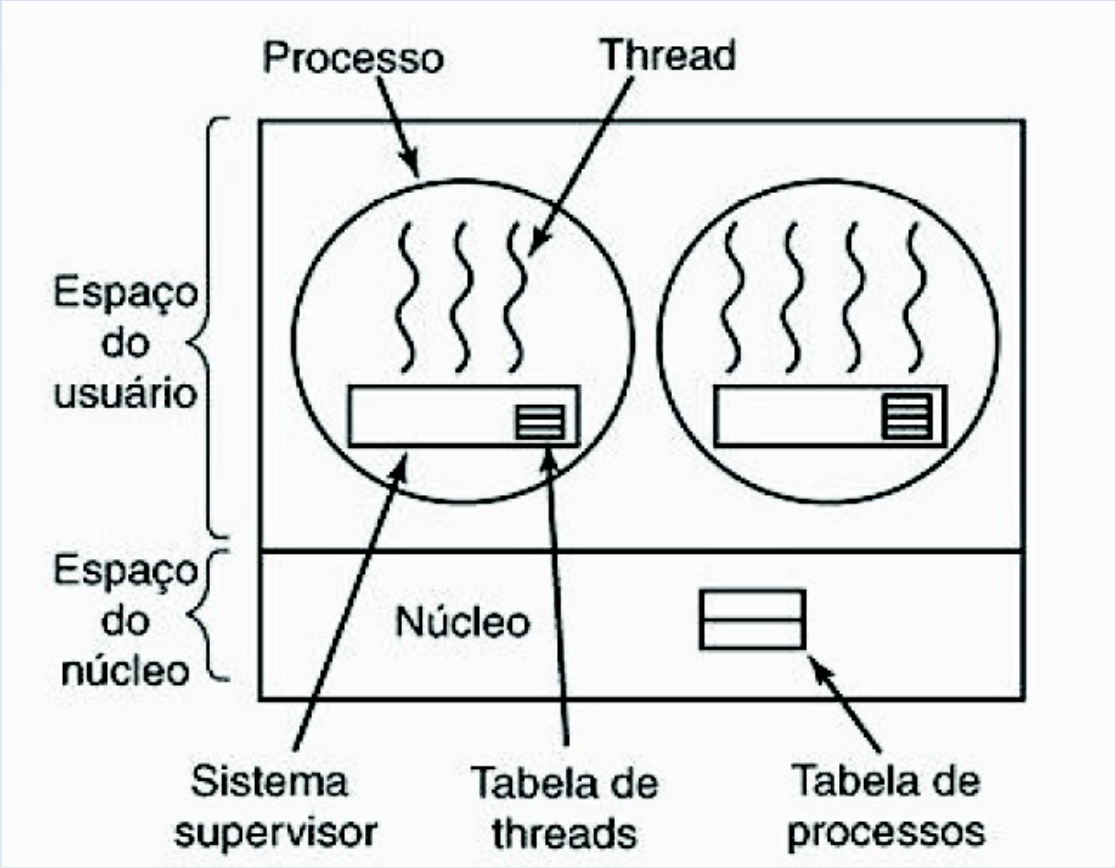
Tabela 2.4 A primeira coluna lista alguns itens compartilhados por todos os threads em um processo. A segunda lista alguns itens específicos a cada thread.

Threads
(vantagens)

- Tempo de resposta
  - Uma aplicação interativa pode continuar sendo executada se parte dela está bloqueada, ou executando uma operação lenta
- Compartilhamento de recursos
  - Por padrão as threads compartilham: memória e qualquer recurso alocado pelo processo ao qual são subordinadas
  - Não é necessária a alocação de mais recursos no sistema
- Economia
  - É mais econômico criar e trocar o contexto das threads
- Utilização de arquiteturas multiprocessadas
  - Cada thread pode ser executada de forma paralela, em processadores distintos

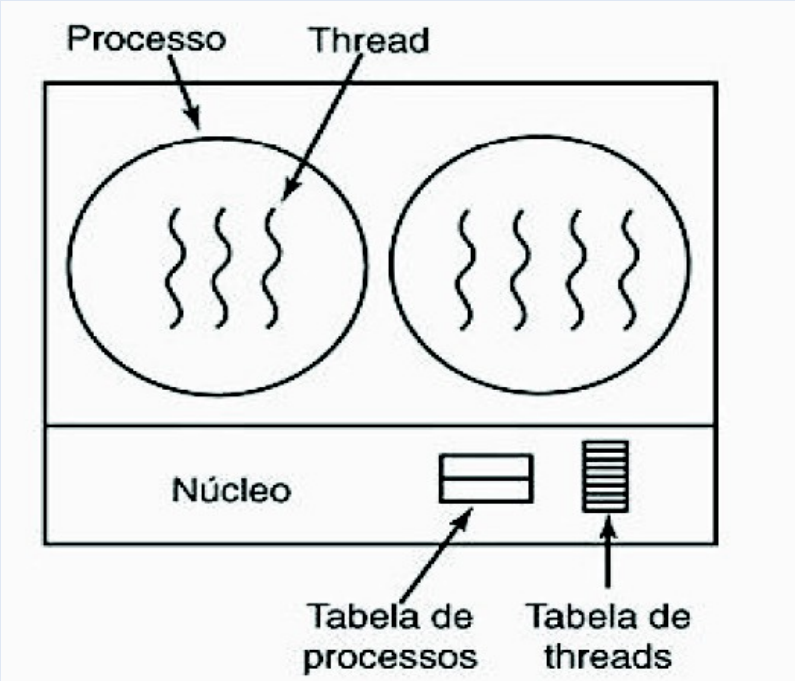
Threads
(gerenciamento)

- Existem dois modos principais de implementar threads
  - Threads no espaço de usuário
    - São admitidas no nível do usuário e gerenciadas sem o suporte do núcleo (kernel)



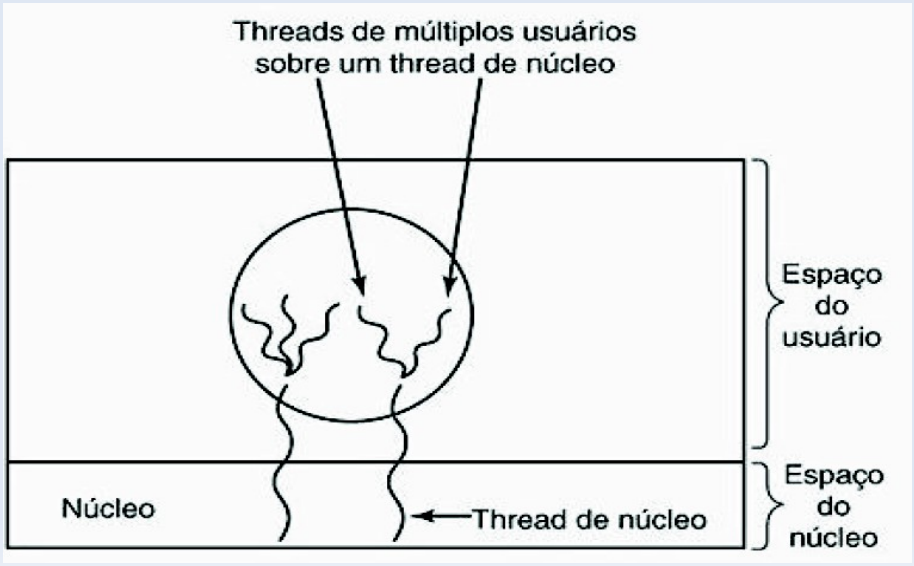
Threads
(gerenciamento)

- Existem dois modos principais de implementar threads
  - Threads no núcleo (kernel)
    - São admitidas e gerenciadas diretamente pelo sistema operacional
    - Quase todos os sistemas operacionais admitem threads de kernel
      - Exemplos: Windows XP, Linux, Mac OS X, Solaris



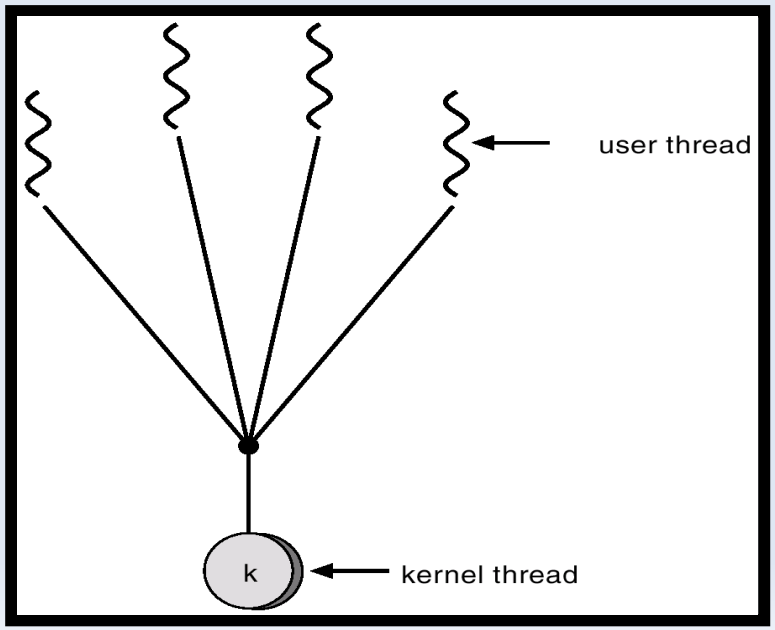
Threads
(gerenciamento)

- Implementação híbridas
  - Tenta combinar as vantagens dois dois modos anteriores
    - Modo usuário: rápida criação e chaveamento entre threads
    - Modo núcleo: o processo todo não é bloqueado pelo bloqueio de uma thread
  - A idéia é utilizar algumas threads de núcleo e multiplexar threads de usuários sobre elas
    - O usuário decide quantas threads utilizar, tendo uma maior flexibilidade



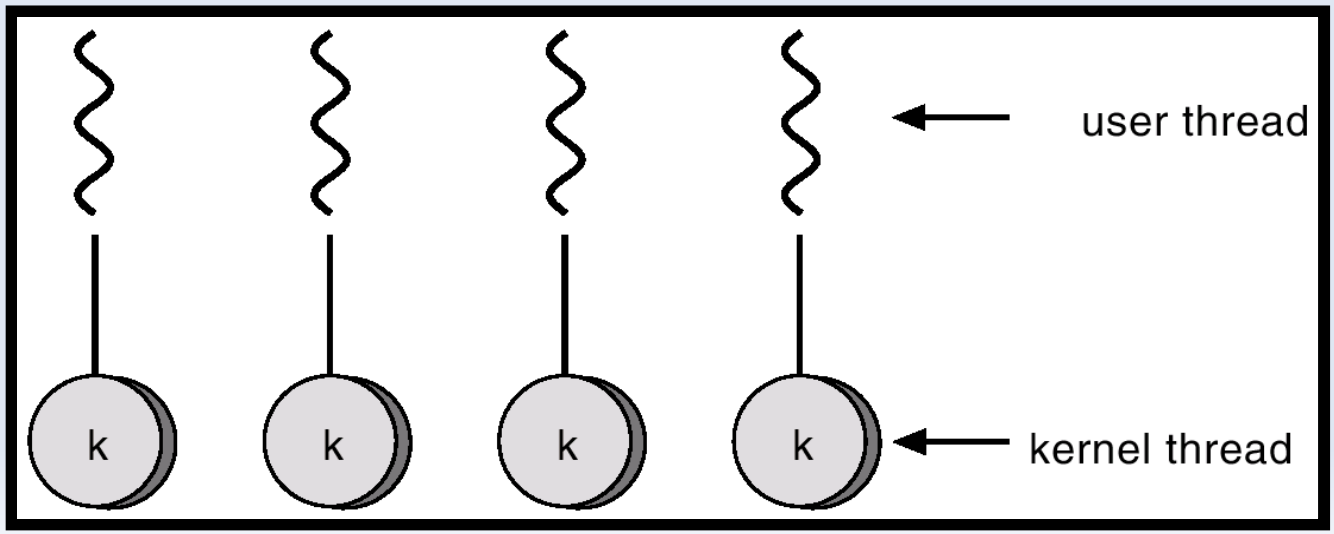
Threads
(modelos de multithreading)

- Modelo N para 1
  - N threads de usuários para 1 thread do núcleo (thread de sistema)
  - O gerenciamento das threads é realizado pela biblioteca de threads no nível de usuário
  - Se um thread fizer uma chamada ao sistema que bloqueia o processamento, todo o processo será bloqueado
  - Utilizado em sistemas que não suportam threads em nível de sistema
  - Alguns sistemas operacionais que suportam este modelo: Solaris e Linux



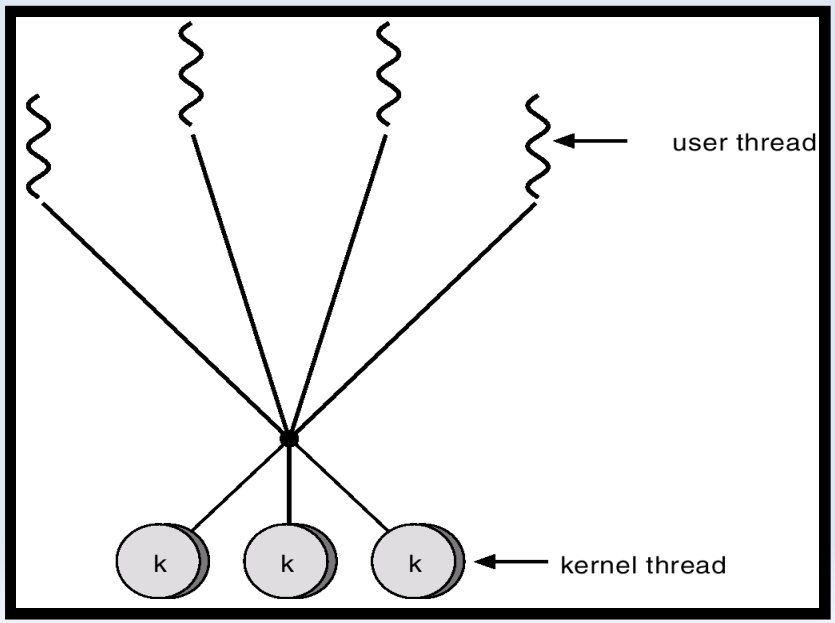
Threads
(modelos de multithreading)

- Modelo 1 para 1
  - Associa cada thread de usuário para 1 thread do núcleo
  - Provê maior concorrência do que o modelo anterior, permitindo que outra thread seja executada quando uma thread faz uma chamada bloqueante
  - Permite que várias threads sejam executadas em paralelo em multiprocessadores
  - Desvantagem: a criação de um thread de usuário é mais rápida do que a criação de uma thread de núcleo
  - Alguns sistemas operacionais que suportam este modelo: versões atuais do Linux, Windows 95/98/NT/2000.



Threads
(modelos de multithreading)

- Modelo N para M
  - N threads de usuário para M threads do núcleo
  - O número de threads de núcleo pode ser específico a determinada aplicação ou a determinada máquina
  - Quando um thread realiza uma chamada de sistema bloqueante, o escalonador do sistema operacional pode escolher outra thread do mesmo processo
  - Alguns sistemas operacionais que suportam este modelo: Solaris 2, Windows NT/2000 (com o Pacote ThreadFiber)



Threads
(bibliotecas de threads)

- Uma biblioteca de thread fornece ao programador uma API (application programming interface) para a criação e gerenciamento de threads
- Existem duas maneiras de implementar bibliotecas
  - Fornecer uma biblioteca no espaço do usuário, sem suporte do núcleo
  - Fornecer uma biblioteca no nível do núcleo, com suporte direto do sistema operacional
- As três bibliotecas de threads mais comuns são:
  - POSIX Pthreads (nível de usuário ou de núcleo)
  - Win32 (nível de núcleo)
  - Java (no nível de usuário, mas usa sempre a biblioteca do sistema operacional hospedeiro)

Threads
(escalonamento de threads)

- Em sistemas operacionais que admitem a criação de threads no nível do núcleo, são as threads que são escalonadas, e não os processos
- As threads em nível de usuário não são conhecidas pelo sistema operacional
- Nas implementações dos modelos N para 1 ou N para M, a biblioteca de suporte a threads (sistema supervior do processo) escalona as threads em nível de usuário
  - Esse esquema é conhecido como Escopo de Disputa do Processo (*Process Contention Scope* – PCS)
  - Pois a disputa pela CPU ocorre entre as *threads* pertencentes ao mesmo processo

Threads
(escalonamento de threads)

- As *threads* no nível do núcleo são escalonadas para o uso da CPU
  - Esse esquema é conhecido como Escopo de Disputa do Sistema (*System Contention Scope* – SCS)
  - A disputa pela CPU com escalonamento SCS ocorre entre todas as *threads* no sistema
- Os sistemas operacionais que utilizam o modelo 1 para 1 (Windows XP, Solaris 9 e Linux) escalonam as *threads* usando apenas o SCS
- Normalmente, o PCS é realizado de acordo com a prioridade, o escalonador seleciona a *thread* executável com a maior prioridade para execução
  - As prioridades das *threads* do usuário são definidas pelo programador