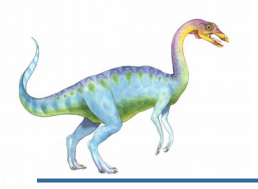
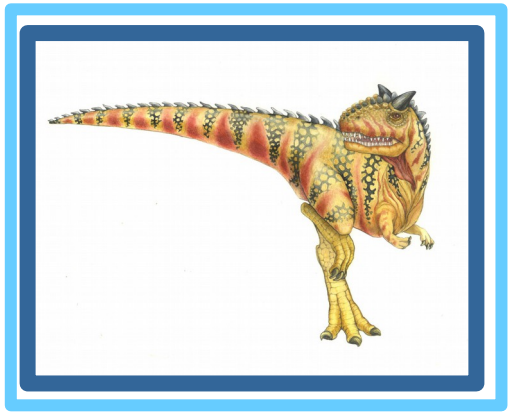
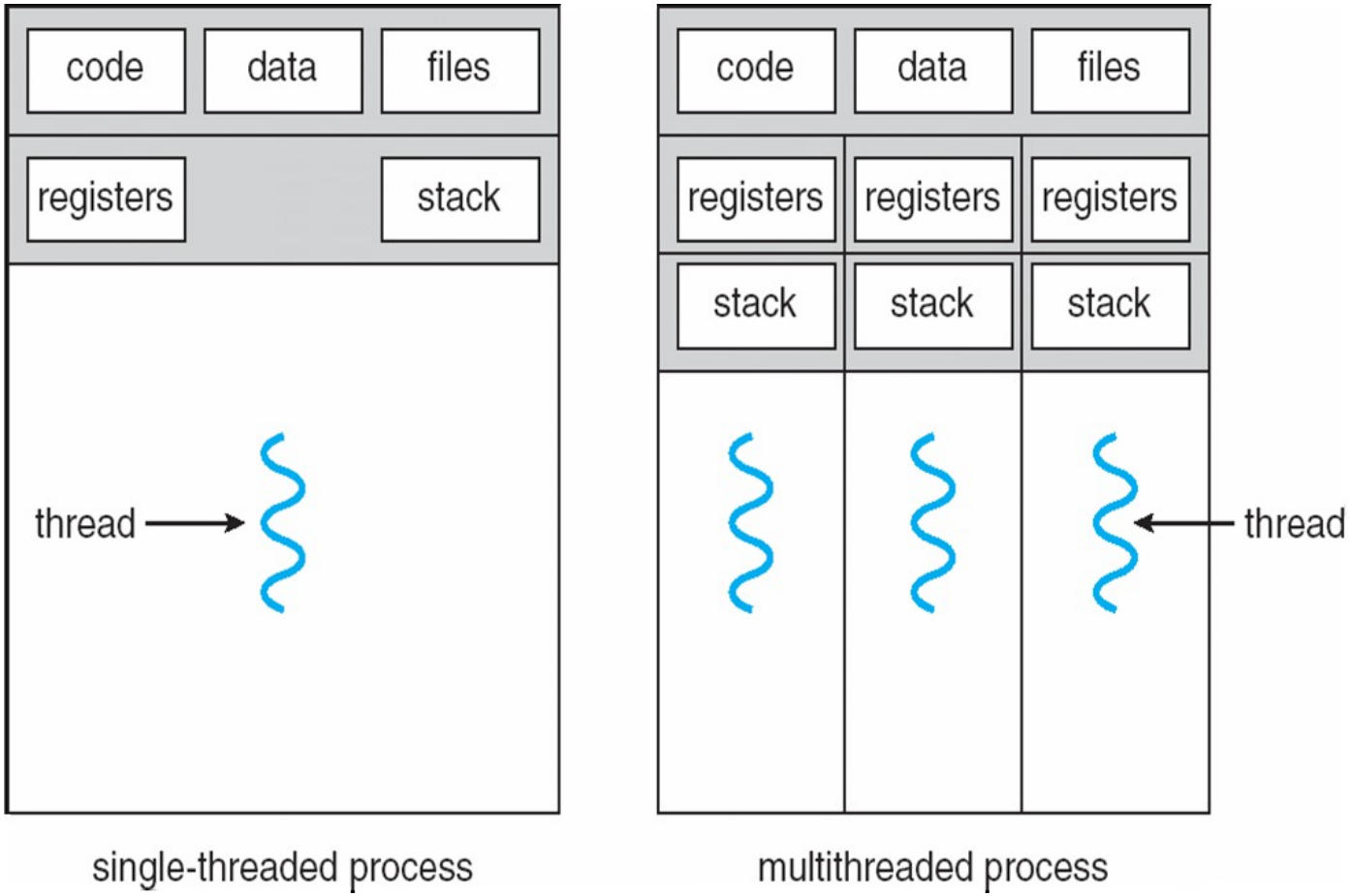


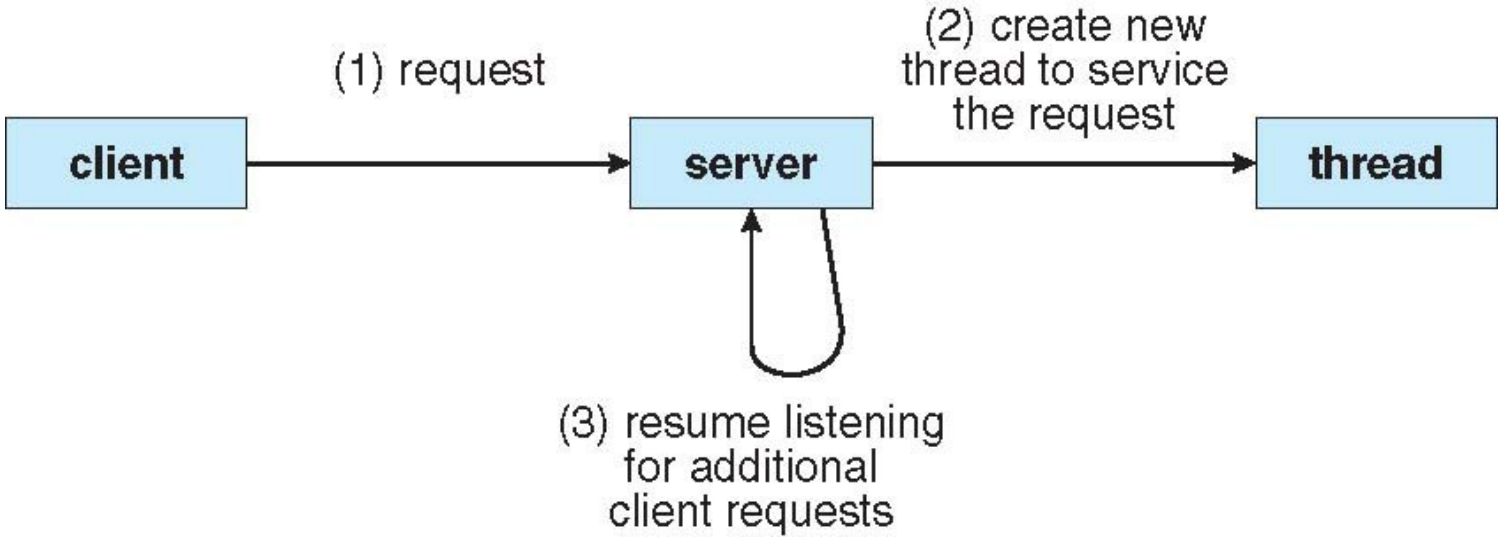
Capítulo 4: Threads



Processos com Uma e Várias Threads (Single e Multithreaded)



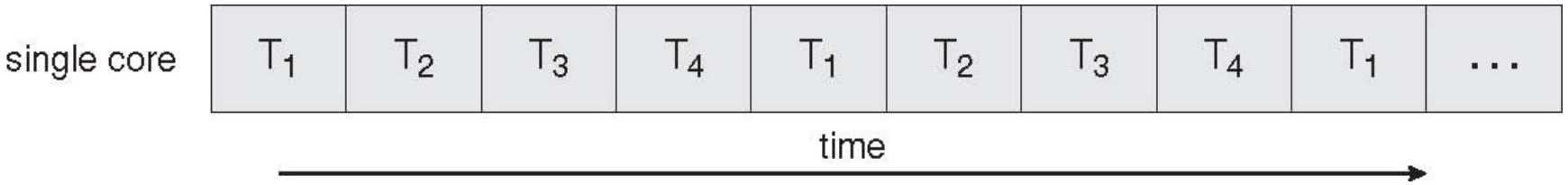
Arquitetura de Servidores Multithread



Benefícios

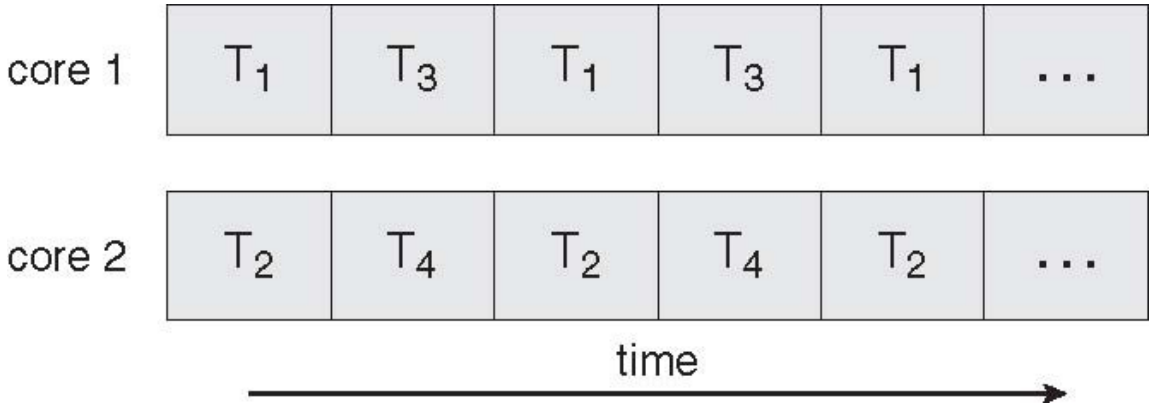
- Tempo de resposta
  - programa dividido em várias linhas de execução
  - Ex. Writer: user interface, keystrokes, spellchecker, file I/O
- Compartilhamento de recursos
  - threads compartilham código e dados (ex.: var. globais) do processo pai
  - processos precisam de mecanismos de IPC (shared mem., msg.)
- Economia
  - É mais barato criar threads no processo do que criar novos processos filhos
  - Ex. Solaris: criação 30x +rápida, troca de contexto 5x +rápida
- Escalabilidade
  - threads podem rodar em paralelo em diferentes núcleos de CPU
  - um processo de uma thread só pode rodar em um núcleo só

Execução **Concorrente** em um Sistema de UM Núcleo



1 CPU com 1 núcleo executa cada threads concorrentemente
Neste exemplo dual-core, cada um das 4 threads é executada novamente após 4 ciclos de clock

Execução **Paralela** em um Sistema Multicore



2 CPUs com 1 núcleo (ou 1 CPU com 2 núcleos) executa 2 threads paralelamente
Neste exemplo dual-core, cada um das 4 threads é executada novamente após 2 ciclos de clock

Sistemas atuais oferecem melhor desempenho com HW que melhora o desempenho de *threads*

Ex.: há CPUs que suportam 1, **2, 4 ou até 8 *threads* por núcleo**

OBS.: **cada núcleo executa 1 thread por vez**, porém as outras *threads* já estão nele carregadas, acelerando a troca de contexto entre elas

- Sistemas *multicore* proveem **novos desafios para os programadores** utilizarem melhor os múltiplos núcleos de computação:
  - **Identificar tarefas**
    seções do programa que podem ser divididas entre *threads*
  - **Balanceamento**
    equalizar o trabalho a ser realizado por cada thread
  - **Divisão dos dados**
    para que possam ser acessados/manipulados em *multicore*
  - **Dependência de dados**
    entre duas ou mais *threads*, cujo acesso deve ser sincronizado
  - **Teste e depuração**
    muitos caminhos diferentes de execução são possíveis ao rodar o programa

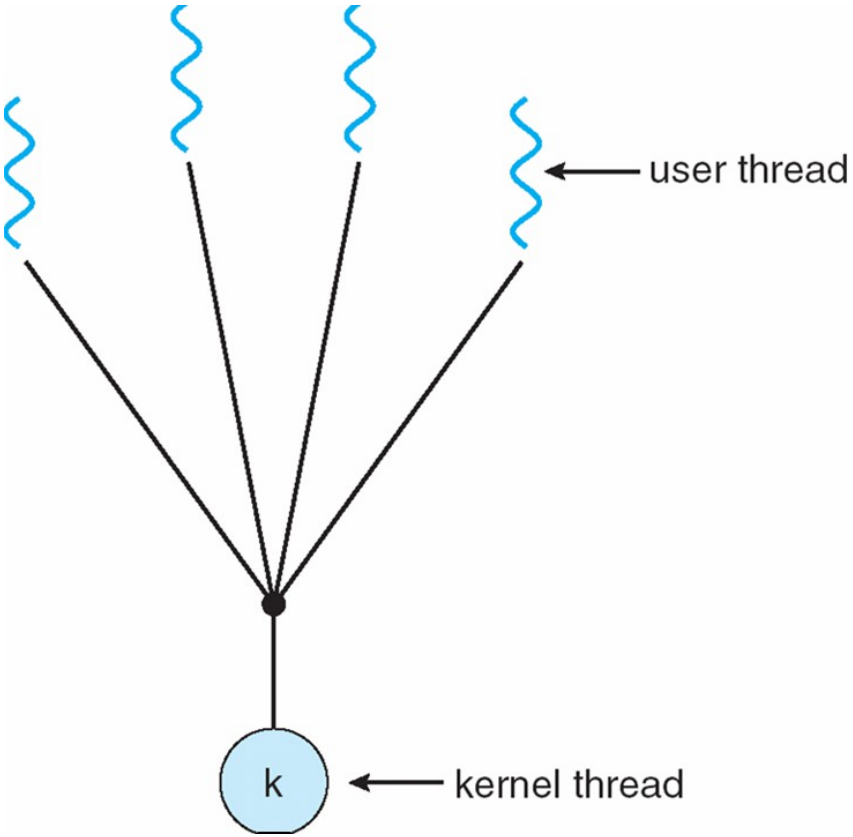
Threads: do Usuário VS do Kernel

- Threads do Usuário:
  - seu gerenciamento é feito por uma biblioteca de threads no nível do usuário
- Três principais bibliotecas para threads:
  - POSIX Pthreads
  - Win32 threads
  - Java threads
- Threads do Kernel
  - suportadas pelo Kernel
- Exemplos
  - Windows
  - Solaris
  - Linux
  - Mac OS X

Modelos Multithread

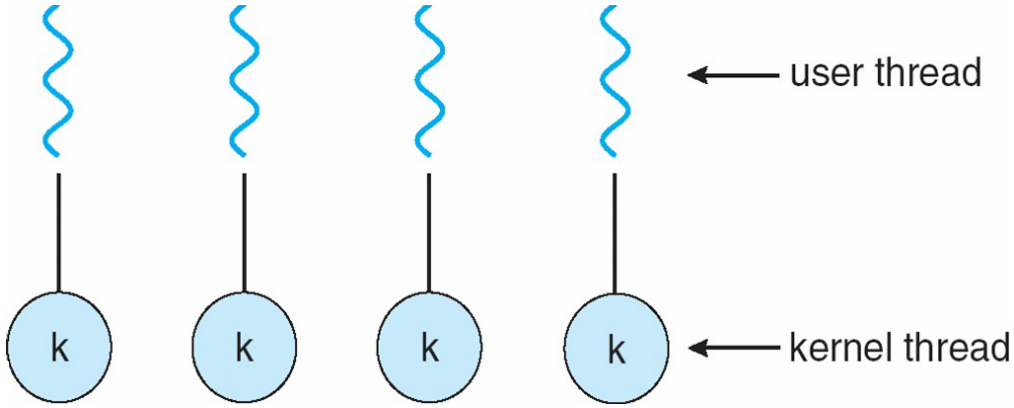
- Muitos-para-Um
- Um-para-Um
- Muitos-para-Muitos

Modelo Muitos-para-Um



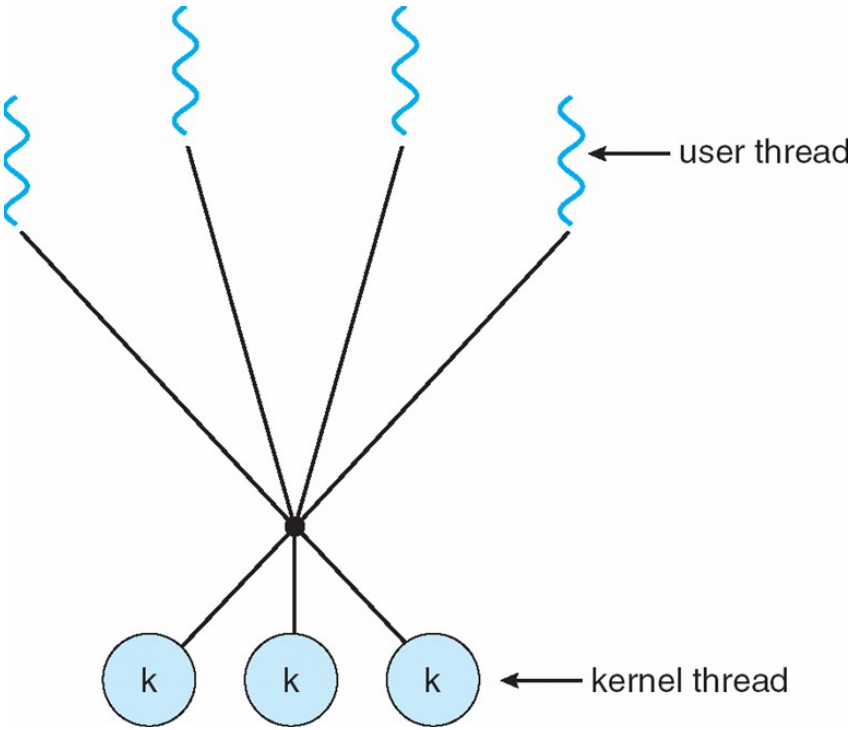
- Muitas threads de nível de usuário mapeadas para uma única thread do kernel
- Exemplos:
  - Green Threads no Solaris
  - Portable Threads no GNU

Modelo Um-para-Um



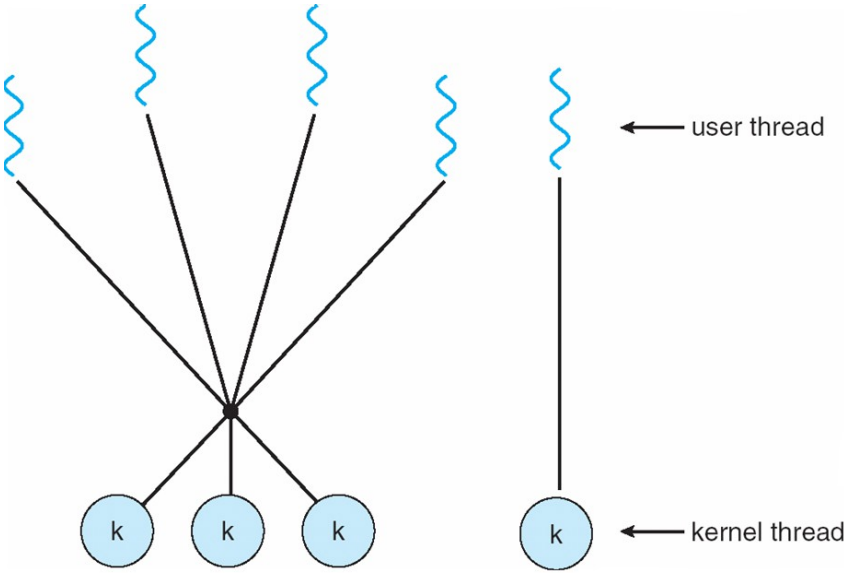
- Cada thread de nível do usuário mapeada para uma thread do kernel
- Exemplos
  - Windows NT/XP/2000
  - Linux
  - Solaris a partir da versão 9

Modelo Muitos-para-Muitos



- Permite muitas threads do nível do usuário serem mapeados para muitas threads do kernel
- Permite que o sistema operacional crie um número suficiente de threads do kernel
- Exemplos:
  - Solaris, versões anteriores a 9
  - Windows NT/2000 com o pacote *ThreadFiber*

Modelo de Dois-Níveis



- Similar ao modelo M:M, exceto pela permissão de uma thread do usuário ser **ligada** (**bound)** à thread do kernel
- Exemplos
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 e versões anteriores

Bibliotecas de Threads

- Fornecem ao programador uma API para a criação e gerência de threads
- Duas formas principais de implementação
  - biblioteca **totalmente no espaço do usuário**
  - biblioteca **no nível do kernel, suportada pelo S.O.**

Pthreads

- Uma API padrão POSIX (IEEE 1003.1c) para a criação e sincronização de threads

  → pode ser fornecida tanto no nível do usuário quanto no nível do kernel

- **API especifica o comportamento** da biblioteca de thread

- Implementação é feita no desenvolvimento da API

- Comum nos sistemas operacionais Unix-like
  → ex.: Linux, Mac OS X, Solaris

Threads Java

- Threads Java são gerenciadas pela JVM
  - implementada no modelo de threads fornecido pelo S.O. subjacente

- Threads Java podem ser criadas por:
  - Estendendo a classe Thread
  - Implementando a interface Runnable

Questões Relacionadas à Threads

- Semântica das chamadas de sistema **fork()** e **exec()**
- Cancelamento de uma Thread
  - Assíncrono ou postergado (deferred)
- Tratamento de Sinais
- Thread pools
- Dados específicos da Thread
- Ativações do Escalonador - Scheduler activations

Semântica do fork() e exec()

- **fork()** duplica somente a thread que a chamou ou todas as threads?

Cancelamento de uma Thread

- Terminação de um thread antes de sua finalização
- Duas abordagens no geral:
  - **Cancelamento Assíncrona** termina a thread alvo imediatamente
  - **Cancelamento Postergado (Deferred cancellation)** permite que a thread alvo periodicamente verifique se ela deve ser cancelada

Tratamento de Sinal

- Sinais são usados em sistemas UNIX para notificar um processo que um evento em particular ocorreu
- Um tratamento de sinal é usado para processar sinais
  1. Um sinal é gerado por um evento em particular
  2. Um sinal é entregue a um processo
  3. O sinal é tratado
- Opções:
  - Entregar o sinal para a thread a qual o sinal se aplica
  - Entregar o sinal para toda thread no processo
  - Entregar o sinal para certas threads no processo
  - Associar uma thread específica para receber todos os sinais do processo

Thread Pools

- Cria um número de threads em um pool onde aguardam por trabalho
- Vantagens:
  - Normalmente mais rápido servir uma requisição com um thread existente do que criar uma nova thread
  - Permite o número de threads da aplicação(ões) serem limitadas(bound) ao tamanho do pool

Dados Específicos de uma Thread

- Permite cada thread ter sua própria cópia do dado
- Útil quando você não tem controle sobre o processo de criação da thread (ex: quando usando um pool de threads)

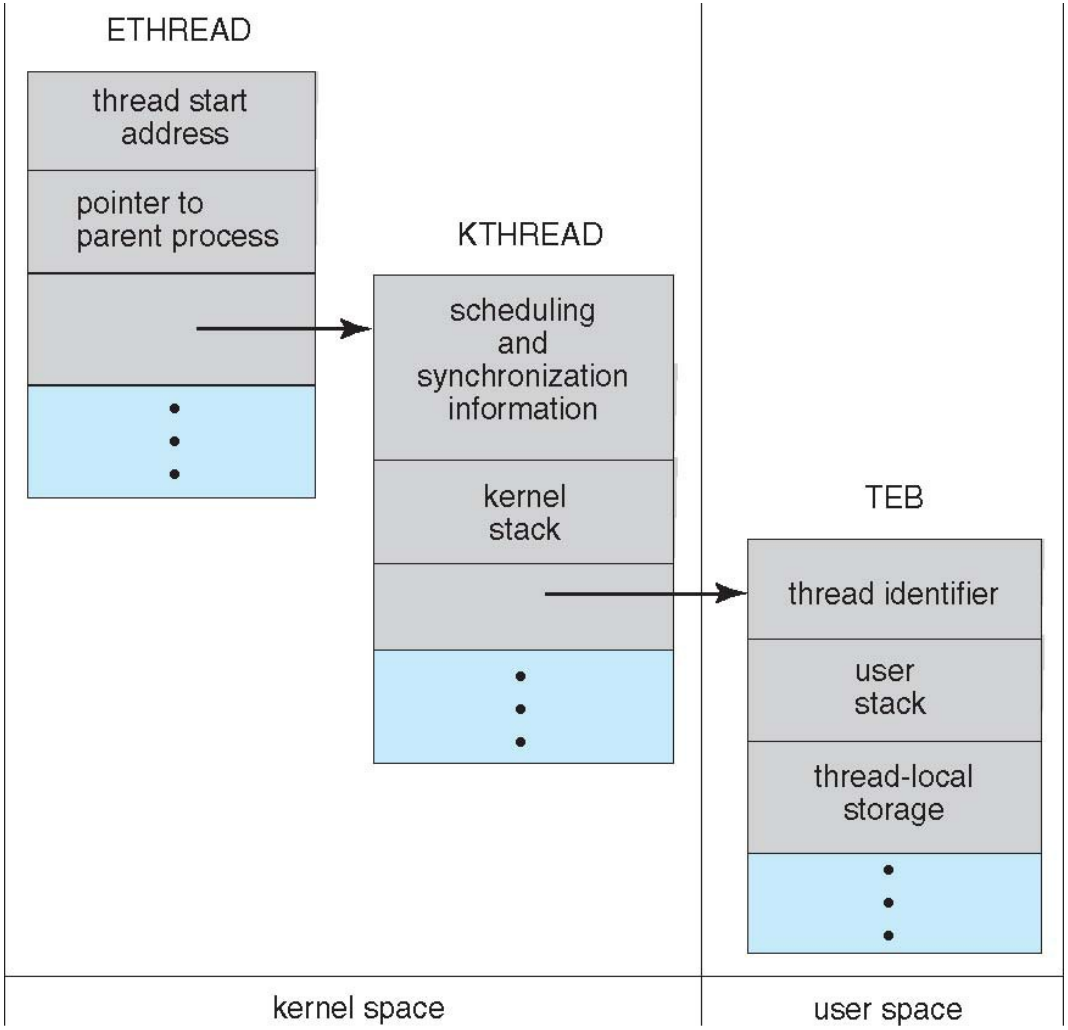
Ativação de Escalonadores

- Ambos models M:M e Dois-Níveis, requerem comunicação para manter o número apropriado de threads do kernel alocado para a aplicação
- Ativação de escalonadores fornece upcalls – um mecanismo de comunicação do kernel para a biblioteca de threads
- Esta comunicação permite que uma aplicação mantenha o número correto de threads do kernel

Exemplos de Sistemas Operacionais

- Threads no Windows XP
- Threads no Linux

Threads no Windows XP



Threads no Linux

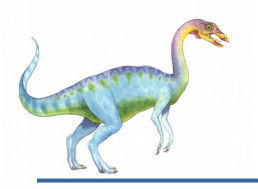
flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

Threads no Windows XP

- Implementa o mapeamento um-para-um, nível do kernel
- Cada thread contém
  - Um identificador da thread
  - Um conjunto de registradores
  - Pilhas separadas para o usuário e o kernel
  - Área privada de armazenamento de dados
- O conjunto de registradores, pilhas, e área privada de armazenamento são conhecidos como o contexto das threads
- As principais estruturas de dados de uma thread são:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

Threads no Linux

- Linux refere-se à elas como *tarefas(tasks)* em vez de *threads*
- A criação de threads é feita via chamada de sistema **clone()**
- **clone()** permite que uma tarefa filha compartilhe o espaço de endereçamento da tarefa pai (processo)



Fim do Capítulo 4