

Sistemas Operativos

Licenciatura em Engenharia Informática

Licenciatura em Engenharia Computacional

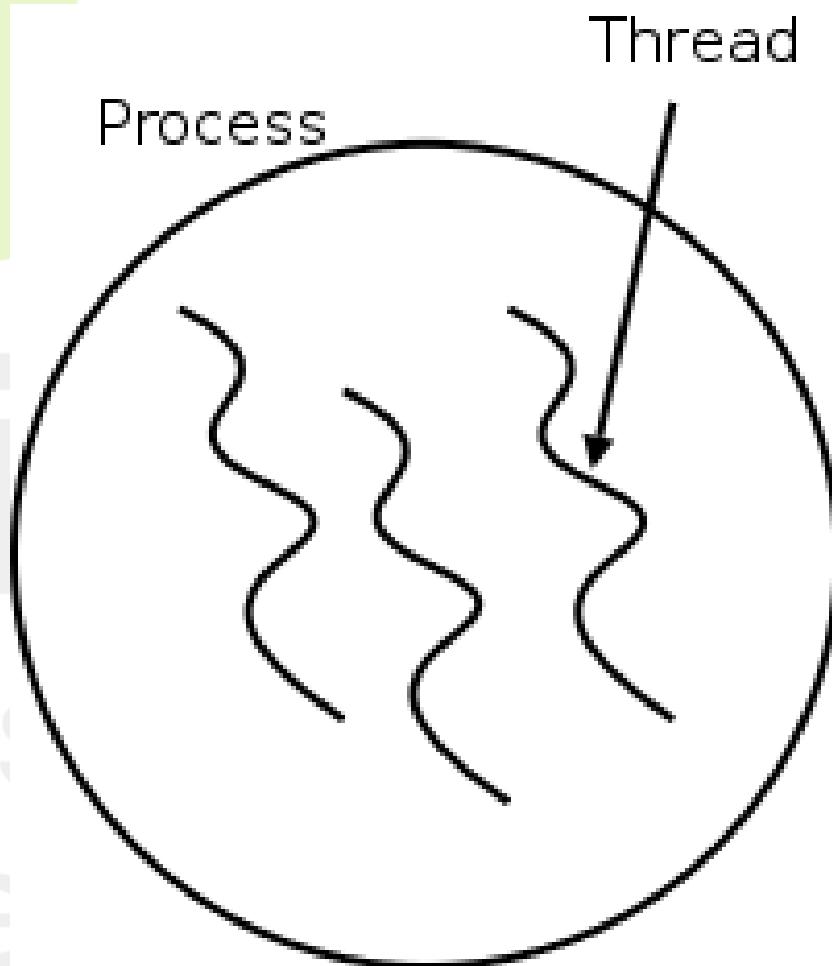
Licenciatura em Física (opcional)

Ano letivo 2025/2026

Pedro Azevedo Fernandes (paf@ua.pt)

Threads – 1

A ideia subjacente a uma thread é a de ter linhas de controlo separadas correndo no mesmo espaço de endereçamento de um único processo, como apresentado no diagrama.



Threads – 2

- O espaço de endereçamento, que é um conceito de gestão de memória, pode ser considerado como a memória em que o processo corre
 - na realidade, também inclui o mapeamento dos endereços virtuais – endereços do programa – para endereços físicos – endereços da máquina

Threads – 3

Uma thread não é um processo!

- Uma thread é parecida com um processo na medida em que, por exemplo, partilha o processador com outras threads
- Uma thread é diferente de um processo na medida em que, por exemplo, o espaço de endereçamento pertence ao processo no qual a thread está a correr

Threads – 4

Utilização das threads

- Quando um processo P bloqueia (p.ex., à espera de E/S), há cálculos que, ainda assim, podem ser efetuados para a aplicação
- Outro processo não os pode efetuar, uma vez que não tem acesso à memória do processo P
- Mas duas threads no mesmo processo partilham memória, pelo que esse problema não se coloca

Threads – 5

Desvantagens das threads

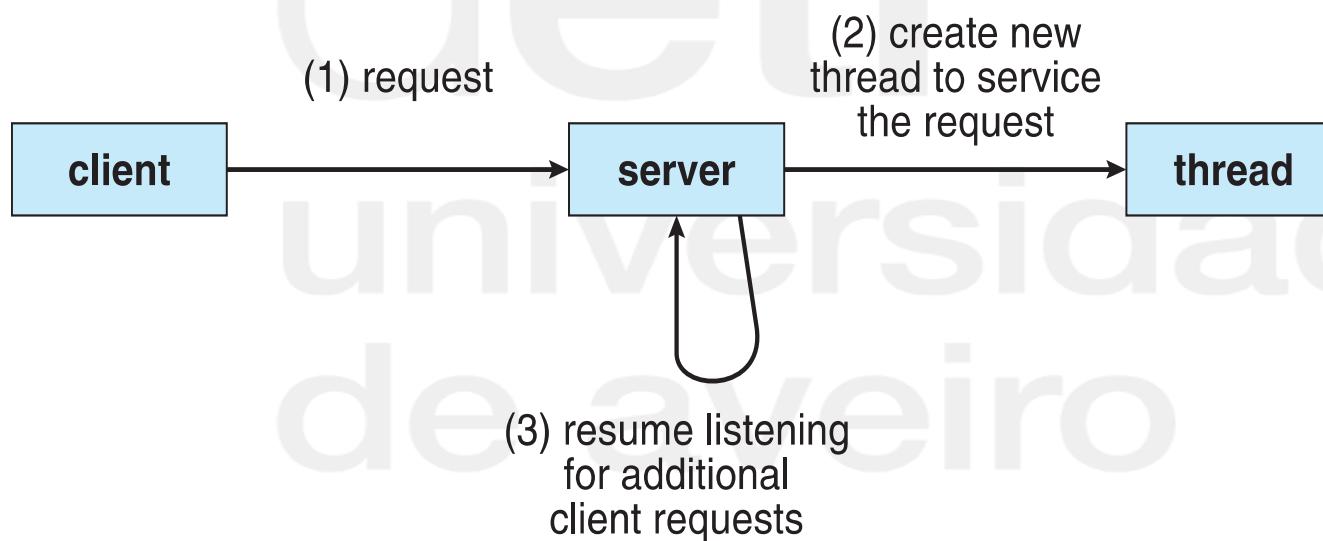
- Uma vez que as threads de um mesmo processo partilham memória, cada thread não se encontra protegida de outras threads desse processo
- Possuir várias threads a aceder concorrentemente à mesma memória dá origem a bugs de difícil deteção
- Deste modo, a utilização das threads implica um compromisso entre o desempenho e a simplicidade

Threads – Motivação - 1

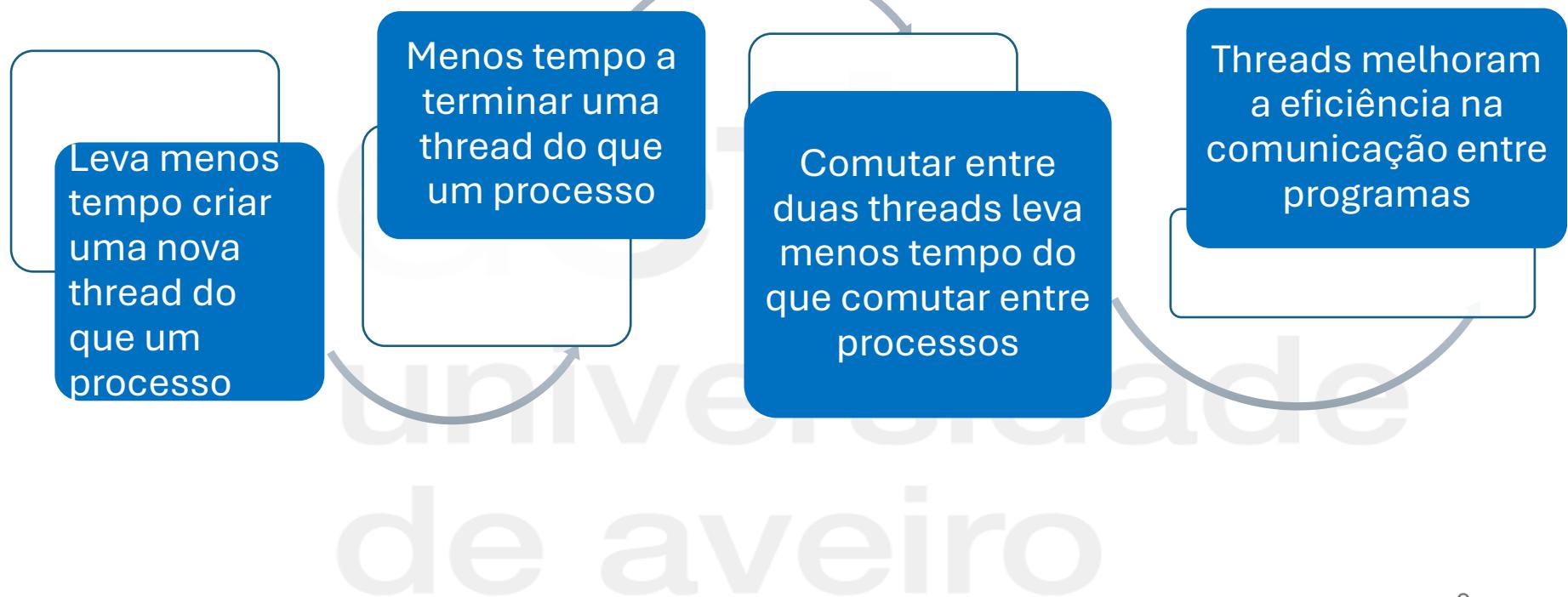
- A maioria dos SO atuais é multithreaded
- As threads correm dentro das aplicações
- Multitarefas da aplicação podem ser implementadas com threads separadas
- Exemplos:
 - Atualizar o ecrã
 - Adquirir dados
 - Responder a um pedido de rede
 - Verificação de erros

Threads – Motivação - 2

- A criação de processos é pesada
- A criação de threads é leve
- Com threads pode-se simplificar o código e melhorar a performance e eficiência
- Os kernel são geralmente multithreaded



Threads – Benefícios - 1



Threads – Benefícios - 2

- **Resposta rápida do sistema** – pode permitir a execução continuada quando parte do processo está bloqueado, o que é importante em especial para os interfaces de utilizador
- **Partilha de recursos** – as threads partilham recursos do processo, o que se torna mais fácil do que a partilha de memória ou a passagem por mensagens

Threads – Benefícios - 3

- **Economia** – mais económico do que a criação de processos em termos de utilização de recursos, a comutação de contexto das threads tem menor carga computacional do que a comutação de processos
- **Escalabilidade** – os processos podem aproveitar as vantagens dos sistemas com arquitecturas multiprocessador

Threads – Utilização

- Trabalho em primeiro e segundo plano
- Processamento assíncrono
- Aumento da velocidade de execução
- Estrutura de programação modular

Threads – Caracterização

- **Thread (ou processo leve)** – A unidade de dispatching -- também referida como processo leve
- **Processo (ou tarefa)** – a unidade que possui os recursos para execução
- **Multithreading** – a capacidade de um SO suportar linhas de execução múltiplas e concorrentes, no contexto de um processo

Uma ou mais threads num processo

- Cada thread tem
 - Um estado de execução (a correr, preparado, ...)
 - Contexto da thread guardado quando esta não se encontra a correr
 - Uma stack de execução
 - Algum armazenamento estático por thread para variáveis locais
 - Acesso à memória e recursos do seu processo (todas as threads de um processo partilham essa informação)

Abordagem de uma só thread

- Uma só thread por processo, na qual o conceito de thread não é reconhecido, é referida como uma abordagem **single threaded**

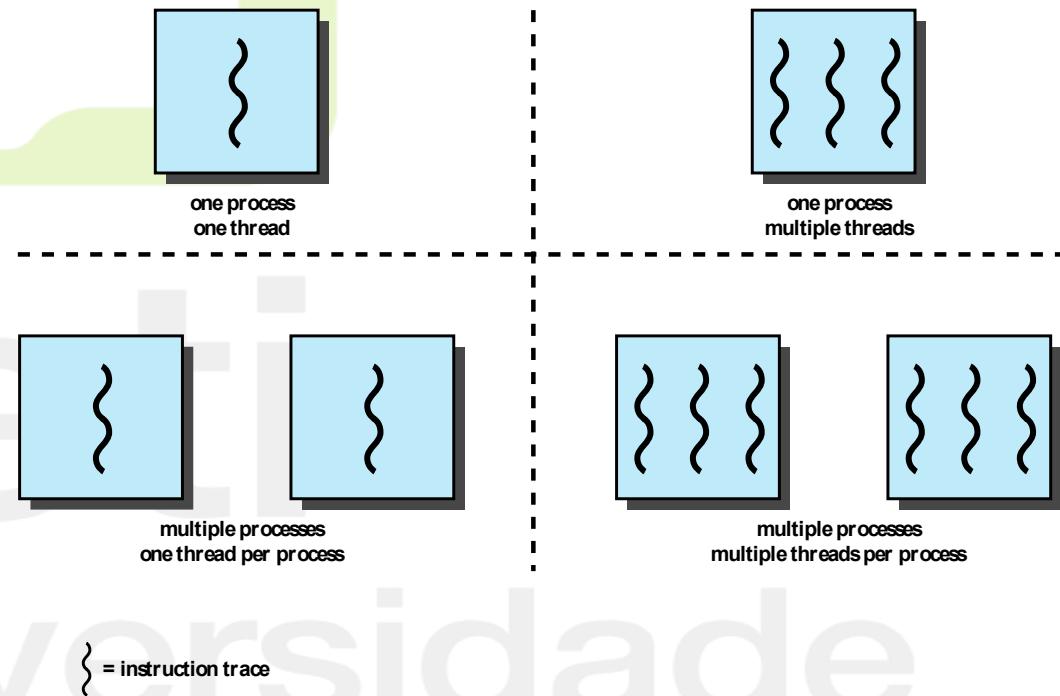


Figure 4.1 Threads and Processes

Abordagem de várias threads

- O exemplo da direita representa uma abordagem **multithreaded**

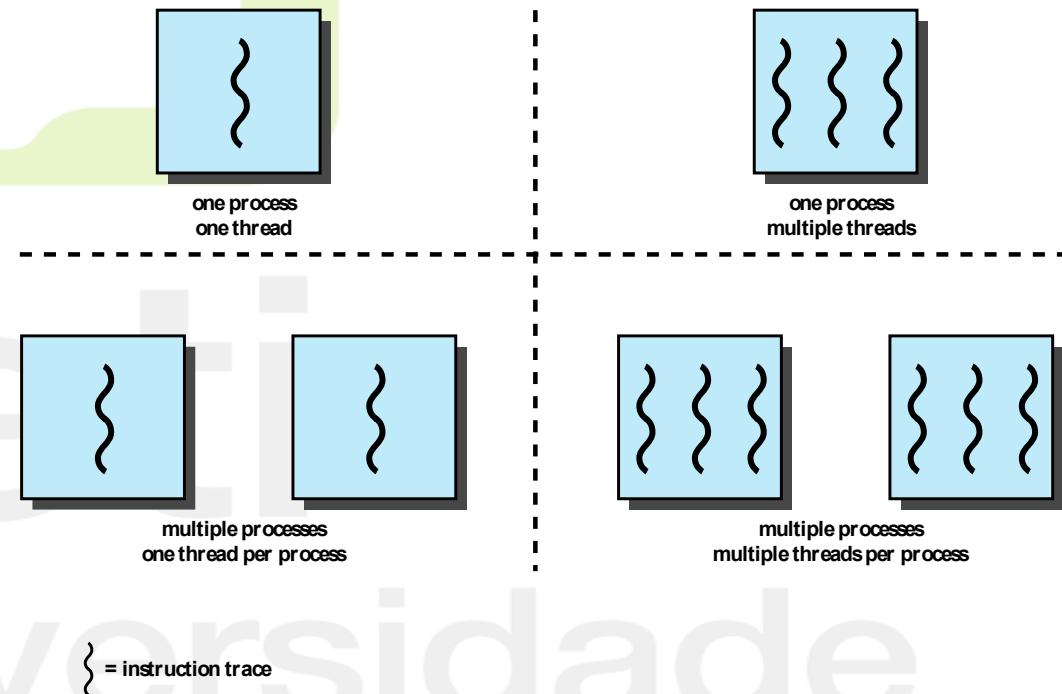
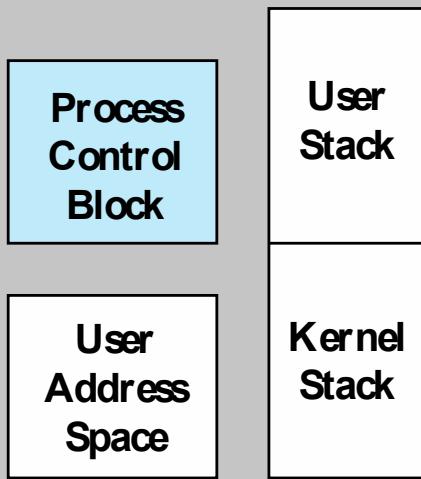


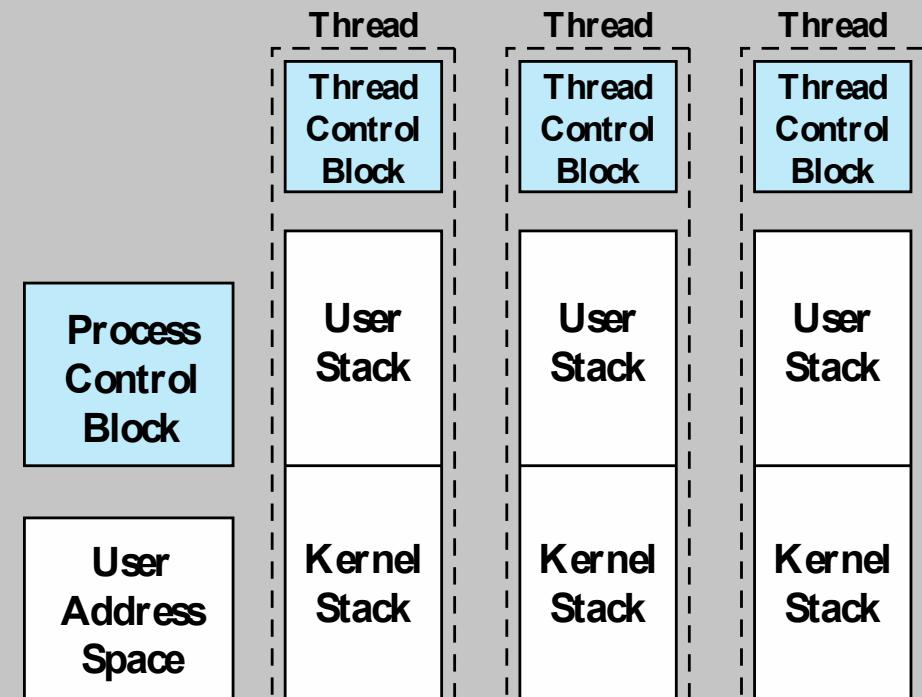
Figure 4.1 Threads and Processes

Modelos de Processos Single Threaded e Multithreaded

**Single-Threaded
Process Model**



**Multithreaded
Process Model**



Programação Multicore - 1

- A programação em sistemas multicore, ou multiprocessador, coloca pressão no programador e cria desafios, tais como:
 - Divisão das atividades
 - Equilíbrio
 - Separação dos dados
 - Dependência dos dados
 - Testes e debugging

Programação Multicore - 2

- **Paralelismo** – implica que um sistema pode executar mais do que uma tarefa simultaneamente
- **Concorrência** – suporta mais do que uma tarefa em progressão
 - Em sistemas monoprocessador/monocore, o scheduler fornece suporte para a concorrência

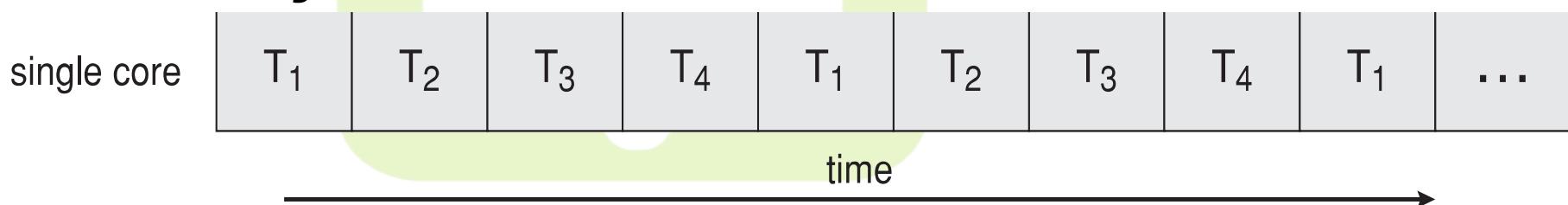
Programação Multicore - 3

Tipos de Paralelismo

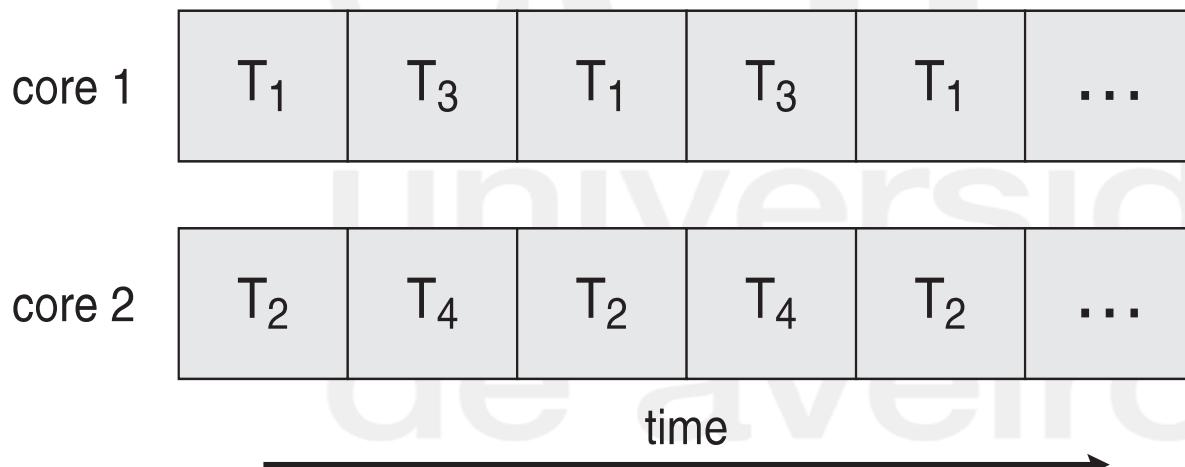
- **Paralelismo de Dados** – distribui subconjuntos dos mesmos dados pelos diversos cores, com operações análogas em cada um deles
- **Paralelismo de Tarefas** – distribui threads pelos diversos cores, com cada thread a executar operações diferentes

Threads – Concorrência vs. Paralelismo - 1

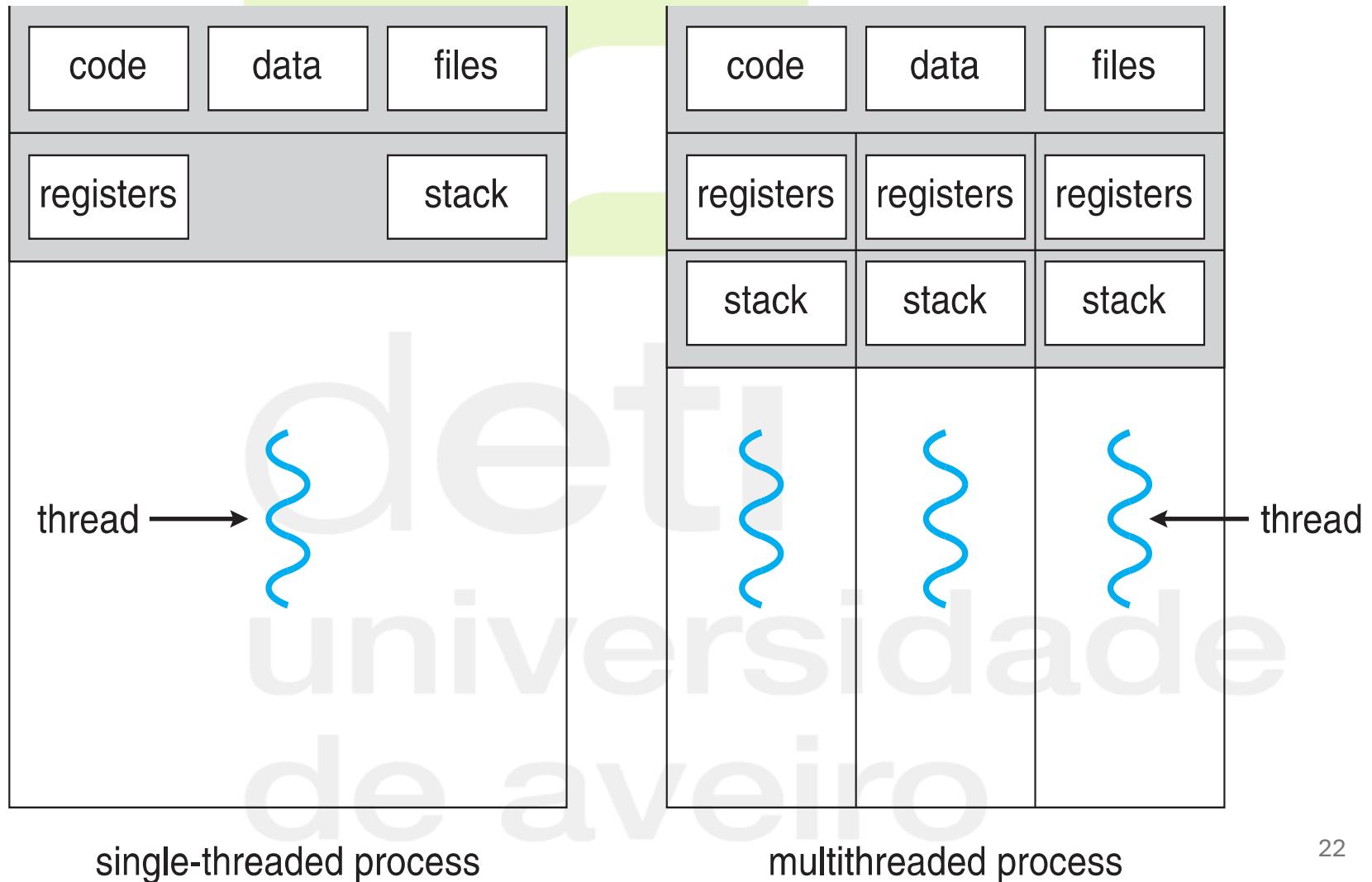
- Execução concorrente num sistema monocore



- Paralelismo num sistema multicore



Threads – Concorrência vs. Paralelismo - 2



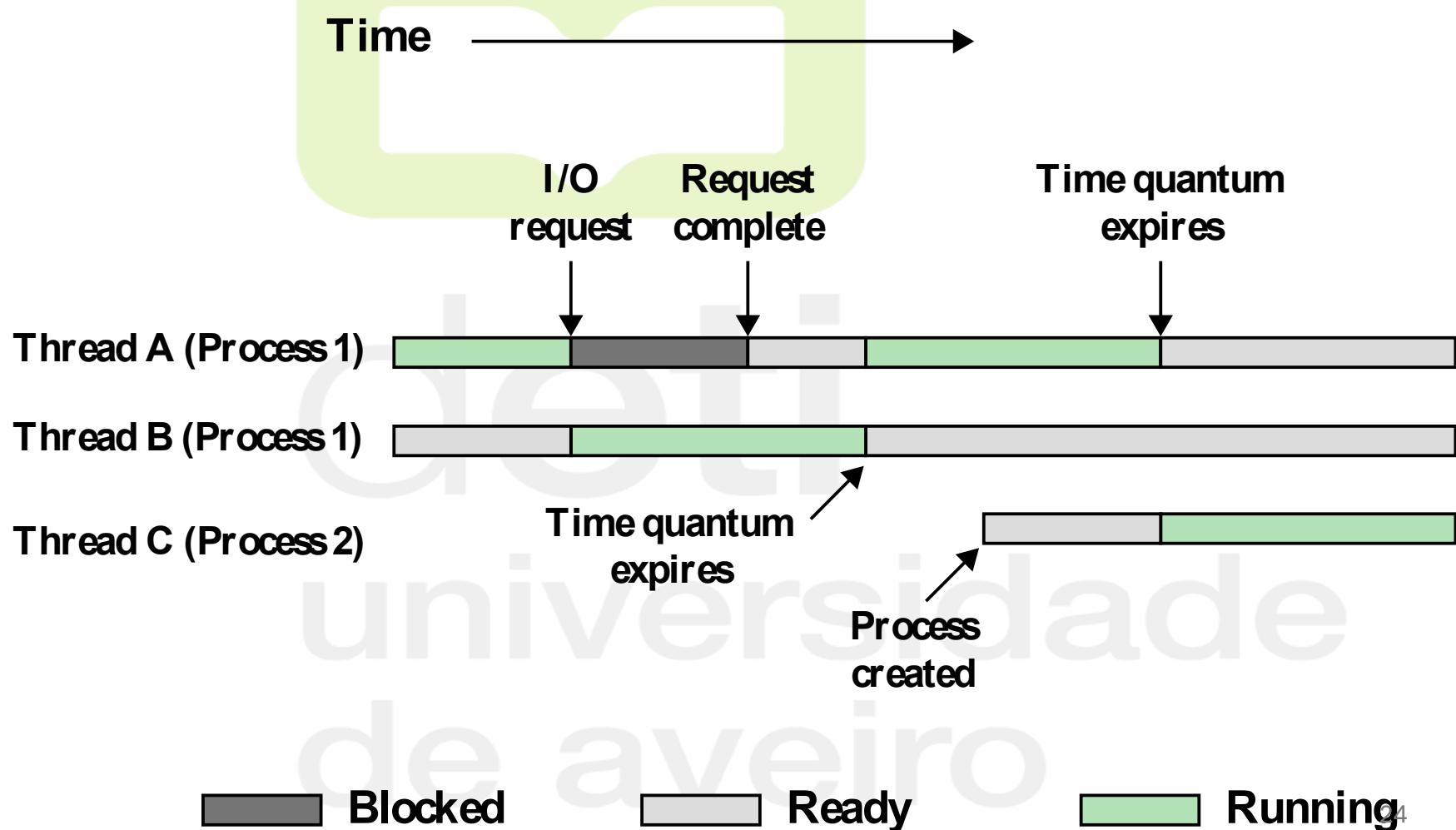
single-threaded process

multithreaded process

Os Estados de Execução de Threads

- Os estados chave de uma thread são:
 - Correr
 - Preparada
 - Bloqueada
- As operações de threads associadas com uma mudança de estado numa thread são:
 - Gerar (spawn)
 - Bloquear
 - Desbloquear
 - Terminar

Exemplo de Multithreading num Sistema Monoprocessador



Sincronização de Threads

- Torna-se necessário sincronizar as atividades das várias threads:
 - Todas as threads de um processo partilham o mesmo espaço de endereçamento e outros recursos
 - Qualquer alteração de um recurso por uma thread afeta as outras threads do mesmo processo

Tipos de Threads

User Level
Thread (ULT)

Kernel level Thread
(KLT)

User Level Threads e Kernel Level Threads

User threads – Gestão efetuada por libraries de threads ao nível do utilizador

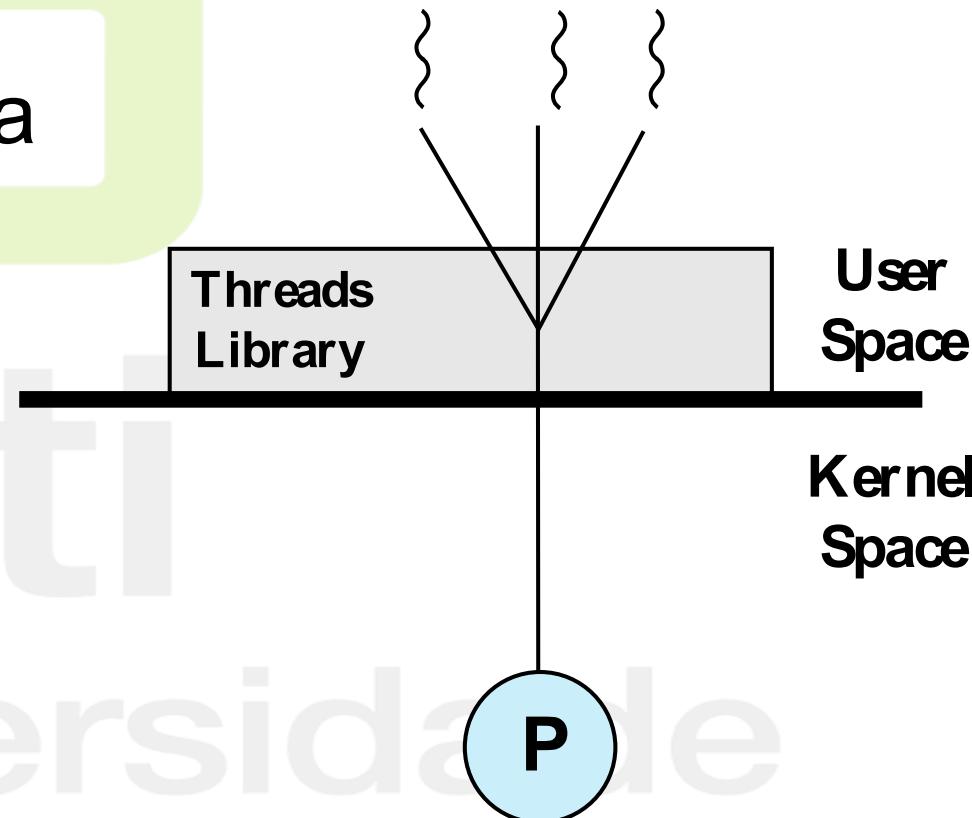
- Existem três tipos de libraries de threads:
 - POSIX Pthreads, Windows threads, Java threads

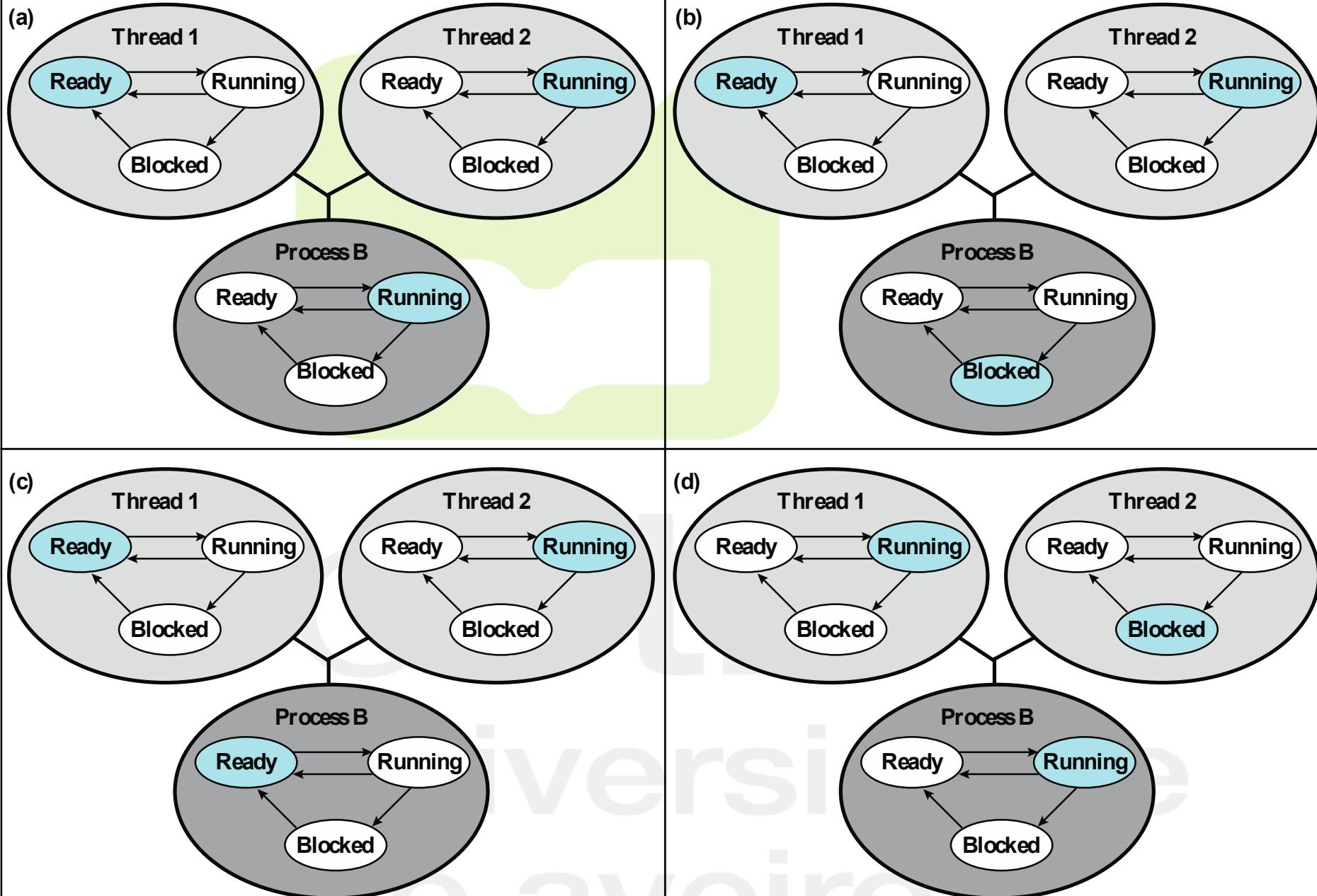
Kernel threads – Suportadas pelo kernel

- Exemplos: virtualmente todos os SO, incluindo:
 - Windows, Solaris, Linux, Tru64 UNIX, Mac OS X

User Level Threads (ULTs)

- Toda a gestão de threads é efetuada pela aplicação
- O kernel não tem conhecimento da existência das threads





Colored state
is current state

Vantagens das ULTs

A comutação de threads
não requer privilégios ao
nível do kernel mode

Scheduling pode
ser específico de
cada aplicação

ULTs podem
correm em
qualquer OS

Desvantagens das ULTs

- Num SO típico, muitas chamadas ao sistema implicam bloqueio
 - Em resultado disso, quando uma ULT executa uma chamada ao sistema, não só essa thread é bloqueada, como todas as outras threads desse processo também o são
- Num sistema com estratégia ULT pura, uma aplicação multithreaded não consegue aproveitar favoravelmente o multiprocessamento

Ultrapassando as Desvantagens das ULTs

Jacketing

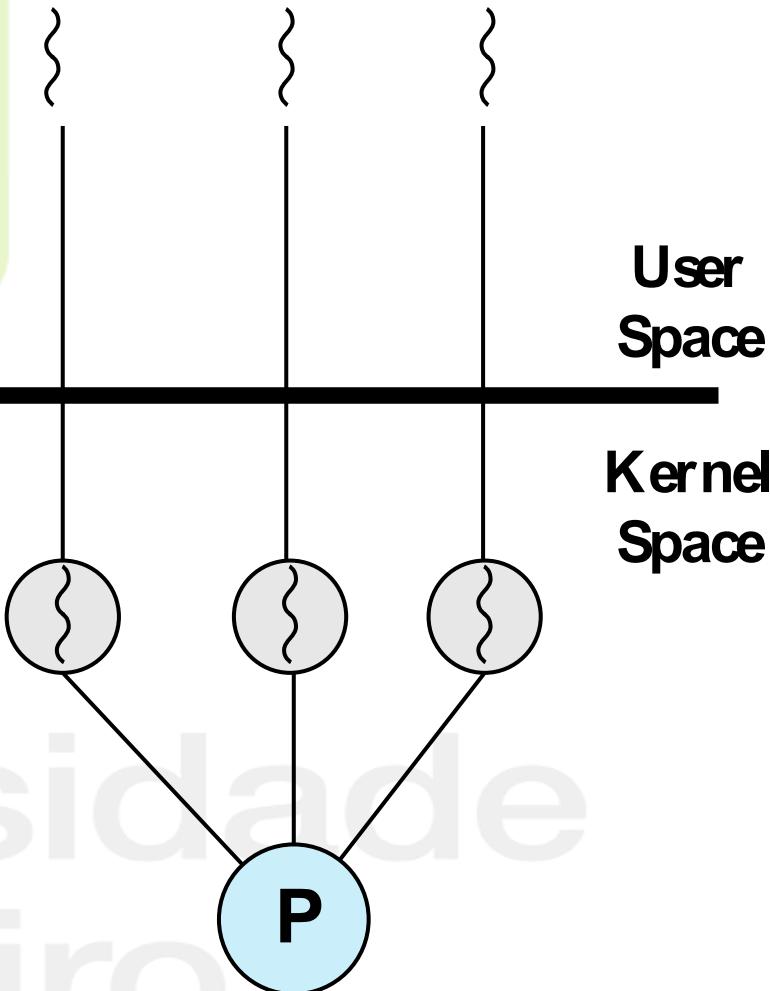
- Converte uma chamada ao sistema que bloqueia numa chamada ao sistema que não bloqueia

Desenvolver uma aplicação como múltiplos processos, em vez de múltiplas threads



Kernel Level Threads (KLTs)

- A gestão de threads é efetuada pelo kernel
- As aplicações não efetuam gestão de threads
- Um exemplo desta abordagem é o Windows



Vantagens das KLTs

- O kernel pode fazer em simultâneo o scheduling de múltiplas threads do mesmo processo em múltiplos processadores
- Se uma thread num processo se encontra bloqueada, o kernel pode fazer o schedule de outra thread do mesmo processo
- Rotinas do kernel podem ser multithreaded

Desvantagens das KLTs

- A transferência de controlo de uma thread para outra num mesmo processo requer uma comutação de modo para o kernel
- Latências da operação de Threads e Processos

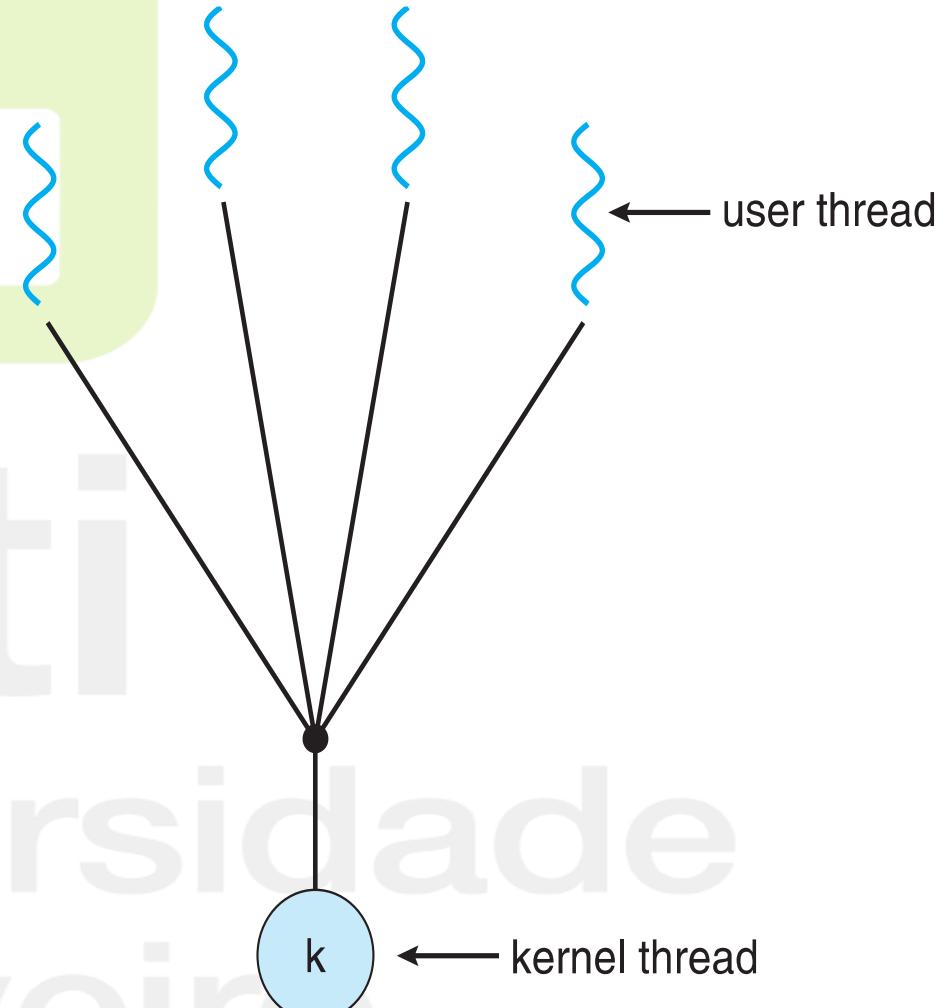
Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Modelos Multithreading

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

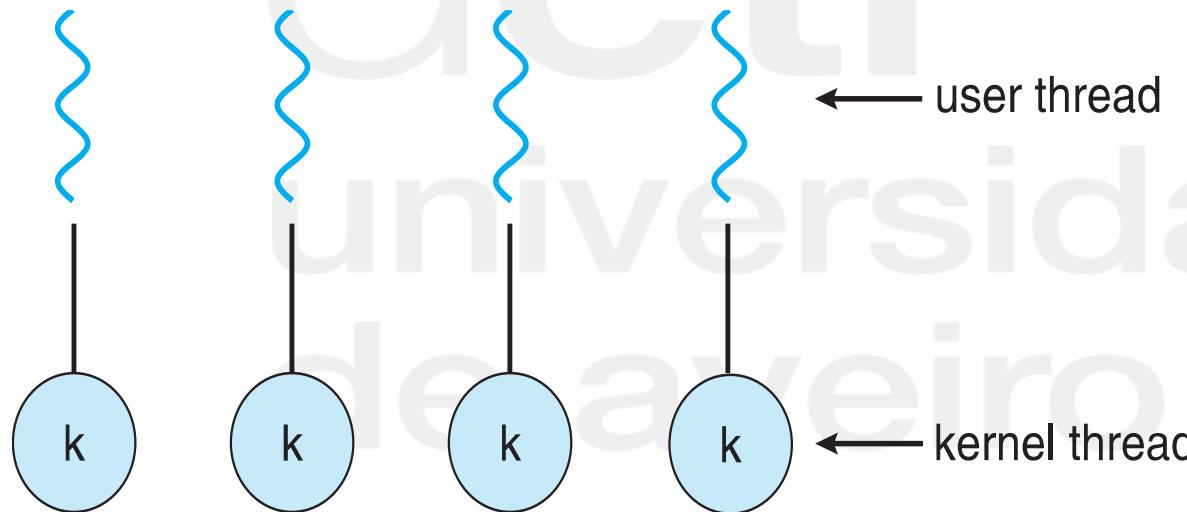
Muitos-para-Um (equivalente às ULTs)

- Muitas user-level threads mapeadas numa única thread ao nível do kernel
- O bloqueio de uma thread leva ao bloqueio de todas as outras
- Threads múltiplas podem não poder correr em paralelo em sistemas multicore porque apenas uma pode estar no kernel de cada vez
- Poucos sistemas usam este modelo



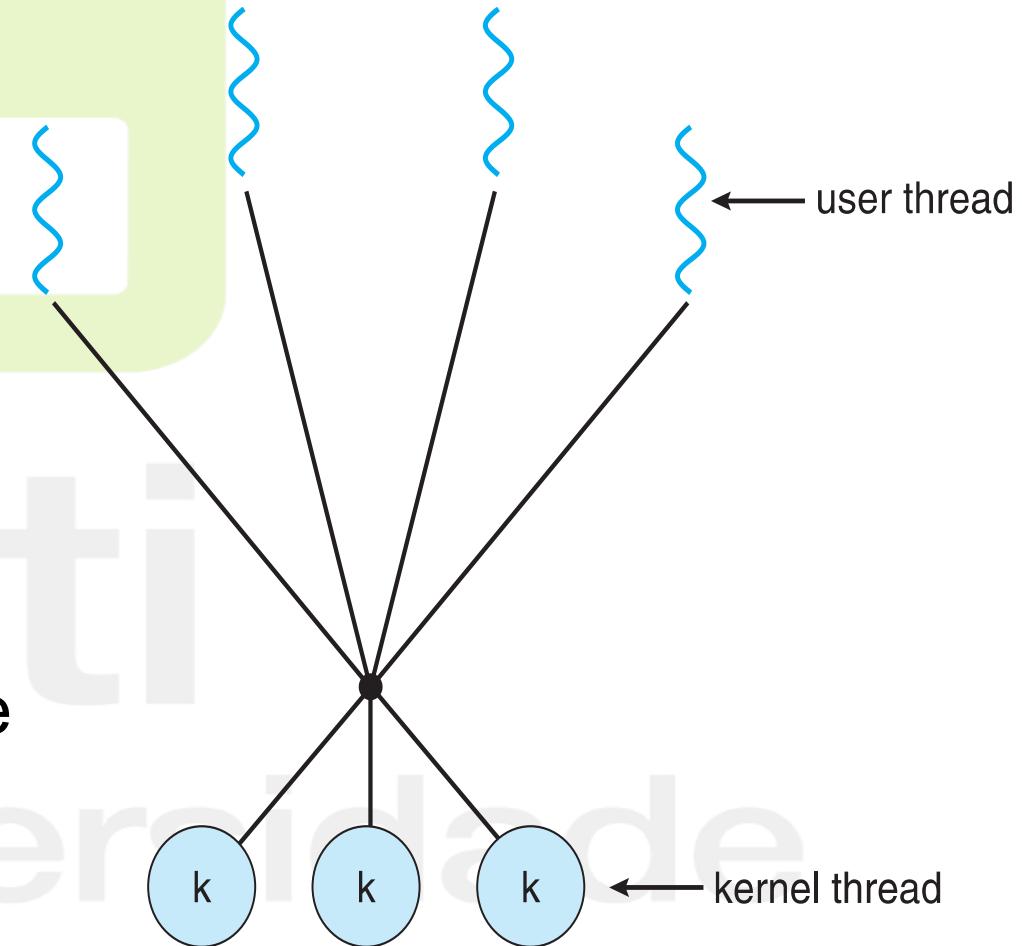
Um-para-Um (equivalente às ULTs)

- Cada user-level thread mapeia para uma thread kernel-level
- A criação de uma user-level thread leva à criação de uma thread ao nível do kernel
- Mais concorrência que o modelo Muitos-para-Um
- Por vezes há restrições ao número de threads por processo devido à sobrecarga
- Exemplos: Windows, Linux, Solaris 9 e seguintes



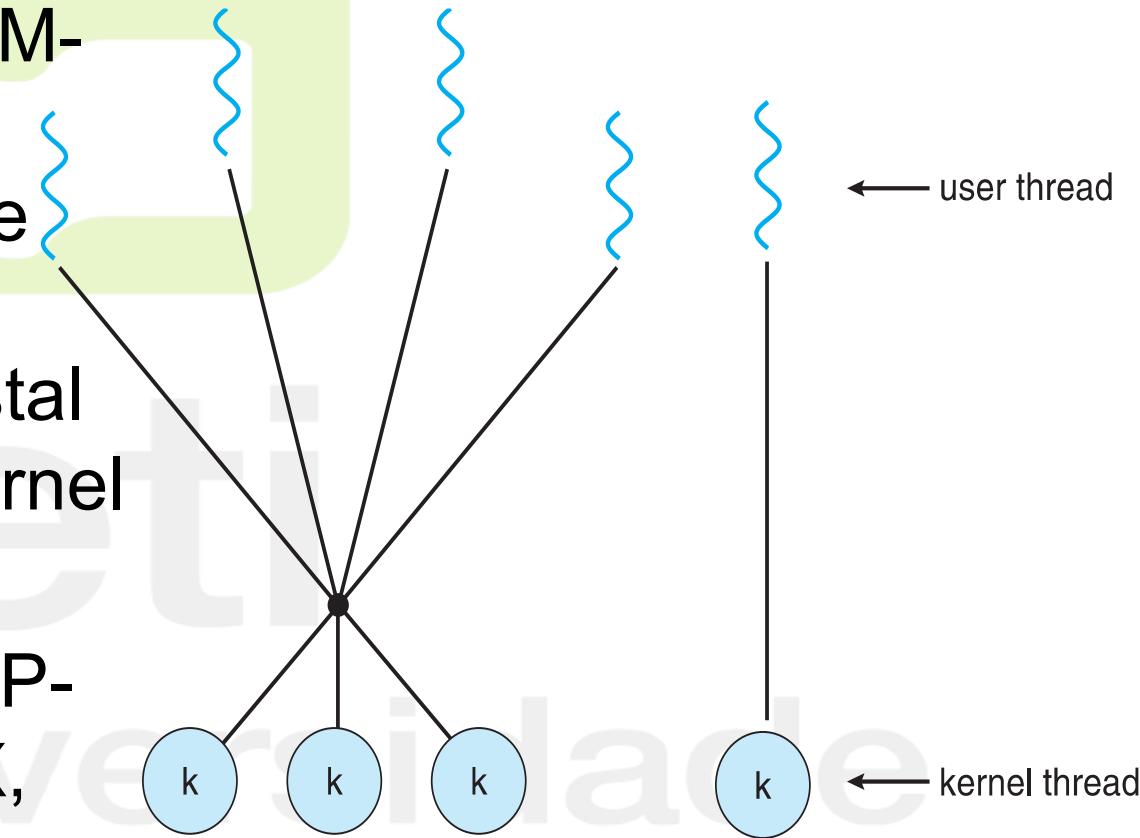
Muitos-para-Muitos (abordagens combinadas)

- Permite que muitas user-level threads possam ser mapeadas para muitas threads ao nível do kernel
- Permite ao SO criar um número suficiente de kernel threads
- Exemplo: versões do Solaris anteriores à 9



Modelo de Dois níveis (abordagens combinadas)

- Semelhante ao M-M, excepto que este permite que uma user-level thread possa estar ligada a uma kernel thread
- Exemplo: Irix, HP-UX, Tru64 UNIX, Solaris 8 e anteriores



Resumo dos Modelos

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Lei de Amdahl - 1

- Identifica os ganhos de desempenho resultantes do acrescentar cores adicionais a uma aplicação que possui componentes de execução em série e paralelo
- S é a componente série
- N representa o número de cores

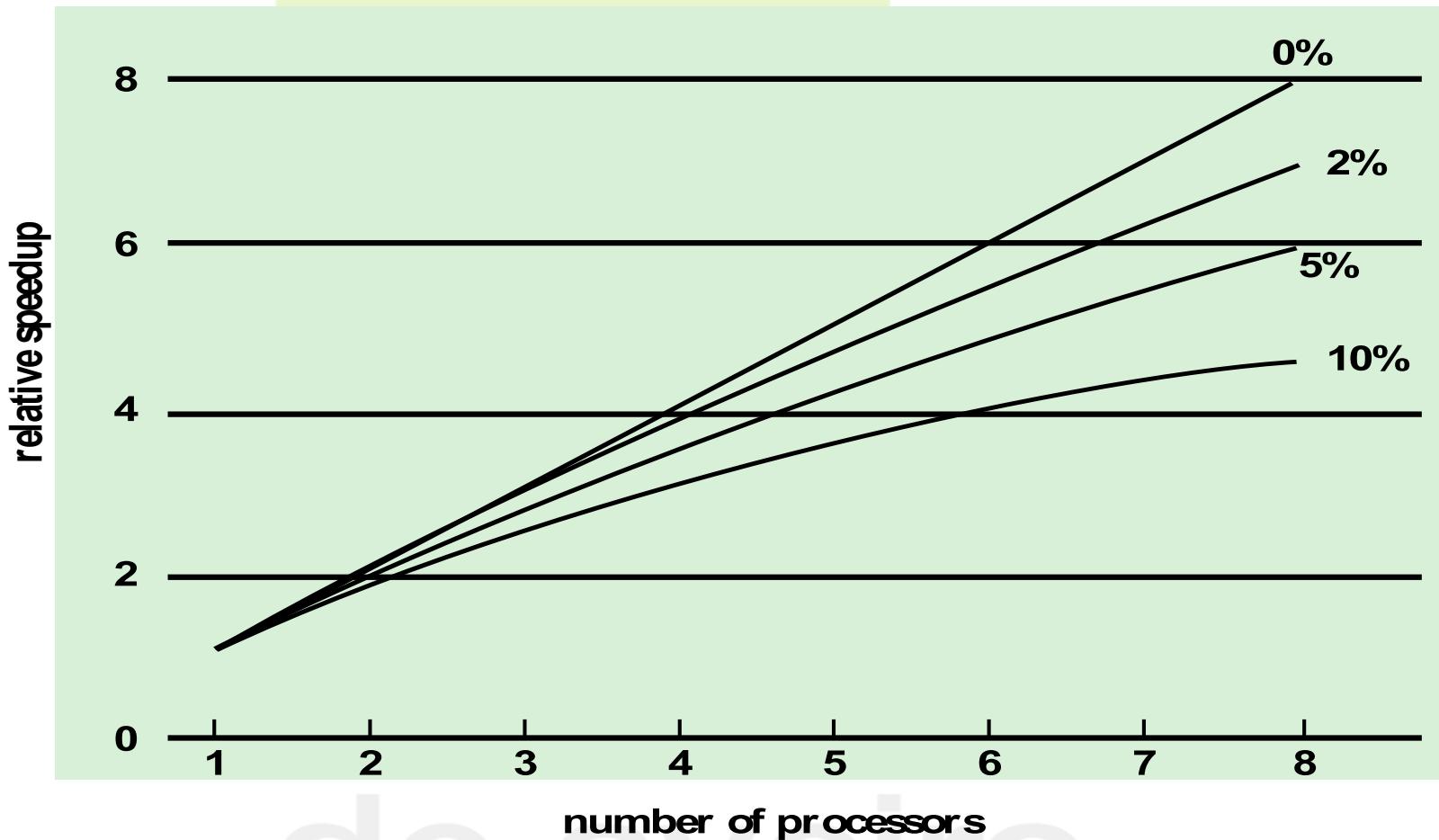
$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

Lei de Amdahl - 2

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- Ou seja, se a aplicação é 75% paralela e 25% série, passar de um para dois cores resulta num aumento de desempenho de 1,6 vezes
- À medida que N se aproxima de infinito, a melhoria de desempenho aproxima-se de 1/S
- A componente série de uma aplicação tem um peso desproporcionado nos ganhos de desempenho pelo acrescentar de cores adicionais

Efeitos dos Multicores no Desempenho



(a) Speedup with 0% , 2% , 5% , and 10% sequential portions