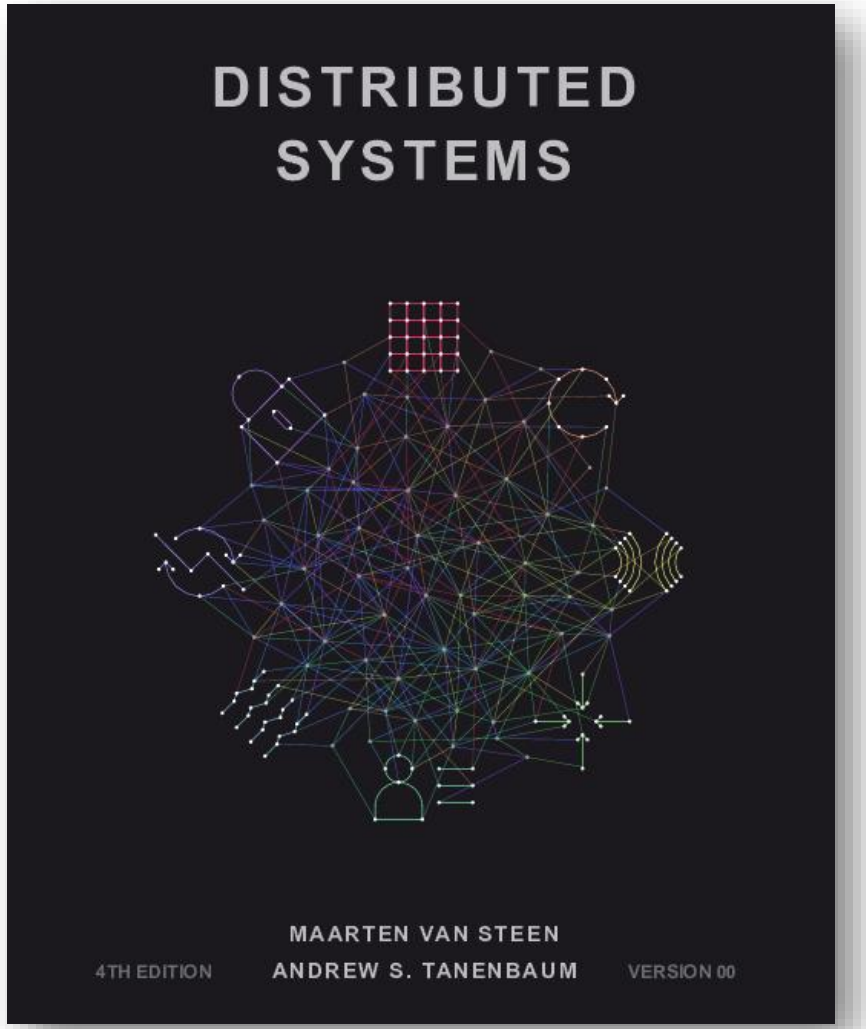
DCC 602 – Sistemas Distribuídos

3 – Processos

Departamento de CIÊNCIA DA COMPUTAÇÃO

UFRR

Sumário



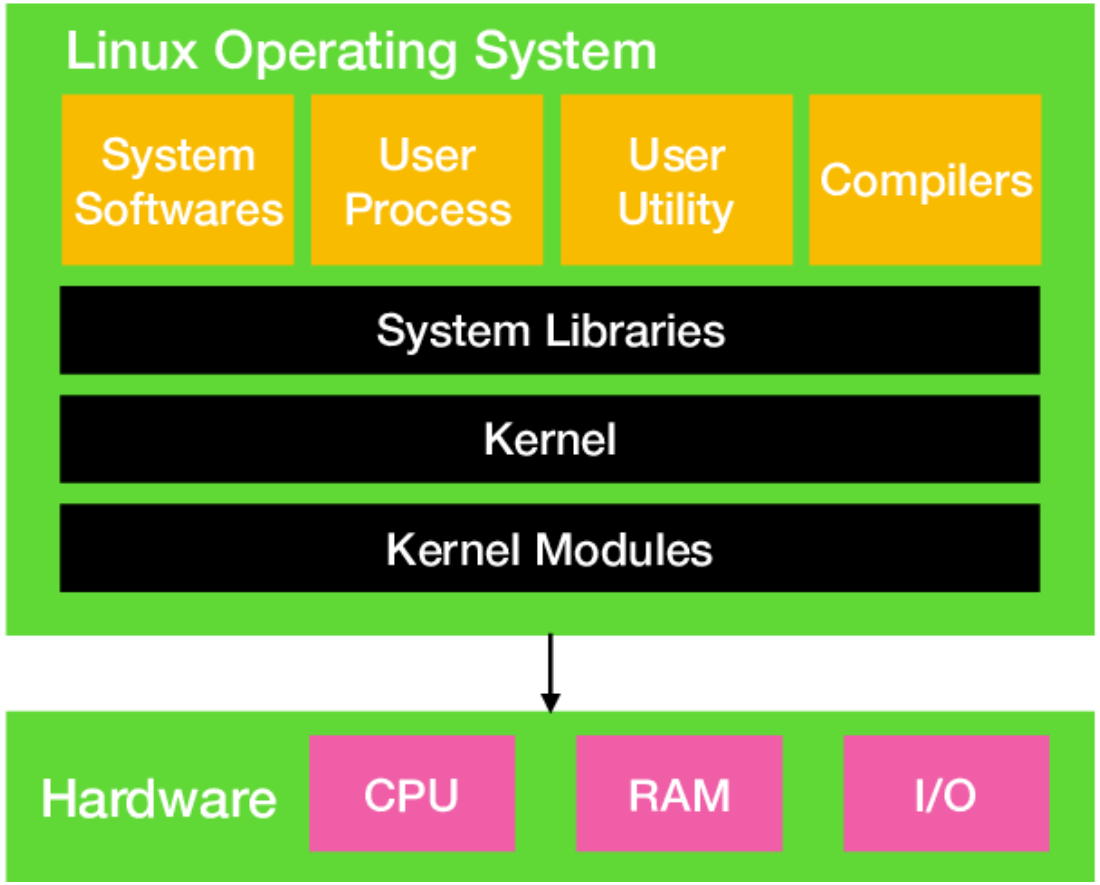
Departamento de CIÊNCIA DA COMPUTAÇÃO

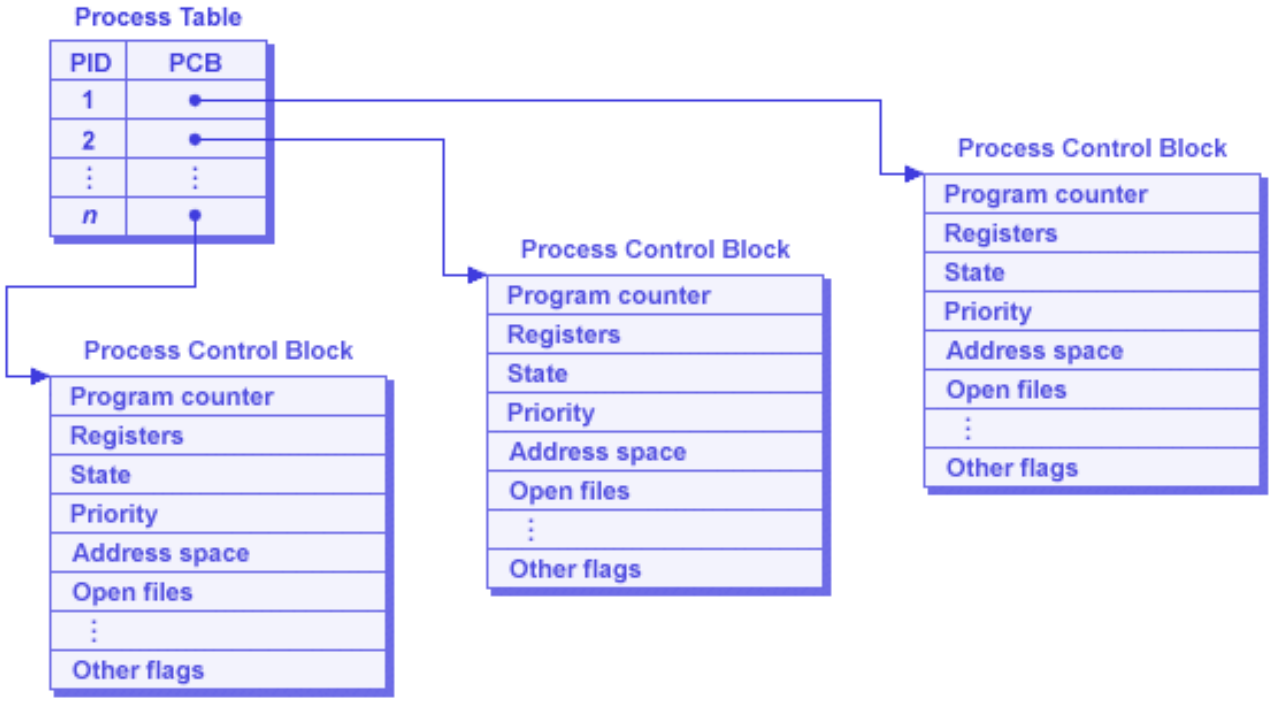
UFRR

3.1 Threads

- Em sistemas distribuídos, compreender o papel das threads requer uma compreensão dos processos e sua relação com as threads.
- Para executar um programa, o sistema operacional cria **processadores virtuais**, cada um executando um programa diferente.
- O sistema operacional mantém **uma tabela de processos** para acompanhar esses processadores virtuais, que contém entradas para armazenar valores de registro de CPU, mapas de memória, arquivos abertos, informações contábeis, privilégios e outros dados.
- Essas entradas formam coletivamente um **contexto de processo.**
- **O contexto do processador** contém pelo menos o contador de programa e, em alguns casos, outros valores de registro, como o ponteiro de pilha.

3.1 Threads

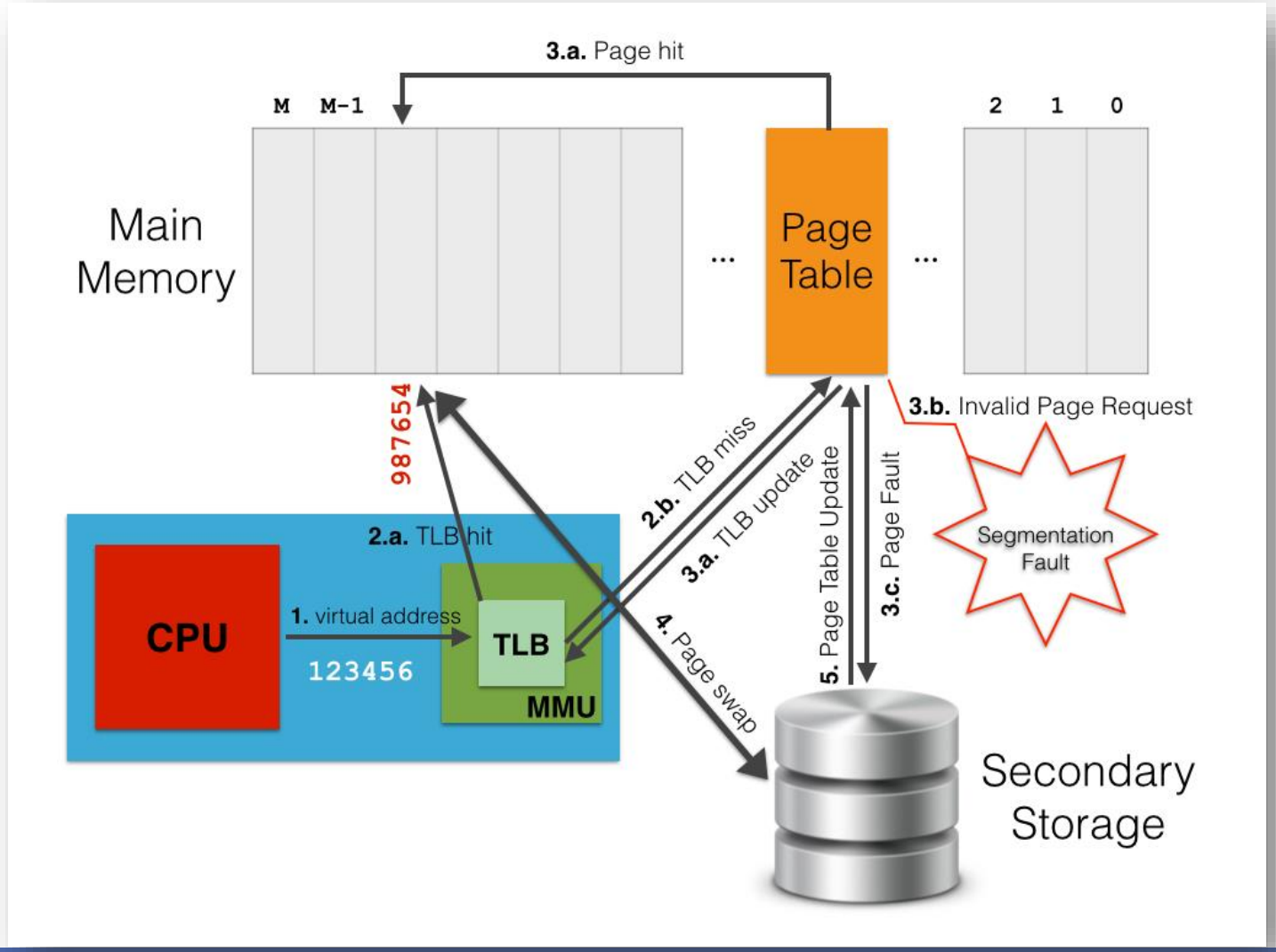

Linux Operating System
System Softwares
User Process
User Utility
Compilers
System Libraries
Kernel
Kernel Modules
Hardware
CPU
RAM
I/O


Process Table
PID
PCB
1
2
n
Process Control Block
Program counter
Registers
State
Priority
Address space
Open files
Other flags
Process Control Block
Program counter
Registers
State
Priority
Address space
Open files
Other flags
Process Control Block
Program counter
Registers
State
Priority
Address space
Open files
Other flags

3.1 Threads

- O sistema operacional toma muito cuidado para garantir que processos independentes não possam afetar maliciosamente ou inadvertidamente a correção do comportamento um do outro.
- Essa transparência de concorrência tem um custo, já que o sistema operacional precisa criar um espaço de endereço independente completo cada vez que um processo é criado, o que pode envolver a inicialização de segmentos de memória, a cópia do programa associado e a configuração de uma pilha de dados temporária.
- Além disso, a mudança da **CPU** entre dois processos pode exigir a modificação de registros da **Unidade de Gerenciamento de Memória (MMU)** e a invalidação de caches de tradução de endereços, como no **Translation Lookaside Buffer (TLB).**
- Se o sistema operacional suportar mais processos do que pode manter simultaneamente na memória principal, pode ser necessário trocar processos entre a memória principal e o disco antes que a troca real possa ocorrer.

3.1 Threads

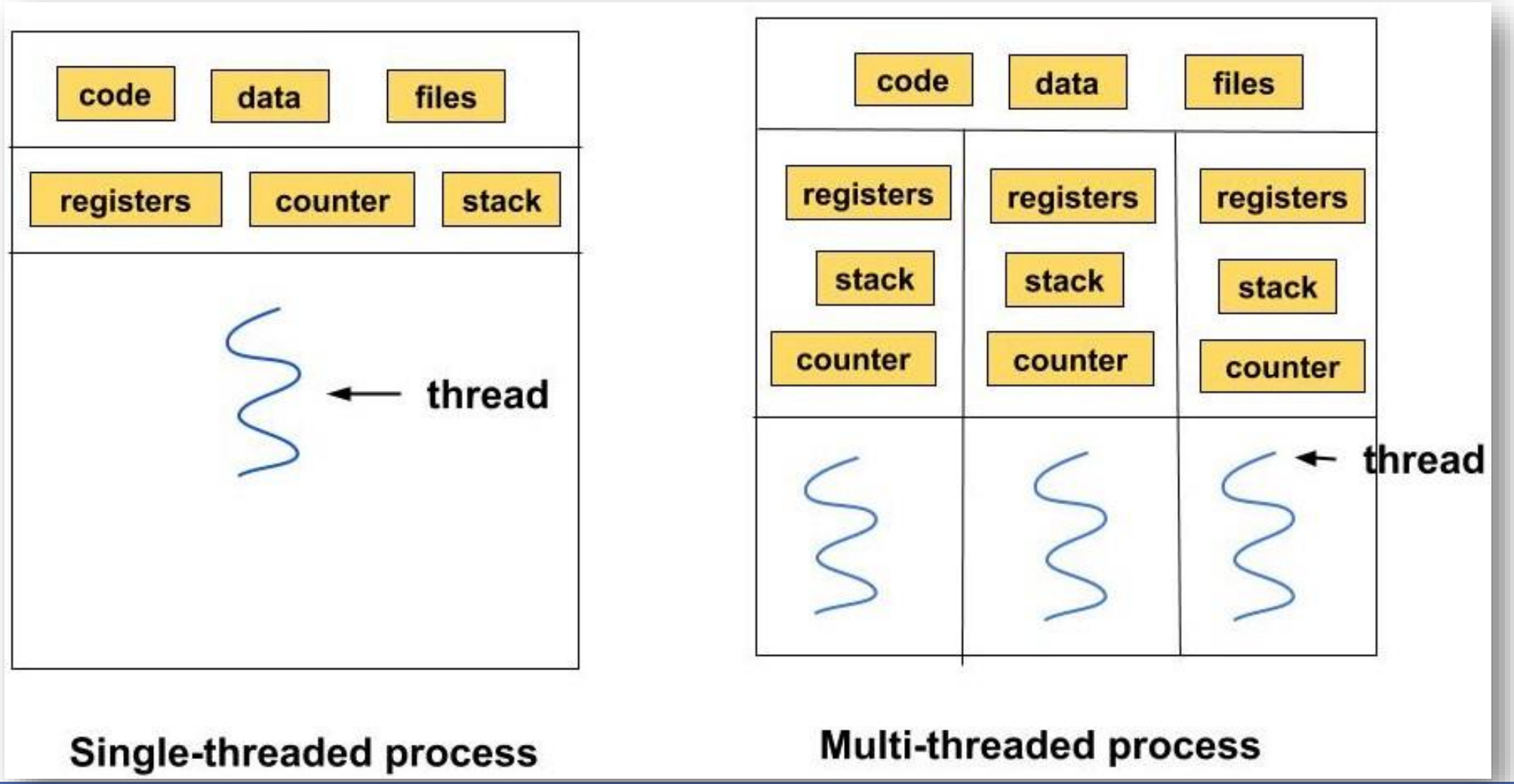


UFRR

3.1 Threads

- Threads são executadas independentemente umas das outras, assim como os processos.
- Porém, diferentemente dos processos, não é feita uma tentativa de alcançar uma alta transparência de concorrência se isso resultar em uma degradação de desempenho.
- Um sistema de threads geralmente mantém apenas as informações mínimas para permitir que uma CPU seja compartilhada por várias threads.
- O contexto de uma thread geralmente consiste apenas do contexto do processador, juntamente com algumas outras informações para gerenciamento de threads.

3.1 Threads


code
data
files
registers
counter
stack
thread
Single-threaded process
code
data
files
registers
registers
registers
stack
stack
stack
counter
counter
counter
thread
Multi-threaded process

3.1 Threads

- A implantação de threads tem duas implicações importantes.
  - Primeiro, o desempenho de um aplicativo multithreaded dificilmente será pior do que o de sua contraparte singlethreaded.
    - Na verdade, muitas vezes, o multithreading leva a um ganho de desempenho.
  - Segundo, como as threads não são automaticamente protegidas umas das outras da mesma forma que os processos, o desenvolvimento de aplicativos multithreaded requer esforço intelectual adicional.

3.1 Threads

- Benefícios do uso de threads
  - Em processos single-threaded, uma chamada de sistema bloqueante pode interromper todo o processo
  - Exemplo: em um programa de planilha eletrônica, uma mudança em uma célula pode desencadear uma série de cálculos, o que bloquearia a interação com o usuário
  - A solução é usar pelo menos duas threads: uma para interação com o usuário e outra para atualização da planilha
  - Uma terceira thread pode ser usada para backup enquanto as outras duas estão em execução

3.1 Threads

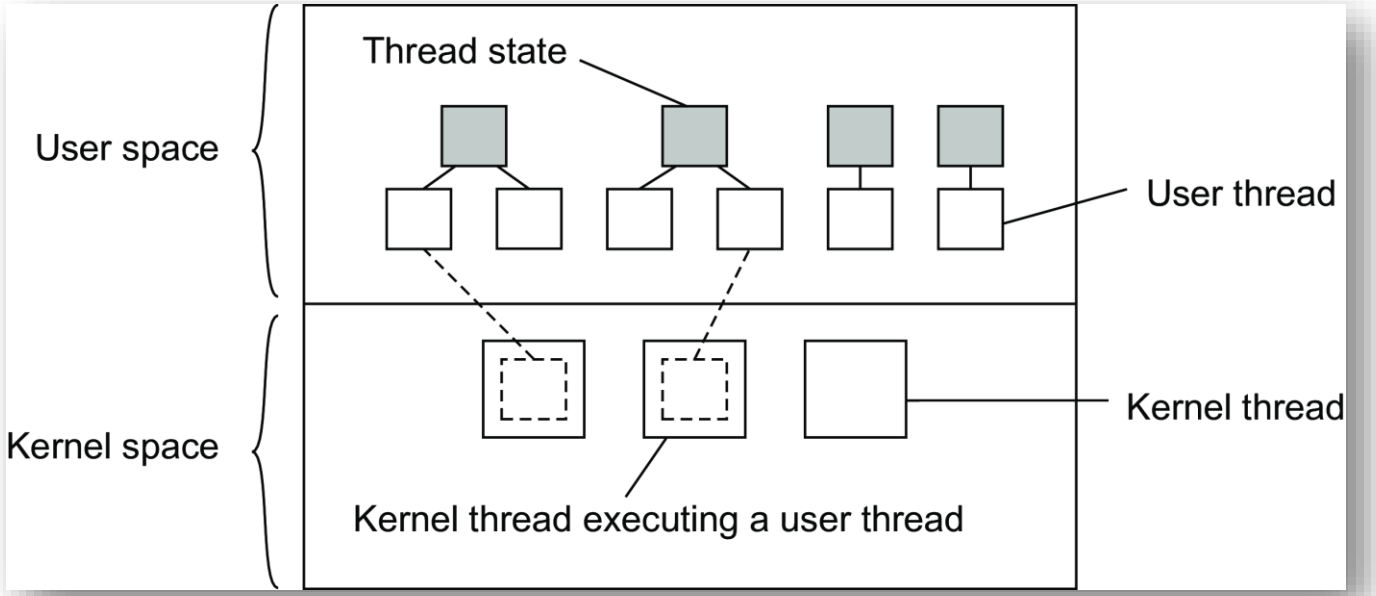
- Benefícios da Multithreading
  - Multithreading permite o uso de paralelismo em sistemas multiprocessadores e multicore
  - Com processamento paralelo, cada thread é atribuída a uma CPU ou core diferente
  - Isso pode ser transparente ao usuário e tornar a execução do processo mais rápida
  - IPC (interprocess communication) pode ser evitado com threads e aumentar a performance
  - Multithreading pode ser mais fácil de estruturar para aplicações com tarefas independentes

3.1 Threads

- Aplicações práticas da Multithreading
  - Servidores em aplicativos cliente-servidor podem se beneficiar da Multithreading
  - Dispositivos como smartphones também podem se beneficiar de processamento paralelo
  - Multithreading pode ser usado para evitar IPC em aplicativos grandes
  - Em vez de usar processos, aplicativos podem ser construídos com threads para comunicação entre partes do sistema

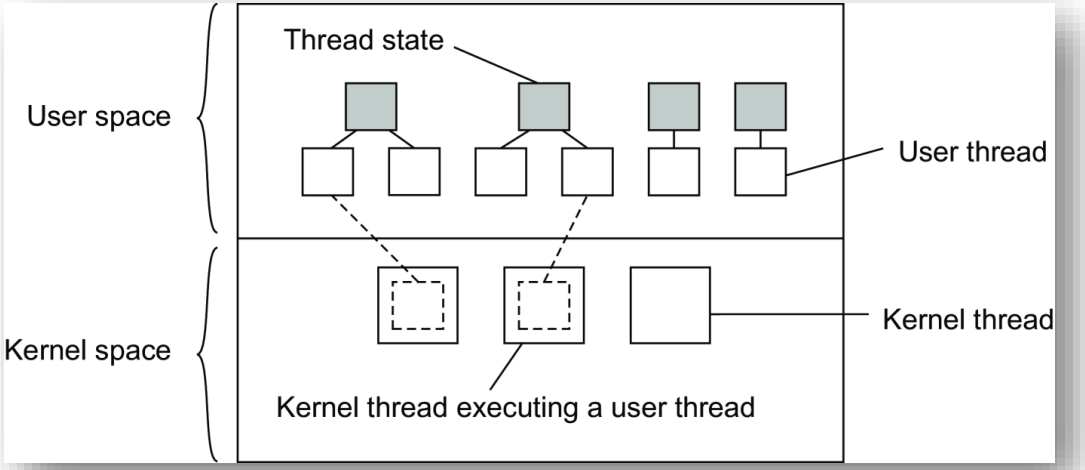
3.1 Threads

- **Implementação de Threads**
  - As threads são uma forma de execução concorrente em um programa
  - Existem duas formas de implementar threads: user space e kernel space



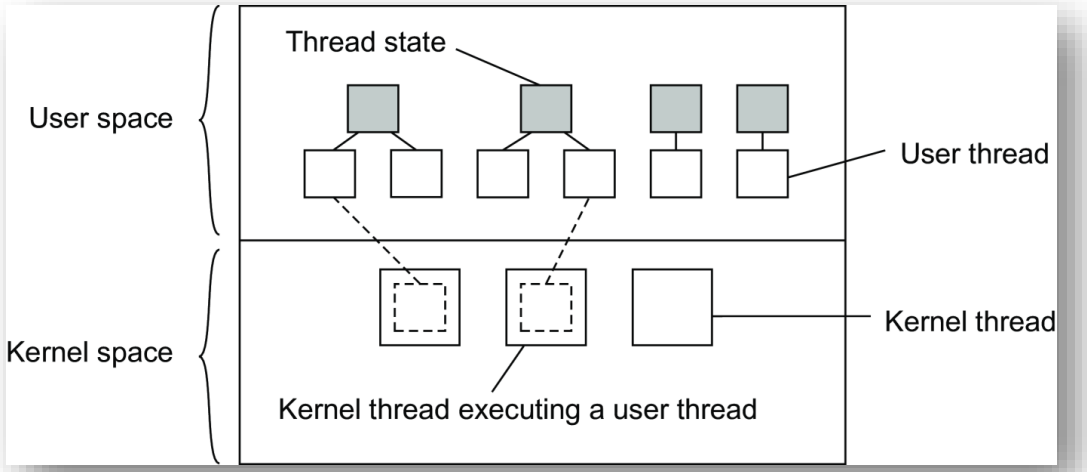
3.1 Threads

- **User Space Threads**
  - As threads são gerenciadas pela aplicação, sem a intervenção do sistema operacional
  - Mais leves, já que não exigem trocas de contexto do kernel
  - Porém, uma thread bloqueada pode bloquear todas as outras threads do processo
  - Implementações comuns em Python são a threading e a asyncio



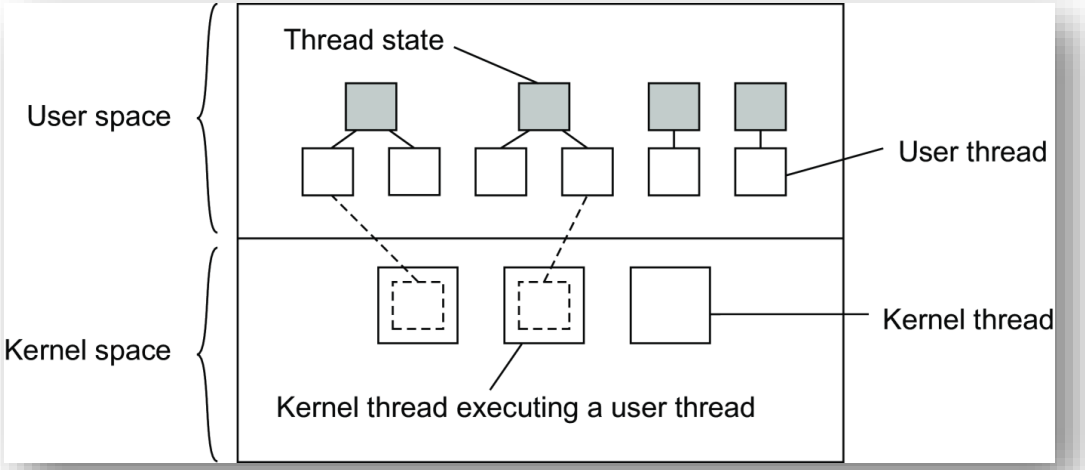
3.1 Threads

- **Kernel Space Threads**
  - As threads são gerenciadas pelo kernel, que as trata como processos leves (também chamados de threads do sistema)
  - O sistema operacional é responsável pelo agendamento das threads
  - Possibilitam que uma thread bloqueada seja escalonada e outras threads continuem a ser executadas
  - Implementações comuns em Python incluem a multiprocess e a concurrent.futures



3.1 Threads

- **Comparação entre User Space e Kernel Space Threads**
  - User Space Threads são mais leves, mas podem bloquear todas as outras threads do processo
  - Kernel Space Threads são mais pesadas, mas possuem melhor escalonamento e permitem que uma thread bloqueada seja escalonada
  - A escolha entre as duas formas de threads depende do tipo de aplicação e de suas necessidades de escalabilidade e bloqueio.



3.1 Threads

- **Proxima aula Programação Multihread em Sistemas Distribuídos**

Continua!