

Sistemas Operacionais I

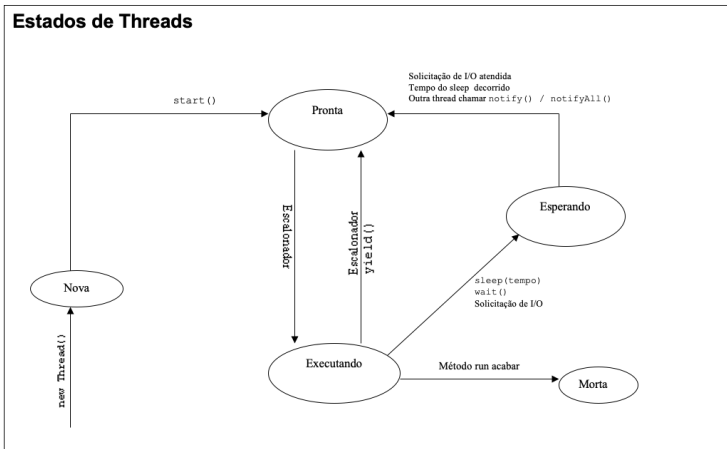
Profa. Kalinka Regina Lucas Jaquie Castelo Branco
kalinka@icmc.usp.br

Universidade de São Paulo

Setembro de 2020

- “*Thread*, ou processo leve, é a unidade básica de utilização da CPU, consistindo de: contador de programa, conjunto de registradores e uma pilha de execução.”
- “*Thread* são estruturas de execução pertencentes a um processo e assim compartilham os segmentos de código e dados e os recursos alocados ao sistema operacional pelo processo. O conjunto de *threads* de um processo é chamado de *Task* e um processo tradicional possui uma *Task* com apenas um *thread*.” **Silberschatz**

Estados de Threads



- O conceito de *thread* foi criado com dois objetivos principais:
 - Facilidade de comunicação entre unidades de execução;
 - Redução do esforço para manutenção dessas unidades.

- Isso foi conseguido por meio da criação dessas unidades dentro de processos, fazendo com que todo o esforço para criação de um processo, manutenção do Espaço de endereçamento lógico e BCP, fosse aproveitado por várias unidades processáveis, conseguindo também facilidade na comunicação entre essas unidades.

- **Processo** - é um espaço de endereço e uma única linha de controle
- **Threads** - é um espaço de endereço e múltiplas linhas de controle
 - O modelo do processo
 - Agrupamento de recursos (espaço de endereço com texto e dados do programa; arquivos abertos, processos filhos, tratadores de sinais, alarmes pendentes etc)
 - Execução
 - O modelo da *Thread*
 - Recursos particulares (PC, registradores, pilha)
 - Recursos compartilhados (espaço de endereço – variáveis globais, arquivos etc)
 - Múltiplas execuções no mesmo ambiente do processo – com certa independência entre as execuções
- **Analogia**
 - Execução de múltiplos *threads* em paralelo em um processo (*multithreading*) e Execução de múltiplos processos em paralelo em um computador

Itens por Processo	Itens por <i>Thread</i>
<ul style="list-style-type: none"> ■ Espaço de endereçamento ■ Variáveis globais ■ Arquivos abertos ■ Processos filhos ■ Alarmes pendentes 	<ul style="list-style-type: none"> ■ Contador de programa ■ Registradores (contexto) ■ Pilha ■ Estado

- Compartilhamento de recursos;
- Cooperação para realização de tarefas.

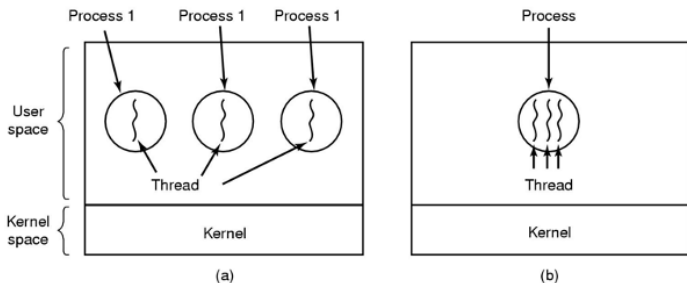


Processo com uma única *thread*



Processo com várias *threads*

Modelo de *Thread*

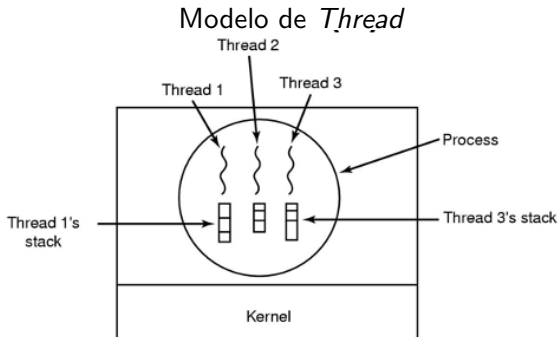


(a) Três processos, cada um com um *thread*

(b) Um processo com três *threads*

- Um *thread* é similar aos programas sequenciais, pois possui um início, sequência de execução e um fim e em qualquer momento uma *thread* possui um único ponto de execução.
- Contudo, um *thread* não é um programa, ele não pode ser executado sozinho e sim inserido no contexto de uma aplicação, onde essa aplicação sim, possuirá vários pontos de execuções distintos, cada um representado por um *thread*.

- Não há nada de novo nesse conceito de processo com um único *thread*, pois o mesmo é idêntico ao conceito tradicional de processo.
- O grande benefício no uso de *thread* é quando temos vários *thread* em um mesmo processo sendo executados simultaneamente e podendo realizar tarefas diferentes.

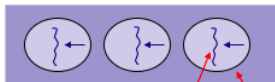


Cada thread tem sua própria pilha de execução

- Dessa forma pode-se perceber facilmente que aplicações *multithreads* podem realizar tarefas distintas ao “mesmo tempo”, dando ideia de paralelismo.
- Exemplo: navegador web HotJava
 - consegue carregar e executar *applets*;
 - executar uma animação;
 - tocar um som;
 - exibir diversas figuras;
 - permitir rolagem da tela;
 - carregar uma nova página; entre outros
- para o usuário todas essas atividades são simultâneas, mesmo possuindo um único processador (possível devido a execução de vários *threads*, provavelmente, uma para cada tarefa a ser realizada.)

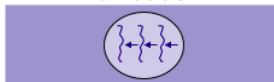
- Tradicionalmente, processos possuem apenas um contador de programas, um espaço de endereço e apenas uma *thread* de controle (ou fluxo de controle);
- *Multithreading*: Sistemas atuais suportam múltiplas *threads* de controle.

a) Três processos



Thread Processo

b) Um processo com três *threads*



- As três *threads* utilizam o mesmo espaço de endereço

- *Thread* é uma entidade básica de utilização da CPU.
 - ou processos leves (*lightweight process*).
- Processos com múltiplas *threads* podem realizar mais de uma tarefa de cada vez;
- Processos são usados para agrupar recursos; *threads* são as entidades escalonadas para execução na CPU
 - A CPU alterna entre as *threads* dando a impressão de que elas estão executando em paralelo;

- Não há proteção entre *threads*, pois é desnecessário
 - Como cada *thread* pode ter acesso a qualquer endereço de memória dentro do espaço de endereçamento do processo, uma *thread* pode ler, escrever ou apagar a pilha de outra *thread*.

Razões para existência de *threads*:

- Em múltiplas aplicações ocorrem múltiplas atividades “ao mesmo tempo”, e algumas dessas atividades podem bloquear de tempos em tempos;
- As *threads* são mais fáceis de gerenciar do que processos, pois elas não possuem recursos próprios - o processo é que tem!
- Desempenho: quando há grande quantidade de E/S, as *threads* permitem que essas atividades se sobreponham, acelerando a aplicação;
- Paralelismo Real em sistemas com múltiplas CPUs.

Considere um servidor de arquivos:

- Recebe diversas requisições de leitura e escrita em arquivos e envia respostas a essas requisições;
- Para melhorar desempenho, o servidor mantém um cache dos arquivos mais recentes, lendo do cache e escrevendo no cache quando possível;
- Quando uma requisição é feita, um *thread* é alocada para seu processamento. Suponha que esse *thread* seja bloqueada esperando uma transferência de arquivos. Nesse caso, outros *threads* podem continuar atendendo a outras requisições.

Considere um navegador WEB:

- Muitas páginas WEB contêm muitas figuras que devem ser mostradas assim que a página é carregada;
- Para cada figura, o navegador deve estabelecer uma conexão separada com o servidor da página e requisitar a figura - tempo;
- Com múltiplas *threads*, muitas imagens podem ser requisitadas ao mesmo tempo melhorando o desempenho.

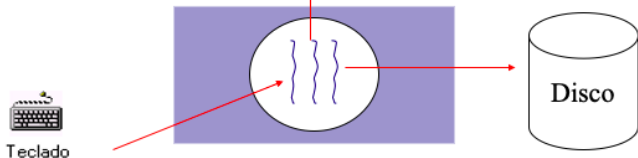
Considere um Editor de Texto:

- Editores mostram documentos formatados que estão sendo criados em telas (vídeo);
- No caso de um livro, por exemplo, todos os capítulos podem estar em apenas um arquivo, ou cada capítulo pode estar em arquivos separados;
- Diferentes tarefas podem ser realizadas durante a edição do livro;
- Várias *threads* podem ser utilizadas para diferentes tarefas.

Threads para diferentes tarefas

Pipeline, antes uma exótica técnica exclusiva dos computadores topo de linha, tem se tornado um lugar comum no projeto de computadores.

A técnica de pipeline nos processadores é baseada no mesmo princípio das linhas de montagem das fábricas: não precisa-se esperar até que a unidade esteja completamente montada para começar a fabricar a próxima.



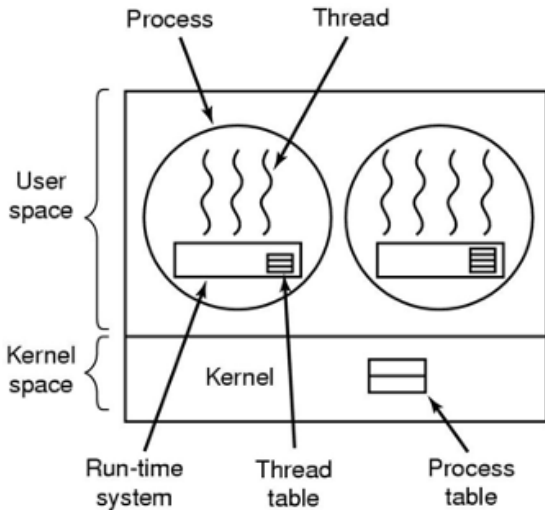
Benefícios

- Capacidade de resposta: aplicações interativas; Ex.: servidor WEB;
- Compartilhamento de recursos: mesmo endereçamento; memória, recursos;
- Economia: criar e realizar chaveamento de *threads* é mais barato;
- Utilização de arquiteturas multiprocessador: processamento paralelo.

Tipos de *Threads*

- Em modo usuário (espaço do usuário): implementadas por bibliotecas no nível do usuário:
 - Criação e escalonamento são realizados sem o conhecimento do kernel:
 - Sistema Supervisor (run-time system): coleção de procedimentos que gerenciam as *threads*;
 - Tabela de *threads* para cada processo.
- Cada processo possui sua própria tabela de *threads*, que armazena todas as informações referentes à cada *thread* relacionada àquele processo.

Threads em Modo Usuário



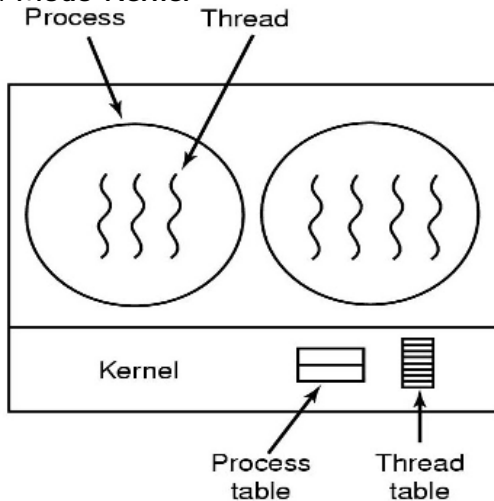
Threads em Modo Usuário

- Vantagens:
 - Alternância de *threads* no nível do usuário é mais rápida do que alternância no *kernel*;
 - Menos chamadas ao *kernel* são realizadas;
 - Permite que cada processo possa ter seu próprio algoritmo de escalonamento.
- Principal desvantagem:
 - Processo inteiro é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema.

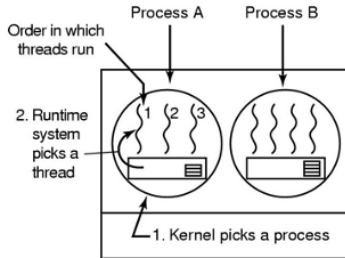
Threads em Modo *Kernel*

- Em modo *kernel*: suportadas diretamente pelo SO;
- Criação, escalonamento e gerenciamento são feitos pelo *kernel*;
 - Tabela de threads e tabela de processos separadas;
 - as tabelas de *threads* possuem as mesmas informações que as tabelas de *threads* em modo usuário, só que agora estão implementadas no *kernel*.

Threads em Modo Kernel



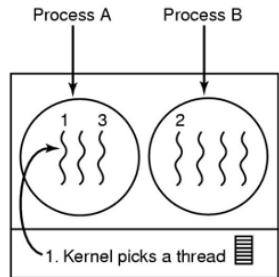
Threads em Modo Kernel e Usuário



Possible: A1, A2, A3, A1, A2, A3

Not possible: A1, B1, A2, B2, A3, B3

Threads em modo usuário



Possible: A1, A2, A3, A1, A2, A3

Also possible: A1, B1, A2, B2, A3, B3

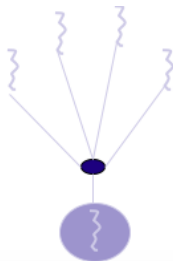
Threads em modo *kernel*

Threads em Modo Kernel

- Vantagens:
 - Processo inteiro não é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema.
- Desvantagem:
 - Gerenciar *threads* em modo *kernel* é mais caro devido às chamadas de sistema durante a alternância entre modo usuário e modo *kernel*.

Modelos *Multithreading*

- Muitos-para-um
 - Mapeia muitas *threads* de usuário em apenas uma *thread* de *kernel*;
 - Não permite múltiplas *threads* em paralelo.

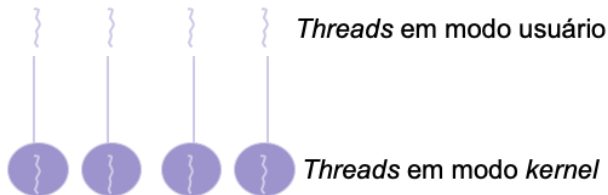


Threads em modo usuário

Thread em modo *kernel*

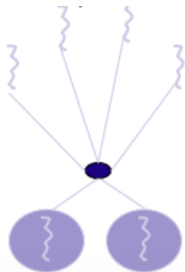
Modelos *Multithreading*

- Um-para-um (Linux, Família Windows, OS/2, Solaris 9)
 - Mapeia para cada *thread* de usuário uma *thread* de *kernel*;
 - Permite múltiplas *threads* em paralelo.



Modelos *Multithreading*

- Muitos-para-muitos (Solaris até versão 8, HP-Ux, Tru64 UNIX, IRIX)
 - Mapeia para múltiplos *threads* de usuário um número menor ou igual de *threads* de *kernel*;
 - Permite múltiplas *threads* em paralelo.



Threads em modo usuário

Thread em modo *kernel*

- Estados: executando, pronta, bloqueada;
- Comandos para manipular *threads*:
 - *Thread_create*;
 - *Thread_exit*;
 - *Thread_wait*;
 - *Thread_yield* (permite que uma *thread* desista voluntariamente da CPU).

Porque threads?

- Simplificar o modelo de programação (aplicação com múltiplas atividades - decomposição da aplicação em múltiplas *threads*);
- Gerenciamento mais simples que o processo (não há recursos atachados – criação de *thread* 100 vezes mais rápida que processo);
- Melhoria do desempenho da aplicação (especialmente quando *thread* é orientada a E/S)
- Útil em sistemas com múltiplas CPUs.

Continuemos com **THREADS**