Departamento de Computação

SISTEMAS OPERACIONAIS 1
21270 A

THREADS

Departamento de Computação
Prof. Kelen Cristiane Teixeira Vivaldini

Departamento de Computação

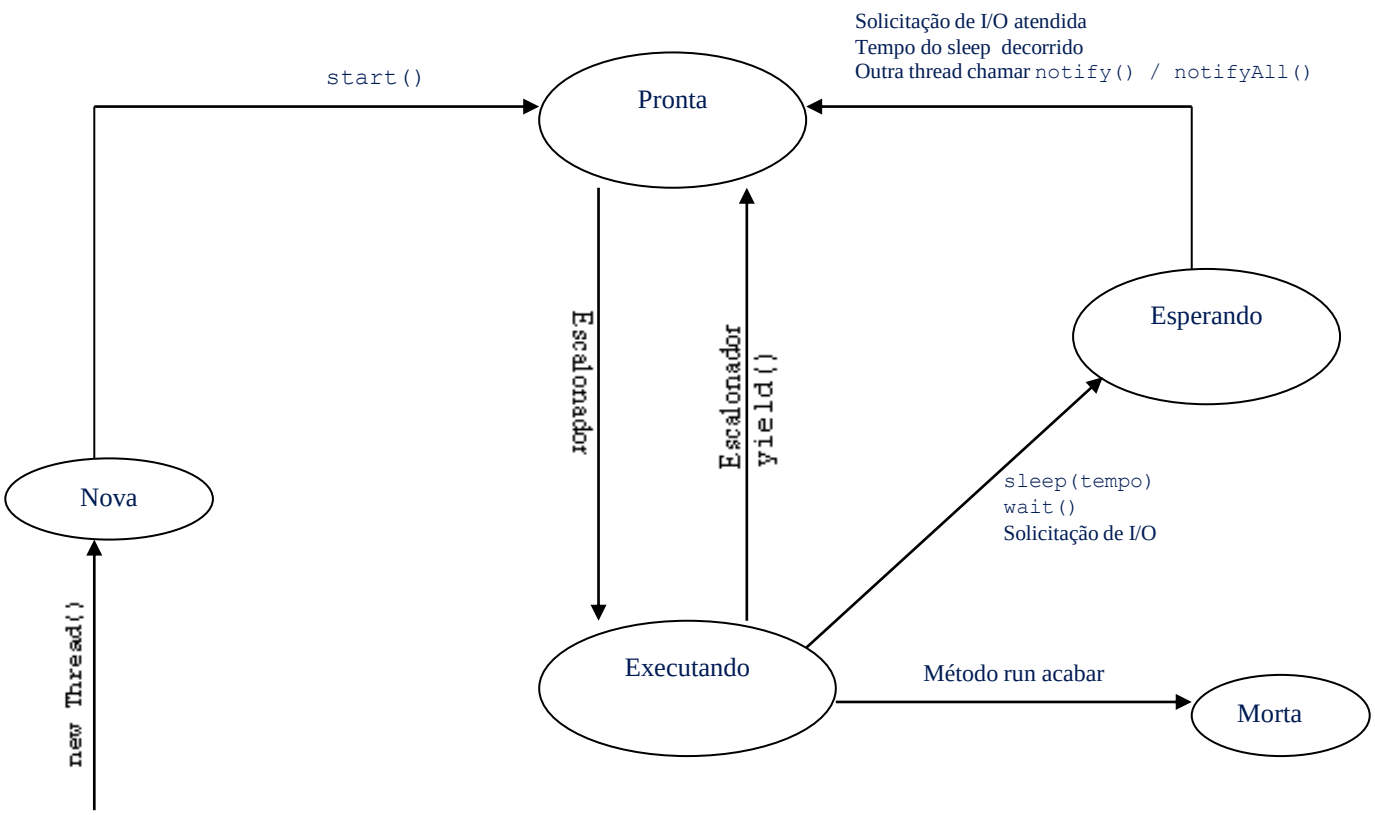
Thread

- "*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**

Thread

Estados de Threads



Departamento de Computação

Thread

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

Thread

- Isso foi conseguido através 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 PCB, fosse aproveitado por várias unidades processáveis, conseguindo também facilidade na comunicação entre essas unidades.

Departamento de Computação

Thread

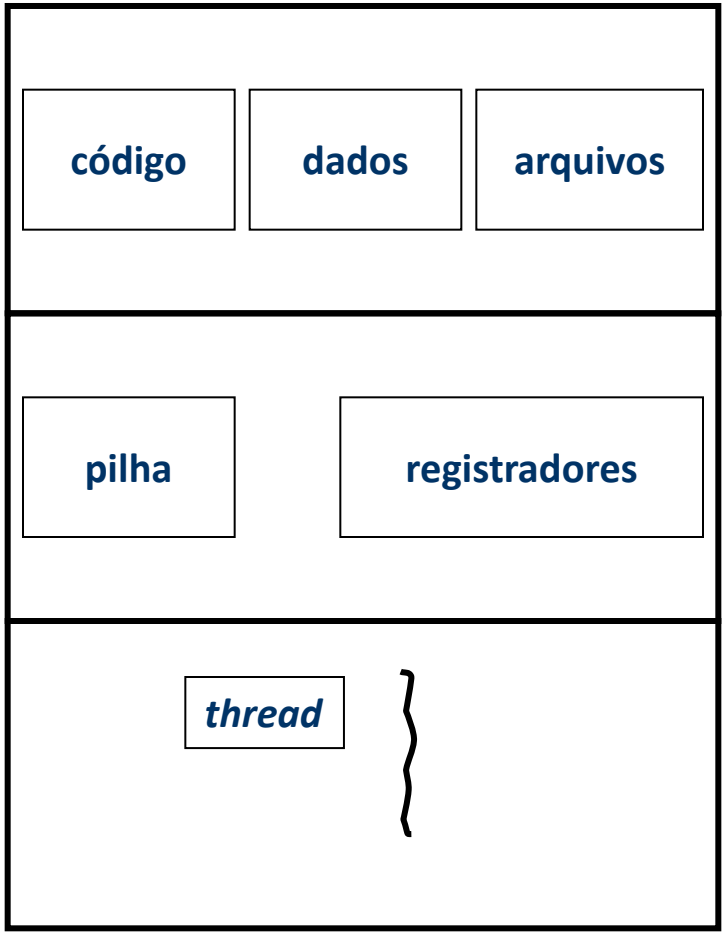
- **Processo ->** um espaço de endereço e uma única linha de controle
  - O Modelo do Processo
    - Agrupamento de recursos (espaço de endereço com texto e dados do programa; arquivos abertos, processos filhos, tratadores de sinais, alarmes pendentes etc)
    - Execução
- **Threads ->** um espaço de endereço e múltiplas linhas de controle
  - 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

Threads

Itens por Processo	Itens por *Thread*
Espaço de endereçamento Variáveis globais Arquivos abertos Processos filhos Alarmes pendentes	Contador de programa Registradores (contexto) Pilha Estado

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

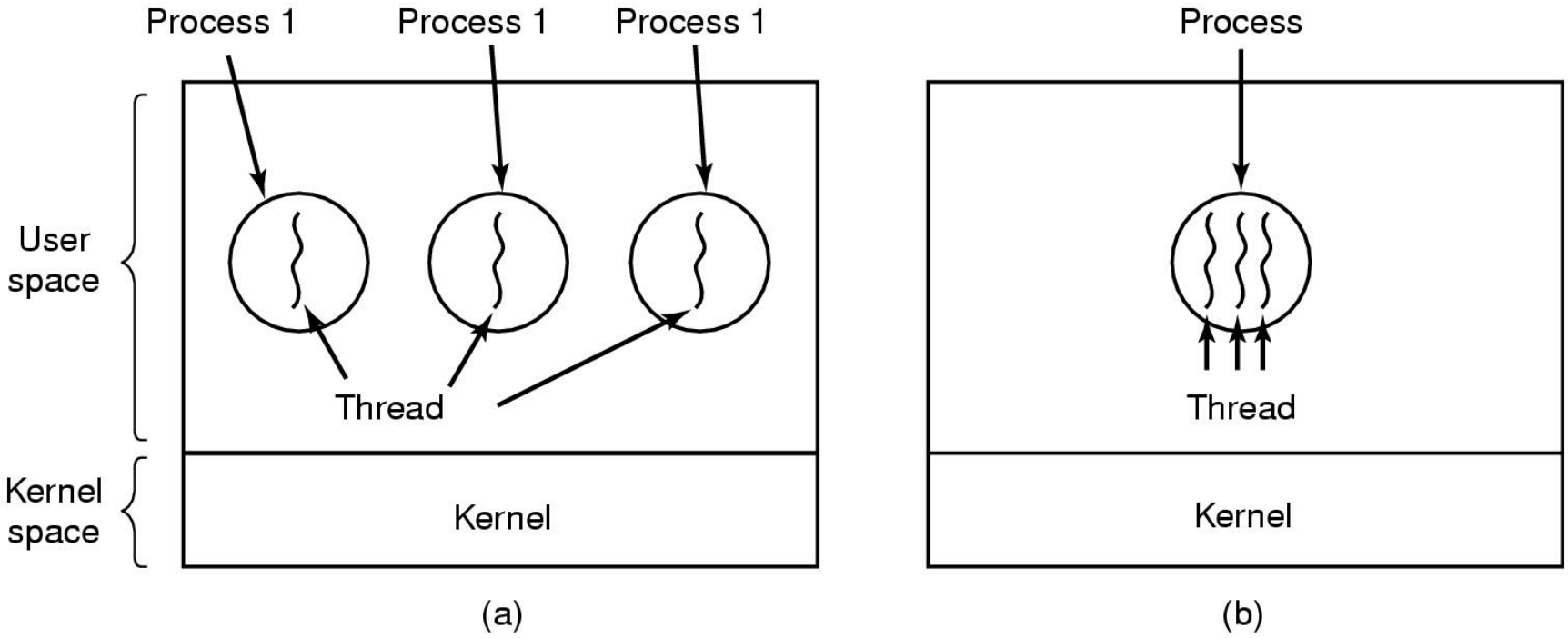
Threads



Processo com uma única *thread*

Processo com várias *threads*

Threads
O Modelo Thread (1)



(a) Três processos, cada um com um *thread*
(b) Um processo com três *threads*

Departamento de Computação

Thread

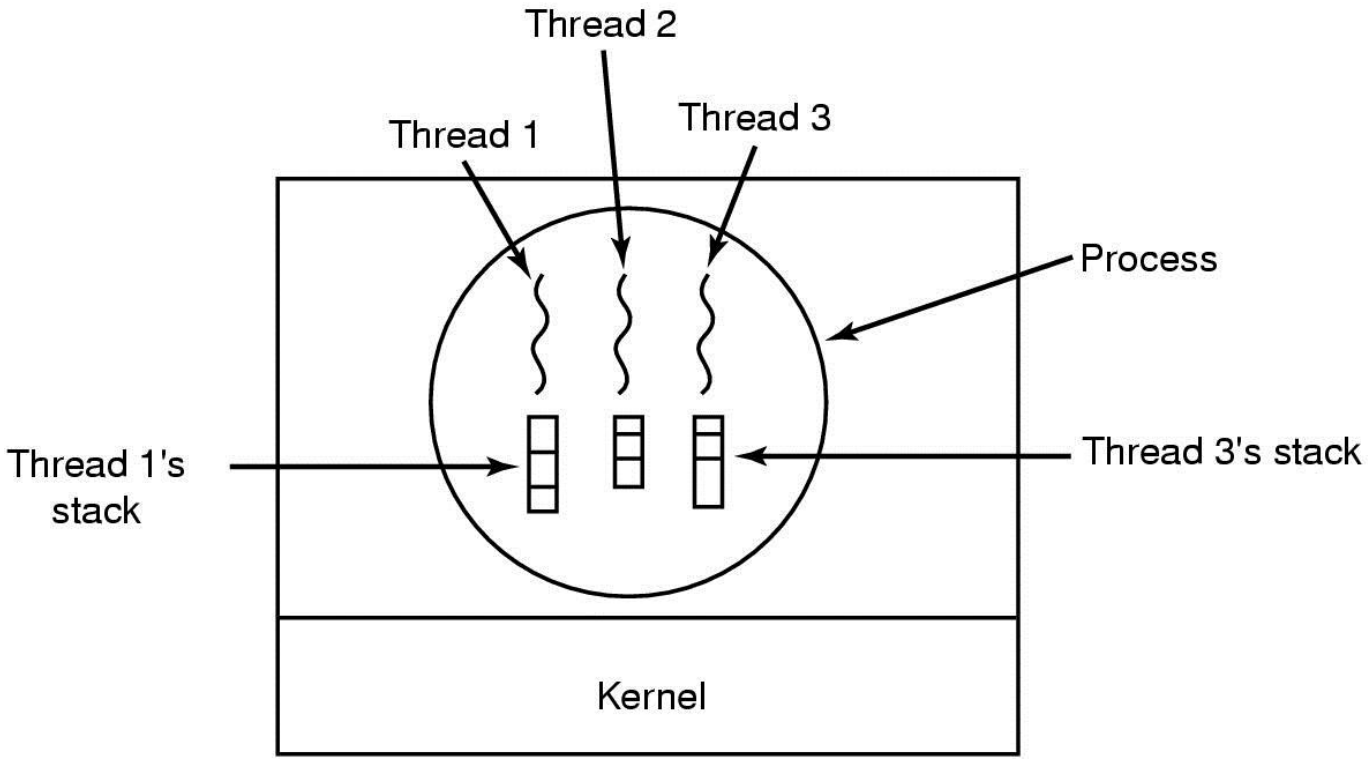
- Um *thread* é similar aos programas seqüenciais, pois possui um início, seqüência de execução e um fim e em qualquer momento uma thread possui um único ponto de execução.
- Contudo, um *thread* não é um programa, ele não pode ser executado sozinho e sim inserido no contexto de uma aplicação, onde essa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

Departamento de Computação

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* num mesmo processo sendo executados simultaneamente e podendo realizar tarefas diferentes.

O Modelo Thread (3)



Cada thread tem sua própria pilha de execução

Departamento de Computação

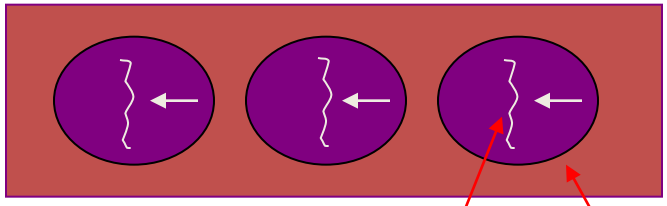
Thread

- Dessa forma pode-se perceber facilmente que aplicações multithreads podem realizar tarefas distintas ao “mesmo tempo”, dando idéia 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.)

Threads

- 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;

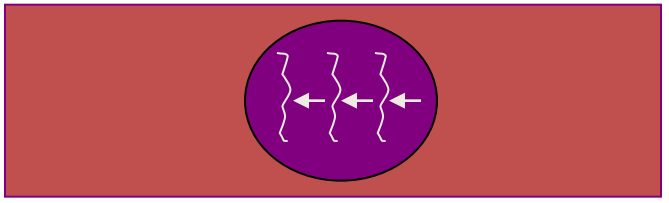
a) Três processos



Thread

Processo

b) Um processo com três *threads*



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

Departamento de Computação

Threads

- *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;

Threads

- 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*;

Departamento de Computação

Threads

- 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.

Departamento de Computação

Threads

- 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, uma *thread* é alocada para seu processamento. Suponha que essa *thread* seja bloqueada esperando uma transferência de arquivos. Nesse caso, outras *threads* podem continuar atendendo a outras requisições;

Departamento de Computação

Threads

- 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;

Threads

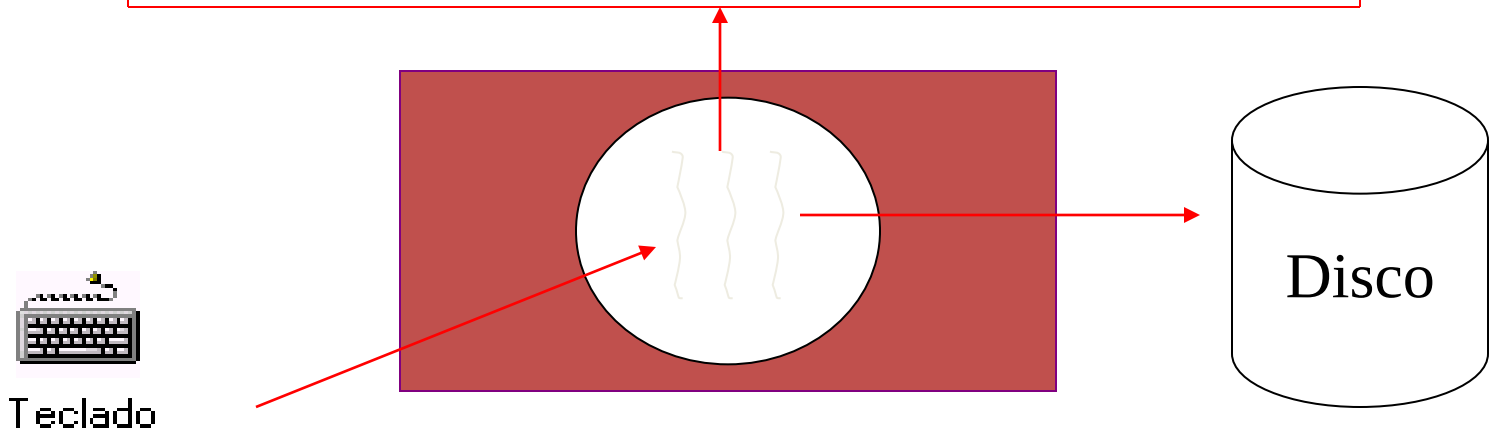
- Considere um Editor de Texto:
  - Editores mostram documentos formatados que estão sendo criados em telas (vídeo);
  - No caso de um livro, por exemplo, todos os capítulos podem estar em apenas um arquivo, ou cada capítulo pode estar em arquivos separados;
  - Diferentes tarefas podem ser realizadas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

Threads

- *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.



Threads

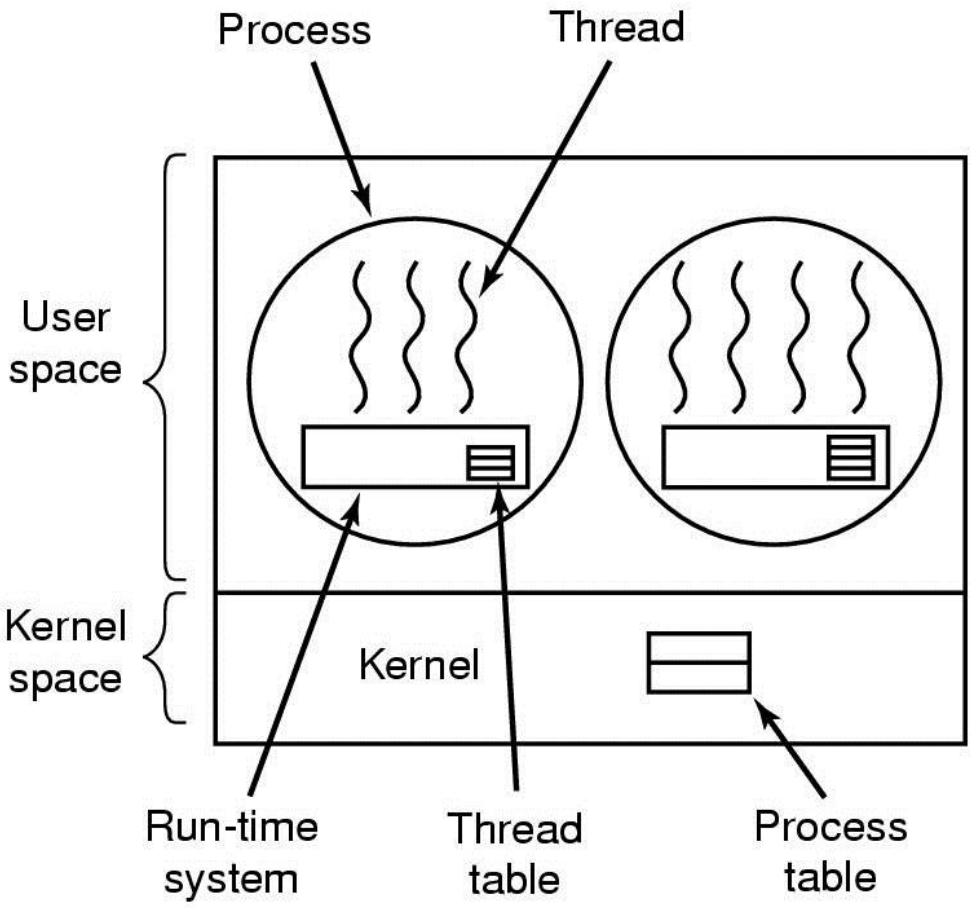
- Benefícios:
  - Capacidade de resposta: aplicações interativas; Ex.: servidor WEB;
  - Compartilhamento de recursos: mesmo endereçamento; memória, recursos;
  - Economia: criar e realizar chaveamento de *threads* é mais barato;
  - Utilização de arquiteturas multiprocessador: processamento paralelo;

Threads

- 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 a informações referentes à cada *thread* relacionada àquele processo;

Departamento de Computação

Threads em modo usuário



Departamento de Computação

Threads em modo usuário

- Tipos de *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;

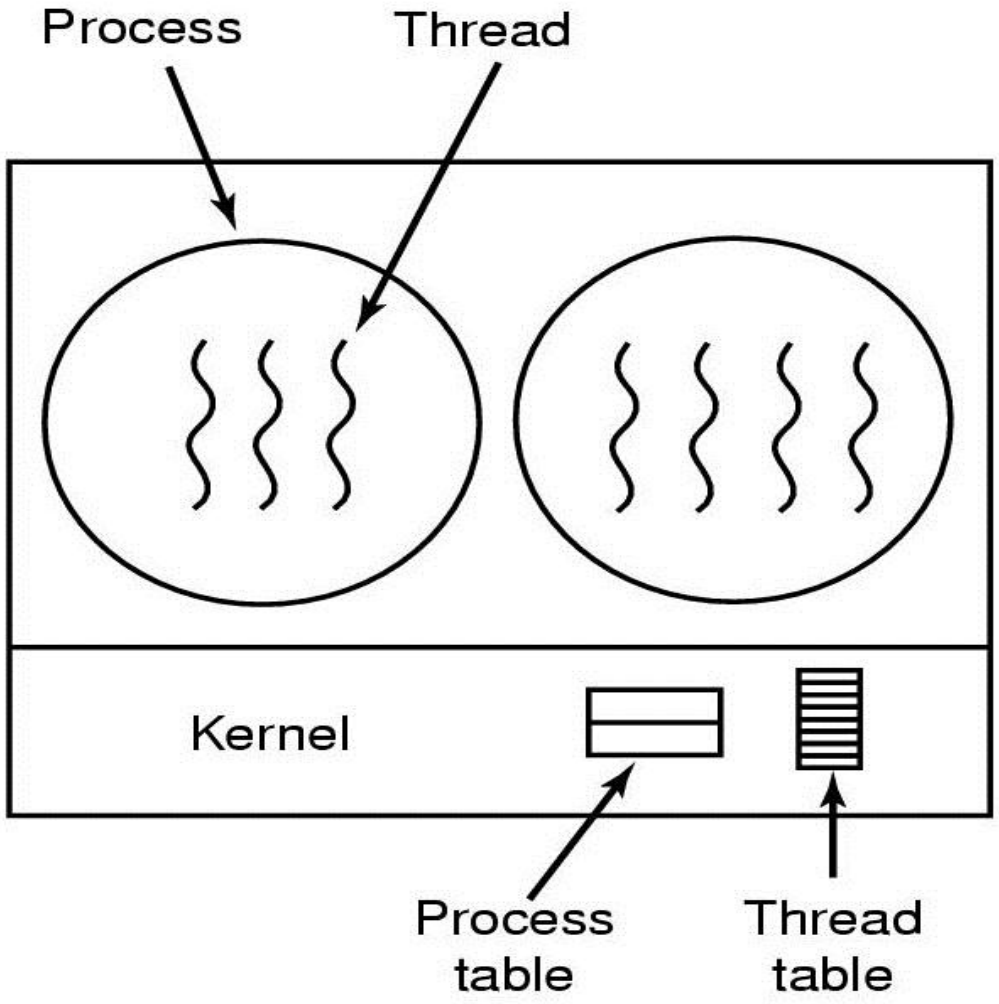
Departamento de Computação

Tipos de *Threads*

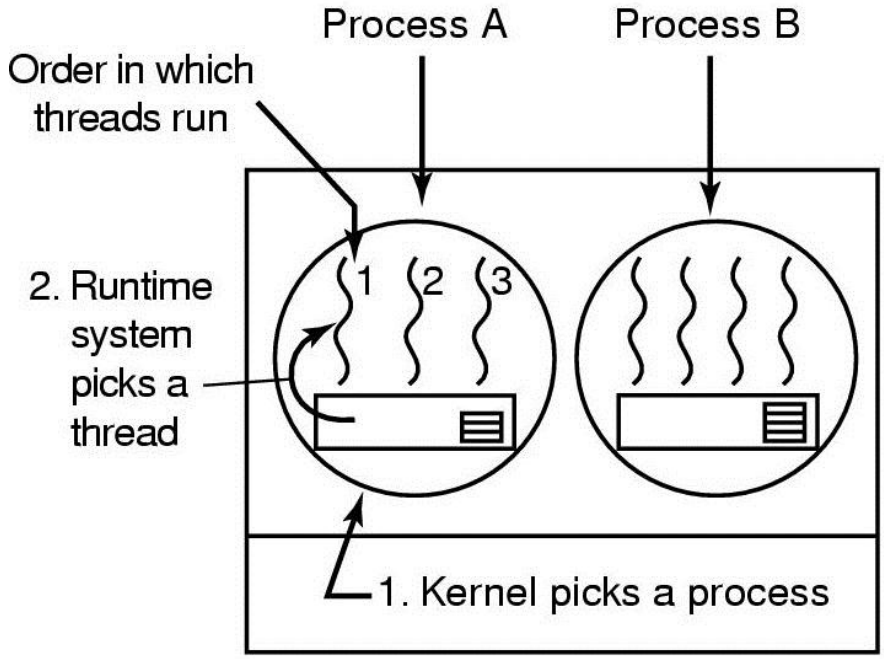
- Tipos de *threads*:
  - Em modo *kernel*: suportadas diretamente pelo SO;
  - Criação, escalonamento e gerenciamento são feitos pelo *kernel*;
    - Tabela de *threads* e tabela de processos separadas;
      - as tabelas de *threads* possuem as mesmas informações que as tabelas de threads em modo usuário,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

Departamento de Computação

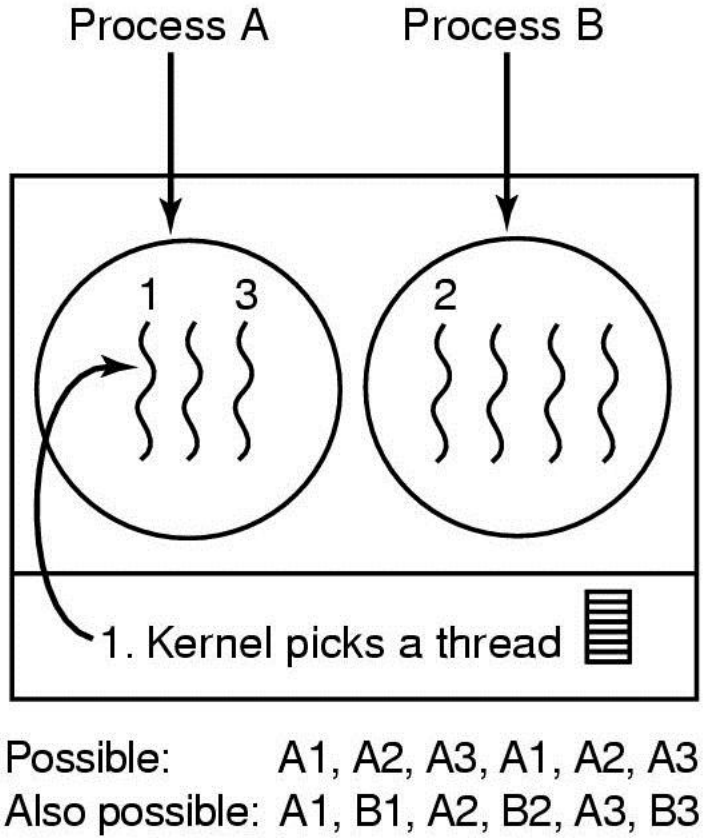
Threads em modo *kernel*



Threads em modo Usuário *x* *Threads* em modo *Kernel*



Threads em modo usuário



Threads em modo *kernel*

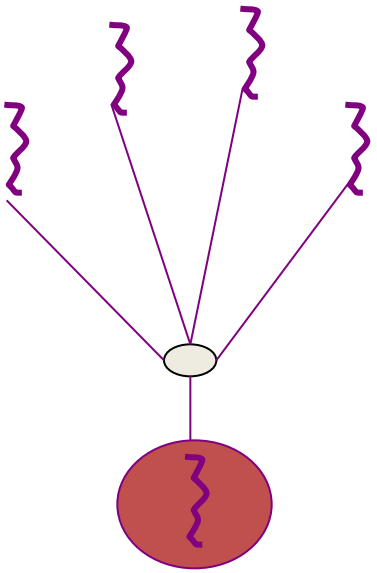
Departamento de Computação

Threads em modo *kernel*

- Tipos de *threads*: em modo *kernel*
- Vantagem:
  - 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*;

Threads

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

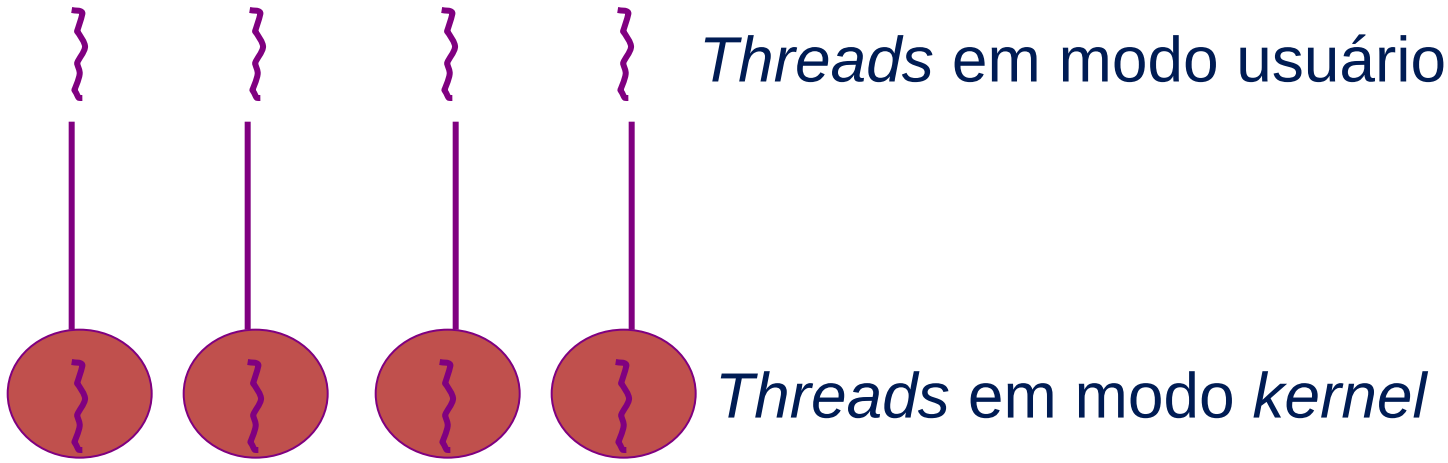


Threads em modo usuário

Thread em modo *kernel*

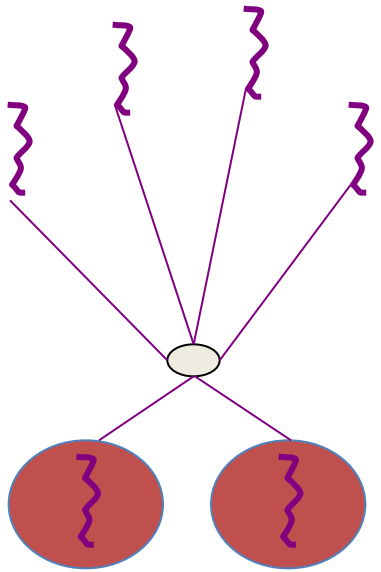
Threads

- 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;



Threads

- 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*

Departamento de Computação

Threads

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

Thread

- **Porquê 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

Thread
Como trabalhar com threads

- Veja os comandos:
  pthread create
  pthread join
  pthread exit
  Para mais informações: man <comando>

Thread
Como criar uma thread

```
int pthread_create(pthread_t *thread,
        pthread_attr_t *attr,
        void * (*start_routine)(void *),
        void *arg);
```

Veja o código: create0.c, mas antes
veja a revisão sobre apontadores para funções em C

Apontadores para funções em C

Veja o código fs.h

```
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

- **base**: apontador para a base do vetor
- **nmemb**: numero de elementos ´
- **size**: tamanho (em bytes) de cada elemento
- **compar**: apontador para uma função que compara elementos e pode retornar 0, valores positivos ou negativos conforme o resultado da comparação.

Veja o código: qsort.c e executa-funcao.c

Como esperar pelo termino de uma thread

```
int pthread_join(pthread_t thr,
        void **thread_return);
```

Veja os codigos: join0.c

Departamento de Computação

Como passar argumentos para uma thread

Exemplo: cada thread pode precisar de um identificador único.

Veja os códigos: create1.c, create2.c, create3.c, create4.c e create5.c

Como encerrar a execução de uma thread

- Comando *return* na função principal da thread (passada como parâmetro em pthread create)
- Análogo ao comando *return* na função main()

Veja os códigos: return0.c, return1.c pthread return.c

Como encerrar a execução de uma thread

- void pthread_exit(void *retval);
- Analogo ao comando exit(status);

Veja os códigos: exit0.c, exit1.c e pthread ´ exit0.c

Create e Join

```
int pthread_create(pthread_t *thread,
        pthread_attr_t *attr,
        void * (*start_routine)(void *),
        void *arg);
int pthread_join(pthread_t thr,
         void **thread_return);
```

Veja o codigo: create ´ join.c

Pilhas de execução

```
int pthread_create(pthread_t *thread,
        pthread_attr_t *attr,
        void * (*start_routine)(void *),
        void *arg);
int pthread_join(pthread_t thr,
         void **thread_return);
```

Veja o codigo: create ´ join.c