Aula de Threads

Estagiário: Maycon Leone M. Peixoto
maycon@icmc.usp.br

Slides de autoria do
Prof. Marcos José Santana baseados no livro
Sistemas Operacionais Modernos de A. Tanenbaum

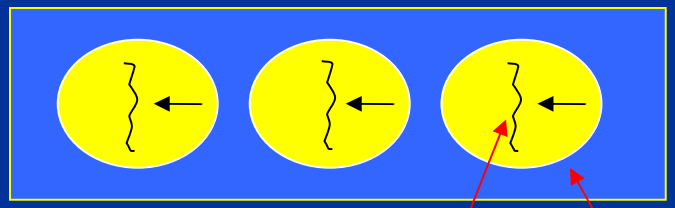
Processos

- Sistemas Operacionais tradicionais:
  - Cada processo tem um único espaço de endereçamento e um único fluxo de controle
- Existem situações onde é desejável ter múltiplos fluxos de controle compartilhando o mesmo espaço de endereçamento:
  - Solução: threads

Threads

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

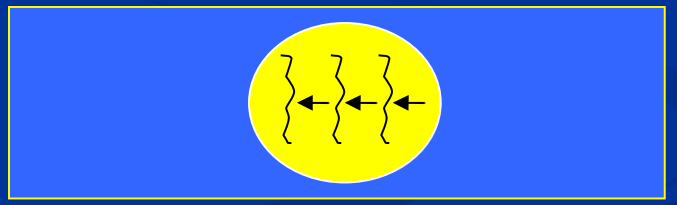
a) Três processos



Thread

Processo

b) Um processo com três *threads*

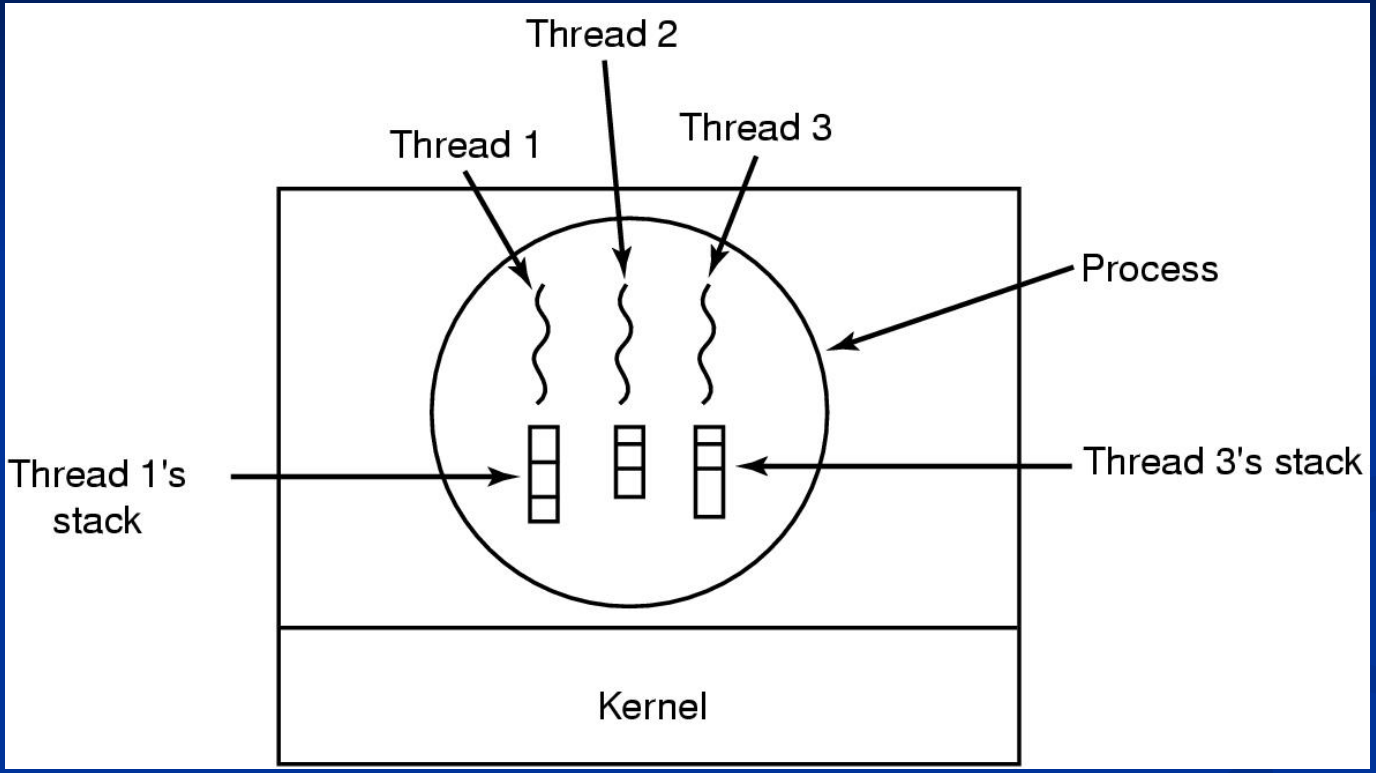


• As três *threads* utilizam o mesmo espaço de endereço

Threads

- ***Thread*** (processo leve) é uma entidade básica de utilização da CPU;
- Processos com **múltiplas *threads*** podem realizar mais de uma tarefa de cada vez;

Threads

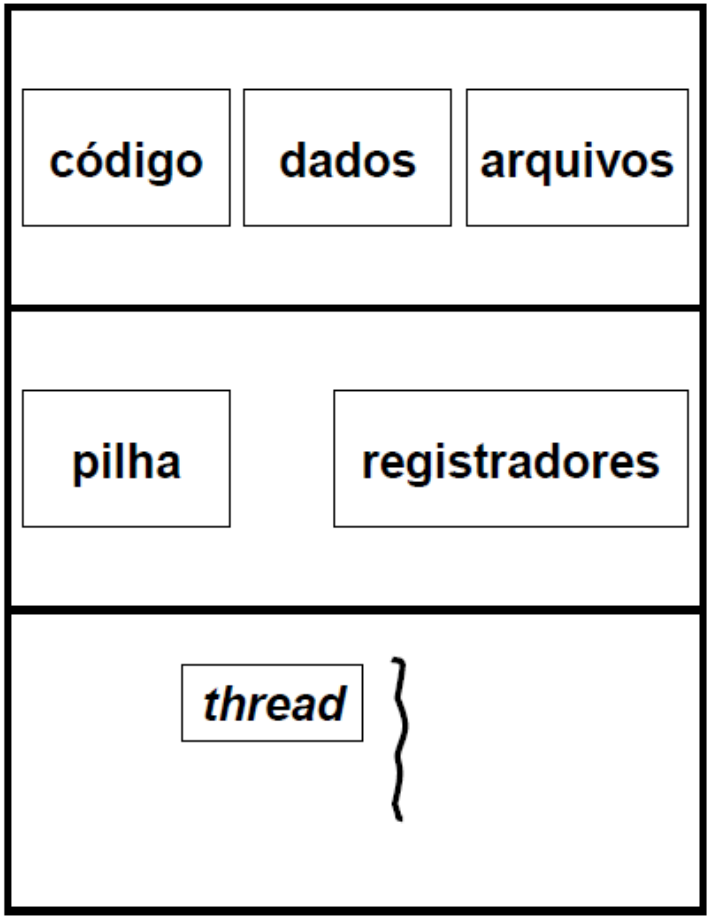


Cada *thread* tem sua pilha de execução

Threads

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

Threads



Processo com uma única *thread*

Processo com várias *threads*

Threads

- 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*;
- Não existe proteção pois:
  - É impossível
  - Não é necessário pois, diferente dos processos que podem pertecem a

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

Threads

- **Exemplo - servidor de arquivos:**
  - Recebe diversas requisições de leitura e escrita em arquivos e envia respostas a essas requisições;
  - Para melhorar desempenho, o servidor mantém uma *cache* dos arquivos mais recentes, lendo da *cache* e escrevendo na *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

Threads

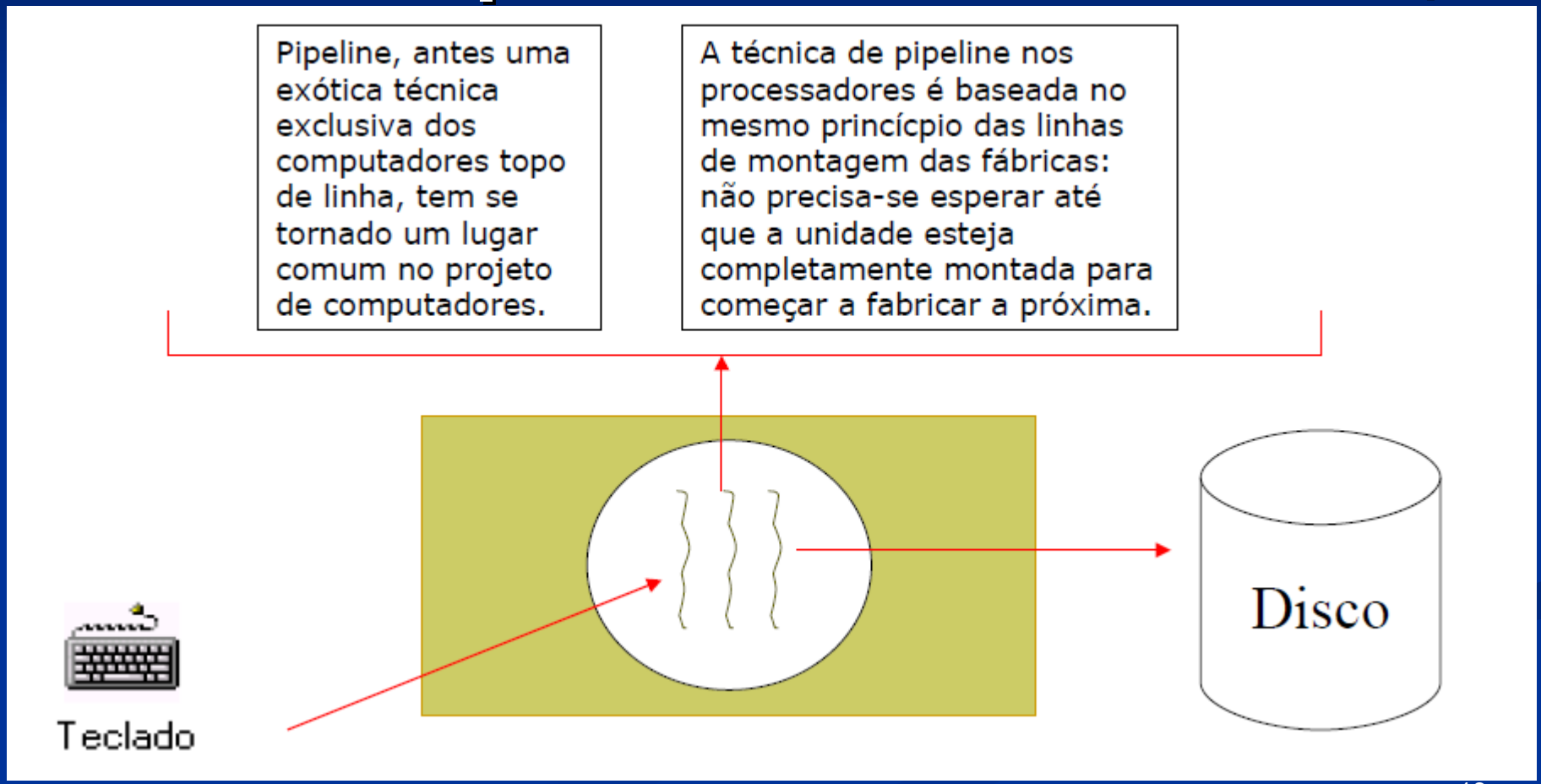
- **Exemplo - navegador WEB:**
  - Muitas páginas WEB contêm muitas figuras que devem ser mostradas assim que a página é carregada;
  - Para cada figura, o navegador deve estabelecer uma conexão separada com o servidor da página e requisitar a figura → consumo de tempo;
  - Com múltiplas *threads*, muitas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

Threads

- **Exemplo - 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;**



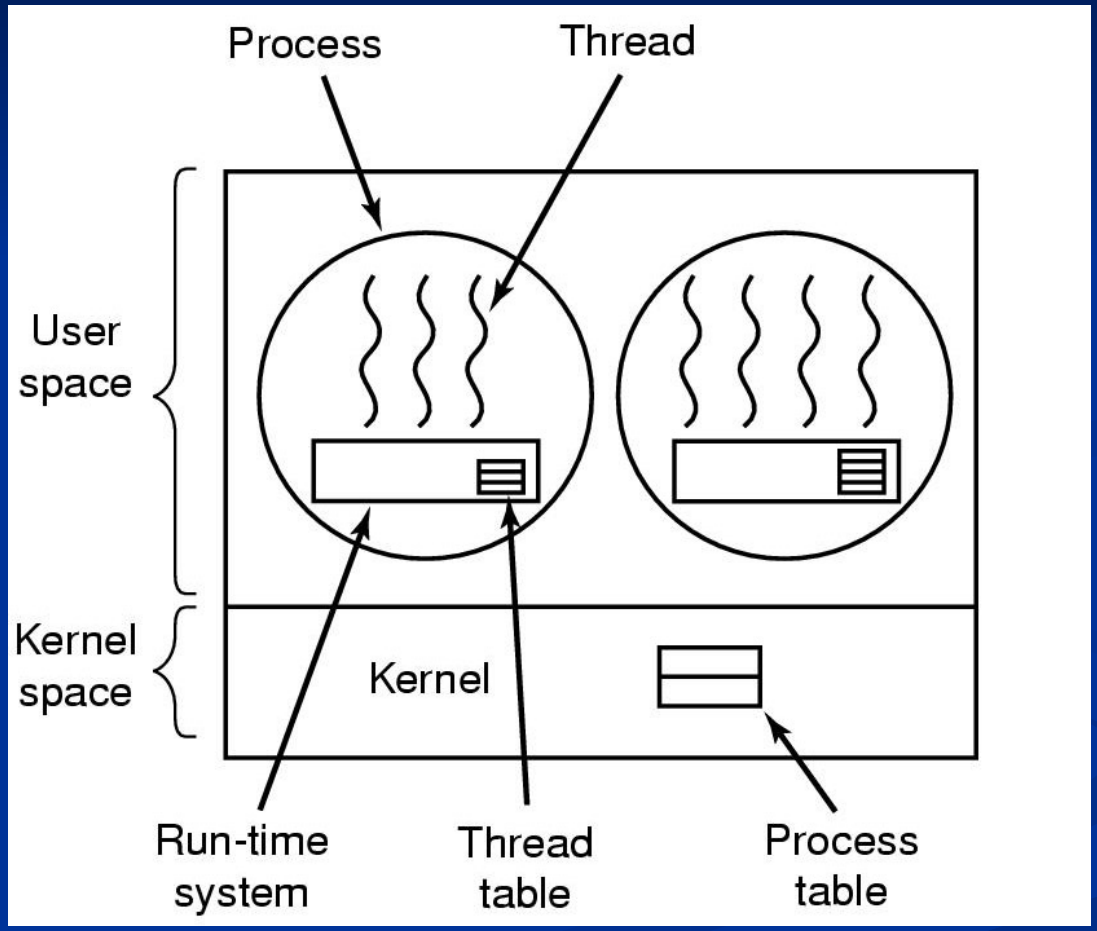
Threads

- **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*:**
  - De usuário: implementadas por bibliotecas no nível do usuário; (Solaris, Mach)
    - Criação e escalonamento são realizados sem o conhecimento do *kernel*;
      - Sistema Supervisor (*run-time system*);
      - Tabela de threads para cada processo;
    - Processo inteiro é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema;

Threads de Usuário



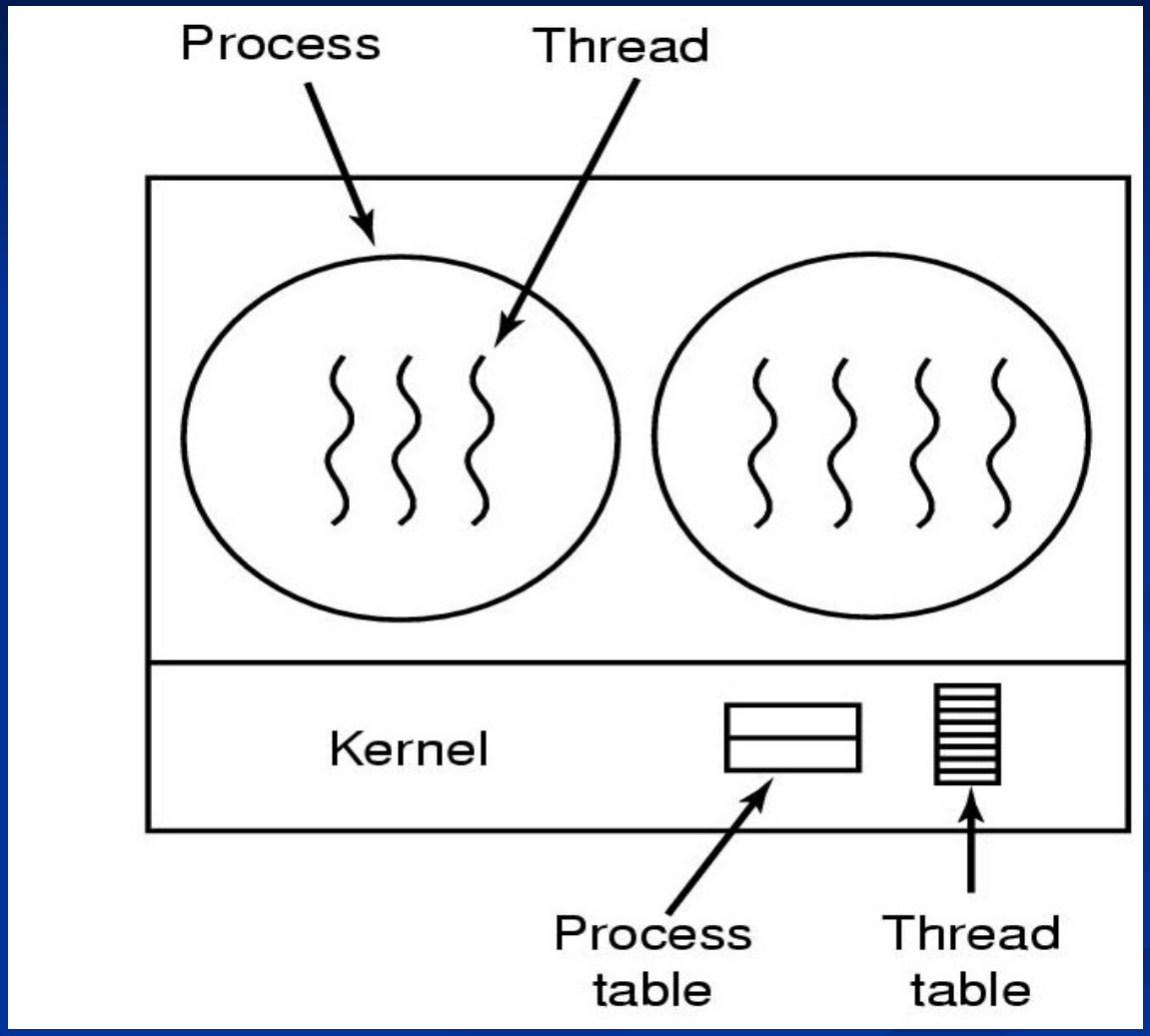
Threads

- Tipos de *threads*:
  - **De *kernel*:** suportadas diretamente pelo SO; (Solaris, WinNT, Digital UNIX)
    - Criação, escalonamento e gerenciamento são feitos pelo *kernel*;
      - Tabela de threads e tabela de processos separadas;
    - Processo inteiro não é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema;

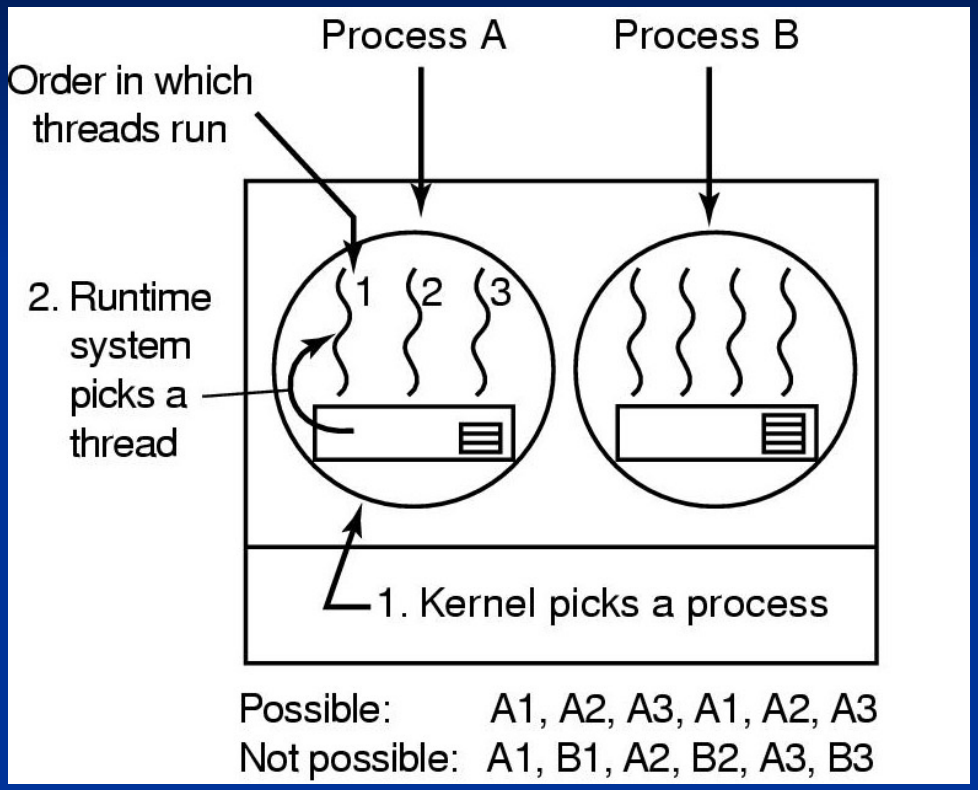
Threads

- Implementação em espaço de usuário; problemas:
  - Como permitir chamadas bloqueantes se as chamadas ao sistema são bloqueantes e uma chamada bloqueante irá bloquear todas as threads?
    - Mudar a chamada ao sistema para não bloqueante, mas isso implica em alterar o SO -> não aconselhável
    - Verificar antes se uma determinada chamada irá bloquear a thread e, se for bloquear, não a executar, simplesmente mudando de thread
  - Page fault
    - Se uma thread causa uma page fault, o kernel, não sabendo da existência da thread, bloqueia o processo todo até que a página que está em falta seja buscada
  - Se uma thread não liberar a CPU voluntariamente, ela executa o quanto quiser
    - Uma thread pode não permitir que o processo escalonador do processo tenha sua vez

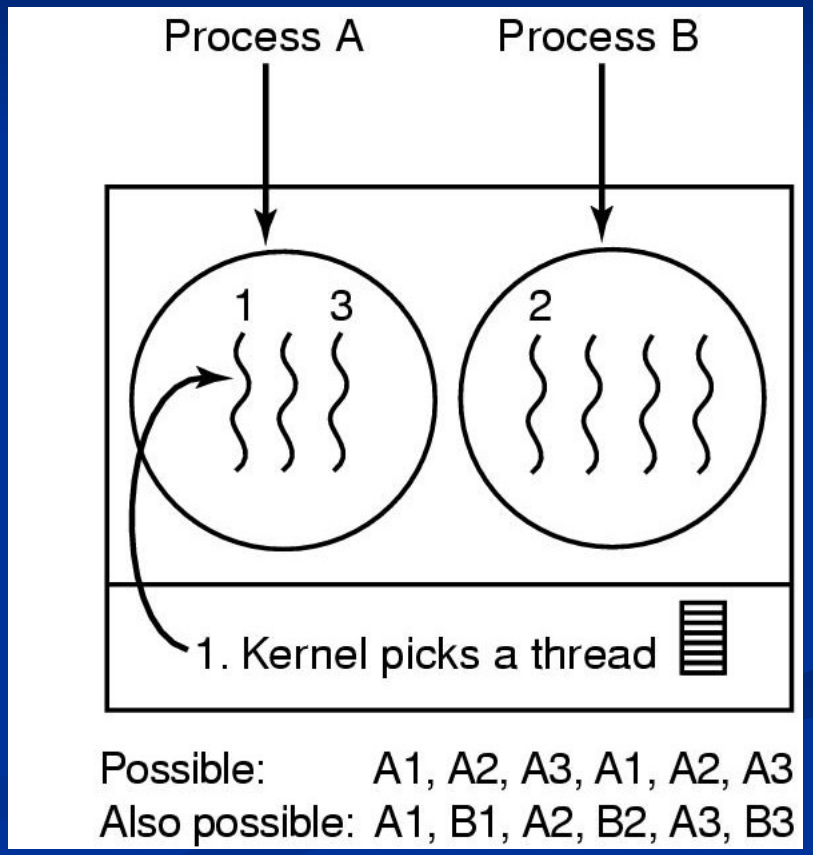
Threads de *Kernel*



Threads de Usuário *x* *Threads* de *Kernel*



Threads de usuário



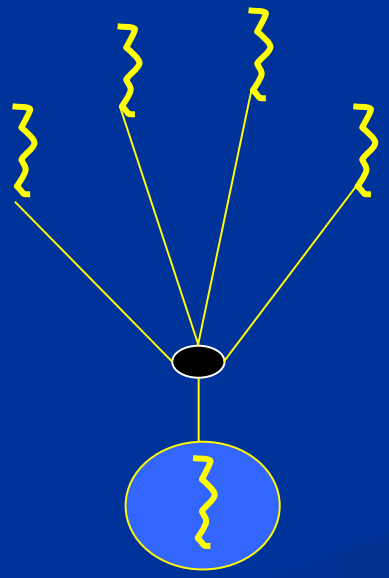
Threads de *kernel*

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;

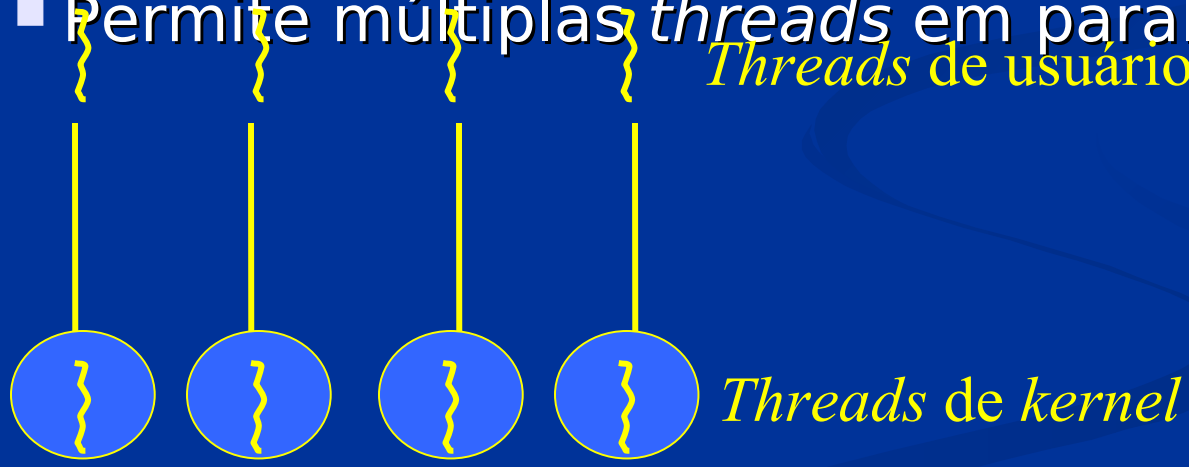


Threads de usuário

Thread de *kernel*

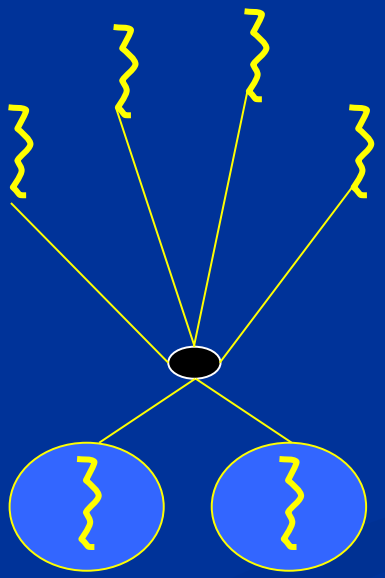
Threads

- **Modelos *Multithreading***
  - Um-para-um: (WinNT, OS/2)
    - Mapeia para cada *thread* de usuário uma *thread* de *kernel*;
    - Permite múltiplas *threads* em paralelo;



Threads

- **Modelos *Multithreading***
  - Muitos-para-muitos: (Solaris, Digital UNIX)
    - Mapeia para múltiplos *threads* de usuário um número menor ou igual de *threads* de *kernel*;
    - Permite múltiplas *threads* em paralelo;



Threads de usuário

Thread de *kernel*

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);;

    . . .

Threads

Threads em Java

- **Corpo de um Thread**
  - Método run()
    - Porção do código que se repete em todas as threads.
  - Duas maneiras distintas de implementação
    - Criar uma subclasse de Thread
    - Implementar a Interface Runnable

Threads em Java

- Um maneira facil de criar um thread é através da herança: subclassing java.lang.Thread

```
class BasicThread extends Thread {
    char c;
        BasicThread(char c) {
          this.c = c;
        }
}
```

Threads em Java

- Para realmente executar o thread deve-se invocar seu método start( );

```
 BasicThread bt  = new
BasicThread('!');
 BasicThread bt1 = new
BasicThread('*');
 bt.start();
 bt1.start();
```

Threads em Java

- O método start( ) aloca recursos do sistema que são requisitados pela thread, escalona a thread para rodar e invoca o método run( ).
- Sobre este código visto: Ele ainda não faz nada!
- Para este thread fazer alguma coisa é necessário reescrever o método run( ).
- O método run( ) é definido na

Threads em Java

```
public void run() {
        for(int i=0; i<100; i++) {
            System.out.print(c);
        }
}
```

- Uma saída possível é:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*!*!*!*!*!*!*!*!*!*!*!*!
*!*!*!*!*!*!*!*!*!*!*!*!*!*!*!*!*!*!*!*!*!!*!*!*!*!*!*!*!*!*!*!*!*!*!*!
*!*!*!*****************************************************
```

- A saida é intercalada porque as threads estão rodando de forma concorrente.

Código

- A corrida dos sapos.

Questão para discussão

- 1) Caso você esteja implementando um servidor Web, qual tipo de thread você utilizaria? (Threads de usuário ou Threads de núcleo). Por quê?