Universidade Federal
de Mato Grosso
Campus Rondonópolis

Sistemas Operacionais

Aula 4 - Threads

Prof. Msc. Cleyton Slaviero

cslaviero@gmail.com

Relembrando...

- Processos
  - PCB
    - Pilha de execução (stack)
    - Ponteiro de pilha
    - Contador de programa
    - Registradores
    - ...
  - Estados
    - Pronto, Bloqueado, Terminado, Executando…
- …por processo!

Universidade Federal
de Mato Grosso
Campus Rondonópolis

Motivação

- Um jogo de realidade aumentada qualquer...

Motivação

- Quais as tarefas precisam ser realizadas?
- Sobre que recursos eles são realizados?

Universidade Federal de Mato Grosso
Campus Rondonópolis

Motivação

- Processos resolveriam?
- Qual o problema?
  - Troca de contexto
  - Compartilhamento de recursos

Universidade Federal de Mato Grosso
Campus Rondonópolis

Solução...

 Threads 

Universidade Federal
de Mato Grosso
Campus Rondonópolis

Porque pensar em threads?

- Em múltiplas aplicações ocorrem múltiplas atividades “ao mesmo tempo”, e algumas dessas atividades podem bloquear de tempos em tempos;
  - Threads são boas quando há muitas ações CPU-bound **e** E/S-bound
- Compartilhamento de espaço de endereçamento e dados entre threads
- Threads são mais leves que processos
  - Mais fácil (e rápido) para criar e destruir
- Threads são boas em computadores paralelos

Mais um exemplo...

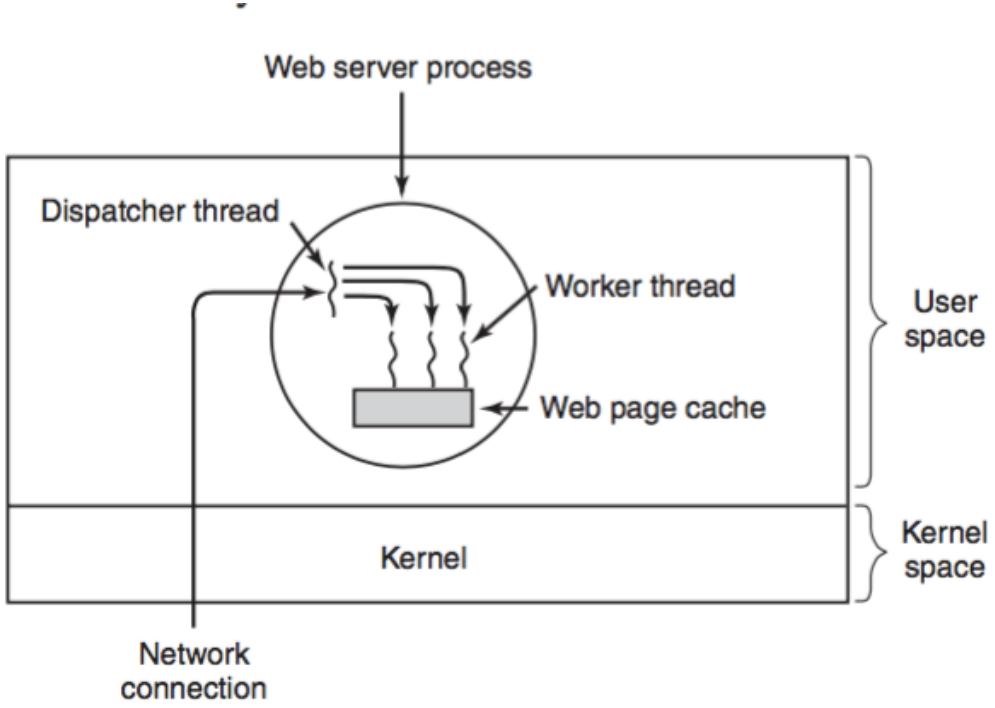
- Escrevendo um livro
  - Quero apagar uma linha do texto
  - Em seguida, vou para uma página específica
- Com threads…
  - Uma thread fica responsável pela reorganização do texto
  - Outra thread busca a página a ser exibida
  - Uma terceira thread pode salvar tudo em disco
- Três processos funcionariam?
  - NÃO! Não seria possível trabalhar com um mesmo arquivo

Outro exemplo

- 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;

Outro exemplo (2)

- Como organizar um servidor web?
  - Uma Thread ”escalonadora” (dispatcher)
    - Lê requisições para a rede
  - Várias threads ”trabalhadora” (worker)
    - São desbloqueadas para lidar com uma nova requisição
    - Se a requisição pode ser feita pela cache, retorna; caso contrário, lê do disco
    - Enquanto bloqueada, outra thread pode ser executada
  - Um conjunto de threads sequenciais



Outro exemplo (3)

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

(a)

Thread dispatcher

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

(b)

Thread worker

Universidade Federal de Mato Grosso
Campus Rondonópolis

Outro exemplo (4)

- E se não usássemos threads?
  - Seria necessário esperar as respostas de cada requisição

Voltando ao processo....

- **O conceito de um Processo pode ser dividido em dois :**
- **Agrupador de recursos**
  - Um espaço de endereçamento (virtual adress space) que contém o texto do programa e dos dados
  - Uma tabela de descritores dos arquivos abertos
  - Informação sobre os processos filhos
  - Código para tratar de sinais (signal handlers)
  - Informação sobre permissões
- **Um contexto de execução (thread)**
  - Uma thread tem um contador de programa (PC) que guarda o endereço sobre a próxima instrução a executar
  - Registradores – valores das variáveis atuais
  - Pilha (stack) – contém o histório de execução com um ”frame” para cada procedimento chamado mas não terminado

Thread - objetivos

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

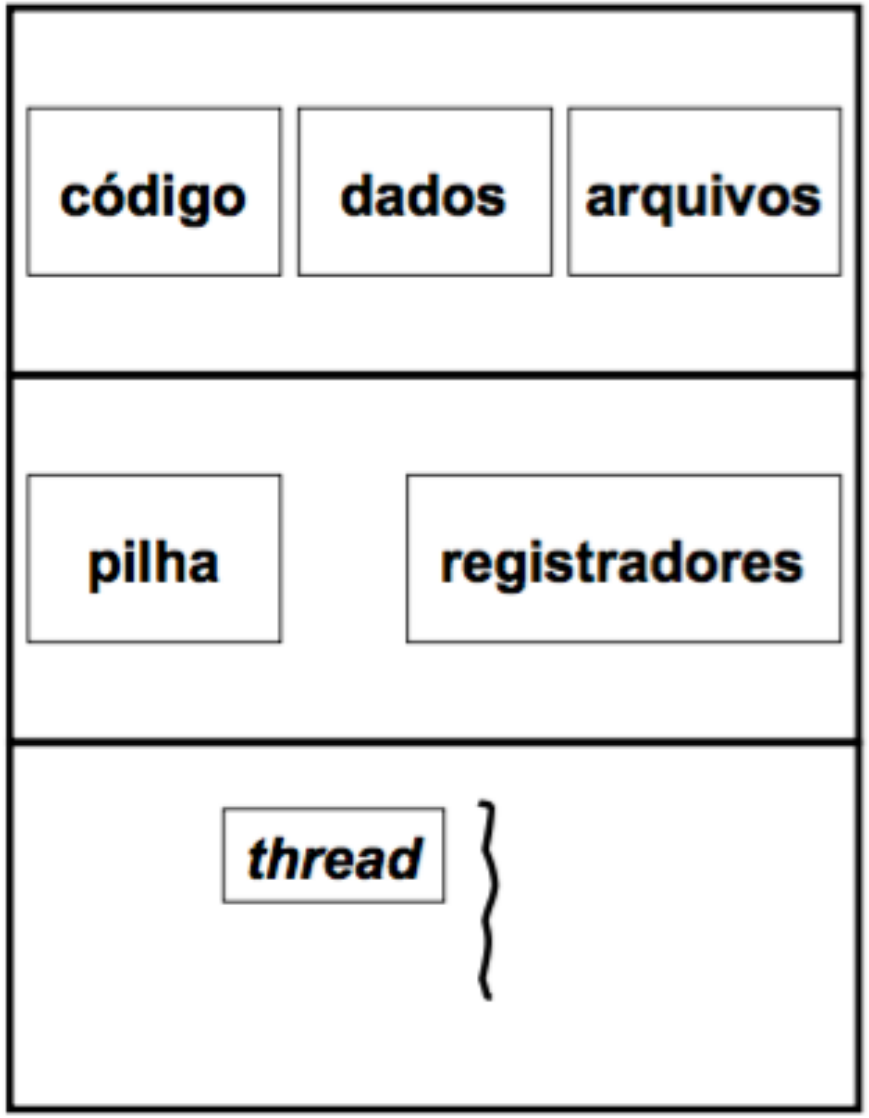
Thread

- **Processo** → um espaço de endereço e uma única linha de controle
- **Threads ->** 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)*
  - Execução de múltiplos processos em paralelo em um computador

Threads

Itens por processo	Itens por thread
Espaço de endereçamento	Contador de programa
Variáveis globais	Registradores (contexto)
Arquivos abertos	Pilha
Processos filhos	Estado
Alarmes pendentes	
Sinais Handlers de sinal	
....	

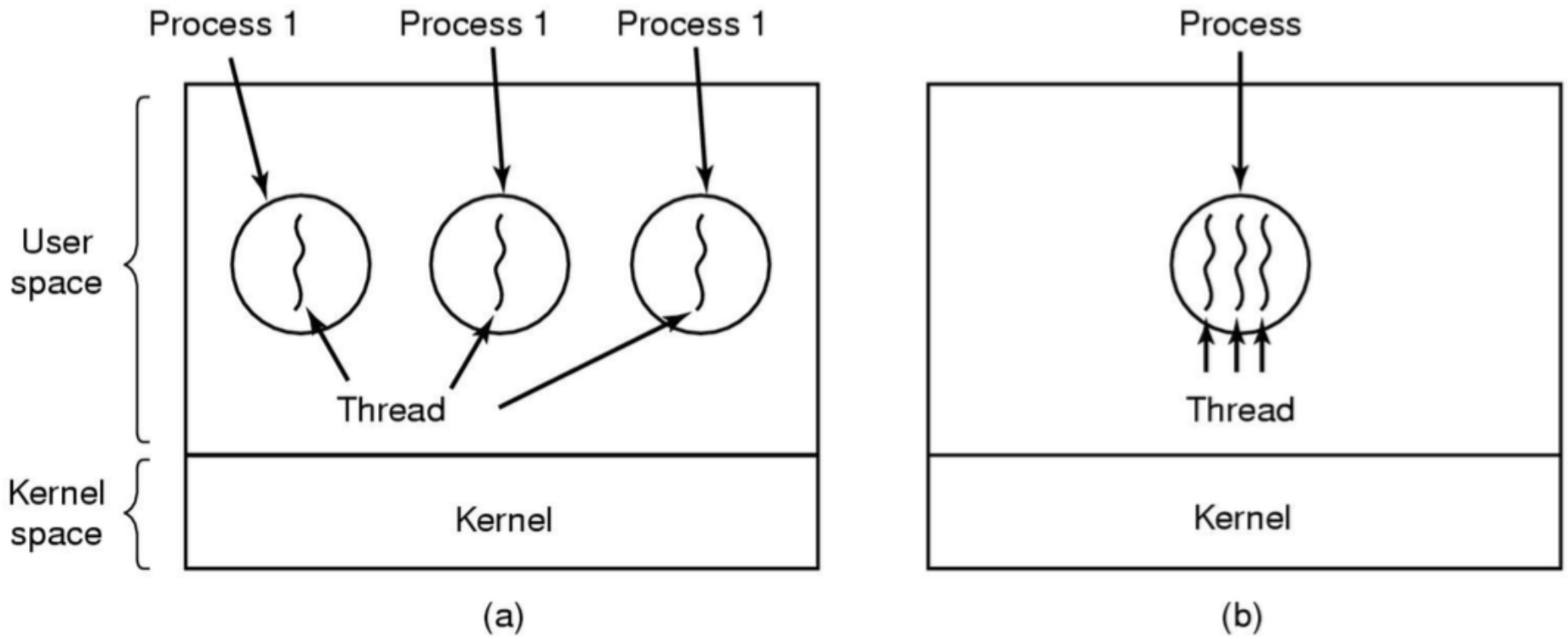
- Compartilhamento de recursos
- Cooperação para realização das tarefas



Processo com uma única *thread*

Processo com várias *threads*

O modelo thread

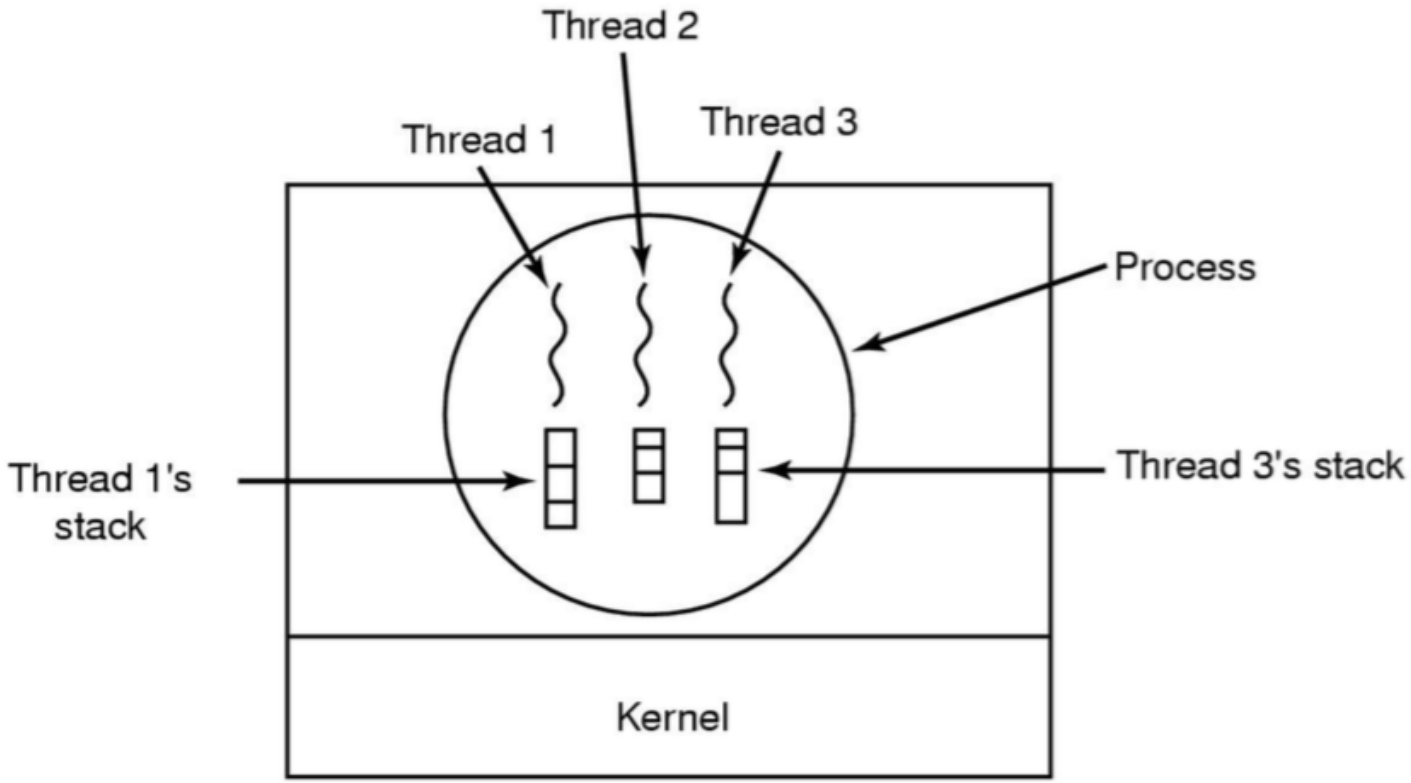


a) Três processos, cada um com uma thread
b) Um procesos com três threads

O modelo thread

- Um processo como já vimos é um processo ”single-threaded” (com um único thread)
- O benefício do uso de threads se dá quando temos múltiplos threads em um mesmo processo, executando **simultaneamente,** e podendo realizar tarefas diferentes
- Simultaneamente?
  - Chaveamento entre threads, em computadores monoprocessados

O modelo thread



Cada thread tem sua própria pilha de execução.

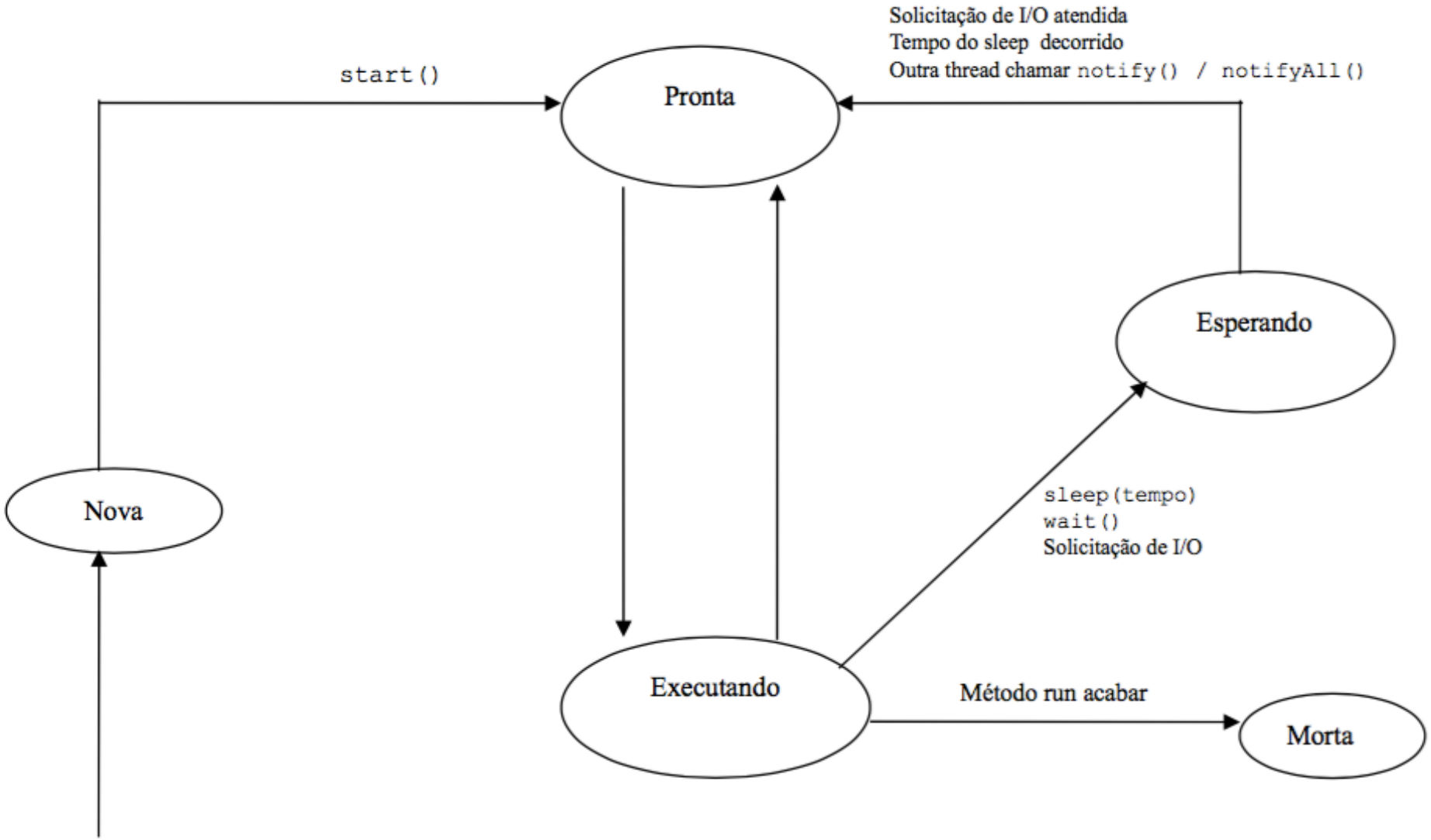
Threads

- Dessa forma pode-se perceber facilmente que aplicações multithreads podem realizar tarefas distintas ao “mesmo tempo”, dando idéia de paralelismo.
- Exemplo: Pokemon Go!
  - Verificar pokemons ao redor
  - Atualizar estatísticas de movimentação (passos andados
  - Buscar ginásios e poké-stops
- 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

- Problema (?)
  - Qualquer thread dentro do espaço de endereçamento do processo pode ler, escrever, ou apagar completamente a pilha de outra thread
- Mecanismo de proteção
  - Não existe
    - É impossível
    - Não é necessário
      - O criador do processo tem controle sobre as threads
- Porém, é preciso sincronizá-las

Estados de Threads



Threads

- Comandos para manipular threads (UNIX):
  - *pthread_create – cria uma nova threads;*
  - *pthread_exit – termina a thread que o chamou;*
  - *pthread_join – espera pela saída de uma outra thread específica;*
  - *pthread_yield* (permite que uma *thread* desista voluntariamente da CPU);

Exemplo

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUMBER OF THREADS 10

void *print hello_world(void *tid) {
        printf("Hello World. Greetings from thread %d\n", tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
        pthread_t threads[NUMBER OF THREADS];
        int status, i;
        for(i=0; i < NUMBER OF THREADS; i++) {
                printf("Main here. Creating thread %d\n", i);

                status = pthread_create(&threads[i], NULL, print hello world, (void *) i);
                if (status != 0) {
                        printf("Oops. pthread create returned error code %d\n", status);
                        exit(-1);
                }
        }
        exit(NULL);
}
```

Implementando threads

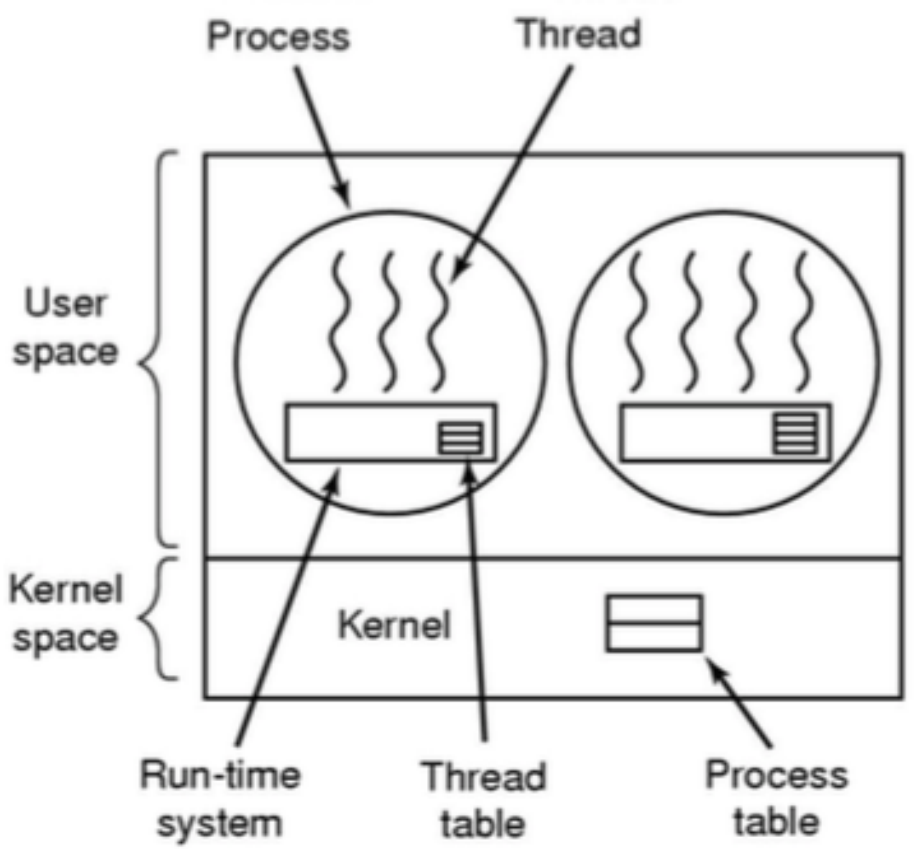
- Três opções
  - Implementar no espaço do usuário
  - Implementar no kernel
  - Implementação híbrida

Tipos de thread

- **Em modo usuário** (espaço do usuário): implementadas por bibliotecas no nível do usuário;
  - Criação e escalonamento são realizados sem o conhecimento do *kernel*;
  - 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;
  - Exemplo no Linux: GNU Portable Thread

Threads em modo usuário

- Processo tem tabela de threads
  - Gerenciada pelo sistema de *runtime*



Threads em modo usuário

- Vantagens:
  - É possível implementar num sistema que não possui threads
  - 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;

Threads em modo usuário

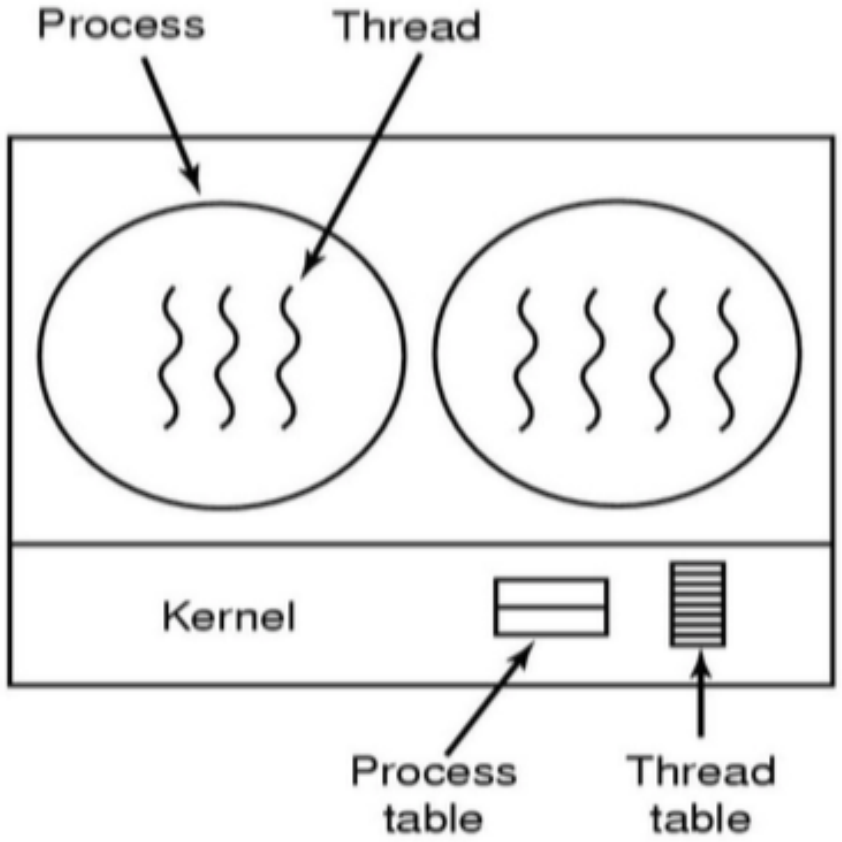
- Desvantagens:
  - Processo inteiro é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema;
  - Thread roda o quanto for preciso
    - Round-robin não é possível em threads
    - Posso ”fazer” um clock, mas pode ser confuso
  - Geralmente são necessários threads em ambientes em que threads bloqueiam muito (exemplo: web servers)

Threads em modo kernel

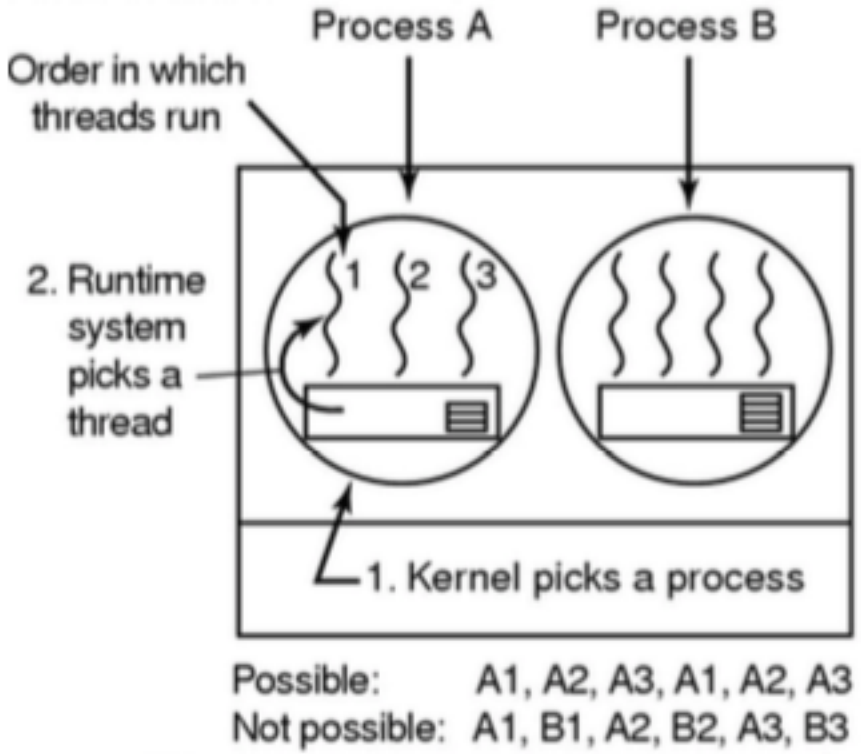
- Suportadas diretamente pelo SO;
- Tabela de *threads* e tabela de processos separadas;
- As tabelas de *threads* possuem as mesmas informações que as tabelas de threads em modo usuário, só que agora estão implementadas no *kernel*;
- No Linux e em C, pode ser criada pelo comando kernel_thread()
- Criação, escalonamento e gerenciamento são feitos pelo *kernel*;

Universidade Federal
de Mato Grosso
Campus Rondonópolis

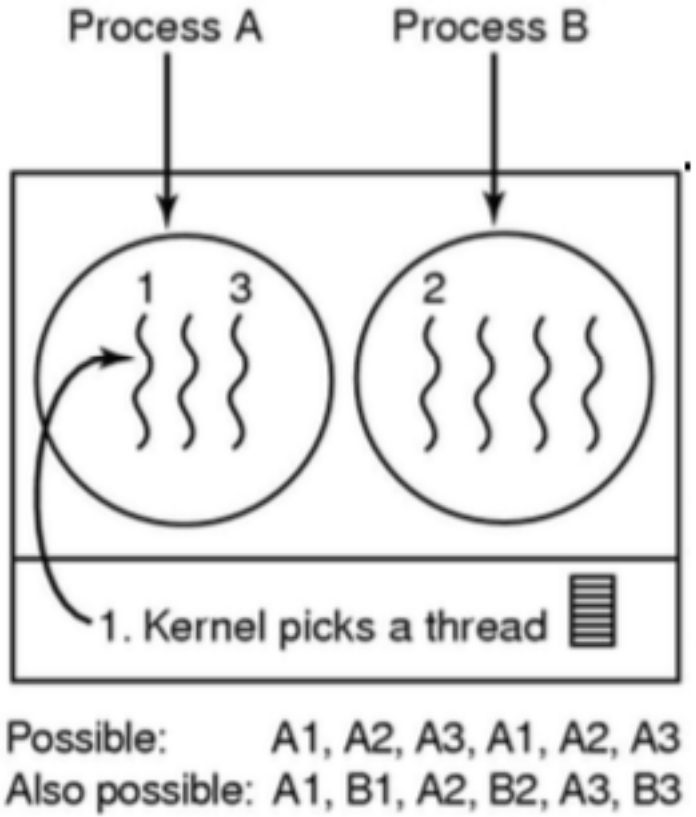
Threads em modo kernel



Threads em modo usuário vs. threads em modo kernel



Threads em modo usuário



Threads em modo kernel

Threads em modo kernel

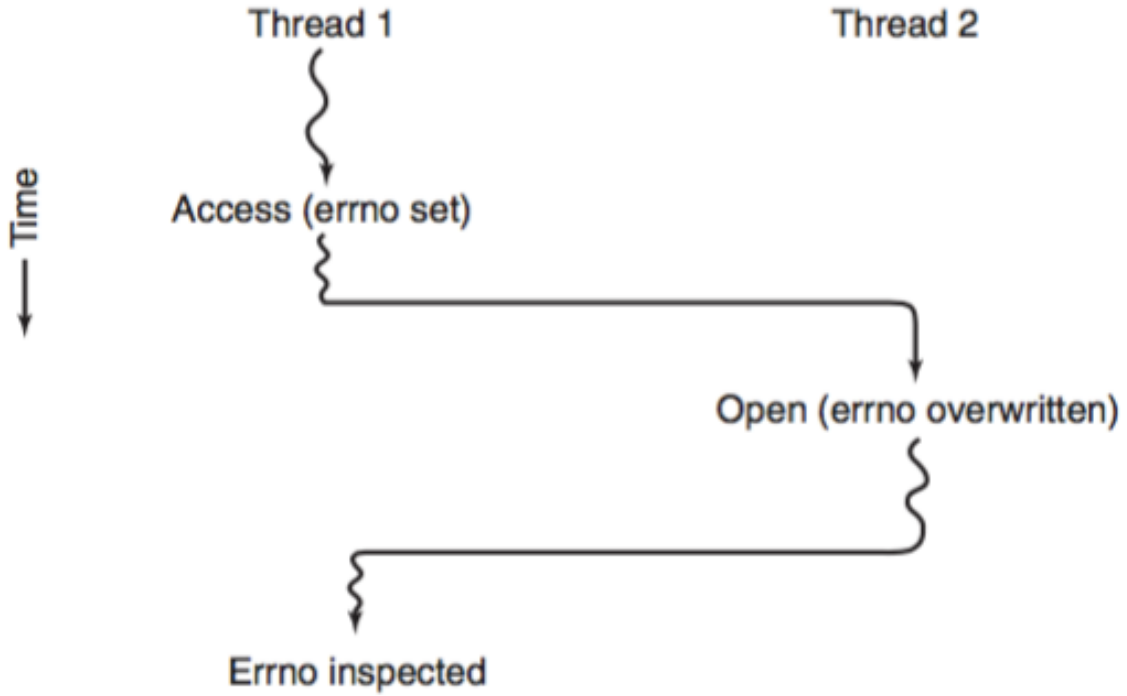
- Vantagem:
  - Processo inteiro não é bloqueado se uma *thread* realizar uma chamada bloqueante ao sistema;
- Desvantagem:
  - Gerenciar threads em modo *kernel* é mais caro devido às chamadas de sistema durante a alternância entre modo usuário e modo *kernel*;

De single-thread para multithreading

- Programas feitos para processos com apenas uma thread
- Questões relacionadas
  - Acesso a variáveis
  - Múltiplas chamadas um mesmo procedimento
  - Alocação de memória
  - Sinais
  - Gerenciamento da pilha

De single-thread para multithreading

- Acesso a variáveis
  - O código de uma thread pode ter vários procedimentos
  - Várias variáveis
    - Variáveis locais, ok
    - Variáveis globais, problemas!
  - Exemplo: *errno*
    - Identificação do erro em uma chamada de sistema
    - Se a thread é interrompida, e a outra também tem um erro
  - Soluções
    - Proibir variáveis globais
    - Variáveis globais privadas
      - Questões de acesso



De single-thread para multithreading

- Múltiplas chamadas a um mesmo procedimento
  - Muitos procedimentos não consideram a possiblidade de uma segunda chamada se a primeira não terminou
  - Exemplo: envio de mensagem pela rede
    - E se uma interrupção de clock força a saída da thread que acabou de construir uma mensagem no buffer?
    - O buffer pode ser sobrescrito!

De single-thread para multithreading

- Alocação de memória
  - Procedimentos de alocação de memória mantém tabelas sobre uso da memória
  - Se o procedimento está atualizando as tabelas, ela pode estar num estado inconsistente
  - Ocorrendo uma troca de threads….
    - ☹

De single-thread para multithreading

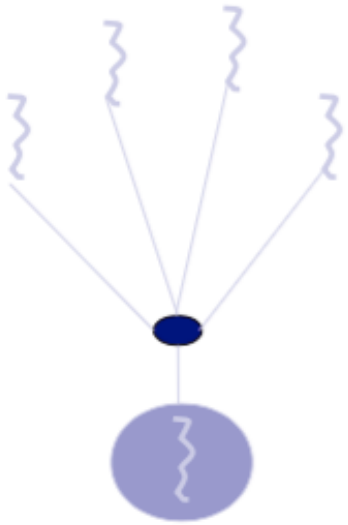
- Sinais
  - Alguns sinais são específicos para a thread, outros não
  - Exemplo: thread chama ’alarm’
    - Se threads são de usuário, para qual thread vai o sinal?
  - Outras não são específicas para thread
    - Mesmos problemas

De single-thread para multithreading

- Gerenciamento de pilha
  - Geralmente, quando uma pilha de processo tem overflow, o SO aumenta o tamanho da pilha
  - Se o processo tem múltiplas threads, também tem múltiplas pilhas
  - Se o kernel não conhece essas threads, não consegue saber quando aumentar

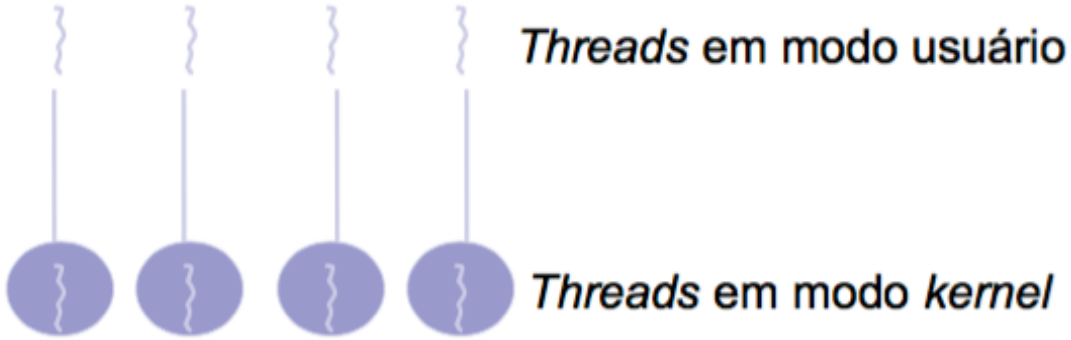
Modelos multithreading

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



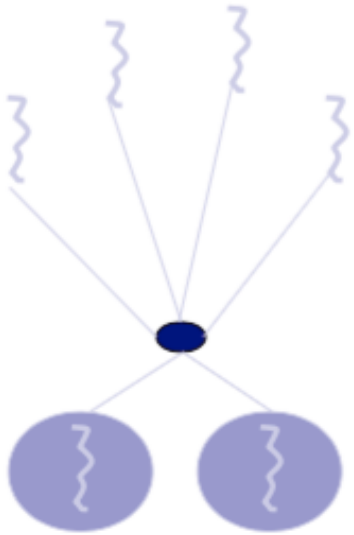
Modelos multithreading

- 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;
  - “Pool” de threads



Threads em modo usuário

Thread em modo *kernel*

Para lembrar...

- **Processos** são usados para **agrupar recursos**.
- **Threads** são as entidades escalonadas para **execução na CPU.**