Sistemas Operacionais

THREAD

Sistemas Operacionais

THREAD

- Os sistemas operacionais até o final dos anos 70 suportavam apenas um programa associado a um processo, este era o conceito monothread.

- Em 1980,com o desenvolvimento do sistema operacional Mach, foi introduzido o conceito de separação entre processo e thread, surgindo o conceito multi thread.

- A partir do conceito multi thread é possível implementar programas do usuário e serviços do sistema operacional que são executados de forma concorrente.

Thread

Ambiente Monothread

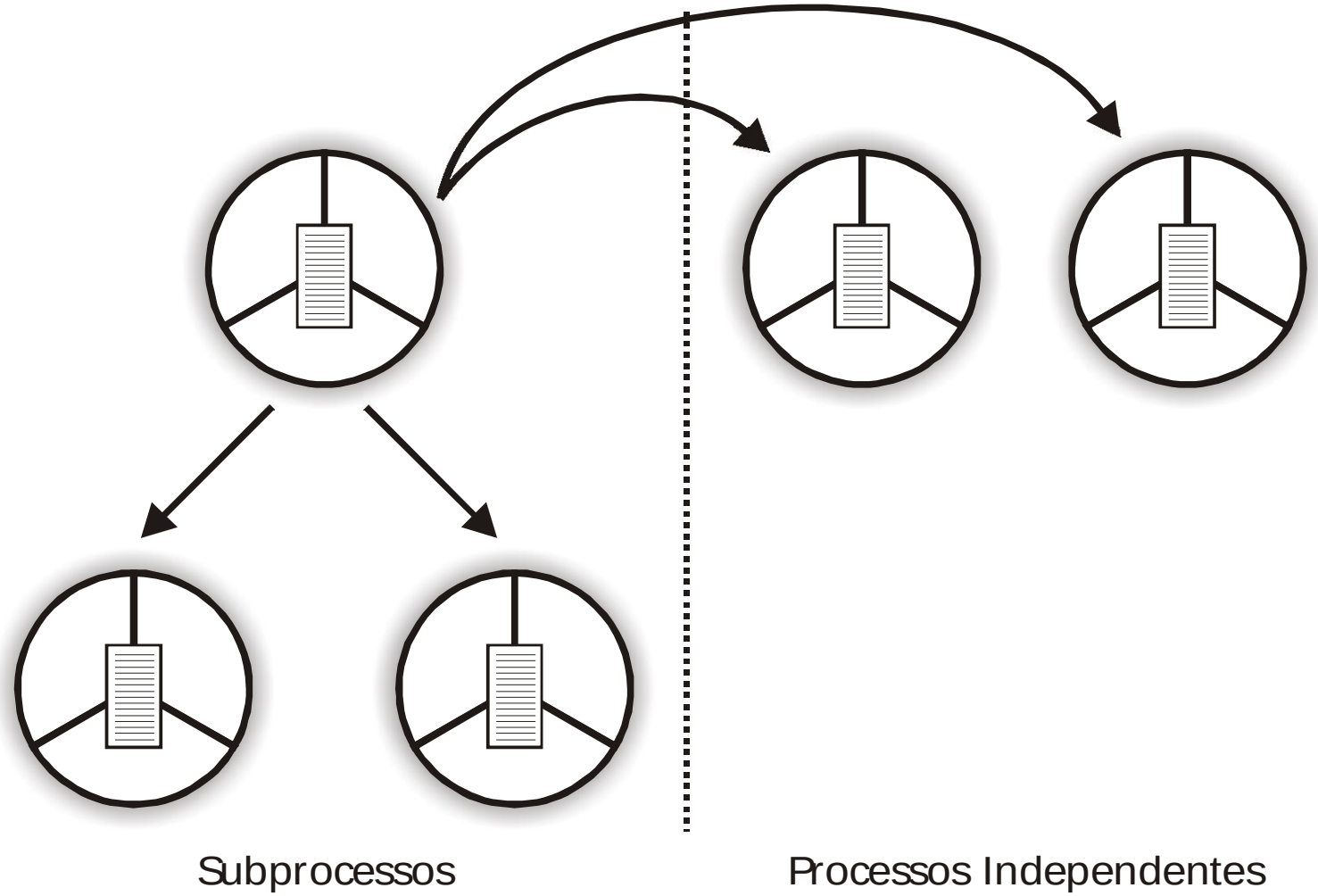
- Um programa é sequência de instruções chamadas por procedimentos e funções. Em um ambiente monothread um processo suporta apenas um único programa em seu espaço de endereçamento. Neste ambiente os sistemas concorrentes são implementados com o uso de múltiplos processos independentes ou subprocessos.

Problemas:

- Quanto mais processos em execução pelo sistema operacional mais recursos são consumidos
- O espaço de endereçamento não compartilhado.
  - Comunicação entre processos mais difícil e lenta
  - Compartilhamento de recursos é mais complicado (ex: arquivos abertos)

Processos

Ambiente Monothread

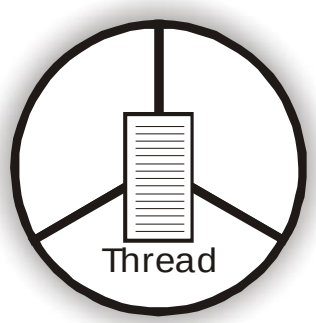


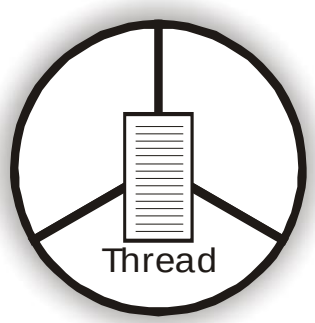
Thread

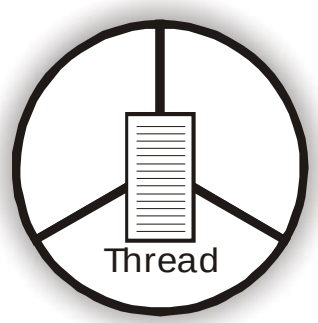
Ambiente Monothread

Processos monothread possuem, cada um, seu próprio contexto de hardware, de software e espaço de endereçamento.

- Exemplos de SO's monothread:
  - MS-DOS, primeiras versões do MS-Windows
  - Primeiros sistemas UNIX



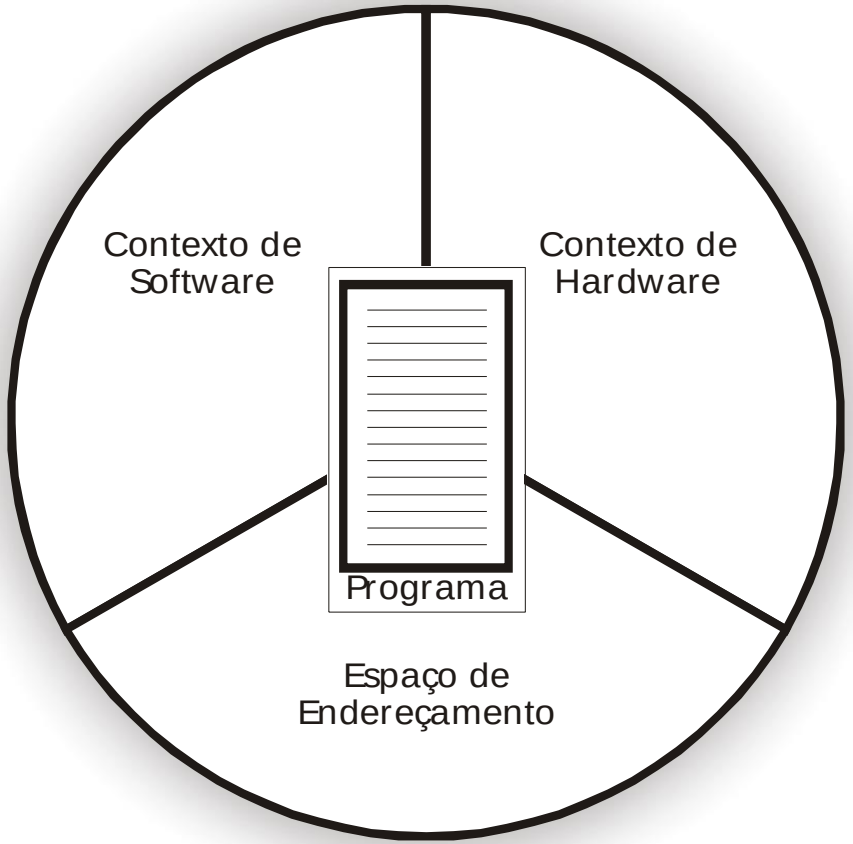




Thread

Recordando:

Estrutura de um processo



Thread

Ambiente Multithread

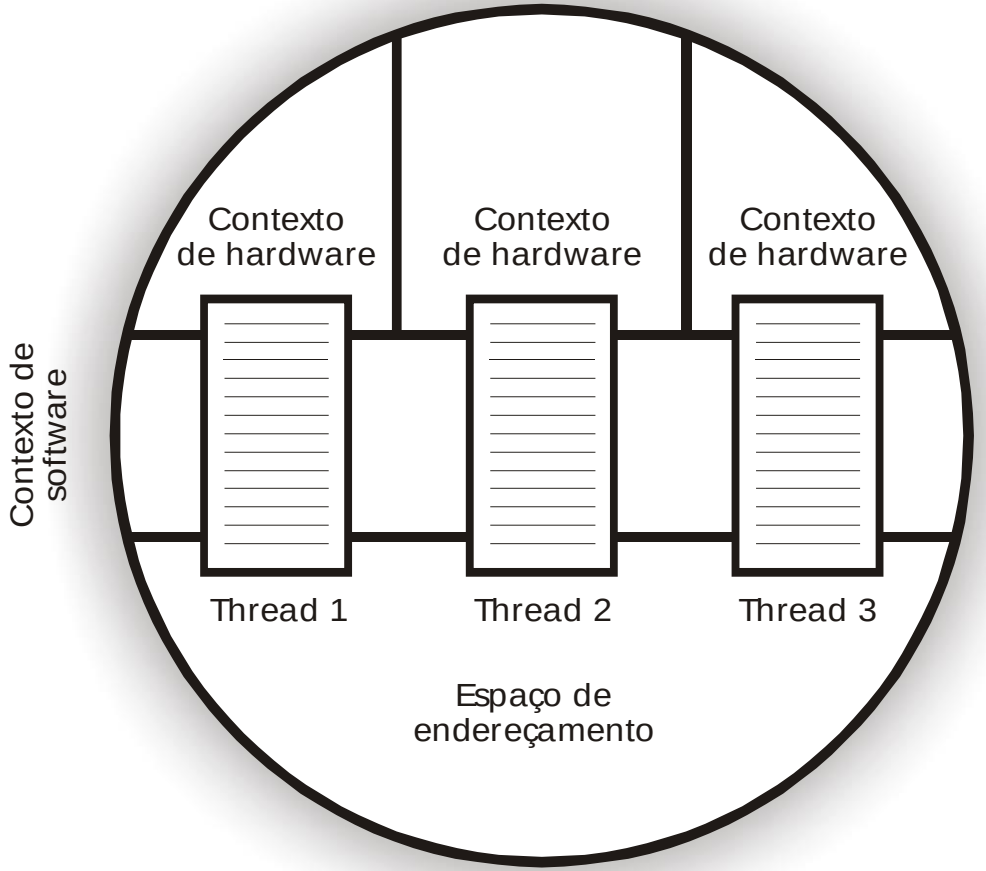
- Em ambientes com multiplos threads a idéia de programas associados a processos. Os programas são associados a threads.
  - Cada programa tem pelo menos uma thread de execução.
  - Podem compartilhar espaço de endereçamento com outras threads criadas pelo programa.

- Uma thread pode ser entendida com uma sub-rotina de um programa que é executada de forma assíncrona (paralela), executada de forma concorrente ao programa principal.
  - Threads criadas dinamicamente, sob demanda.

- A vantagem no uso de threads é minimizar a alocação de recursos com a criação de novos processos.

Thread

Ambiente Multithread



Thread

Aplicação Multithread

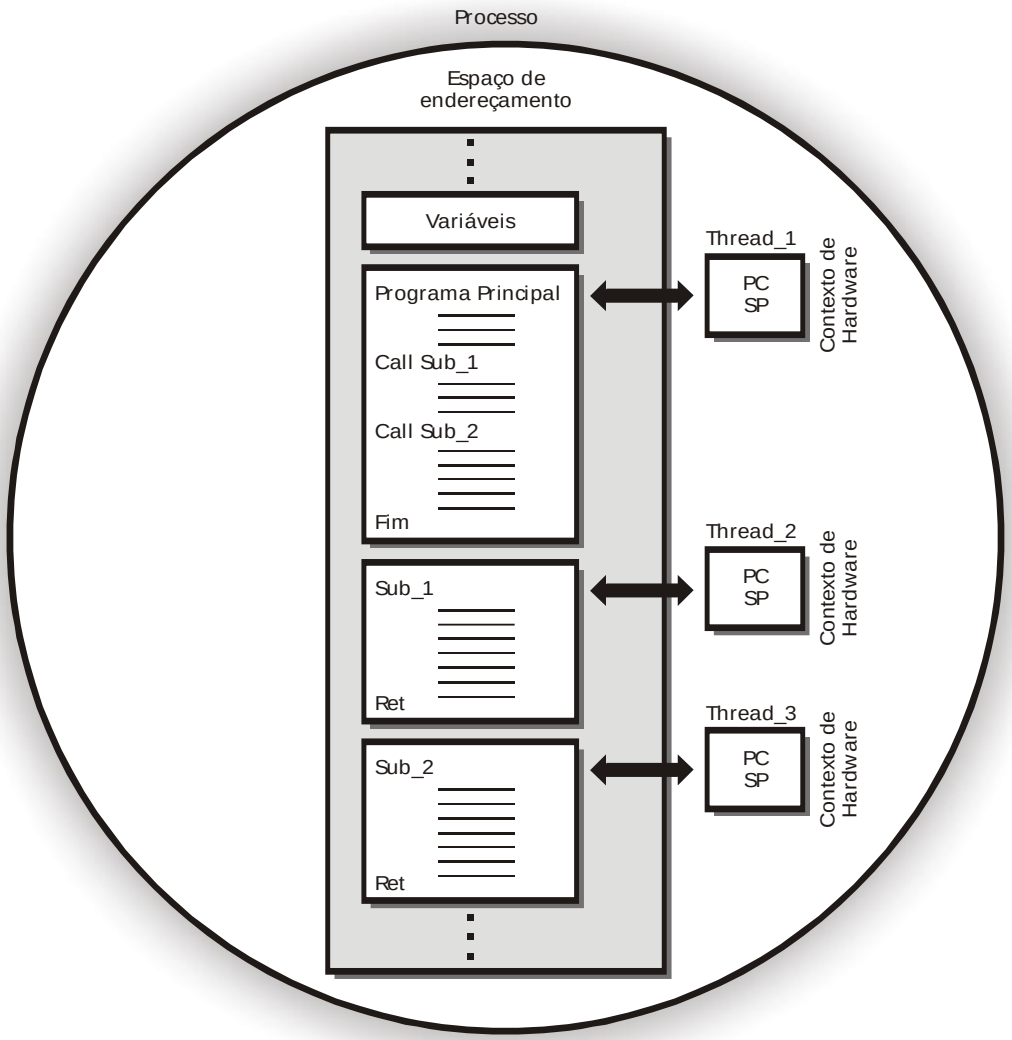
- Threads compartilham o processador da mesma forma que processos. Também passam por mudanças de estados (execução, espera e pronto).
  - Por exemplo, quando uma thread espera uma operação de leitura de arquivo uma outra thread pode ser executada. Para permitir a troca de contexto de thread cada thread possui seu contexto hardware com o conteúdo dos registradores gerais, PC e SP.

- Dentro de um processo as threads compartilham o espaço de endereçamento e o contexto de software porém cada thread possui seu contexto de hardware individual.

- Threads são implementadas internamente na memória principal através de uma estrutura de dados chamada bloco de controle de thread (Thread control block - TCB). O TCB armazena mais algumas informações relacionadas ao thread como prioridade, estado de execução e bits de estado.

Thread

Aplicação Multithread



Thread

Aplicação Multithread

- Unidade de alocação de recursos é o processo
  - Threads criadas compartilham recursos do processo

- Unidade de escalonamento é a thread.
  - SO não escalona o processo para execução, mas sim uma de suas threads

- Compartilhamento de espaço de endereçamento inibe mecanismos de proteção no acesso a este espaço (aplicação deve cuidar disto)

Thread

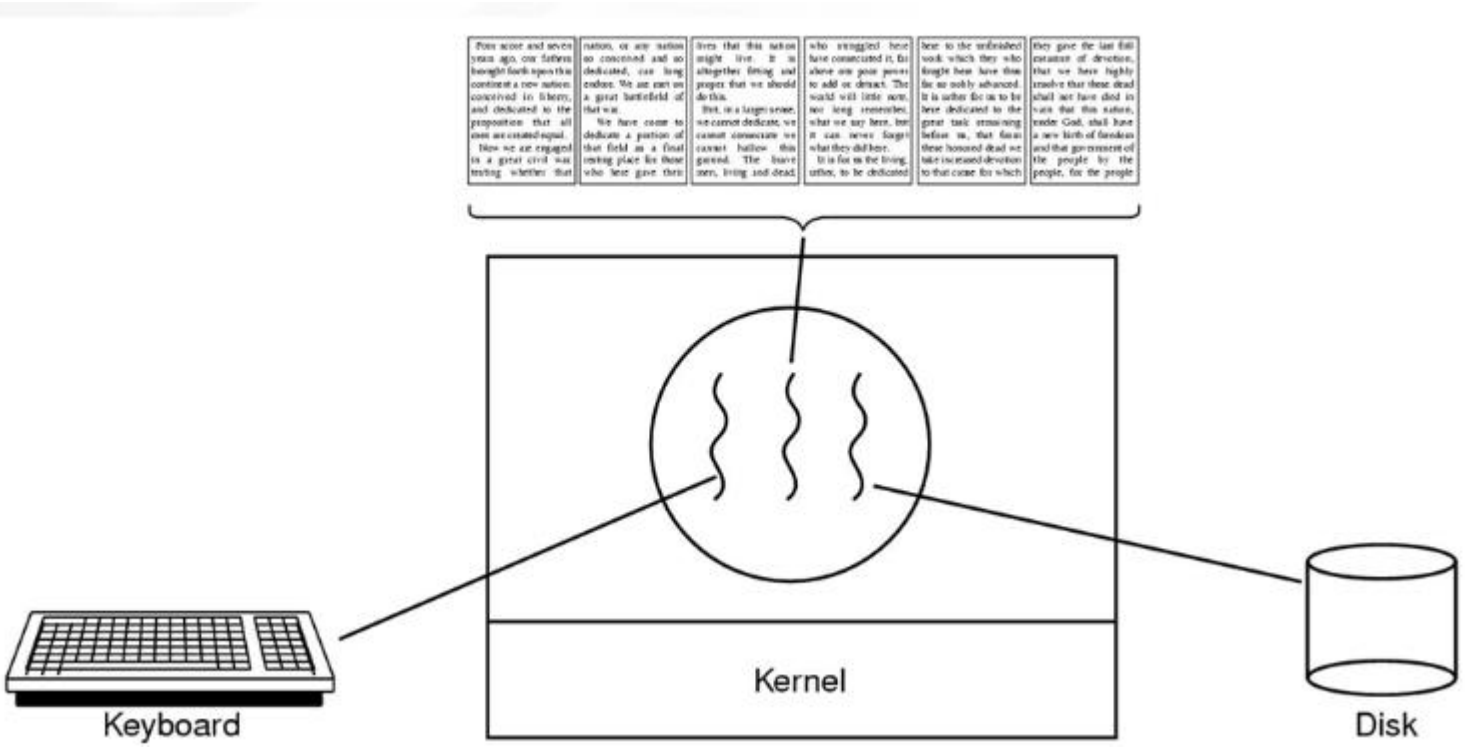
Aplicação Multithread

- Threads de um mesmo processo pode facilmente compartilhar recursos como descritores de arquivos, sinais, temporizadores, etc
- Uso de periféricos pode ser realizado de forma concorrente entre as threads
- Melhora desempenho de algumas aplicações onde tarefas podem ser executadas em background durantes operações de E/S
    - Exs: editores de texto, planilhas, aplicações gráficas, processamento de imagens, jogos, web services, etc

Thread

Exemplos de programas:

- Editor de Texto: Permite que o usuário edite o arquivo enquanto ele ainda está sendo carregado do disco. Processamento assíncrono (salvamento periódico)



Thread

Exemplos de programas:

- Navegador (browser):
  - Consegue fazer o download de vários arquivos ao mesmo tempo, gerenciando as diferentes velocidades de cada servidor e, ainda assim, permitindo que o usuário continue interagindo, mudando de página enquanto os arquivos estão sendo carregados.

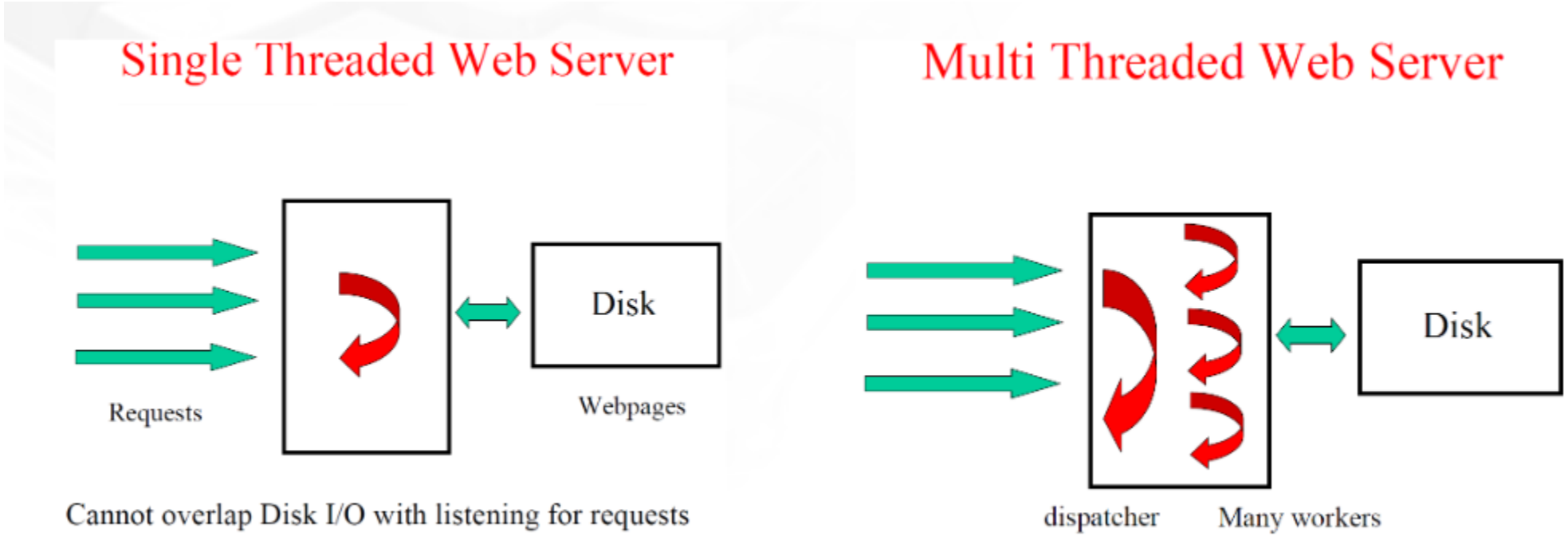
- Programas numéricos (ex: multiplicação de matrizes):
  - cada elemento da matriz produto pode ser calculado independentemente dos outros; portanto, podem ser facilmente calculados por threads diferentes.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix}=\begin{pmatrix} a.e+b.g & a.f+b.h \\ c.e+d.g & c.f+d.h \end{pmatrix}$$

Thread

Exemplos de programas:

- Servidor Web:



Thread

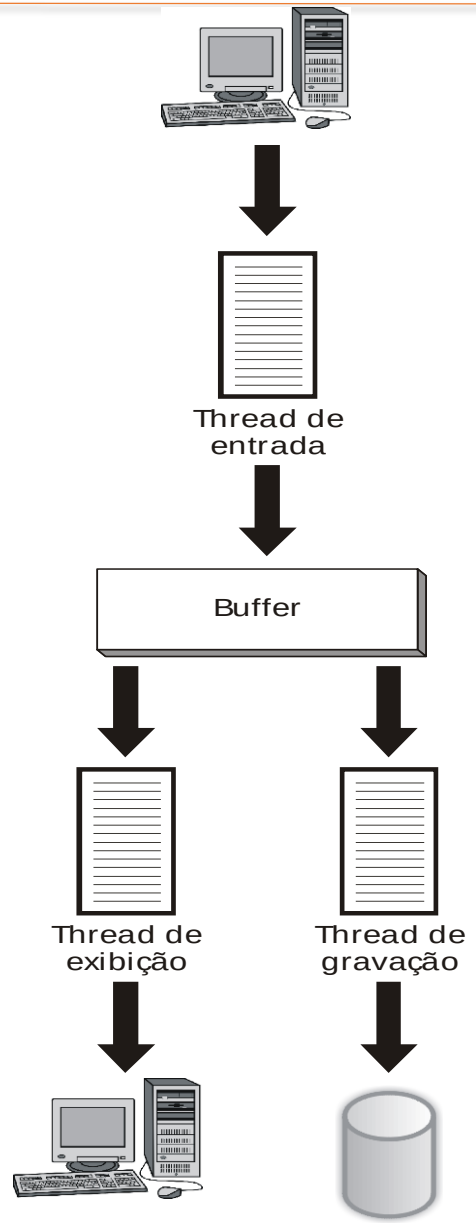
Aplicação Multithread

- Threads de um mesmo processo pode facilmente compartilhar recursos como descritores de arquivos, sinais, temporizadores, etc
- Uso de periféricos pode ser realizado de forma concorrente entre as threads
- Melhora desempenho de algumas aplicações onde tarefas podem ser executadas em background durantes operações de E/S
    - Exs: editores de texto, planilhas, aplicações gráficas, processamento de imagens, jogos, web services, etc

Thread

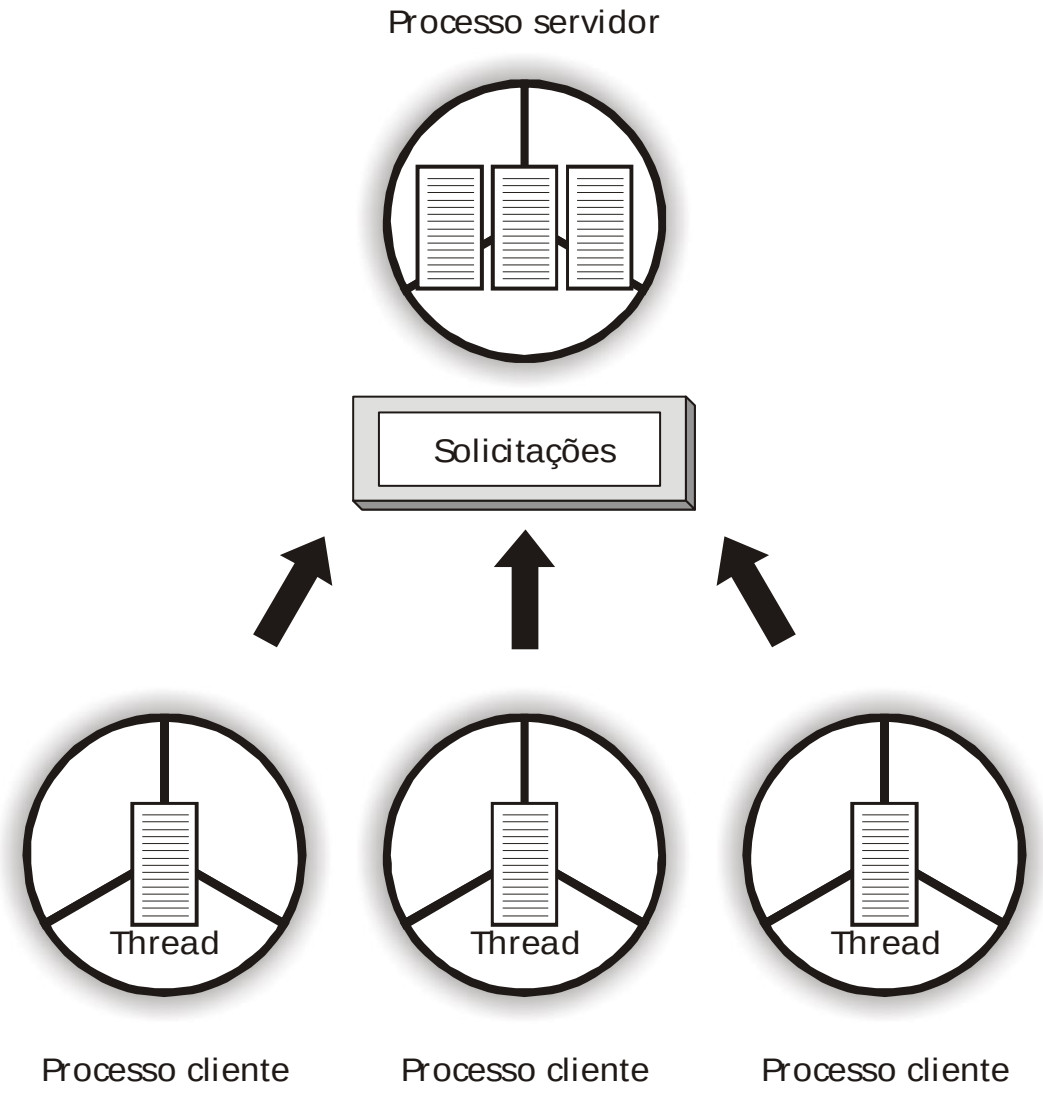
Aplicação Multithread

- Thread principal solicita operações de E/S
- Threads específicas p/ executar as operações de E/S



Thread

Aplicação Multithread



Thread

Aplicação Multithread

Essenciais para arquiteturas cliente-servidor

- Thread no cliente p/ solicitar e aguardar o serviço enquanto thread principal continua executando em background
    - Evita que processo cliente pare aguardando serviço pedido
- Processo servidor dispara threads para atender cada solicitação que chega de maneira simultânea
    - Evita que uma solicitação precise aguardar o término do atendimento das solicitações anteriores

Thread

Aplicação Monothread e Multithread

- A grande diferença entre aplicações monothread e aplicações multithread está no uso do espaço de endereçamento.
- Processos são independentes e portanto cada processo possui seu próprio espaço de endereçamento, enquanto que as threads compartilham o mesmo espaço de endereçamento de um processo.
- Este compartilhamento de memória permite que a troca de dados entre threads de um mesmo processo seja simples e rápida se comparado a comunicação de processos em um ambiente monothread.
- Como threads de um mesmo processo compartilham o mesmo espaço de endereçamento é necessário garantir que uma thread não altere dados de outra thread. O programador deve usar técnicas de sincronização e comunicação entre threads para garantir o acesso seguro aos dados compartilhados no mesmo espaço de endereçamento.

Thread

Arquitetura e Implementação

- Sistemas Operacionais disponibilizam pacotes de threads para serem usados pelas aplicações
- Abordagem usada no pacote influenciará o desempenho, a concorrência e a modularidade das aplicações multithread
- Podem ser oferecidas de quatro formas:
  - Biblioteca de rotinas em modo usuário (fora do núcleo do SO)
  - Rotinas em modo kernel (do núcleo do SO)
  - Modo híbrido (modos kernel + usuário)
  - Modelo de Scheduler Activations

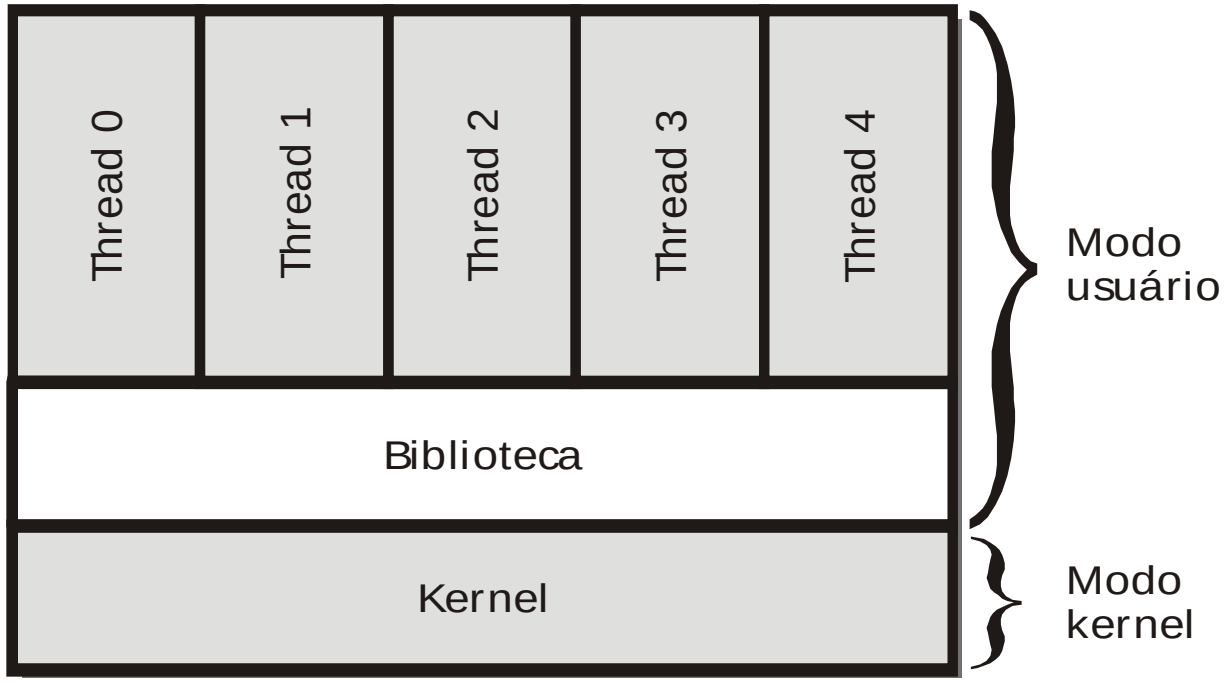
Thread

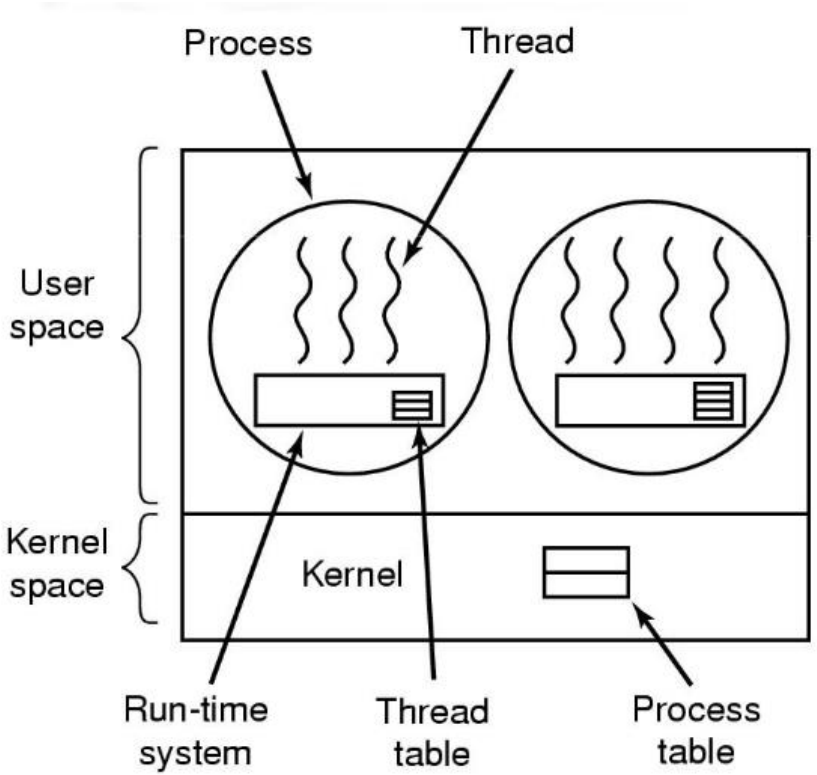
Arquitetura e Implementação

- **Threads em modo usuário:**
  - Threads podem ser criadas através de uma biblioteca de funções fora do núcleo do sistema operacional (modo usuário). Neste caso as threads são implementadas pela aplicação do usuário e não pelo sistema operacional. É a aplicação que deve gerenciar as threads (sincronização e comunicação).
  - SO não gerencia nem sincroniza as múltiplas thread, é responsabilidade da aplicação
  - Vantagem é poder implementar aplicações multithreads em SO's que não suportam threads
  - TMU's são mais rápidas por dispensarem acessos ao kernel do SO, porém mais limitadas

Thread

- **Threads em modo usuário:**
  - Caso exista múltiplos processadores, TMU's não poderão rodar nos diferentes processadores
  - Como o SO só enxerga o processo, todas as TMU's rodarão no mesmo processador
  - Limitação extrema para o paralelismo da aplicação

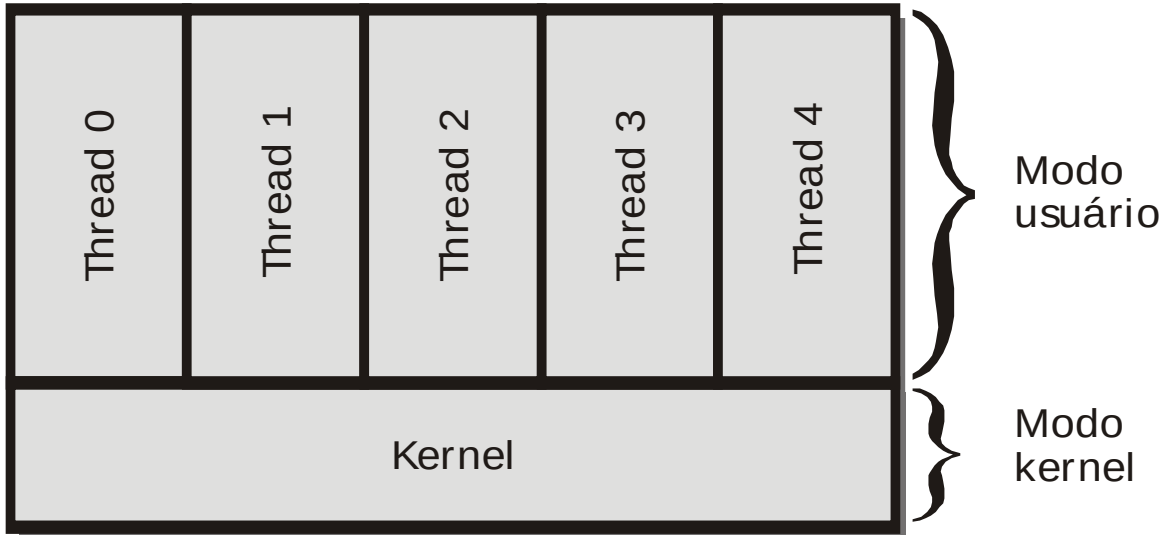




Thread

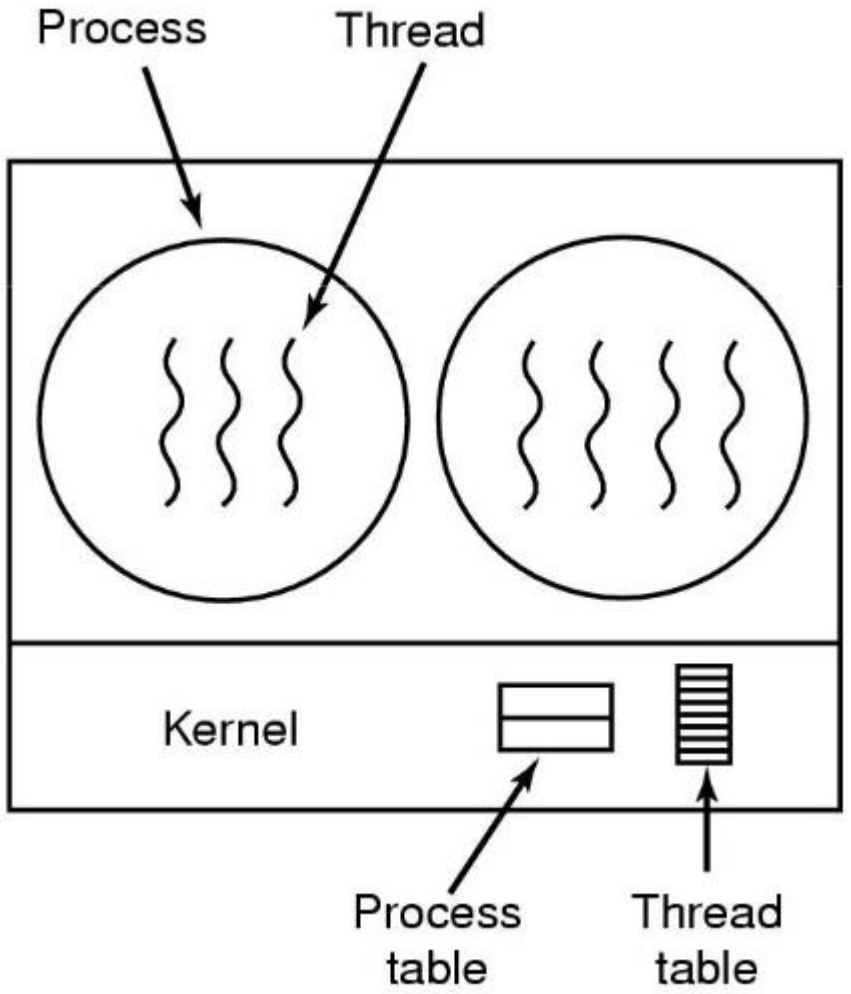
Threads em modo kernel

- Threads podem ser criadas pelo próprio núcleo do sistema operacional (modo kernel) através de chamadas à rotinas do sistema (system calls) que oferecem todas as funções para gerencia de threads (sincronização e comunicação).
  - SO escalona as threads individualmente
  - Utiliza capacidade de múltiplos processadores
  - Baixo desempenho devido às mudanças de modo de acesso (contexto) usuário-kernel-usuário



Thread

Threads em modo kernel



Thread

Threads em modo híbrido

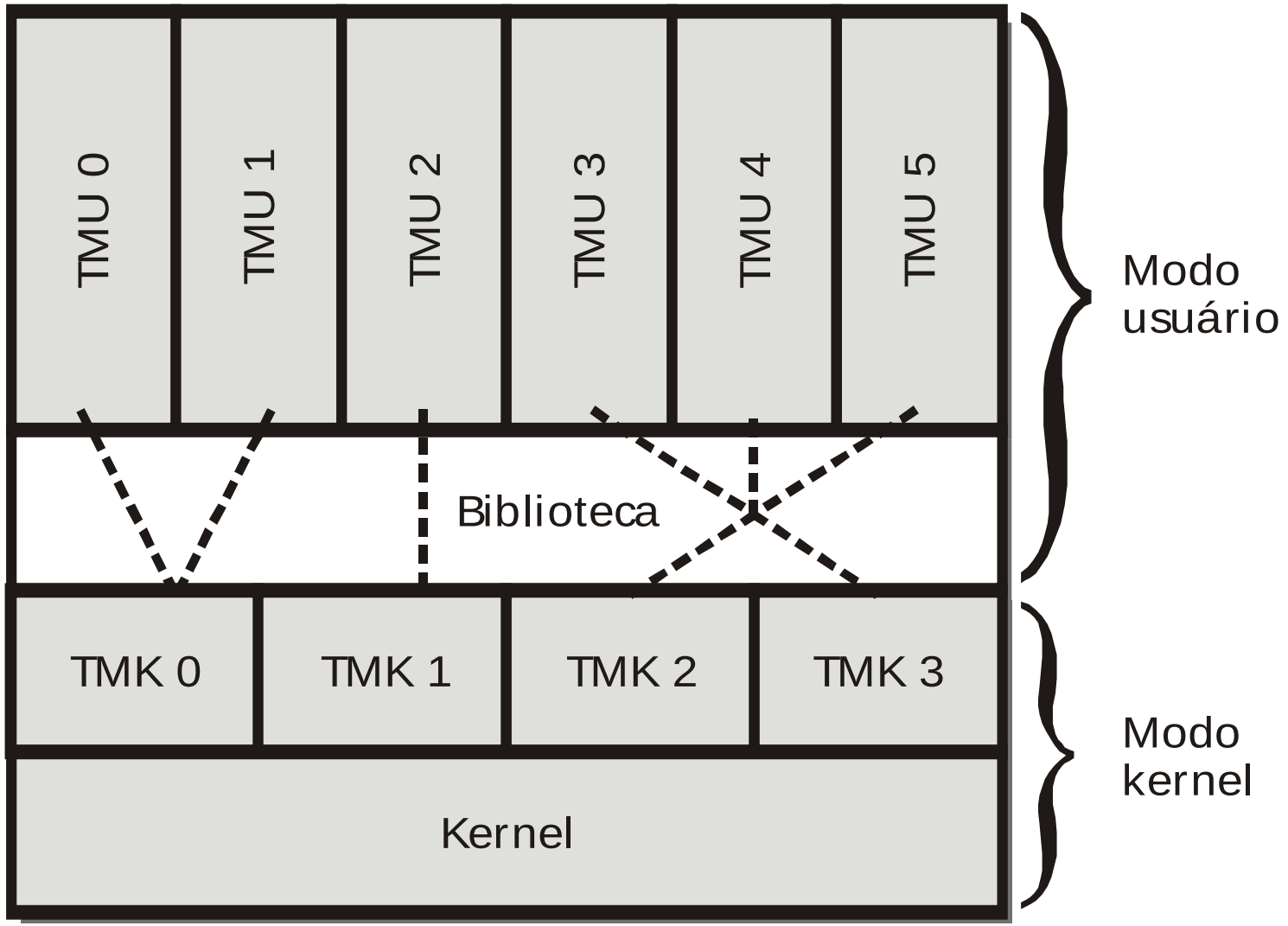
- Threads podem ser criadas por uma combinação de ambos (modo híbrido).
  - um processo pode ter várias threads em modo kernel e cada thread em modo kernel pode ter várias threads em modo usuário.
  - O núcleo do sistema operacional reconhece as threads em modo kernel e pode escalona-las individualmente.

Apesar da flexibilidade, apresenta desvantagens

- Quando Thread modo kernel (TMK) faz uma operação de bloqueio, todas as Threads modo usuário (TMU's) associadas à TMK ficam bloqueadas
- TMU's que precisem rodar em diferentes processadores precisam estar em TMK's distintas

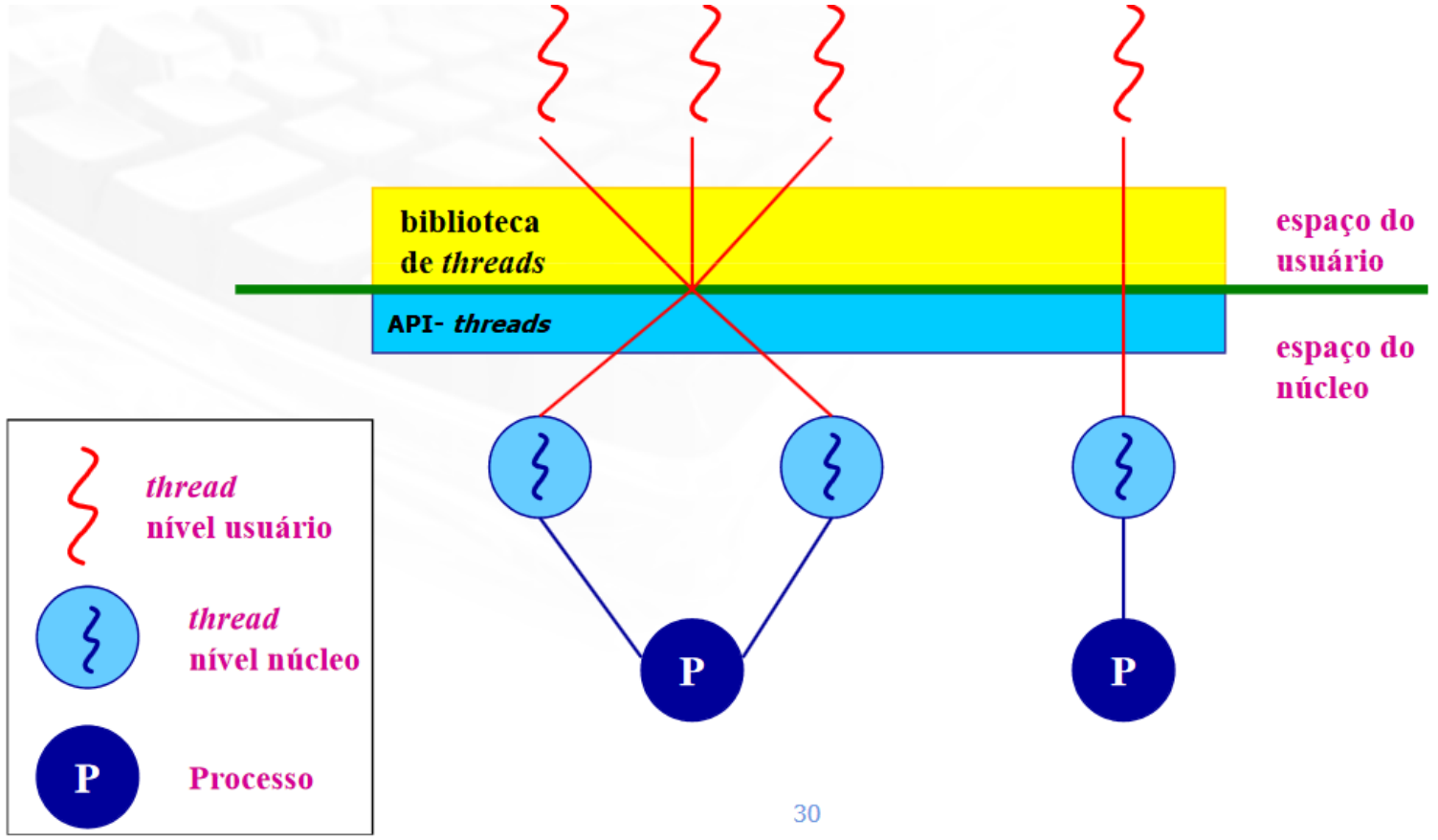
Thread

Threads em modo híbrido



Thread

Threads em modo híbrido



Thread

Threads em Scheduler Activations

- Threads em modo Scheduler Activations usam o melhor do modo kernel e do modo usuário. Usam as facilidades do modo kernel com o desempenho e a flexibilidade do modo usuário.
- O núcleo do sistema troca informações com a biblioteca de threads utilizando uma estrutura chamada scheduler activations.
- Esta implementação evita a troca de modo de acesso (usuario-kernel-usuario) pois o kernel e a biblioteca de threads se comunicam e trabalham de forma cooperativa.

Thread

Threads em Scheduler Activations

- Caso uma thread faça uma chamada bloqueante, biblioteca em modo usuário escalona outra thread em cooperação com modo kernel
- Cada camada implementa seu escalonamento de forma independente, trocando informações quando necessário