Universidade Estadual de Mato Grosso dos Sul
Curso de Ciência da Computação
Disciplina de Algoritmos Paralelos e Distribuídos

Aula 3 – Processos: Threads e Virtualização

Processos

- Nos atuais sistemas operacionais, a execução de um programa é feita por processadores virtuais;

- Para monitorar os processadores virtuais o S.O. tem uma tabela de processos que contém entradas para armazenar valores de registradores de CPU, mapas de memória, arquivos abertos, entre outros.

Processos

- Processos são basicamente programas em execução.
- O S.O. é o responsável por assegurar que processos independentes não afetem (modo intencional, malicioso ou acidental) a correção do comportamento dos outros processos sendo executados.
- Transparência no compartilhamento da mesma CPU e outros recursos de hardware.

Processos

- Transparência implica em custo:
  - Criação de espaço de endereços completamente independente;
  - Chavear a CPU entre dois processos;
  - Salvar o contexto da CPU;
  - Troca de informações entre disco e memória principal.
- Em sistemas tradicionais, cada processo possui o seu próprio espaço de endereçamento e um único fluxo de execução.
- Em alguns casos é desejável haver diversos fluxos de execução compartilhando um único espaço de endereçamento, ou seja, a mesma região de memória.

O que significa um único fluxo de execução?

- Um servidor de arquivos deve esperar por requisições feitas ao disco.
- O fluxo de execução que fez a requisição é bloqueado aguardando a resposta.

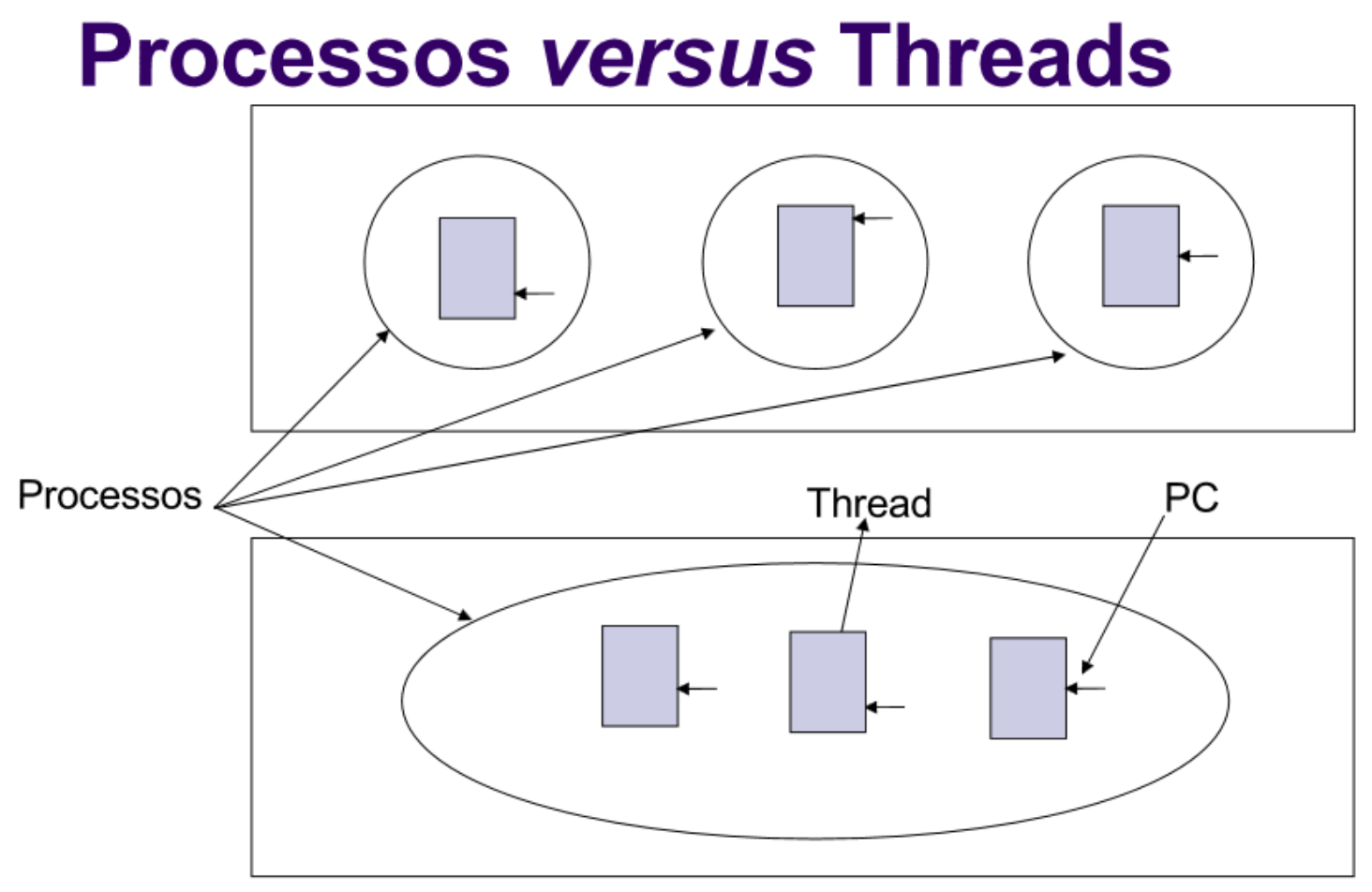
- OCORRE UMA PERDA DE DESEMPENHO!

Solução: vários fluxos de execução.

- Se o servidor de arquivos é implementado usando diferentes fluxos de execução, outras requisições de clientes podem ser processadas, enquanto o primeiro fluxo aguarda a resposta do disco.

- Consequência: Melhor vazão – Throughput – e ganho de desempenho.

Processos *Versus* Threads



Threads

- Cada um dos fluxos de execução de um processo é chamado de *Thread*.
- *Threads* podem ser vistos como mini-processos.
- Cada *thread* executa sua própria porção de código.
- *Threads* compartilham a CPU do mesmo modo que diferentes processos (*timesharing*)

Threads em Sistemas não-Distribuídos

- *Threads* que fazem parte de um mesmo processo não são independentes como o caso de diferentes processos.
- Todas as *Threads* em um mesmo processo possuem a mesma região de memória, compartilhando as mesmas variáveis globais.
- Uma determinada *thread* pode ler, escrever ou mudar a pilha de dados de uma outra *thread*.
- A proteção deve ser feita pela aplicação que está chamando a *thread*.

Threads em Sistemas não-Distribuídos

- As *threads* podem estar em diferentes estados: executando, bloqueado, pronto ou finalizado.

Itens Por Thread
PC
Pilha
Registradores
Threads Filhas
Estado

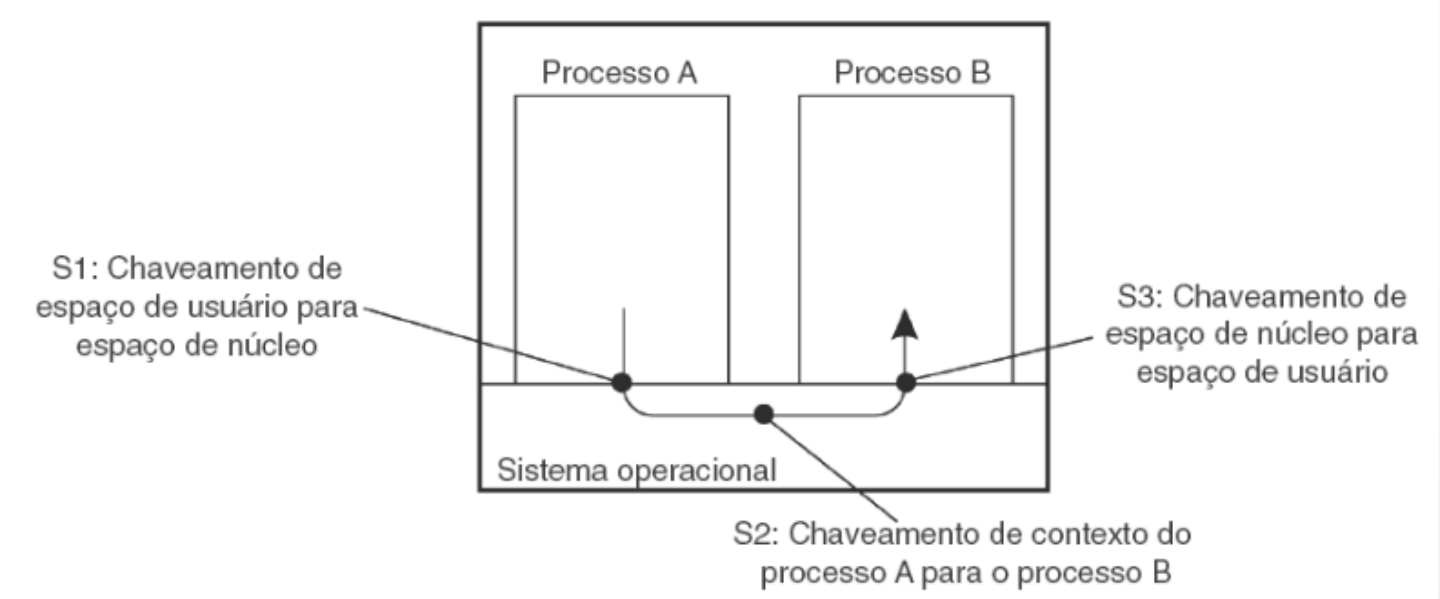
Itens Por Processo
Espaço de endereçamento Variáveis Globais Arquivos abertos Processos filhos Sinais Timers Informações da Conta Semáforos

Threads em Sistemas não-Distribuídos

- Vantagens:
  - Explorar paralelismo ao executar um programa em um sistema multiprocessado.
    - Exemplo: Cada *Thread* é designada a uma CPU, enquanto dados compartilhados são armazenados em memória compartilhada.
  - Grandes aplicações desenvolvidas como um conjunto de programas cooperativos.
    - Evita o chaveamento entre diferentes processos - isso devido à comunicação através de *Interprocess Comunication* (IPC).

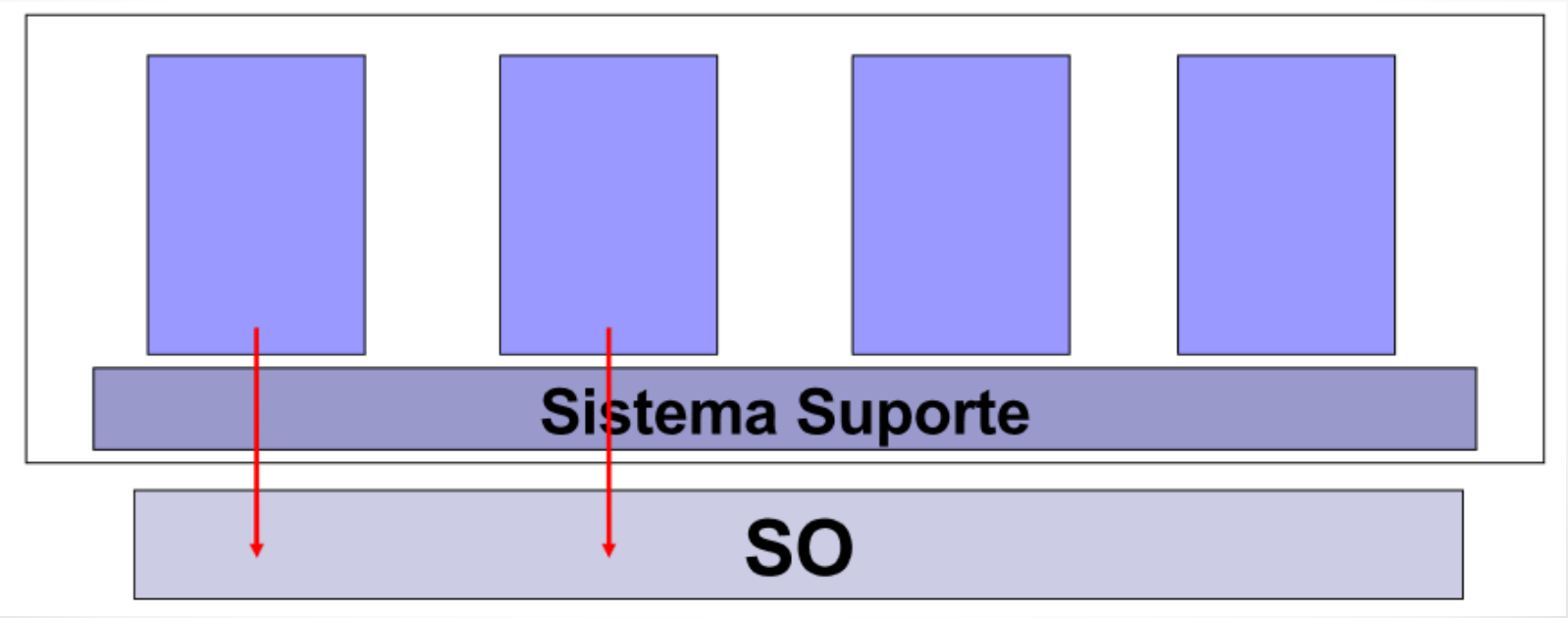
Threads em Sistemas não-Distribuídos

- A comunicação pode ser feita com a utilização de dados compartilhados.
- O chaveamento é o espaço do usuário.
- Exemplo: chaveamento de contexto como resultado de IPC.



Implementação de *Threads* em Sistemas não-Distribuídos

- Implementação no Nível Usuário
  - *Threads* rodam sobre o *runtime system* – coleção de procedimentos que gerenciam as *threads*.
  - Quando uma *thread* executa uma chamada de sistema, entra em execução um semáforo ou um mutex. Enquanto isso a *runtime system* verifica se a thread deve ser suspensa.

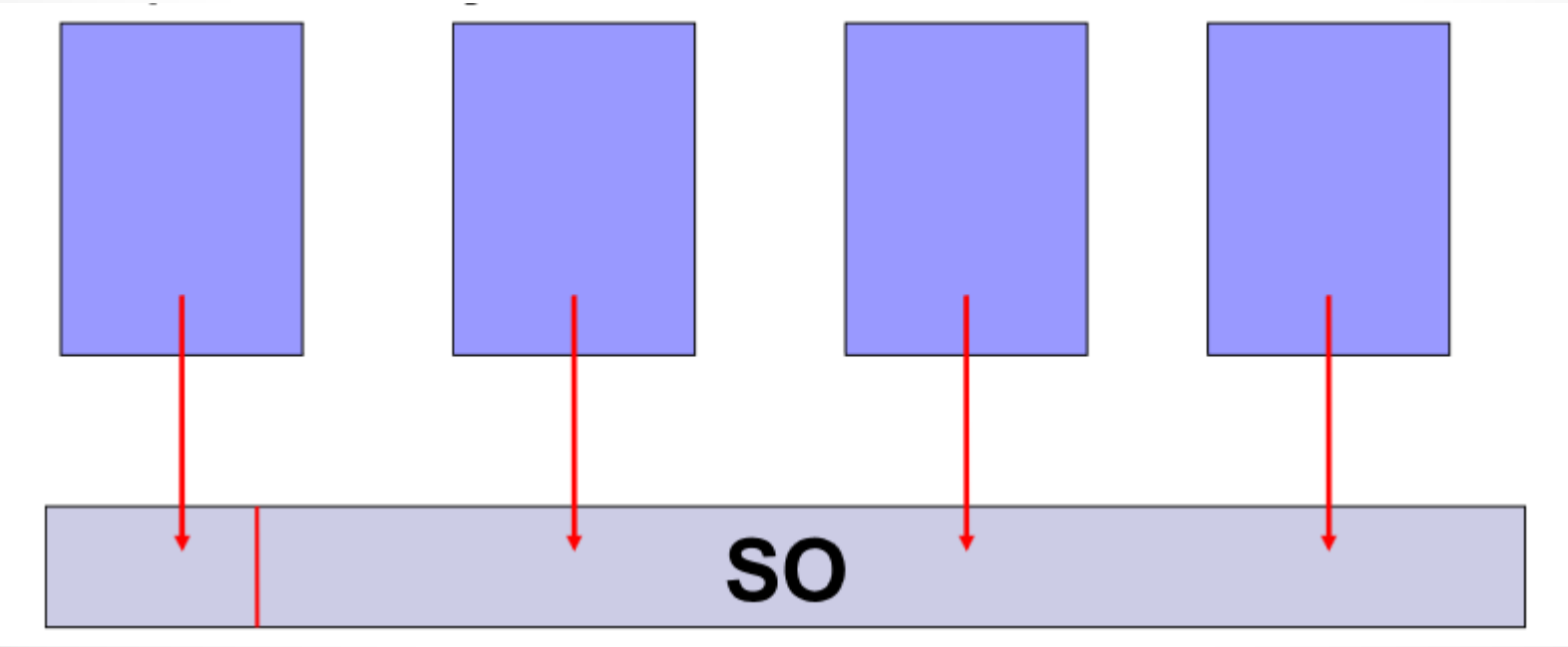


Implementação de *Threads* em Sistemas não-distribuídos

- Nível Usuário
  - Há o custo da criação e da alocação de memória para a pilha.
  - Chaveamento de contexto de *thread* pode ser feito em apenas algumas instruções:
    - Não há a necessidade de mudar mapas de memória, descarregar o TBL (*Translation Lookaside Buffer*).
    - Exemplo: *Threads* que precisam entrar em sincronia.
- Desvantagem de *Threads* em Nível Usuário:
  - O problema está em como chamadas de sistemas bloqueantes são implementadas.
  - Exemplo: ler um *pipe* vazio → chamada de sistema → bloqueio.
    - Todos os *Threads* são bloqueados.

Implementação de *Threads* em Sistemas não-Distribuídos

- Implementação no Kernel



Implementação de *Threads* em Sistemas não-Distribuídos

- Implementação no kernel:
  - Criação, encerramento, sincronização deverão ser feitos pelo kernel.
  - Chamadas de sistema deverão ser realizadas.
  - Chavear contextos de *threads* é tão caro quanto chavear contexto de processos.

Implementação de *Threads* em Sistemas Distribuídos

- Obs.: As *threads* podem proporcionar um meio conveniente de permitir chamadas bloqueantes de sistema sem bloquear o processo inteiro.

- Obs2.: As *threads* possuem um papel importante em Sistemas Distribuídos →facilitam a comunicação no intuito de manter múltiplas conexões lógicas ao mesmo tempo.

Clientes Multithreads

- Sistemas Distribuídos que operam em redes de longa distância → escondem longos tempos de propagação de mensagens entre processos.

- A maneira de ocultar latências de comunicação é iniciar a comunicação e imediatamente prosseguir com outra atividade.

Clientes Multithreads – Navegadores Web

- Um documento Web consiste em: texto, imagens, ícones, etc.
- A cada elemento, o navegador estabelece uma conexão TCP/IP, para ler os dados e passar ao monitor do usuário.
- Operações bloqueantes: estabelecimento da conexão, leitura de dados.
- Os navegadores começam a exibir os dados à medida que as informações vão chegando.
  - Vantagem: o usuário não precisa esperar até que todos os componentes sejam buscados.

Clientes Multithreads – Navegadores Web

- São usados *threads* separadamente para buscar diferentes partes de uma mensagem.
- Caso o servidor esteja com sobrecarga, a disposição de um cliente multithread permitirá estabelecer conexões com diferentes servidores, tendo com isto a transmissão de dados em paralelo.
- O cliente pode manipular fluxos de dados de entrada em paralelo → *Threads*.

Servidores Multithreads

- Acontecimento comum:
  - Um servidor de arquivos normalmente espera pela entrada de uma requisição para uma operação de arquivo e, na sequência, executa a requisição e então devolve a resposta.

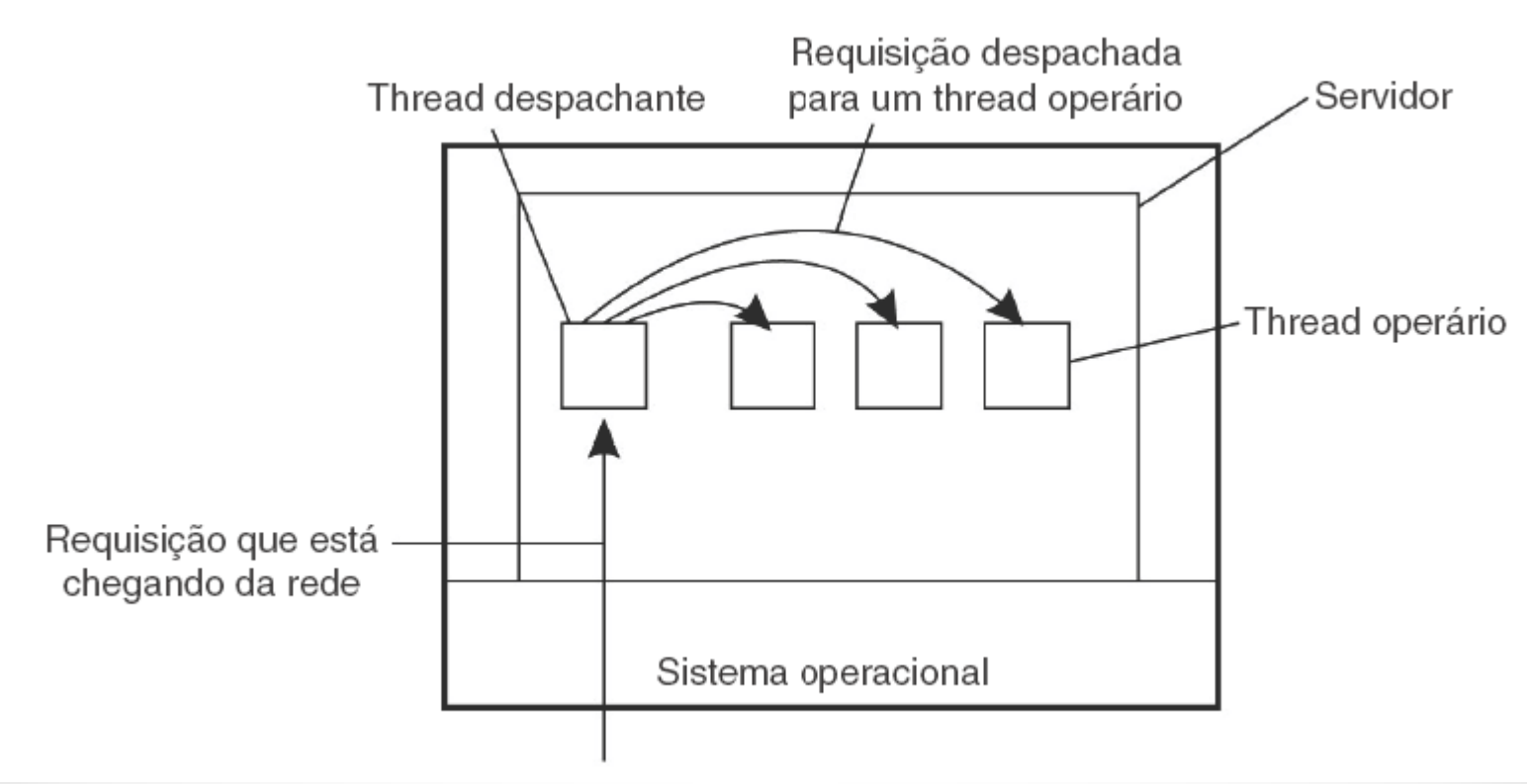
Pergunta: Como aumentar o desempenho??

Servidores Multithreads

- Funcionamento:
  - Requisições são enviadas por clientes para uma porta no servidor.
  - Thread despachante lê as requisições que entram para uma operação de arquivo.
  - O servidor escolhe um thread operário.
  - Se o thread escolhido estiver suspenso, outro *thread* é selecionado para ser executado.
    - Exemplo: o thread despachante pode ser selecionado para adquirir mais trabalho.

Servidores Multithreads

- Servidor multithread organizado de acordo com o modelo despachante/operário.



Servidores Multithreads

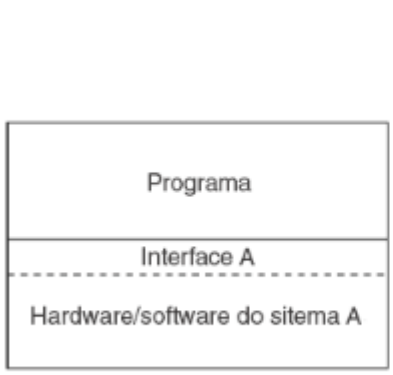
- Se o servidor é inteiramente CPU Bound, não existe a necessidade de diversos threads.
  - CPU Bound → é quando o tempo de processamento depende mais do processador do que das entradas e saídas, fazendo assim com que atrapalhe o tempo total de processamento.
  - Neste caso, você tem um aumento de complexidade sem ganho de desempenho.

Virtualização

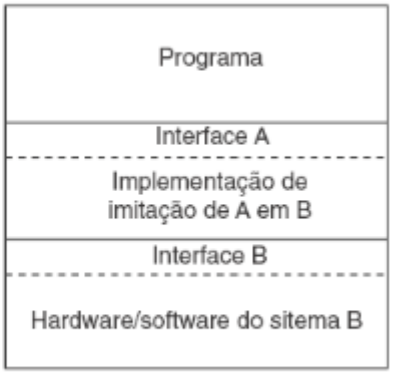
- *Threads* e Processos podem ser vistos como um modo de se fazer diversas tarefas ao mesmo tempo.
- Em computadores monoprocessados, a execução simultânea é uma ilusão → única CPU, somente uma instrução de um único *thread* ou processo será executado por vez.
- Virtualização de Recurso: "Fingir" que um determinado recurso está replicado no sistema.

Virtualização

- Estende ou substitui uma interface existente de modo a imitar o comportamento de um outro sistema.
- Exemplo:
  - A) Organização geral entre programa, interface e sistema;
  - B) Organização geral da virtualização do sistema A sobre o sistema B.



(a)



(b)

Virtualização

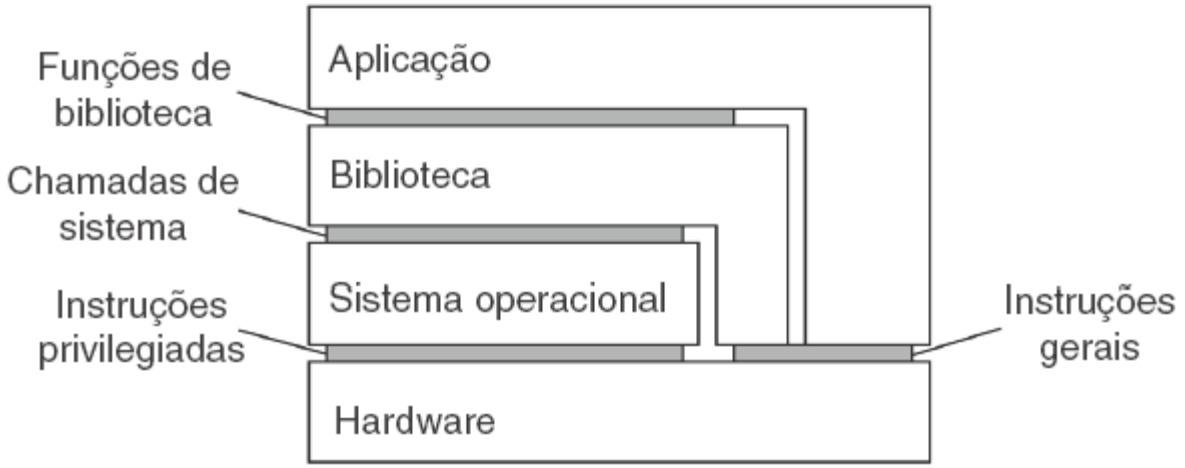
- Na década de 70 a virtualização foi aplicada com sucesso nos *mainframes* da IBM, que ofereciam uma máquina virtual para executar softwares que incluíam aplicações e também os sistemas operacionais.

- Software em nível mais alto são mais estáveis que o hardware e sistemas de software de baixo nível → virtualização pode ajudar transportando as interfaces de software para as novas plataformas

- Novas plataformas são capazes de executar software mais antigos.

Arquiteturas de Máquinas Virtuais

- Existem vários modos pelos quais a virtualização pode ser realizada na prática.
- Existem quatro tipos diferentes de interfaces em quatro níveis diferentes.



Virtualização

- A essência da virtualização é imitar o comportamento das interfaces (instruções de máquina, chamadas de sistema).

Máquina virtual de processo

Aplicações desenvolvidas para um SO são executadas em outro SO

Virtualização feita somente para um único processo

emuladores → imitar chamadas de sistema → Não é trivial

Virtualização

- Monitor de máquina virtual

Fornece o conjunto de instruções completo do hardware

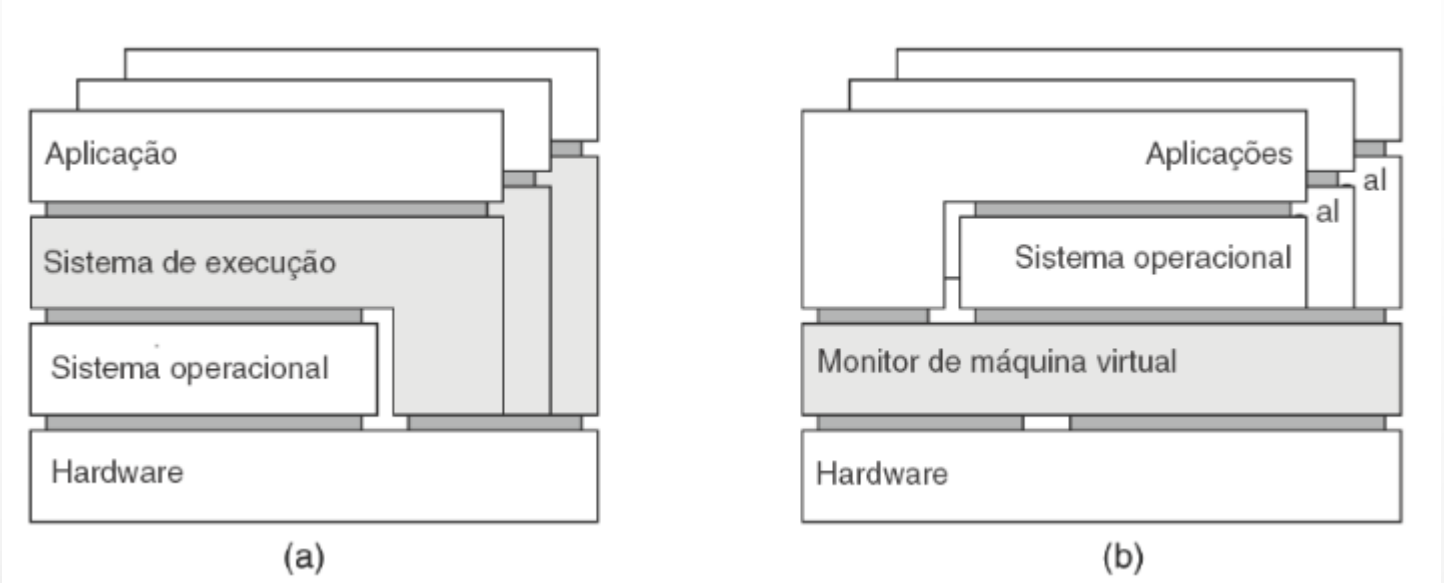
Vários sistemas operacionais diferentes executando independente e concorrentemente na mesma plataforma

Importantes no contexto de confiabilidade e segurança → isolamento de uma aplicação e seu ambiente → falhas não afetam a máquina inteira

Ex.: VMware

Virtualização

- Exemplos:
  - A) Máquina virtual de processo, com várias instâncias de combinações (aplicação, execução).
  - B) Monitor de máquina virtual com várias instâncias de combinações (aplicações, sistema operacional).



Leitura, Estudo e Exercícios

- Este material tem sua fonte no livro:
  - Sistemas Distribuídos: Princípios e Paradigmas.
  - Andrew Tanenbaum e Marteen Steen
  - Capítulo 3

  - Distributed System – Concepts and Design
  - George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair.
  - Capítulo 4