Estrutura e Arquiteturas de SO

Fontes:

Silberschatz cap 2
Tanenbaum cap 1

- Estrutura e organização do SO
- Arquiteturas de SO
  - monolítica
  - em camadas
  - micro-kernel (cliente-servidor)
  - máquina virtual

SO: composição básica

- Um SO consiste, basicamente, de um núcleo (kernel), alguns programas do sistema e aplicações utilitárias que executam diversas tarefas

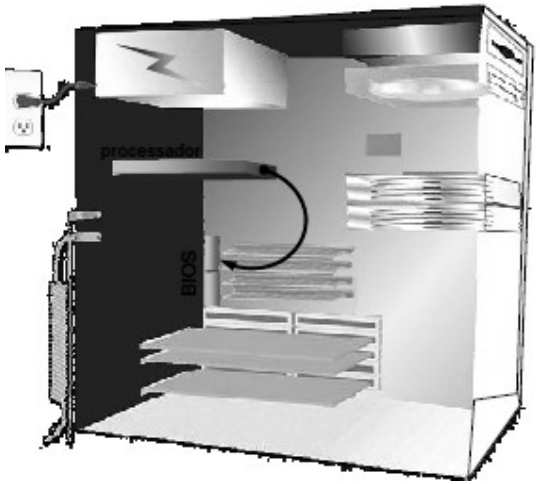
- Kernel
  - Parte central do SO (núcleo)
  - Responsável pela carga do sistema (boot)

Booting (1)

Basic Input/Output System

1) fonte de alimentação fornece energia elétrica p/ as diferentes partes do sistema

2) Processador procura a BIOS (firmware c/ instruções de inicialização)

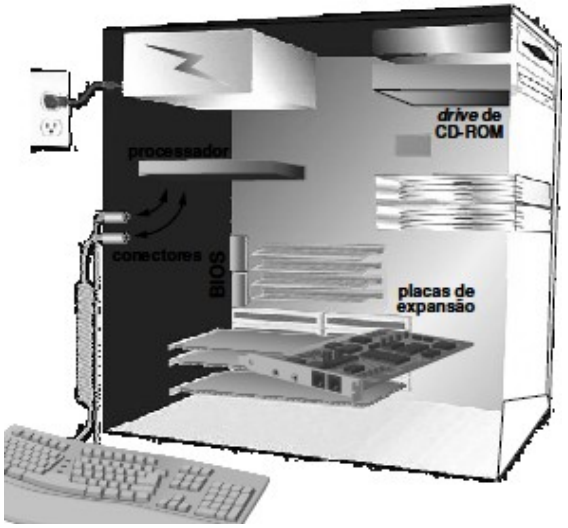


Booting (2)

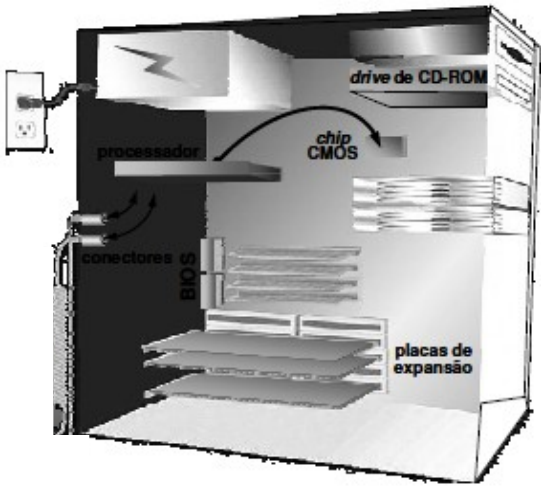
Complementary Metal Oxyde Semiconductor

3) BIOS realiza o POST – teste de verificação de compon. (mouse, teclado, conectores e placas de expansão)

Power-On Self Test

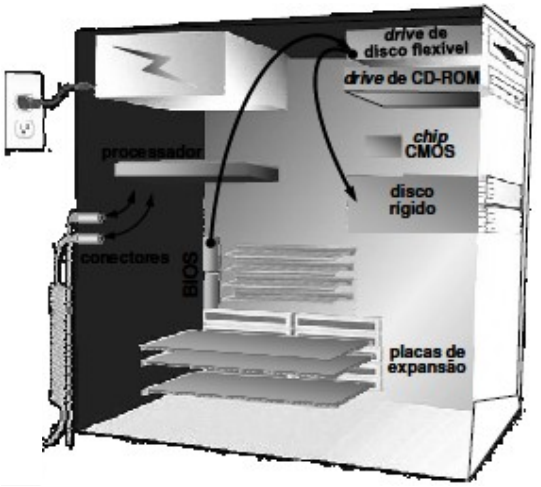


4) Resultados do POST são comparados c/ dados armazenados no chip CMOS (armazena inf. de configuração do comp. e detecta novos dispositivos)

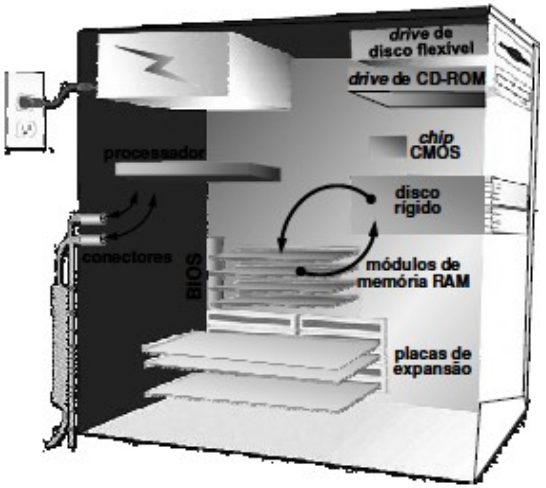


Booting (3)

5) BIOS procura os arquivos de sistema no disco



6) Programa de boot carrega na RAM o kernel do SO (ex. GRUB, LILO, WINLOAD, ...), o qual inicia executando o processo inicial e espera a ocorrência de um evento (**interrupção**)

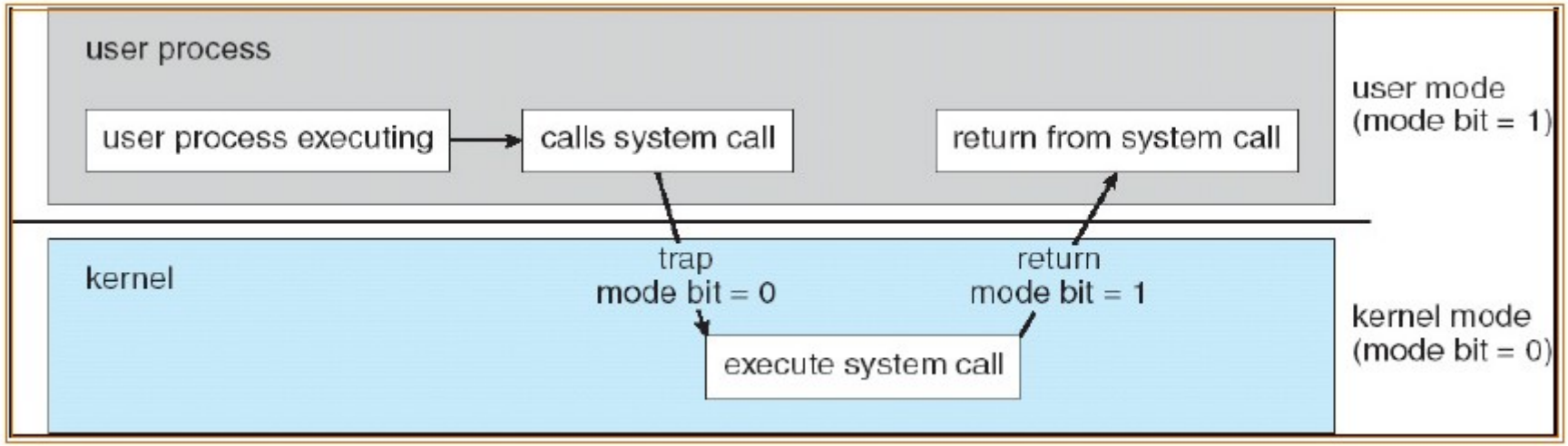


Organização do SO

- Núcleo: conjunto mínimo de serviços executados pelo SO
  - Manipulação de processos, escalonamento
- Chamadas de sistema
  - Funções que os programas dos usuários podem usar p/ acessar os serviços do núcleo
    - ex.: ls, mkdir, cd, …
    - O núcleo assume a execução
- Programas de sistema: serviços menos críticos
    - Compiladores, editores de texto, shell, GUI (Windows), browser, ...

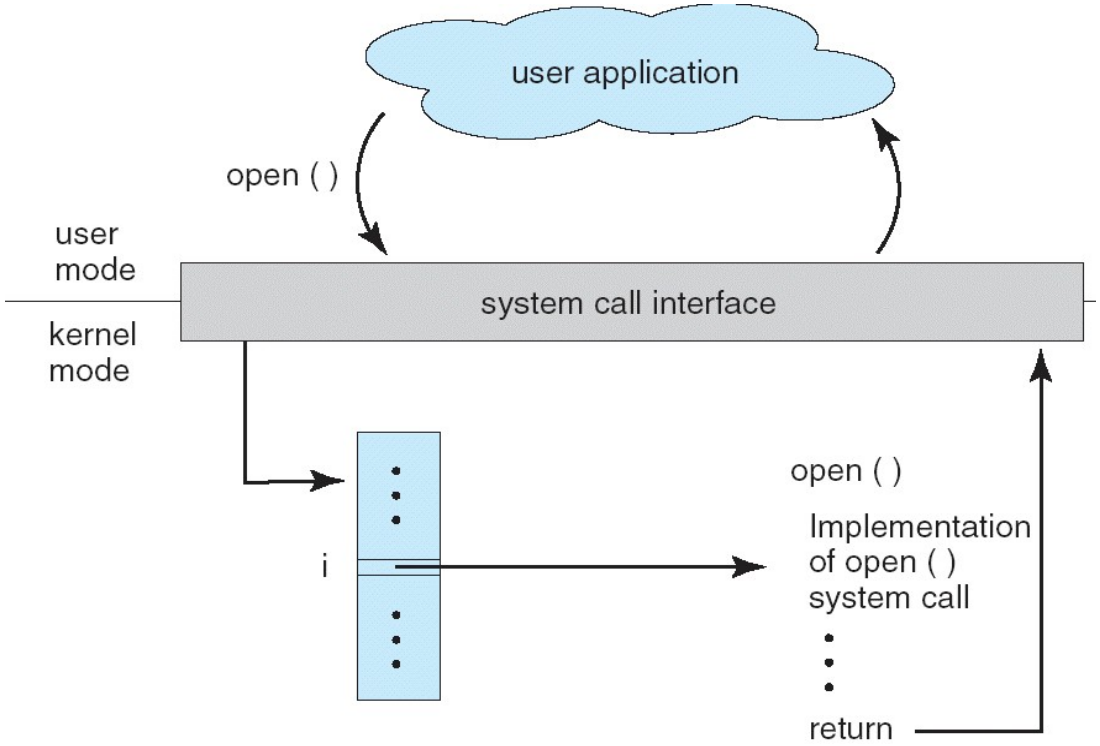
Chamada de sistema - *system call* (1)

- Interface de programação entre um programa em execução (processo) e o SO
- Normalmente acessada via API
  - Ex.: Win32, POSIX Unix, Java API

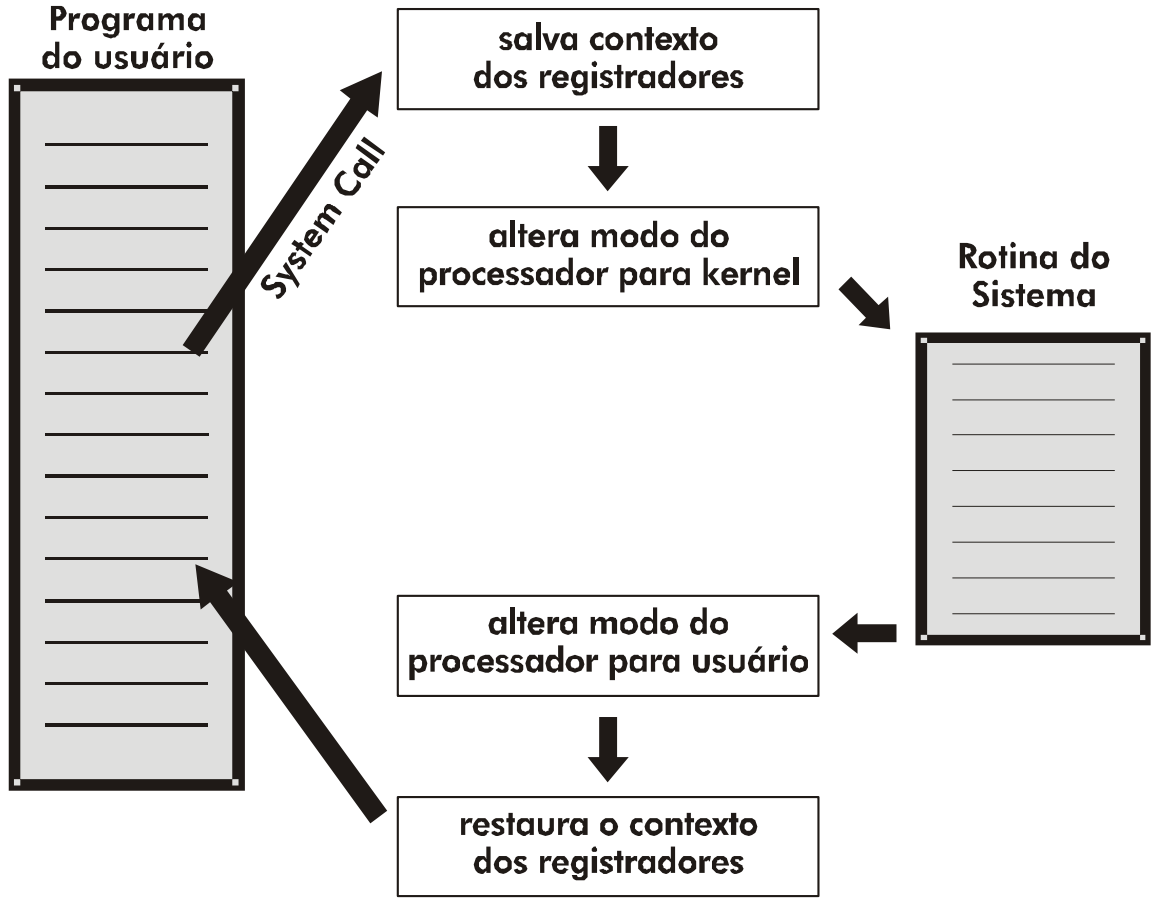


Chamada de sistema - *system call* (2)

- Modo usuário x modo kernel
  - acesso a instruções privilegiadas
  - acesso aos periféricos
  - acesso a áreas de dados específicas

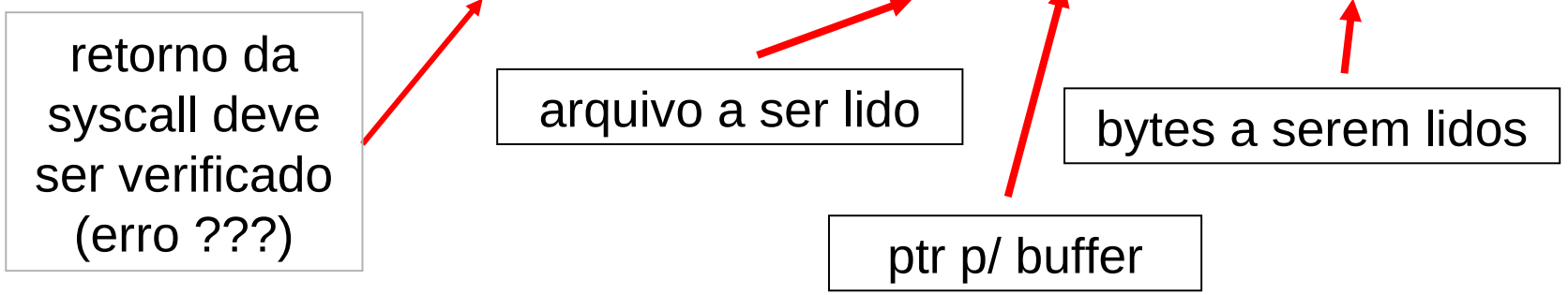


Chamada de sistema (3)

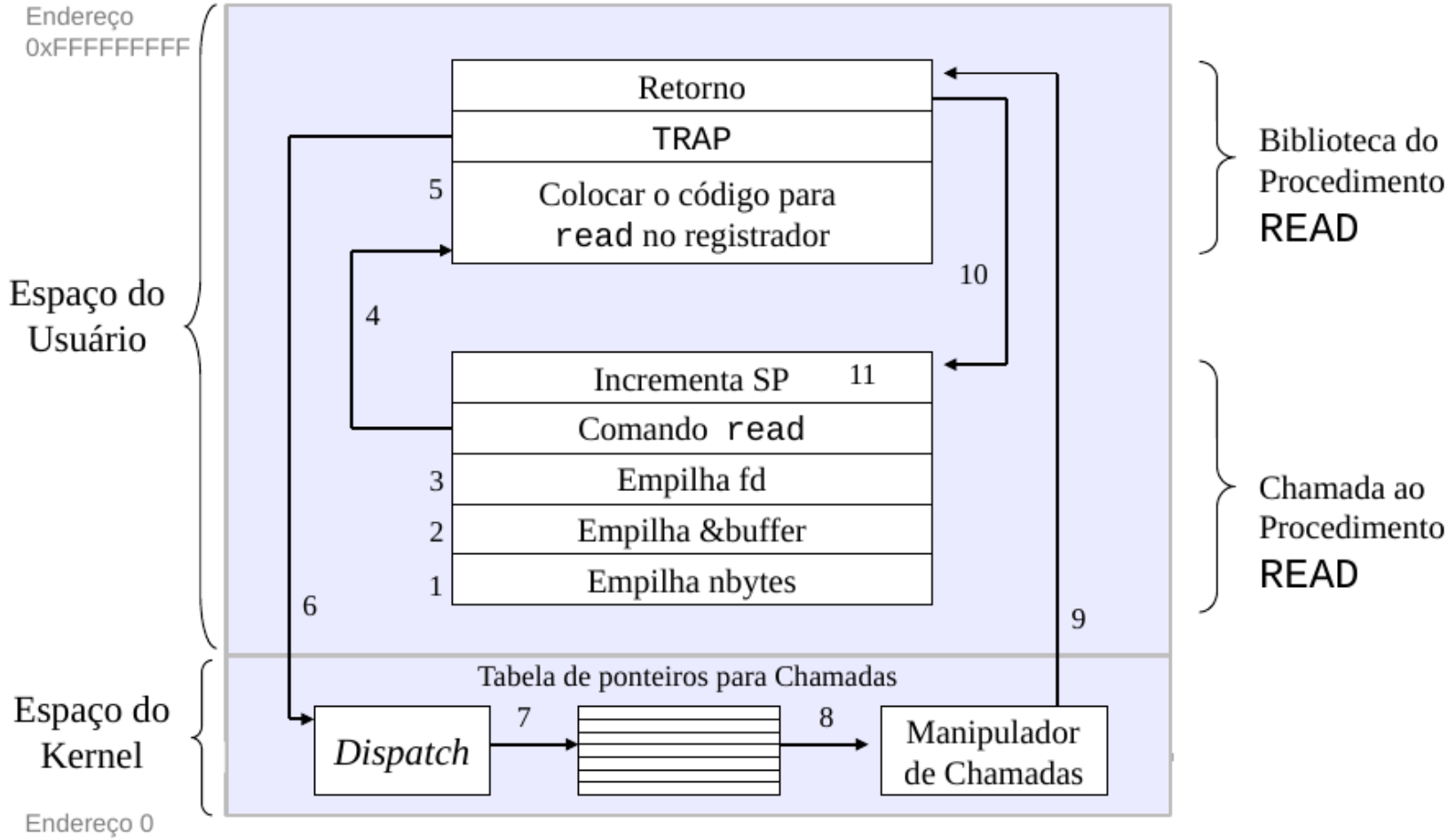


Chamada de sistema (4)

- Tratada pelo HW como *trap* (interrupção de SW)
  - SO examina a instrução de interrupção p/ determinar o que gerou a syscall
  - Parâmetro indica o tipo de serviço solicitado
  - SO analisa, executa e retorna o controle p/ instrução seguinte à syscall
  - Ex.: `count = read(fd,buffer,nbytes);`



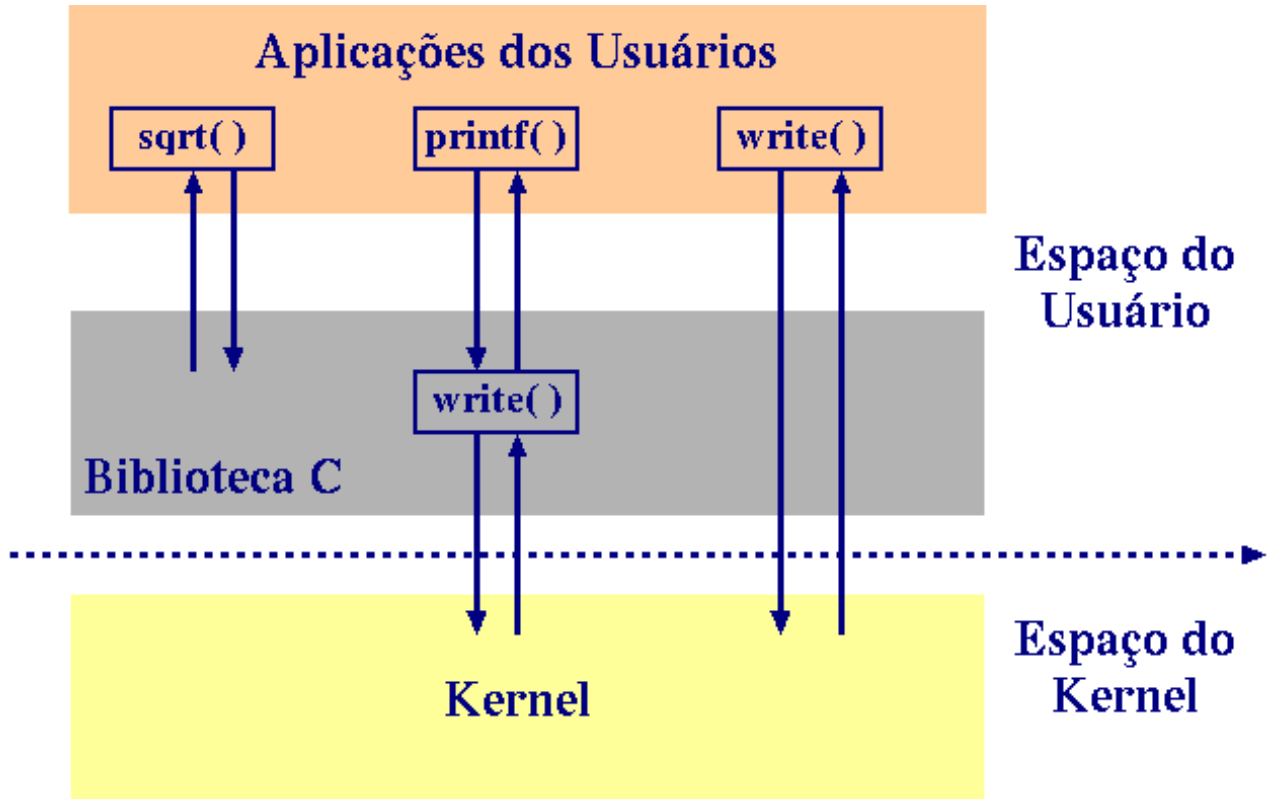
Ex. chamada de sistema - *syscall* **read** passo-a-passo



Chamada de sistema – exemplos

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Chamada de sistema x API



Programas de sistema

- Programas executados fora do kernel (utilitários)
- Implementam tarefas básicas
  - Confundidos com o próprio SO
    - ex. compiladores, assemblers, ligadores, ...
- **Interpretador de comandos**
  - Ativado quando SO inicia sessão de trabalho
    - Ex.: bash, sh, ...
  - Interface gráfica de usuário (GUI)
    - Família Windows, MacOS, ...

Interpretador de Comandos (1)

- Um dos mais importantes *programas do sistema*
- Interface entre usuário e SO
- Alguns SO incluem o interpretador de comandos no kernel

- P/ MS-DOS e Unix é um programa especial:
  - inicia execução quando um processo é inicializado ou quando o 1º usuário se loga (sistema time-sharing)

Interpretador de Comandos (2)

- Programa que lê e interpreta comandos
  - Interpretador de linha de comando → ***shell*** (no Unix)
  - Função: obter o próximo comando do usuário e executar
- Operações:
  - criação e gerência de processos
  - manipulação de E/S
  - gerência de armazenamento secundário, MP
  - acesso a sistema de arquivos
  - proteção
  - ligação em rede

Arquiteturas de Sistemas

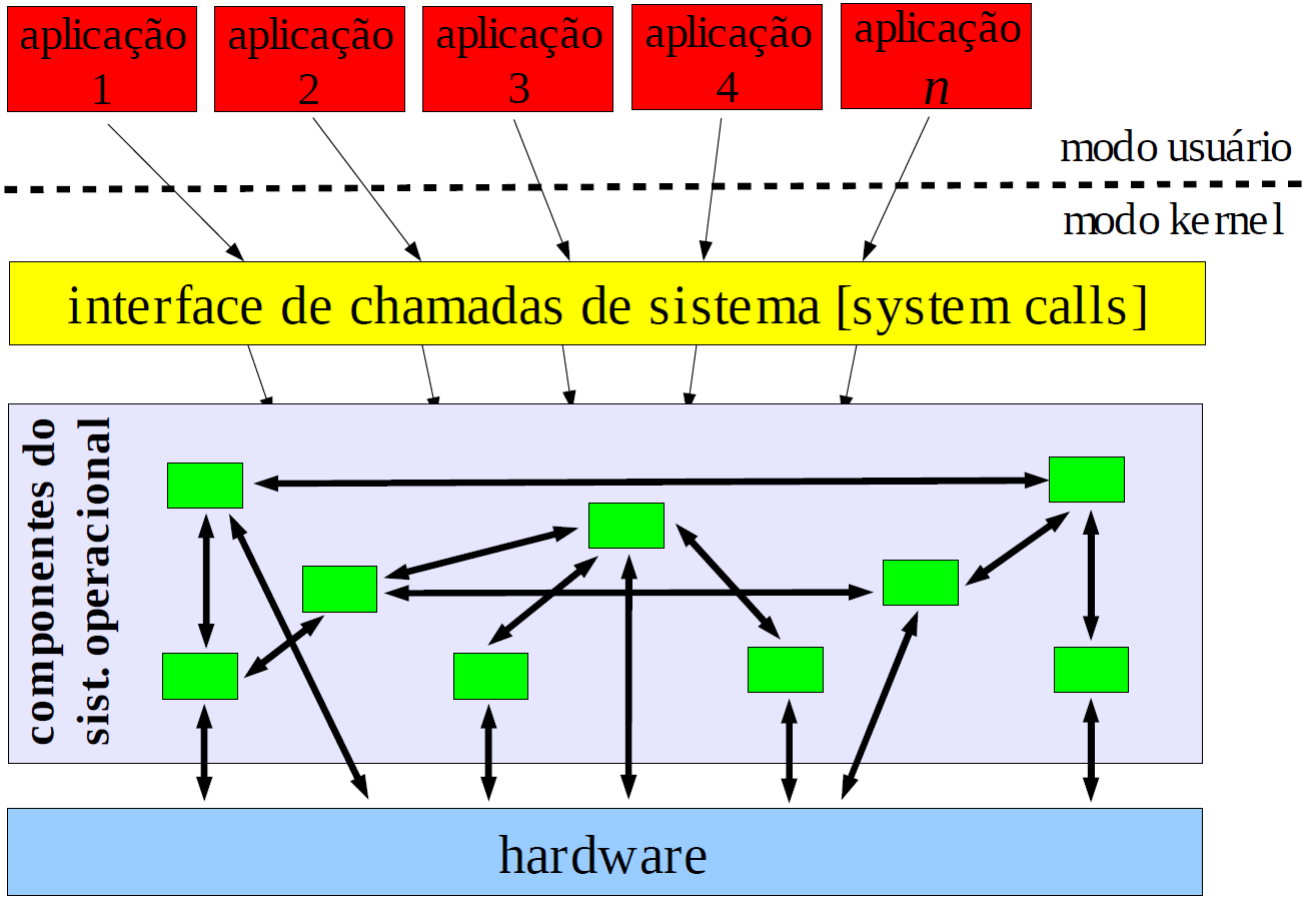
- Projeto e desenvolvimento de um SO devem ser criteriosos

- Arquiteturas:
  - monolítica
  - em camadas
  - micro-kernel (cliente-servidor)
  - híbrida
  - avançadas

Sistema monolítico (1)

- Mantém as principais funções do SO em modo supervisor
- SO = {procedimentos}
- Kernel monolítico
  - Diversos módulos compilados separadamente em um único .exe
- Como todas as funções são concentradas no kernel, não existe uma interface bem definida
- Não é estruturado, nem totalmente desestruturado
  - existe um pouco de estrutura p/ os serviços do sistema que são requisitados via chamadas de sistema

Sistema monolítico (2)



Sistema monolítico (3)

- ☺ estrutura simples
  - bloco único de código que opera em modo núcleo
- ☺ desempenho
  - qualquer componente do núcleo pode acessar diretamente os demais componentes, áreas de memória e dispositivos periféricos
  - essa interação direta leva a sistemas mais rápidos e compactos
  - sem necessidade de mecanismos de comunicação entre os componentes do núcleo

Sistema monolítico (4)

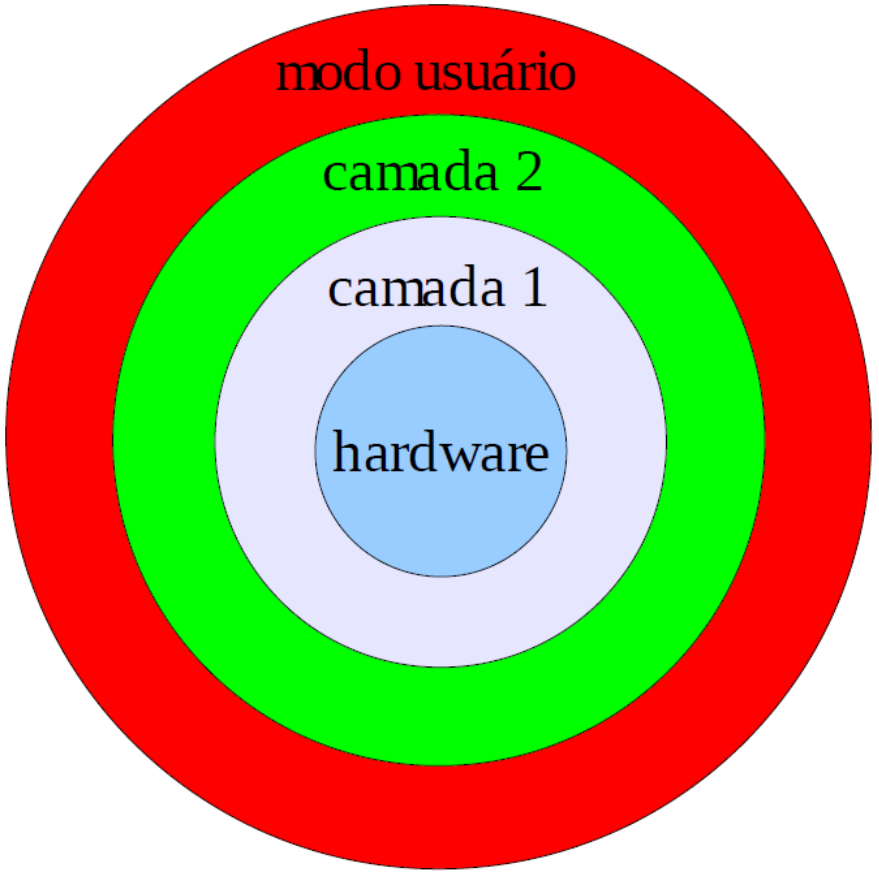
☹ pouco flexível para alterações
  – componentes dependentes entre si

☹ manutenção complexa

- Ex.: 1º sistemas Unix, MS-DOS
- Hoje: FreeBSD, AIX, HP-UX, Linux (monolítico e modular), ...

Sistema em camadas (1)

- SO divido em camadas/níveis
- Funções da camada N são implementadas através de funções na camada N-1
- Possui interface bem definida entre as funções
- Objetivo:
  - dividir sistemas complexos em partes gerenciáveis



Sistema em camadas (2)

☺ modularidade

☺ gerência e manutenção facilitadas

☹ organização das camadas (conteúdo, ordem)

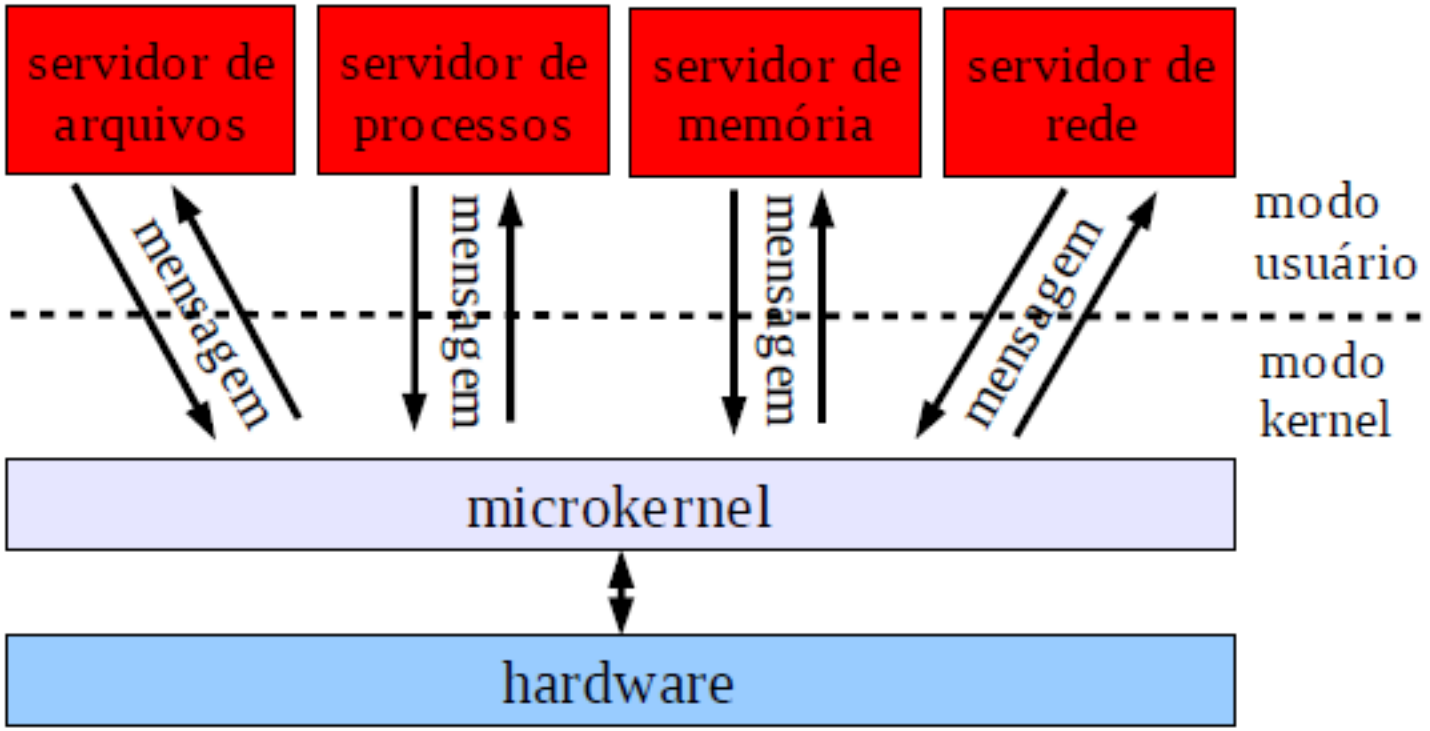
☹ overhead de cada camada – desempenho

- Ex.: MULTICS, THE
- Atualmente:
  - sistemas implementam uma camada inferior de abstração do HW p/ interagir c/ dispositivos → camada HAL (Hardware Abstraction Level), implementada pelo Windows NT e sucessores

Sistema Micro-kernel (1)

- Move código p/ camadas mais altas, deixando um mínimo de kernel
  - Núcleo implementa: espaços de memória protegidos, tarefa (thread, processo,…), comunicação entre tarefas e operações de acesso a E/S, ...
- Restante do SO é organizado em um conjunto de rotinas (serviços)
  - Políticas de uso do processador e da memória, sistema de arquivos, controle de acesso a recursos, protocolos de rede, ...

Sistema Micro-kernel (2)



- Variação: Modelo cliente-servidor

Sistema Micro-kernel (3)

☺ portabilidade

☺ modularidade e escalabilidade

☺ servidores fáceis de gerenciar e depurar

☺ independência dos servidores

☺ proteção (s/ acesso direto ao HW)

☺ serviços executam em modo usuário

☹ desempenho

- Ex.: Mach, Chorus, Minix 3, QNX

Sistema Híbrido

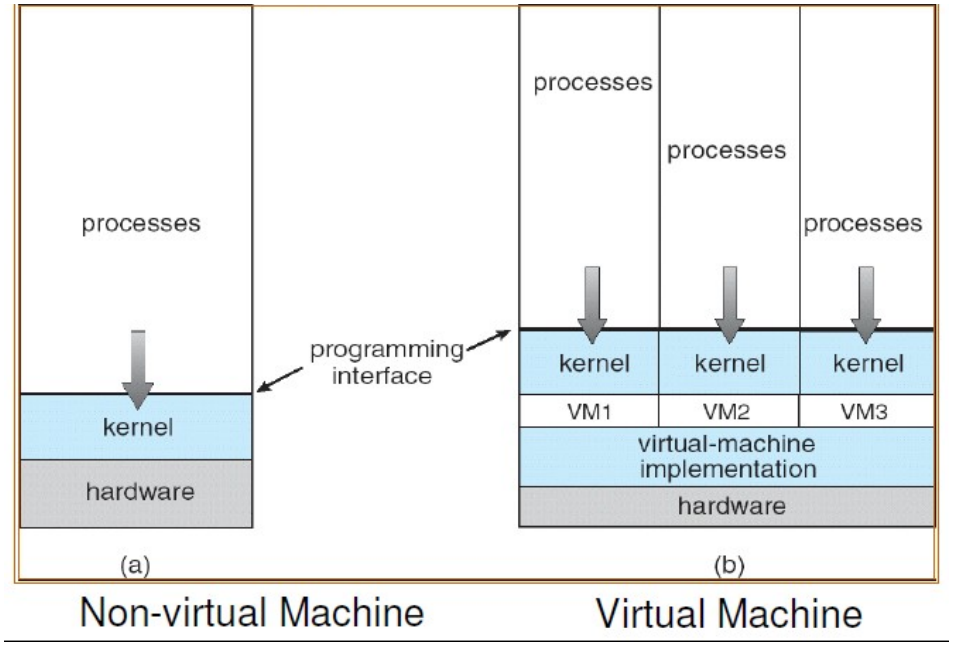
- Abordagem intermediária entre a estrutura monolítica e micro-kernel, com influência da estrutura em camadas
- Objetivo: diminuir o problema do baixo desempenho da estrutura micro-kernel
  - mantém no nível do núcleo os componentes mais críticos – **núcleo híbrido**
- A maioria dos sistemas atuais é híbrida

Arquiteturas avançadas

- Organizam os componentes do SO visando contextos de aplicação específicos
  - Máquina virtual
  - Contêineres

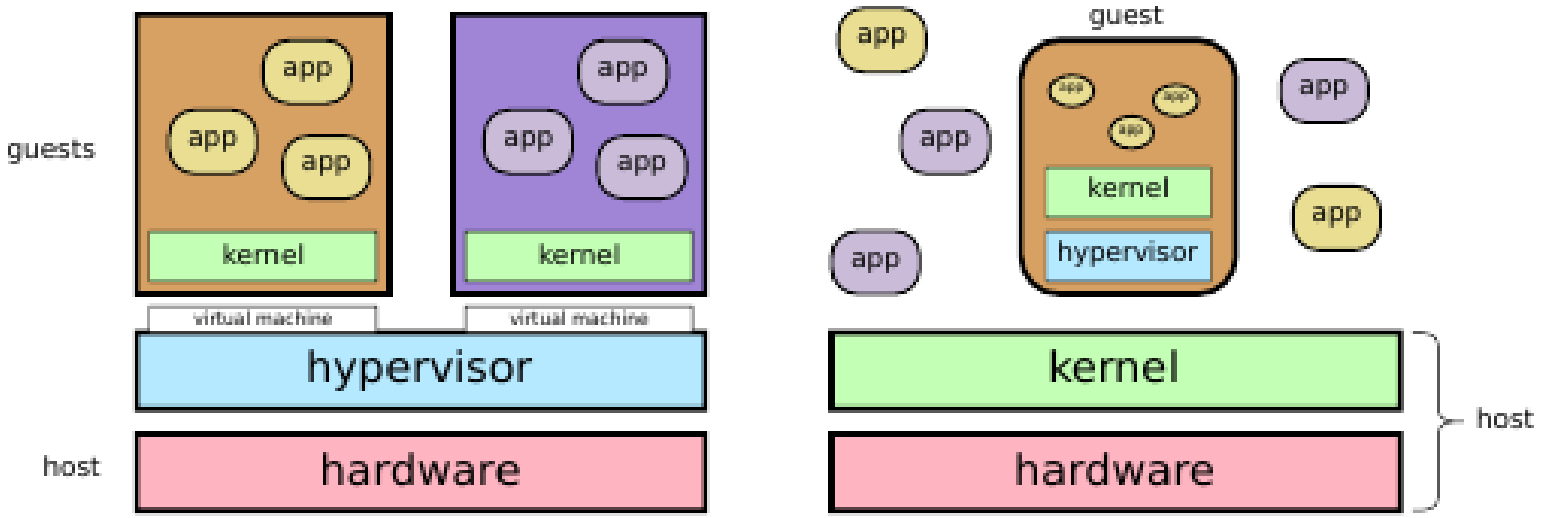
Máquina Virtual (1)

- Virtualização
  - Simula em software um sistema computacional sobre outro sistema



Máquina Virtual (2)

- host: sistema hospedeiro – contém os recursos reais de HW e SW do sistema
- hypervisor (ou VMM – Virtual Machine Monitor) – constrói o sistema computacional virtual a partir dos recursos do sistema
- guest: sistema convidado – executa sobre o sistema virtual



créditos: prof. Maziero

Máquina Virtual (3)

- Hipervisores
  - Quanto ao ambiente virtual oferecido
    - HV de aplicação: suporta aplicação convidada (Java, C#)
    - HV de sistema: suporta SOs convidados (VMWare, VirtualBox)
  - Quanto ao suporte de execução
    - HV nativo: executa diretamente sobre o HW (Xen)
    - HV convidado: executa sobre um SO hospedeiro (VirtualBox)

Máquina Virtual (4)

☺ proteção aos recursos do sistema

☺ independência

☺ flexibilidade

☹ sem compartilhamento de recursos

☹ sobrecarga

☹ complexidade (cópia exata do hw ???)

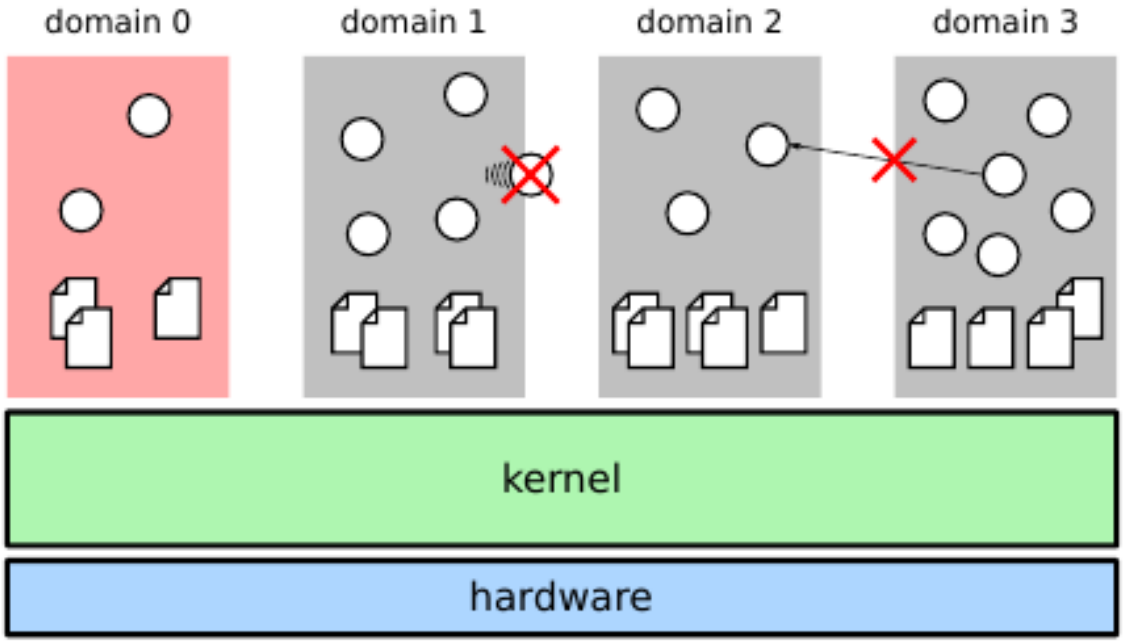
- 1º sistema VM - System/370(IBM)

Contêiner (1)

- Assim como máquina virtual, é uma forma de isolar aplicações ou subsistemas em um SO
  - virtualiza o espaço de usuário do SO em domínios ou contêineres
  - cada domínio recebe uma parcela dos recursos do SO
  - isolamento garantido por uma interface de rede virtual (cada domínio tem seu próprio endereço de rede)
  - Ex.: Jails (FreeBSD), zonas (Solaris), Linux Containers (LXC), Docker, Kubernetes

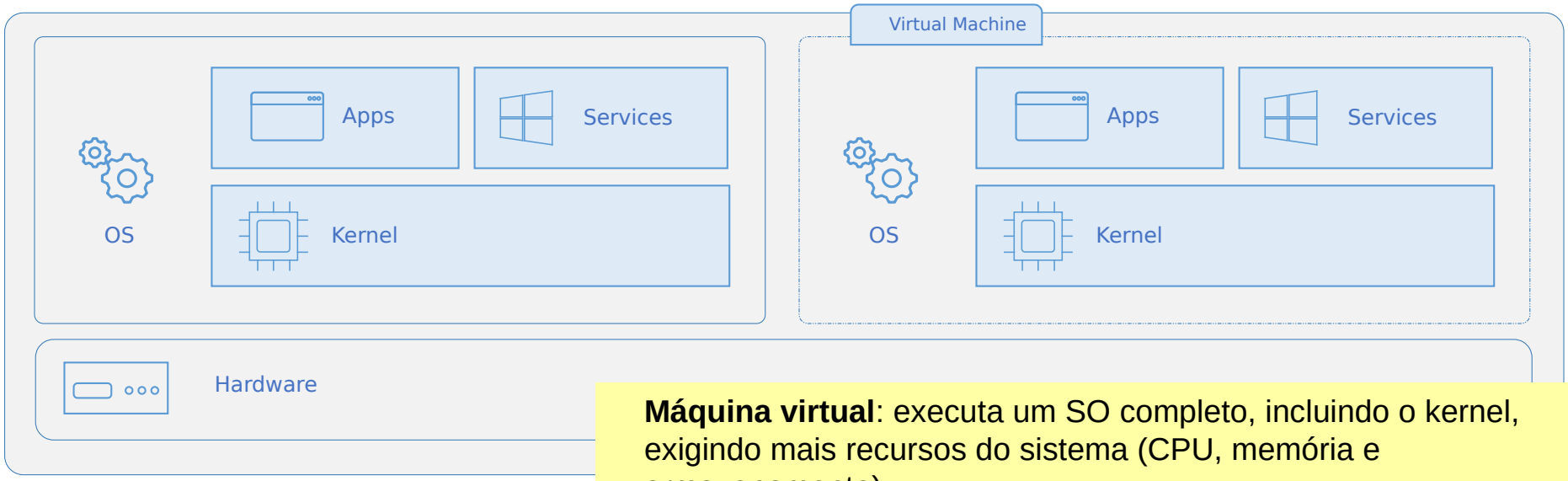
Contêiner (2)

- Processos em um mesmo domínio podem interagir entre si
- Domínio de inicial ou privilegiado (d_0)
  - processos têm visibilidade e acesso a processos e recursos dos demais domínios

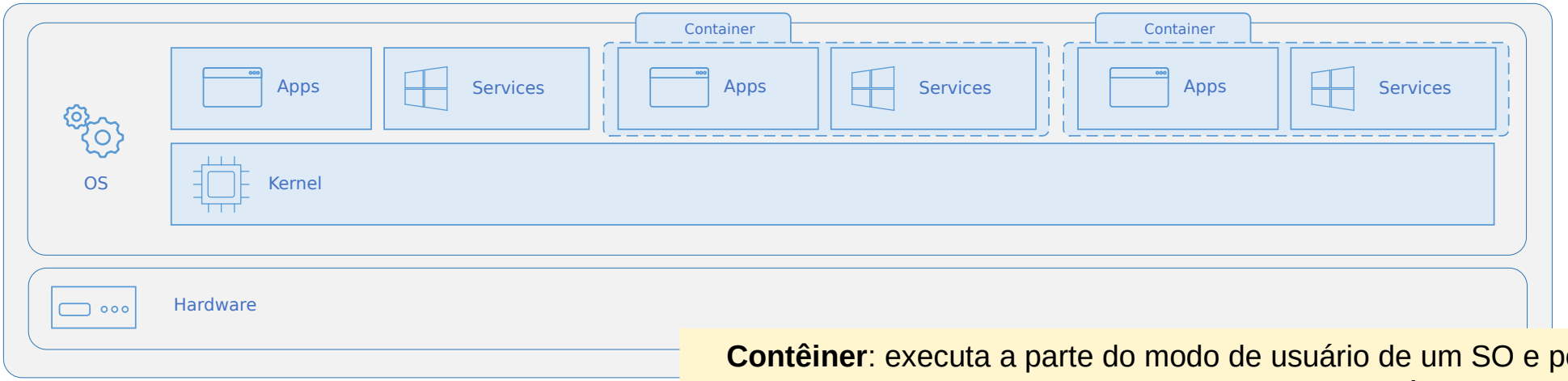


créditos: prof. Maziero

Máquina virtual x Contêiner



Máquina virtual: executa um SO completo, incluindo o kernel, exigindo mais recursos do sistema (CPU, memória e armazenamento)



Contêiner: executa a parte do modo de usuário de um SO e pode ser adaptado p/ conter apenas os serviços necessários p/ sua aplicação, usando menos recusos do sistema