Interfaces e Estruturas de SO

- Interfaces com o SO
- Estruturas de SO
  - monolítica
  - em camadas
  - micro-kernel (cliente-servidor)
  - máquina virtual

Interfaces com o SO

- Programas utilitários e aplicações
- Shell ou Interpretador de Comandos
- Chamadas de sistema (*system calls*)
- API para chamadas de sistema

Programas utilitários e aplicações

- Serviços de mais alto nível
- interface entre usuário e SO
- 4 categorias principais:
  - Manipulação de arquivos
  - Informações sobre o sistema
  - Suporte a linguagens
  - Carga/execução de programas

Interpretador de Comandos (1)

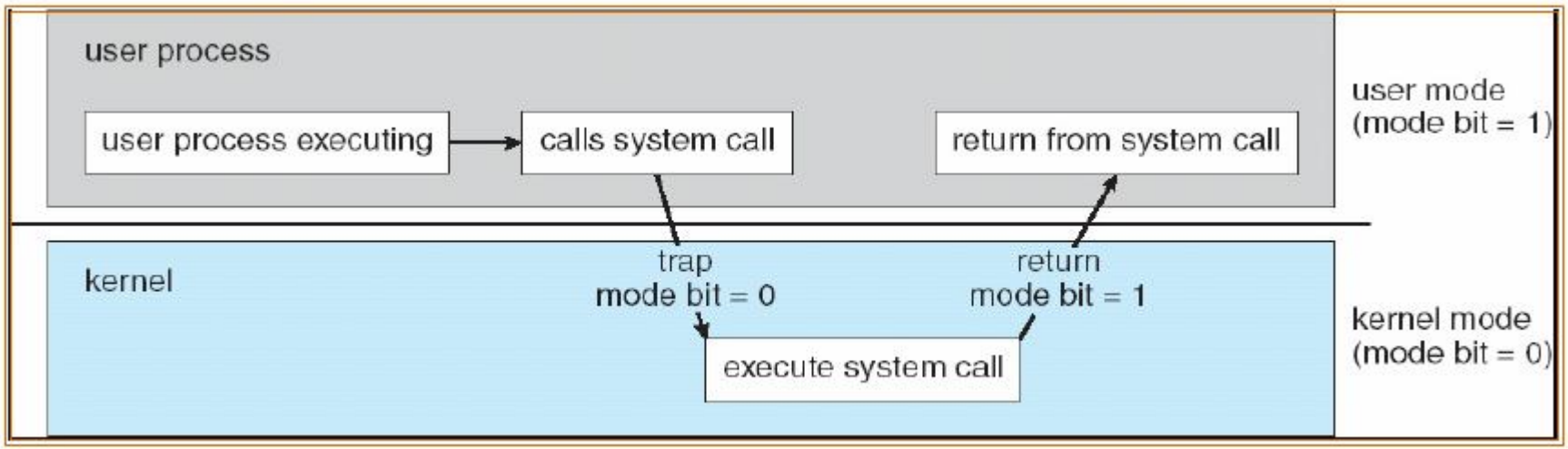
- um dos mais importantes *programas do sistema*
- interface entre usuário e SO
- alguns SO incluem o interpretador de comandos no kernel
- ativado sempre que o SO inicia uma sessão de trabalho
  - ex.: bash, cash, sh, ...
- p/ MS-DOS e Unix é um programa especial:
  - inicia execução quando um processo 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

Chamada de sistema - *system call* (1)

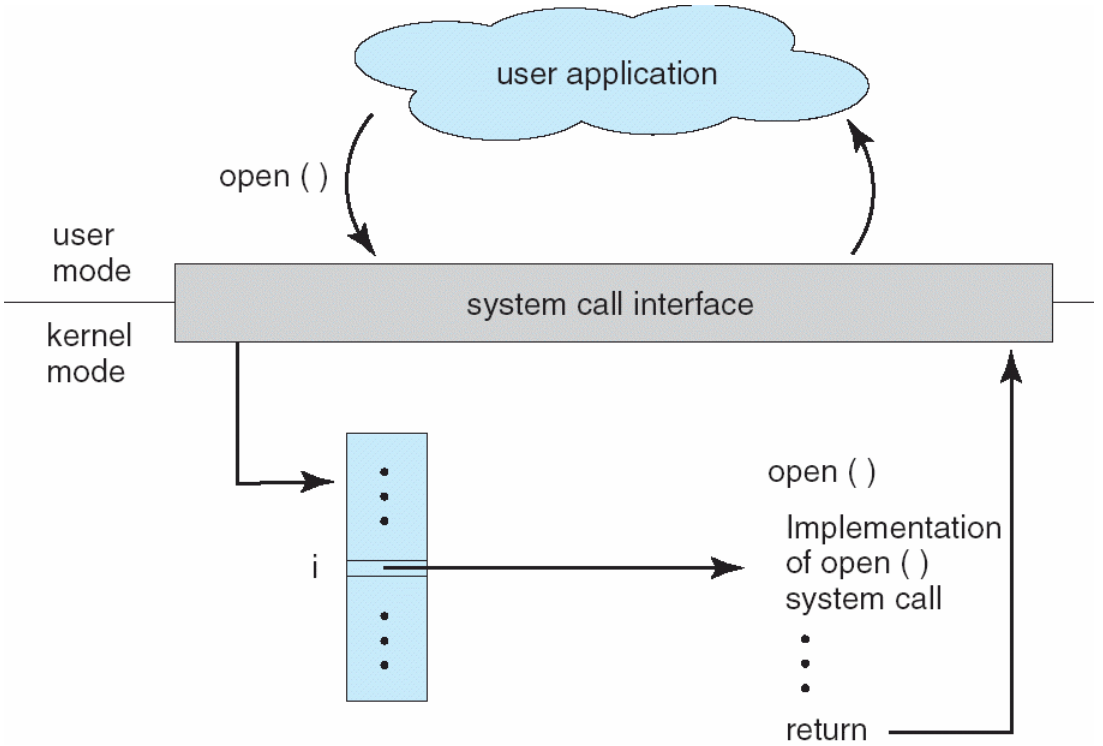
- interface de programação entre um programa em execução (processo) e o SO



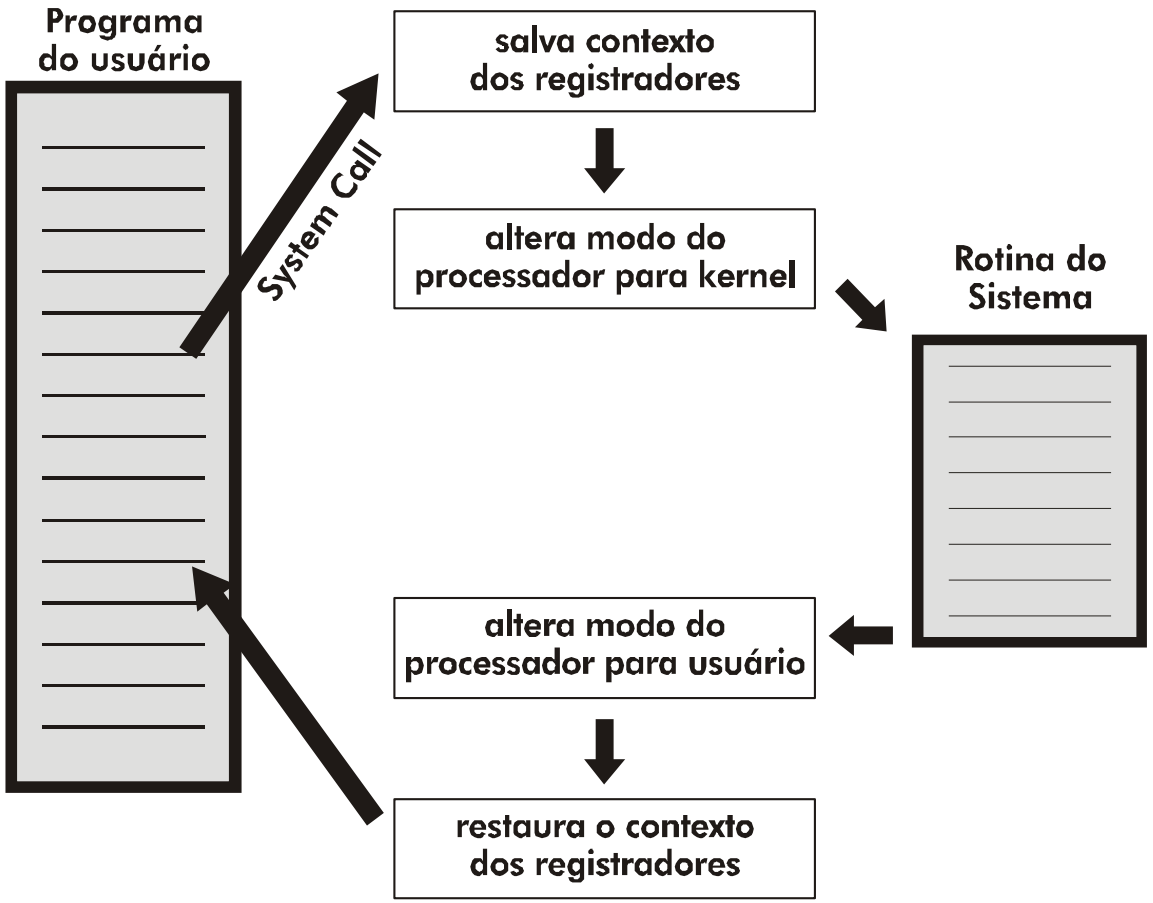
- normalmente acessada via API
  - Ex.: Win32, POSIX Unix, Java API

Chamada de sistema - *system call* (2)

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

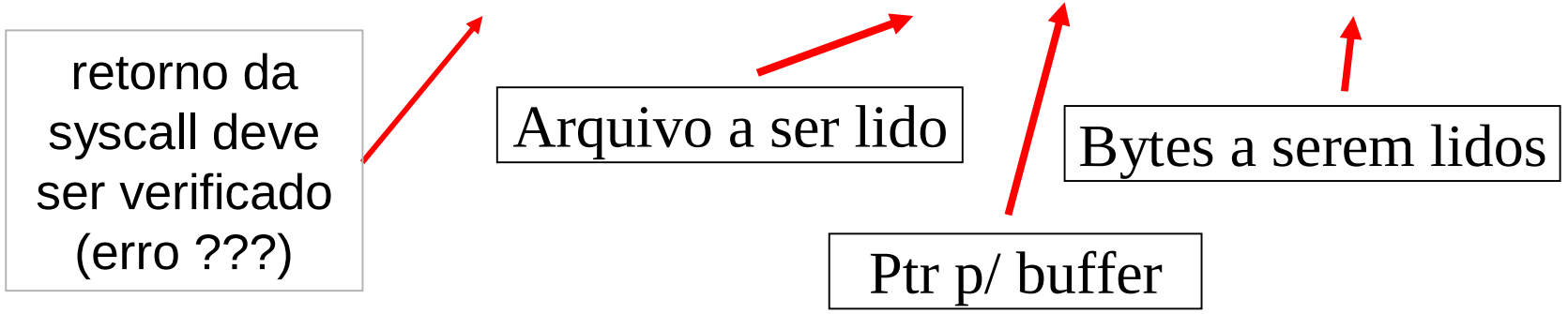


Chamada de sistema (3)

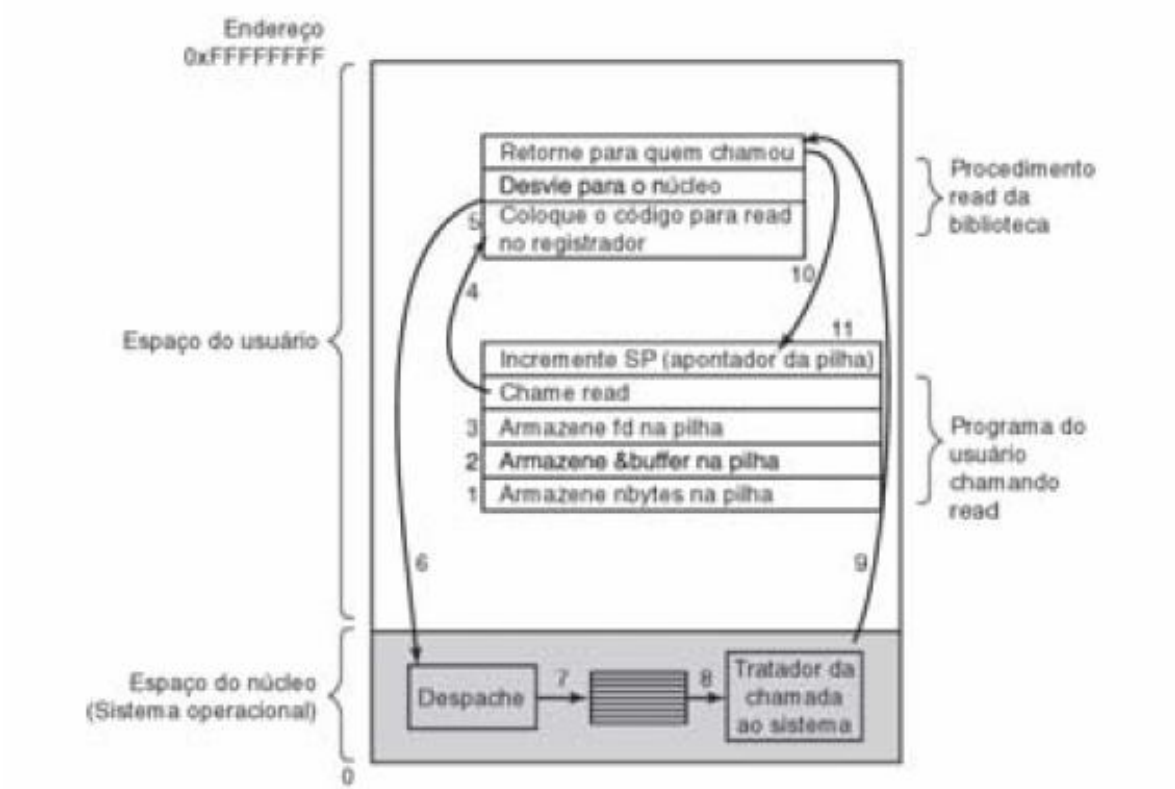


Chamada de sistema (4)

- tratada pelo HW como *trap*
  - 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);`



Chamada de sistema - *syscall read* (5)



Os 11 passos para fazer uma chamada ao sistema

read (fd, buffer, nbytes)

Chamada de sistema – classificação (6)

- Controle de processos
  - end, abort, load, execute, create, terminate, wait time, event, signal, allocate, free memory
- Gerenciamento de arquivos
  - create, delete, open, close, read, write, reposition, get/set attributes
- Gerenciamento de dispositivos
  - request, release, read, write, reposition, get/set attributes
- Gerenciamento de informações
  - get/set time/date, get/set system data
- Comunicação
  - create/delete connection, send, receive, transfer status information

Estruturas de Sistemas

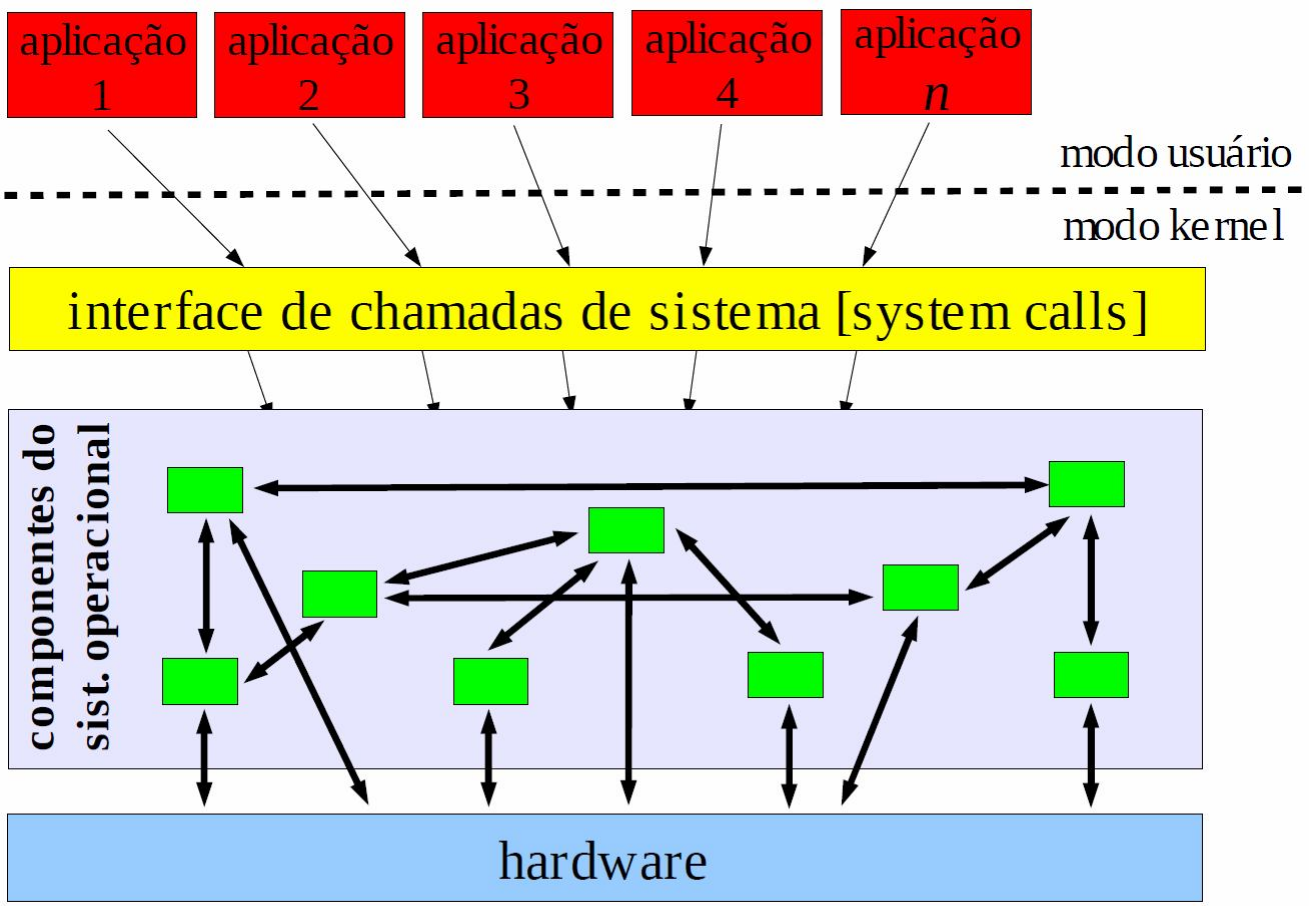
- projeto e desenvolvimento de um SO devem ser criteriosos

- estruturas:
  - monolítica
  - em camadas
  - micro-kernel (cliente-servidor)
  - máquina virtual

Estrutura monolítica (1)

- mantém as principais funções do SO em modo supervisor
- 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

Estrutura monolítica (2)



Estrutura monolítica (3)

☺ estrutura simples

☺ desempenho

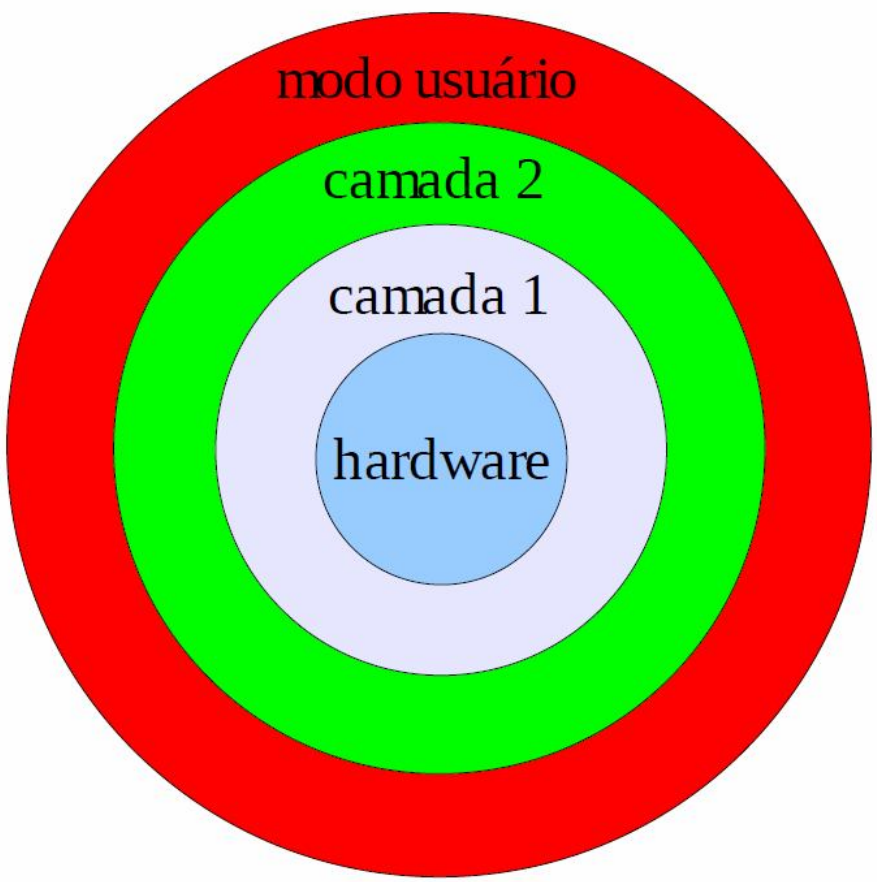
☹ pouco flexível p/ alterações

☹ manutenção

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

Estrutura 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



Estrutura em camadas (2)

☺ modularidade

☺ gerência e manutenção fáceis

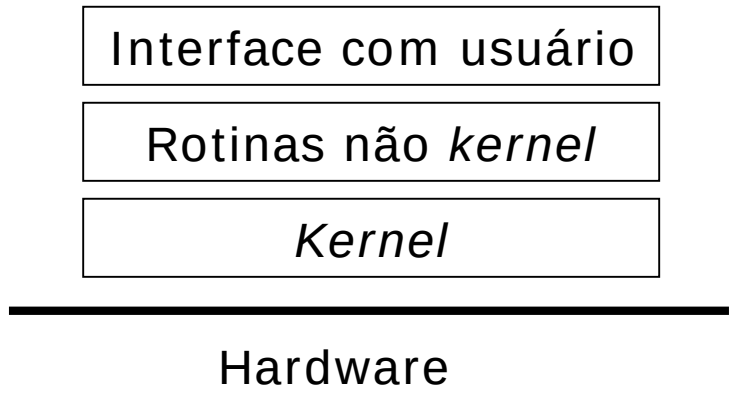
☹ organização das camadas (conteúdo, ordem)

☹ *overhead* de cada camada

- Ex.: THE (sistema batch simples, com 6 camadas), Minix, IBM OS/2, MULTICS, Windows NT, Mac OS X

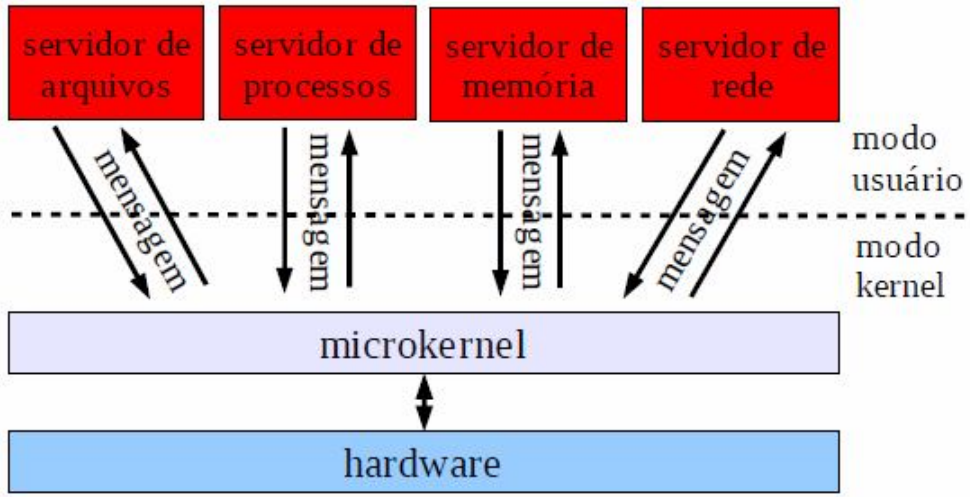
Estrutura Micro-kernel (1)

- Base: modelo cliente-servidor
- move código p/ camadas mais altas, deixando um mínimo de kernel
- *Kernel* = núcleo do SO
- restante do SO é organizado em um conjunto de *rotinas não-kernel*



Estrutura Micro-kernel (2)

- funções do SO → processos de usuário
  - requisição de serviço
    - processo cliente
  - atendimento do serviço
    - processo servidor



  - Kernel
    - gerencia comunicação clientes – servidores (troca de msgs)

Estrutura Micro-kernel (3)

☺ portabilidade

☺ escalabilidade

☺ servidores fáceis de gerenciar e depurar

☺ independência dos servidores

☺ proteção (s/ acesso direto ao hw)

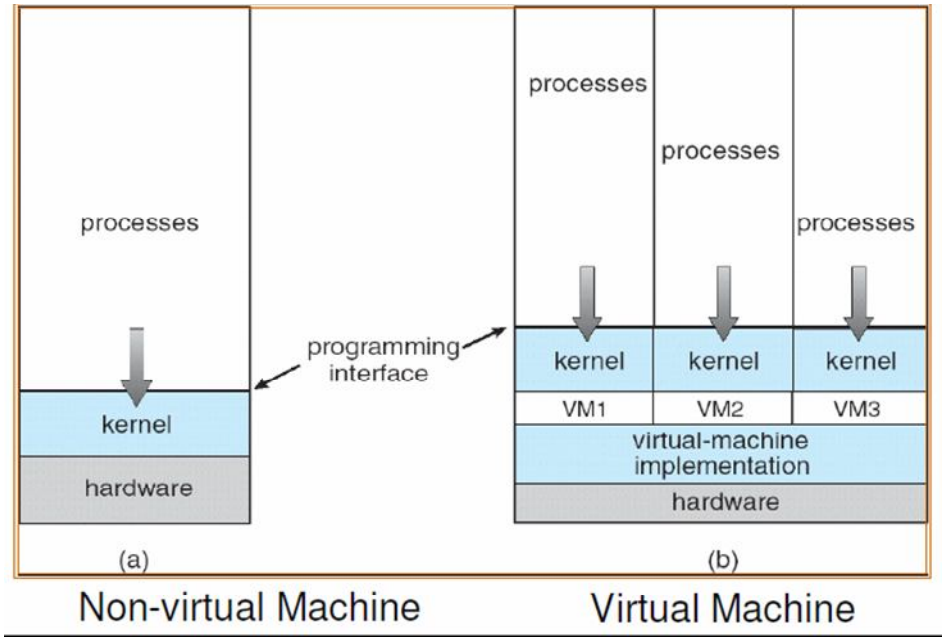
☺ serviços executam em modo usuário

☹ desempenho

- Ex.: Mach, Chorus, QNX, Windows baseados no NT

Estrutura Máquina Virtual (1)

- nível intermediário entre SO e hw
  - diversas MV independentes e isoladas
  - cada MV oferece uma cópia virtual do hw (modos de acesso, interrupções, dispositivos de E/S, etc)
  - cada MV pode rodar um SO diferente



Estrutura Máquina Virtual (2)

☺ proteção aos recursos do sistema

☺ independência

☺ flexibilidade

☹ sem compartilhamento de recursos

☹ sobrecarga

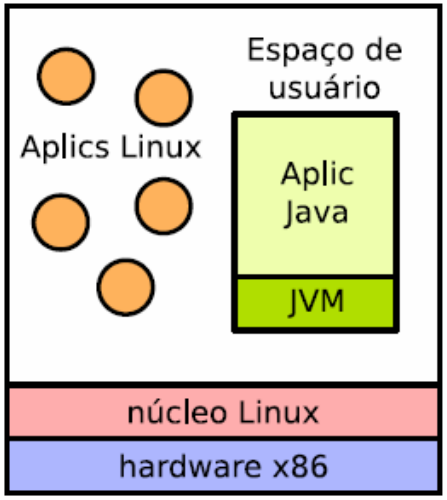
☹ complexidade (cópia exata do hw ???)

- 1º sistema VM - System/370(IBM)

Máquina virtual: hoje (3)

MV de aplicação

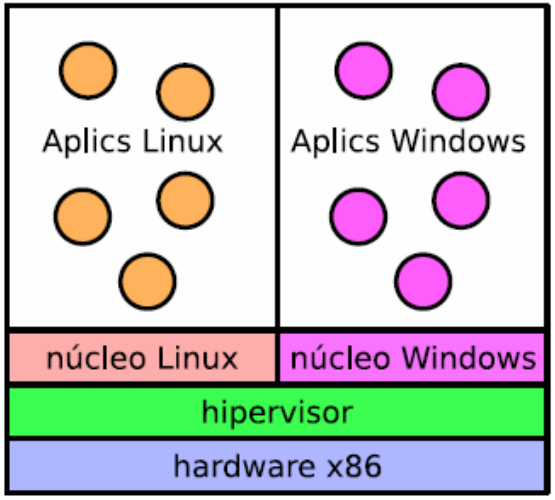
- suporta apenas um processo ou aplicação específica
- Ex.: máquina virtual Java



VM de aplicação

MV de sistema

- suporta SOs completos, com aplicações executando sobre eles
- Ex.: *VMWare, Xen e VirtualBox*



VM de sistema