Sistemas Operacionais

Prof. Rafael Obelheiro
rafael.obelheiro@udesc.br

UDESC

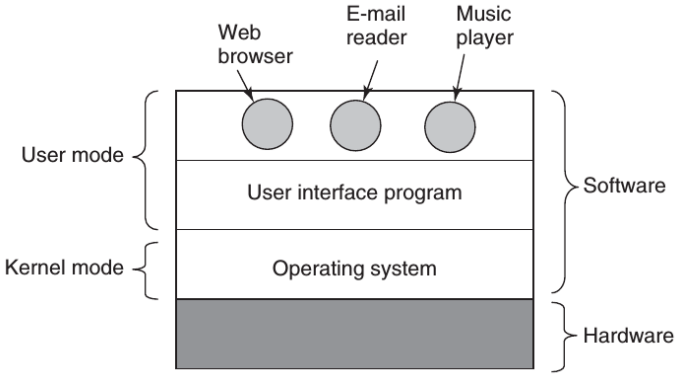
Fundamentos de SO

Sumário

O que é um sistema operacional?

- O gerenciamento de um sistema computacional moderno é sem dúvida uma tarefa complexa
- O computador moderno exige o controle no uso de um ou mais processadores, memória, discos, impressoras de maneira correta e otimizada
- Duas visões
  1. Uma máquina estendida **abstração**
     - ★ esconde os detalhes complicados do funcionamento do hardware
     - ★ oferece uma interface mais amigável para as aplicações
       ⇒ *máquina virtual*
  2. Um gerenciador de recursos **gerência**
     - ★ cada programa utiliza o recurso durante um tempo
     - ★ cada programa ocupa um certo espaço no recurso

Camadas de um SO



Sumário

Histórico dos sistemas operacionais

período	hardware	SO
1945–1955	válvulas painéis de programação	praticamente inexistente
1955–1965	transistores mainframes centros de computação	sistemas em lote
1965–1980	CIs minicomputadores terminais	multiprogramação tempo compartilhado
1980–hoje	computadores pessoais workstations microcontroladores	desktop LANs, cliente-servidor SOs embarcados
2000–hoje	smartphones	mobile

Tipos de sistemas operacionais

- SOs para mainframes
- SOs para servidores
- SOs para multiprocessadores
- SOs para computadores pessoais
- SOs para dispositivos móveis
- SOs de tempo real
- SOs embarcados

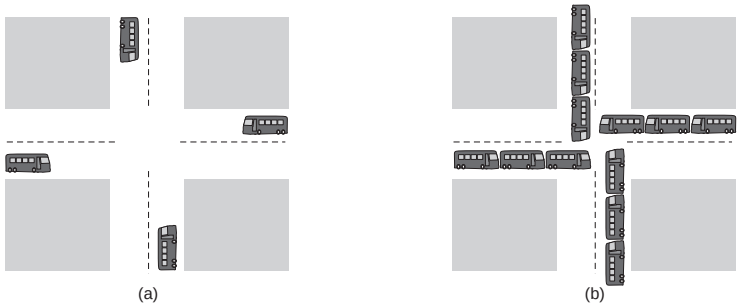
Sumário

Processos

- Um **processo** é basicamente um programa em execução
- Cada processo possui um **espaço de endereçamento**
  - faixa de endereços de memória onde ele pode ler e escrever
- Cada entrada na tabela de processos possui informações sobre o estado corrente do processo
- Processos podem se comunicar → comunicação interprocessos (IPC)

Deadlocks

- Situações que surgem na interação entre processos e onde o progresso é impossível:
  (a) deadlock potencial
  (b) deadlock real
- Muito comuns em programação concorrente

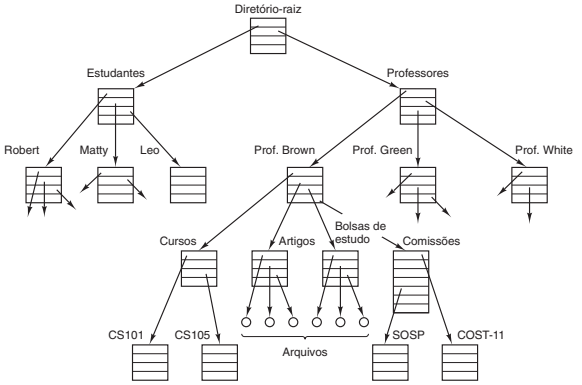


Gerência de memória

- Define como a memória principal é alocada para os processos
- Implementa mecanismos de proteção
- Lida com espaços de endereçamento maiores do que a memória disponível → memória virtual

Gerência de arquivos

- Define uma interface mais refinada para armazenamento persistente de informações
- A maior parte dos sistemas usa o conceito de arquivos organizados em hierarquias de diretórios



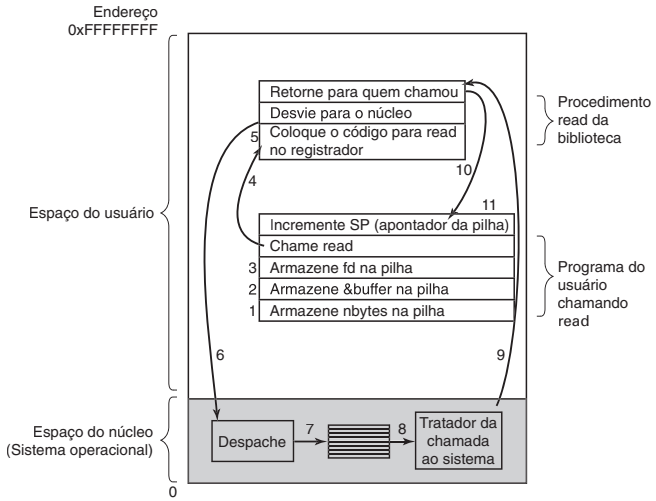
Gerência de E/S

- Subsistema que lida com os mais variados dispositivos de entrada e saída
  - acesso direto ao hardware
  - capaz de lidar com particularidades de cada dispositivo
- Funcionalidades comuns incluem alocação de dispositivos e gerenciamento de buffers

Chamadas de sistema

- As **chamadas de sistema** compõem a interface que o SO oferece às aplicações
- Executam no contexto do SO, usando o modo privilegiado do processador
- Tipos de chamada de sistema
  - processos: criar, sincronizar, terminar
  - memória: alocar, desalocar
  - arquivos e diretórios: criar, ler, escrever, remover, definir permissões
  - ...

Exemplo de chamada de sistema: `read()`



`read(fd, &buffer, nbytes)`

Chamadas de sistema do xv6 (MIT)

chamada de sistema	descrição
1. `fork()`	cria um processo
2. `exit()`	encerra o processo corrente
3. `wait()`	espera o encerramento de um processo filho
4. `kill(pid)`	encerra o processo `pid`
5. `getpid()`	retorna o ID do processo corrente
6. `sleep(n)`	dorme por `n` ticks do relógio
7. `exec(filename, *argv)`	carrega e executa um arquivo
8. `sbrk(n)`	aumenta a memória do processo em `n` bytes
9. `open(filename, flags)`	abre um arquivo; `flags` indica leitura/escrita
10. `read(fd, buf, n)`	lê `n` bytes de um arquivo aberto para `buf`
11. `write(fd, buf, n)`	escreve `n` bytes em um arquivo aberto
12. `close(fd)`	libera o arquivo aberto `fd`
13. `dup(fd)`	duplica `fd`
14. `pipe(p)`	cria um pipe e retorna seus descritores
15. `chdir(dirname)`	muda o diretório corrente
16. `mkdir(dirname)`	cria um novo diretório
17. `mknod(name, major, minor)`	cria um arquivo de dispositivo
18. `fstat(fd)`	retorna informações sobre um arquivo aberto
19. `link(f1, f2)`	cria um outro nome (`f2`) para o arquivo `f1`
20. `unlink(filename)`	remove um arquivo

Exemplo de chamadas de sistema da API Win32

Unix	Win32	Descrição
fork	CreateProcess	Crie um novo processo
waitpid	WaitForSingleObject	Pode esperar um processo sair
execve	(none)	CrieProcesso = fork + execve
exit	ExitProcess	Termine a execução
open	CreateFile	Crie um arquivo ou abra um arquivo existente
close	CloseHandle	Feche um arquivo
read	ReadFile	Leia dados de um arquivo
write	WriteFile	Escreva dados para um arquivo
lseek	SetFilePointer	Mova o ponteiro de posição do arquivo
stat	GetFileAttributesEx	Obtenha os atributos do arquivo
mkdir	CreateDirectory	Crie um novo diretório
rmdir	RemoveDirectory	Remova um diretório vazio
link	(none)	Win32 não suporta ligações (link)
unlink	DeleteFile	Destrua um arquivo existente
mount	(none)	Win32 não suporta mount
umount	(none)	Win32 não suporta mount
chdir	SetCurrentDirectory	Altere o diretório de trabalho atual
chmod	(none)	Win32 não suporta segurança (embora NT suporte)
kill	(none)	Win32 não suporta sinais
time	GetLocalTime	Obtenha o horário atual

Chamadas de sistema: comparação entre SOs

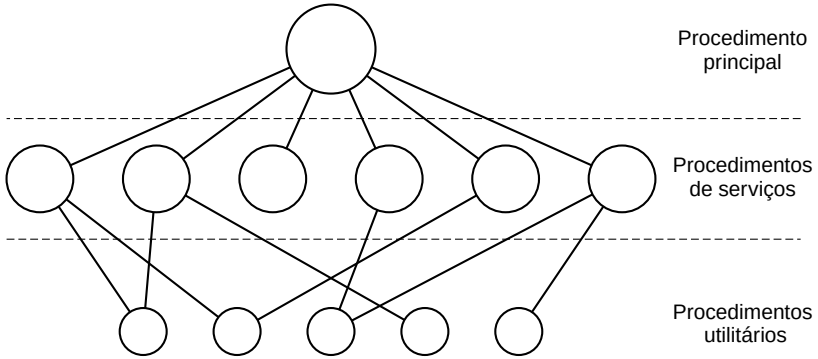
SO	ano	nº de chamadas
Unix V7	1979	≈ 50
FreeBSD 4.3	2000	336
FreeBSD 12.3	2021	576
OpenBSD 2.8	2000	268
OpenBSD 7.0	2021	331
Linux 2.4	2000	116
Linux 5.16.17	2022	398
Windows 2000	2000	248 + 640
Windows 10	2020	417 + 1314

Sumário

Estruturas de SO

- Como o SO é organizado internamente
- Estruturas clássicas
  - monolítico
  - em camadas
  - micronúcleo
  - cliente-servidor
  - máquinas virtuais

Sistemas monolíticos

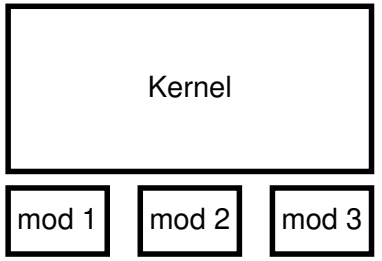


- Coleção de procedimentos
- Invocação livre
- Confiabilidade
- Desempenho

- Procedimentos de serviço implementam chamadas de sistema
- Módulos dinâmicos

Arquitetura monolítica com módulos dinâmicos

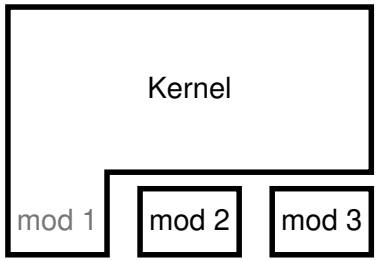
- **Módulos dinâmicos**: componentes que podem ser carregados e descarregados dinamicamente para dentro de um kernel monolítico
  - ajustes em tabelas de ponteiros



kernel monolítico com 3 módulos

Arquitetura monolítica com módulos dinâmicos

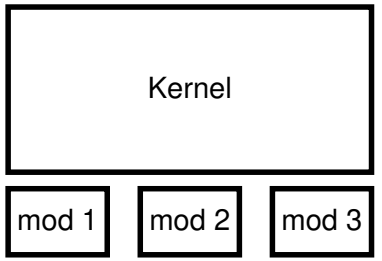
- **Módulos dinâmicos**: componentes que podem ser carregados e descarregados dinamicamente para dentro de um kernel monolítico
  - ajustes em tabelas de ponteiros



kernel monolítico após a carga do módulo 1

Arquitetura monolítica com módulos dinâmicos

- **Módulos dinâmicos**: componentes que podem ser carregados e descarregados dinamicamente para dentro de um kernel monolítico
  - ajustes em tabelas de ponteiros



kernel monolítico após a descarga do módulo 1

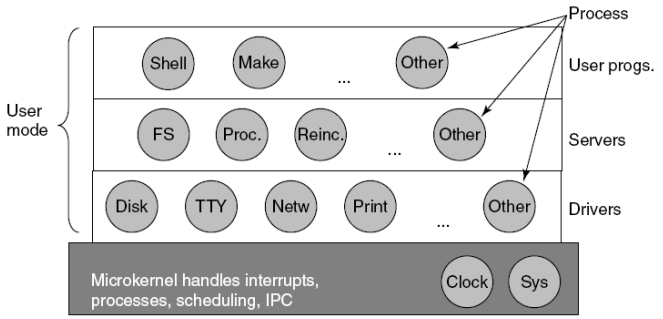
Sistemas em camadas

Camada	Função
5	O operador
4	Programas do usuário
3	Gerenciamento de entrada/saída
2	Comunicação operador-processo
1	Gerenciamento da memória e do tambor magnético
0	Alocação de processador e multiprogramação

Estrutura do sistema operacional THE

- Interfaces bem definidas
- Cada camada usa os serviços da camada inferior
- Nem sempre suportada pelo hardware
- Exemplos: THE, MULTICS
- Em SOs atuais é comum haver subsistemas em camadas
  - exemplos: armazenamento, rede

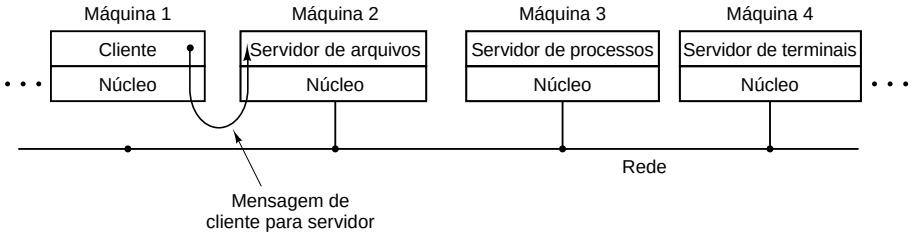
Micronúcleo



Arquitetura do MINIX 3

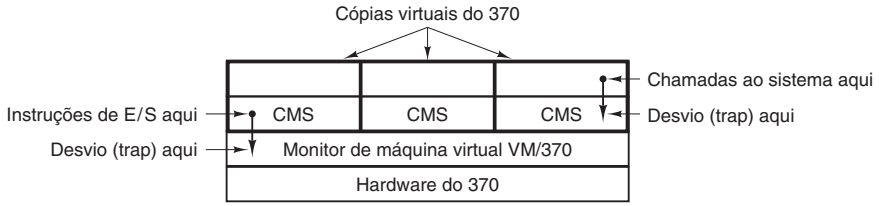
- Funcionalidades do núcleo particionadas
  - micronúcleo (*microkernel*) em modo supervisor
  - drivers e servidores em modo usuário
- Comunicação por troca de mensagens
- Algumas funções exigem modo núcleo
- Confiabilidade vs desempenho

Cliente-servidor



- **Clientes** enviam requisições a **servidores**
  - comunicação por troca de mensagens
- Pode ser usado com sistemas centralizados ou distribuídos
  - transparência de falhas é diferente

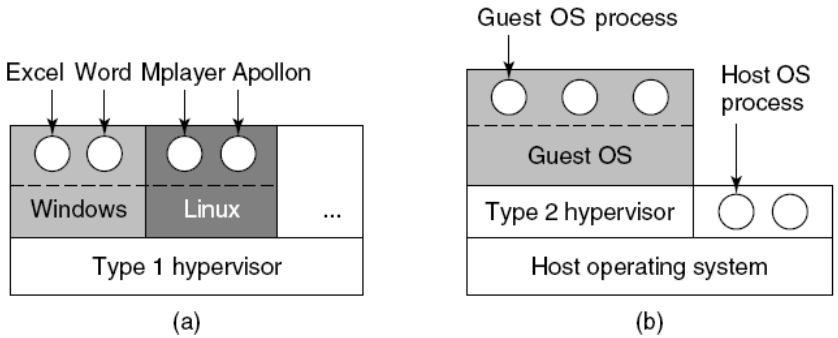
Máquinas virtuais



Estrutura do VM/370 com o CMS

- O MMV/hipervisor fornece uma abstração do hardware para as MVs
- Isolamento entre as MVs
- Diferentes sistemas operacionais nas MVs
- Desempenho depende do suporte do HW
- Consolidação de servidores

Hipervisores tipo 1 e tipo 2



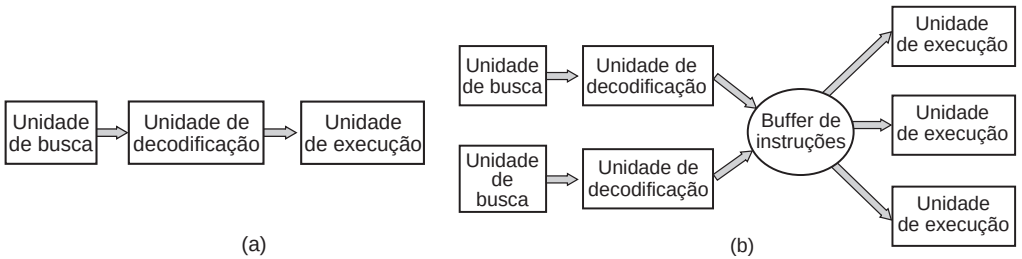
(a) Tipo 1: executa direto sobre o HW
- Xen, VMware ESX/ESXi, IBM z/VM

(b) Tipo 2: executa em um SO hospedeiro → MMV é um processo
- VirtualBox, VMware Workstation

Sumário

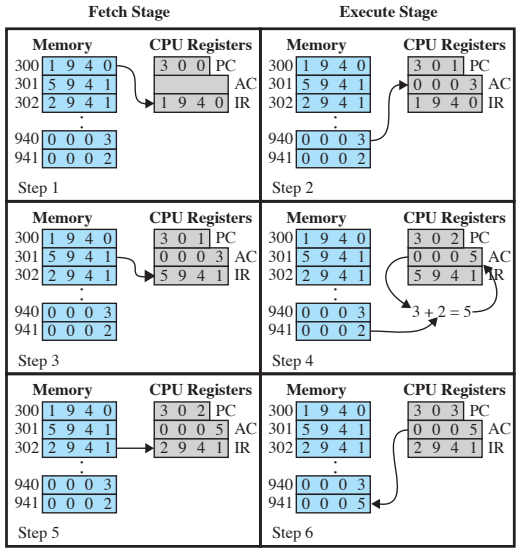
Processador: organização básica

- Ciclo busca-decodifica-executa
- Registradores
  - propósito geral
  - contador de programa / ponteiro de instrução
  - ponteiro de pilha
  - PSW (*program status word*) / flags



(a) Um pipeline de 3 estágios (b) Uma CPU superescalar

Exemplo de busca-decodificação-execução



Formato de instrução *Onnn* *O*: opcode *nnn*: endereço de memória

Opcodes 1: AC ← mem 2: mem ← AC 5: AC ← AC+mem

Registradores na arquitetura MIPS

- Registradores de propósito geral (ou nem tanto)

mnemônico	número	uso
`$zero`	`$0`	constante zero
`$at`	`$1`	reservado para o montador
`$v0`, `$v1`	`$2`, `$3`	valor de retorno de subrotina
`$a0`–`$a3`	`$4`–`$7`	argumentos para subrotina
`$t0`–`$t7`	`$8`–`$15`	temporários
`$s0`–`$s7`	`$16`–`$23`	registradores salvos
`$t8`, `$t9`	`$24`, `$25`	temporários
`$k0`, `$k1`	`$26`, `$27`	reservados para o SO
`$gp`	`$28`	ponteiro global
`$sp`	`$29`	ponteiro de pilha
`$fp`	`$30`	ponteiro de frame
`$ra`	`$31`	endereço de retorno

- Registradores de controle
  - coprocessador 0 (`CP0`)

Busca e execução no MIPS

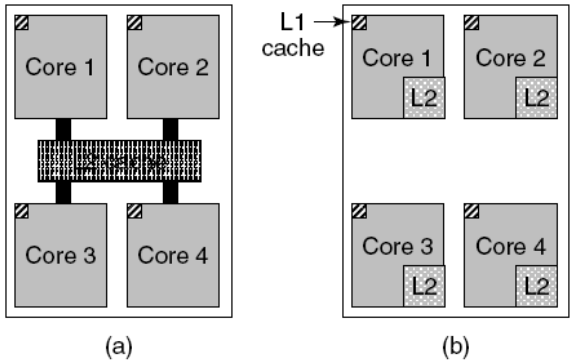
- CPU busca instrução apontada por contador de programa (PC)
  - PC é incrementado automaticamente em 4 bytes
- PC é modificado por desvios e chamadas de sub-rotinas (`jxx`/`bxx`)
  - não existe um registrador que permita manipular o PC diretamente

Chips multithread e multicore (1)

- Durante muitos anos, o desempenho dos processadores foi melhorado basicamente por
  - aumento na densidade de transistores → mais portas por chip
  - aumento na frequência de operação → clock mais rápido
  - exploração do paralelismo de instruções → pipelining, superescalar
- Os ganhos obtidos com essas estratégias são cada vez menores
  - solução: aumento do paralelismo arquitetural
- **Chips multithread**: replicam parte da lógica de controle, permitindo chaveamento rápido (ordem de ns) entre threads quando uma delas precisa acessar dados fora da cache
  - *HyperThreading* da Intel

Chips multithread e multicore (2)

- **Chips multicore**: CPUs independentes



(a) chip quad-core com cache L2 compartilhada (Intel)

(b) chip quad-core com caches L2 separadas (AMD)

Processador: modos de operação

- Modos de operação
  - modo núcleo (*kernel* ou supervisor)
  - modo usuário
- Chaveamento entre os modos
  - *trap*: usuário → núcleo
    - ★ chamadas de sistema (software)
    - ★ traps de hardware: exceções
  - instrução: núcleo → usuário

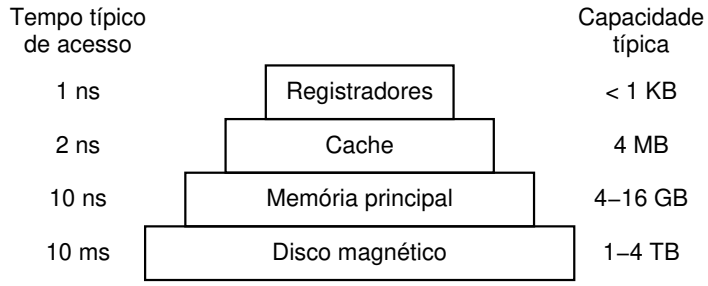
Processador: modos de operação no MIPS

- MIPS tem 3 níveis de privilégio, controlados por 2 bits (`KSU`) no registrador de status (`SR`, *status register*) do `CP0`
  - `KSU` = 0: usuário
  - `KSU` = 1: supervisor [não usado]
  - `KSU` = 2: kernel
- Chaveamento
  - usuário → kernel: `syscall`
  - kernel → usuário: `eret`

Organização da memória principal

- A memória é um vetor de N palavras de B bits
  - do ponto de vista do programador, um vetor de bytes
- A memória armazena dados e instruções
  - arquitetura de von Neumann
  - **A INTERPRETAÇÃO É FEITA PELO SOFTWARE**

Hierarquia de memória

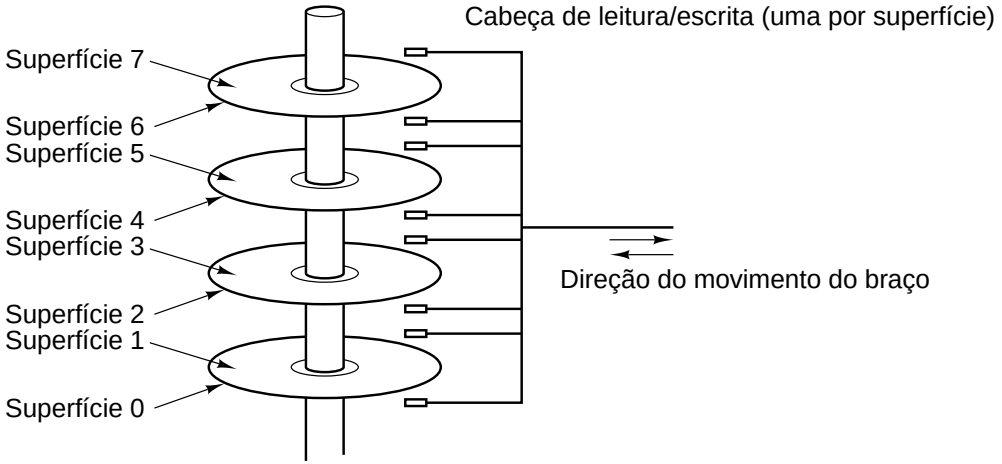


números mostrados são apenas aproximações

Hierarquia de caches

- Sistemas atuais têm geralmente 2 ou 3 níveis de cache
  - níveis sucessivos têm maior capacidade e maior latência
- L1: 8–64 KB
  - tipicamente separados para instruções e dados
- L2: 256 KB–8 MB
  - em chips multicore, pode ser compartilhado ou local
    - ★ cache compartilhado: complica controlador, simplifica coerência
    - ★ cache local: simplifica controlador, complica coerência
- L3: até 16 MB
  - geralmente compartilhado

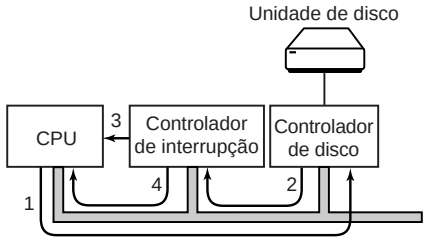
Estrutura de uma unidade de disco



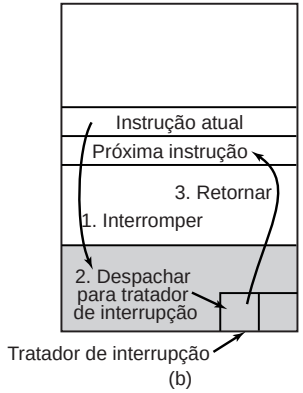
Dispositivos de E/S (1/2)

- Controladores vs dispositivos
  - registradores de dados, controle, status
- Drivers de dispositivo
  - geralmente fornecidos pelo fabricante
  - executam em modo núcleo para ter acesso ao dispositivo
- Modos de operação
  - E/S programada
  - interrupções
  - DMA (*Direct Memory Access*)

Dispositivos de E/S (2/2)



(a)

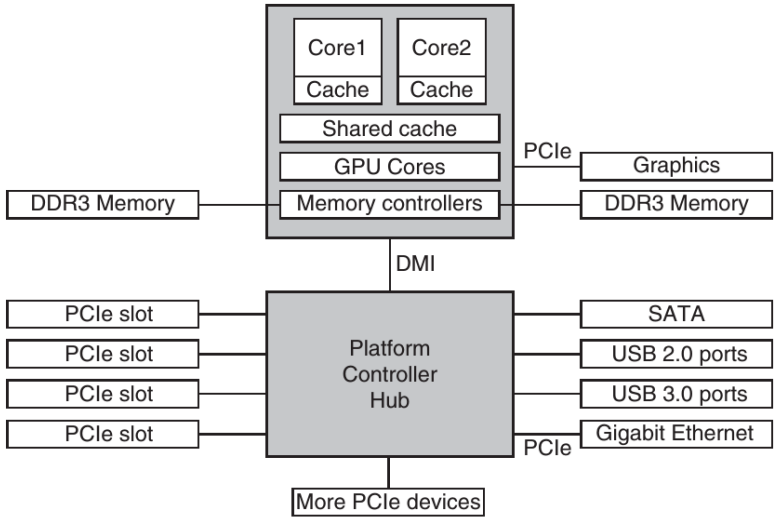


(b)

(a) os passos para iniciar um dispositivo de E/S e obter uma interrupção

(b) o processamento de uma interrupção

Estrutura de um sistema x86 atual



Bibliografia Básica

- Andrew S. Tanenbaum e Herbert Bos.
  Sistemas Operacionais Modernos, 4ª Edição. Capítulo 1.
  Pearson Prentice Hall, 2016.
- Carlos A. Maziero.
  Sistemas Operacionais: Conceitos e Mecanismos. Capítulos 1–3.
  Editora da UFPR, 2019.
  http://wiki.inf.ufpr.br/maziero/doku.php?id=socm:start
- William Stallings.
  Operating Systems: Internals and Design Principles, 6th Ed.
  Capítulos 1 e 2.
  Pearson Prentice Hall, 2009.