Sistemas Operacionais: Fundamentos de Sistemas Operacionais

Prof. Maurício Aronne Pillon
Prof. Rafael R. Obelheiro

UDESC/CCT – Departamento de Ciência da Computação
{mauricio.pillon,rafael.obelheiro}@udesc.br

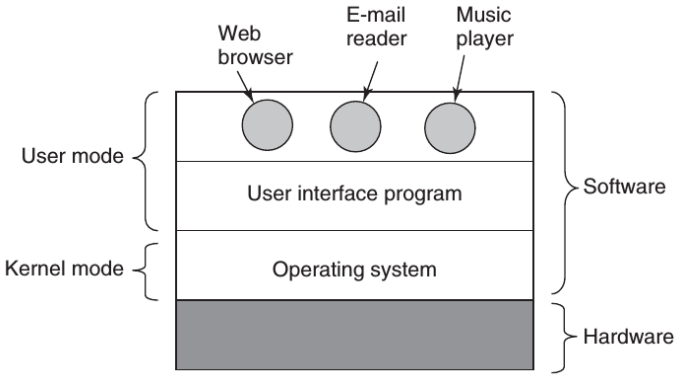
Joinville, fevereiro de 2017

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
     - esconde os detalhes complicados do funcionamento do hardware
     - oferece uma interface mais amigável para as aplicações
       $\Rightarrow$ *máquina virtual*
  2. Um gerenciador de recursos
     - cada programa utiliza o recurso durante um tempo
     - cada programa ocupa um certo espaço no recurso

Camadas de um SO



Sumário

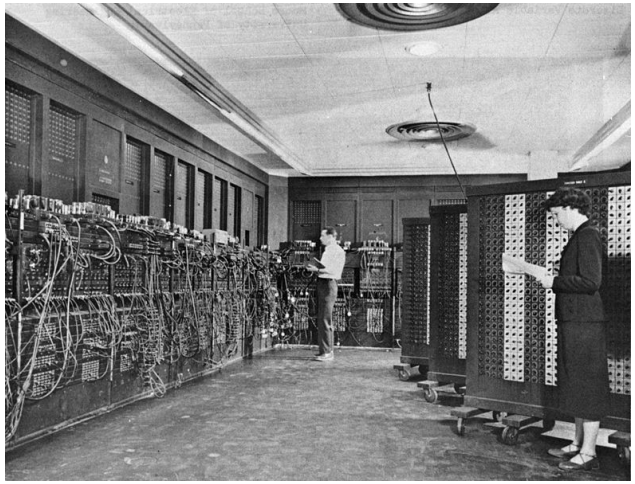
História dos Sistemas Operacionais

1. Primeira geração (1945–1955)
   - válvulas, painéis de programação
2. Segunda geração (1955–1965)
   - transistores, sistemas em lote
3. Terceira geração (1965–1980)
   - CIs e multiprogramação
4. Quarta geração (1980–dias de hoje)
   - computadores pessoais
5. Quinta geração (1990–dias de hoje)
   - dispositivos móveis

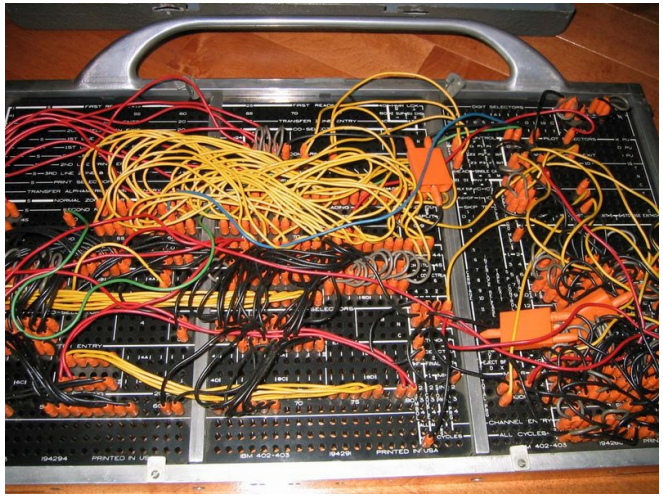
Primeira geração (1945–1955)

- Howard Aiken (Harvard), John von Neumann (Princeton), J. Presper Eckert e William Mauchley (Pensilvânia) e Konrad Zuse (Alemanha) construiram máquinas de calcular
- Sistema totalmente mecânico (relés lentos), com ciclos medidos em segundos
- Primeira evolução: substituição dos relés por válvulas
- O mesmo grupo de pessoas projetava, construía, programava, operava e realizava a manutenção de cada máquina
- Programação feita com código absoluto e muitas vezes conectando plugs em painéis
- Os sistemas operacionais ainda não tinham sido inventados

ENIAC



Quadro de plugues do IBM 402

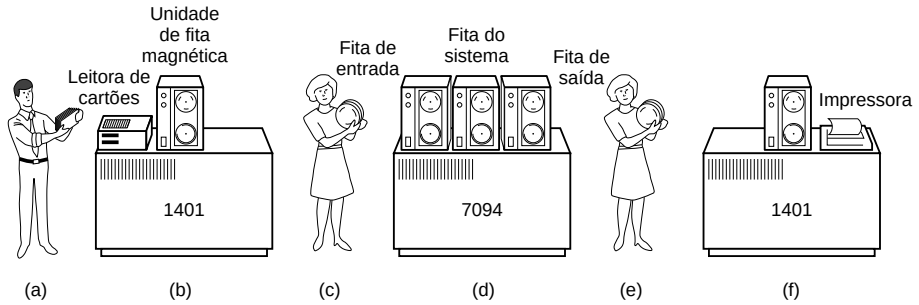


Segunda geração (1955–1965)

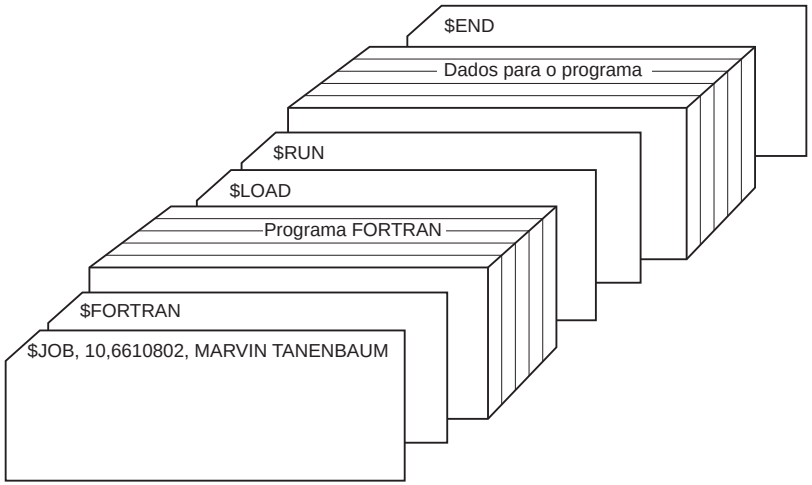
- Introdução do transistor (substituindo as válvulas)
- Computadores mais confiáveis
- Passaram a ser fabricados e comercializados com uma expectativa de "vida útil" mais longa
- Estabeleceu-se uma separação clara entre as funções: projetista, fabricantes, programadores e técnicos de manutenção
- Estas máquinas passaram a ser denominadas de **computadores de grande porte (mainframes)**
- Devido ao alto custo do equipamento, buscou-se uma maneira de reduzir o desperdício, adotando-se o chamado **sistema em lote (batch)**

Sistema em lote (batch)

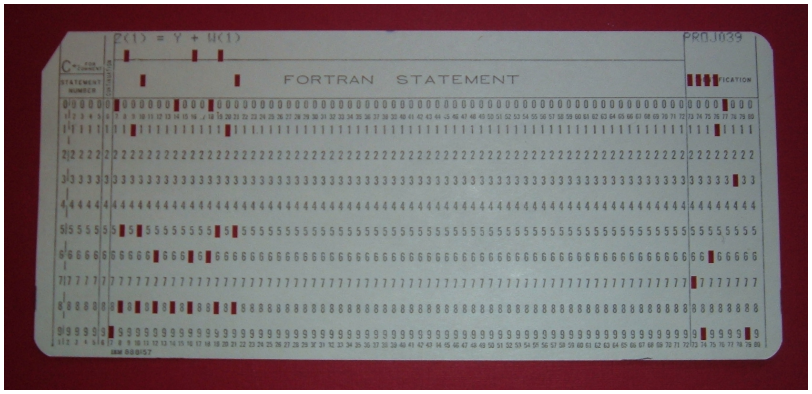
(a) os programadores levavam os cartões para o 1401
(b) o 1401 gravava o lote de jobs em fita
(c) o operador levava a fita de entrada para o 7094
(d) o 7094 executava o processamento
(e) o 1401 imprimia as saídas



Estrutura de um job típico



Cartão Fortran



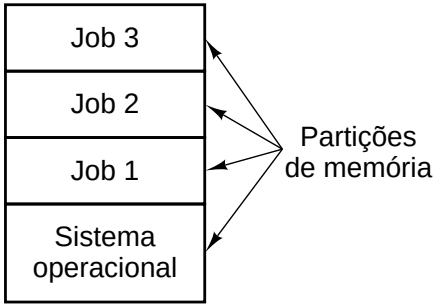

```
Z(1) = Y + W(1)
```


3ª geração: famílias de computadores

- Os computadores de 2ª geração eram radicalmente diferentes entre si, tornando-os incompatíveis:
  - IBM 7094: cálculo intensivo, orientado a palavra
  - IBM 1401: processamento comercial, orientado a caracter
- As diferenças representavam:
  - custos significativos para o fabricante
  - custos **e** inconveniências para os usuários que precisavam migrar de plataforma
- A IBM introduziu a família 360
  - primeiros computadores que usavam CIs (Circuitos Integrados)
  - compatibilidade de software e hardware
- Os requisitos e características diferentes se tornaram um pesadelo para a equipe de desenvolvimento do OS/360
  - milhões de linhas de Assembly, milhares de programadores
  - quando bugs eram corrigidos, outros eram introduzidos
  - *The Mythical Man-Month* (Fred Brooks)

Sistema multiprogramado (3ª geração)

- No 7094, quando um job parava para aguardar uma E/S, a CPU simplesmente permanecia ociosa
- Em processamento comercial (*I/O Bound*), o tempo de espera por E/S chegava a 80 a 90%
- 3 jobs na memória



- Deu origem a sistemas de tempo compartilhado (*time sharing*)

MULTICS

- *MULTiplexed Information and Computing Service*
- Um "computador utilitário", capaz de atender a centenas de usuários simultâneos
  - CPU equivalente a um 386, maior capacidade de E/S
  - nuvens computacionais resgataram o conceito de computação utilitária
- O projeto foi um fracasso comercial (menos de 100 instalações), mas teve usuários fiéis (até década de 90)
- Extremamente influente do ponto de vista técnico
  - lançou diversas idéias usadas até hoje
  - em alguns aspectos ainda superior
- Diversos nomes importantes da computação nos últimos 40 anos fizeram parte do projeto

UNIX

- Ken Thompson (Bell Labs) e o PDP-7 (origem do UNIX)
- Crescimento comercial dos minicomputadores, culminando no PDP-11, sucesso comercial
- BSD: memória virtual e TCP/IP
- Problemas de incompatibilidade sobre as versões UNIX levou o IEEE a lançar um padrão de desenvolvimento, POSIX (*Portable Operating System-IX*)
- Sistema feito por programadores, para programadores
- Características principais:
  - ferramentas de desenvolvimento cooperativo
  - interfaces simples, elegantes, consistentes e sem "frescuras"
  - conceito de ferramentas de software: pequenos programas que desempenham funções específicas e que podem ser combinados

Thompson, Ritchie e um PDP-11 (ca. 1972)

Computadores Pessoais (microcomputadores)

- Os microcomputadores não eram muito diferentes do PDP-11, porém eram muito mais baratos
- Desenvolvimento de CIs em larga escala (*large scale integration – LSI)*, que são chips contendo milhares de transistores em um centímetro quadrado de silício
- Em 1974, a Intel lançou o 8080, primeira CPU de 8 bits de propósito geral
- Em 1977, a Digital Research (Gary Kildall) reescreveu o CP/M (*Control Program for Microcomputers*) para rodar em outras CPUs além do 8080
- Nos anos 80, a IBM solicitou a um desconhecido desenvolvedor de interpretador Basic, Bill Gates, o contato de uma empresa que pudesse desenvolver um SO para o IBM PC
- Bill Gates comprou o DOS (*Disk Operating System*) e vendeu à IBM o DOS/Basic. A Microsoft ainda contratou o desenvolvedor do DOS, Tim Paterson, lançando o conhecido MS-DOS

Computadores Pessoais (microcomputadores)

- CP/M e MS-DOS, entre outros, eram todos baseados na digitação de comandos
- Doug Engelbart (Stanford) inventou uma interface gráfica voltada para o usuário (GUI – *Graphical User Interface*)
- O Apple Macintosh foi o primeiro SO a incorporar esta idéia
- Devido ao sucesso comercial da Apple, a Microsoft também resolveu incorporar esta idéia
- Os computadores tornaram-se cada vez mais poderosos
  - em vez das pessoas esperarem pelo computador, o computador é que espera por elas
  - necessidade de SOs mais sofisticados
- Surgimento de redes de computadores, com sistemas operacionais de rede e sistemas operacionais distribuídos

Dispositivos móveis

- Primeiro vieram PDAs, depois smartphones e tablets
  - alguns dispositivos eram PCs de tamanho reduzido
- Inicialmente SOs embarcados
  - PalmOS, BlackBerry OS, Symbian
- Com a evolução do hardware surgiram versões mobile de SOs para desktop
  - iOS (Mac OS X), Android (Linux), Windows Mobile/Phone

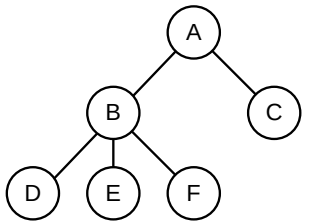
Tipos de sistemas operacionais

- SOs para mainframes
- SOs para servidores
- SOs para multiprocessadores
- SOs para computadores pessoais
- SOs para computadores portáteis (dispositivos móveis)
- SOs de tempo real
- SOs embarcados
- SOs para smartcards

Sumário

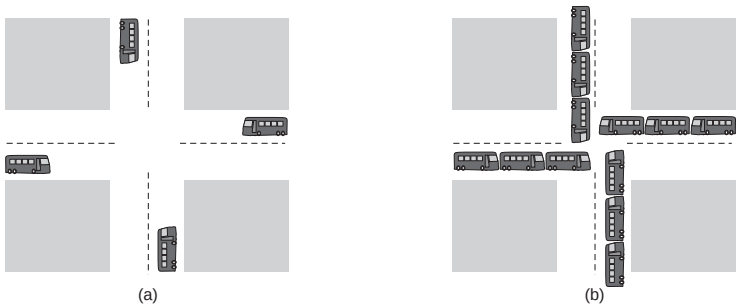
Processos

- 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)
- Árvores de processos



Deadlocks

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

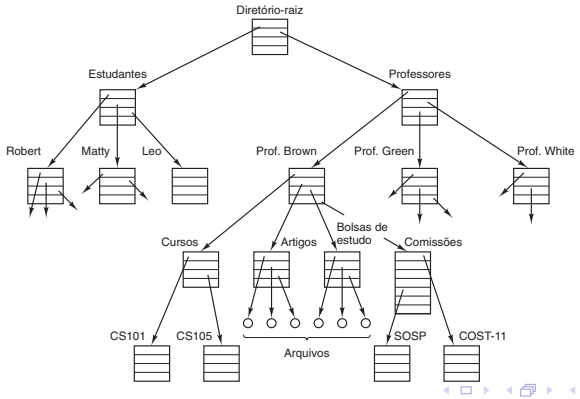


Gerência de memória

- 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

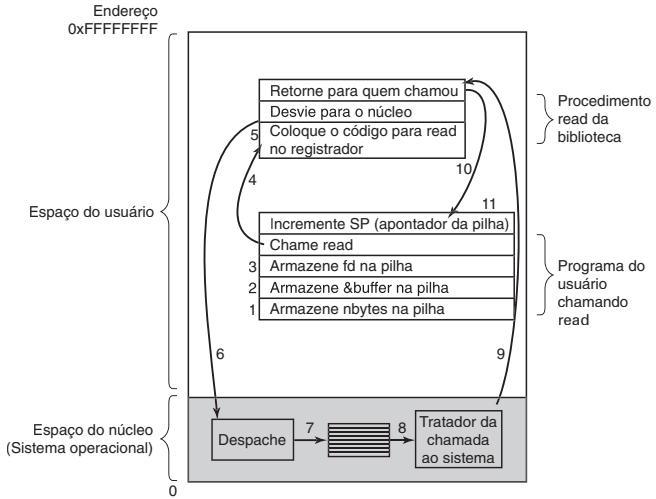
- Define uma interface mais refinada para armazenamento persistente de informações
- A maior parte dos sistemas usa o conceito de arquivos organizados em hierarquias de diretórios



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 POSIX

Gerenciamento de processos

Chamada	Descrição
pid = fork( )	Crie um processo filho idêntico ao processo pai
pid = waitpid(pid, &statloc, options)	Aguarde um processo filho terminar
s = execve(name, argv, environp)	Substitua o espaço de endereçamento do processo
exit(status)	Termine a execução do processo e retorne o estado

Gerenciamento de arquivos

Chamada	Descrição
fd = open(file, how, ...)	Abra um arquivo para leitura, escrita ou ambas
s = close(fd)	Feche um arquivo aberto
n = read(fd, buffer, nbytes)	Leia dados de um arquivo para um buffer
n = write(fd, buffer, nbytes)	Escreva dados de um buffer para um arquivo
position = lseek(fd, offset, whence)	Mova o ponteiro de posição do arquivo
s = stat(name, &buf)	Obtenha a informação de estado do arquivo

Gerenciamento do sistema de diretório e arquivo

Chamada	Descrição
s = mkdir(name, mode)	Crie um novo diretório
s = rmdir(name)	Remova um diretório vazio
s = link(name1, name2)	Crie uma nova entrada, name2, apontando para name1
s = unlink(name)	Remova uma entrada de diretório
s = mount(special, name, flag)	Monte um sistema de arquivo
s = umount(special)	Desmonte um sistema de arquivo

Diversas

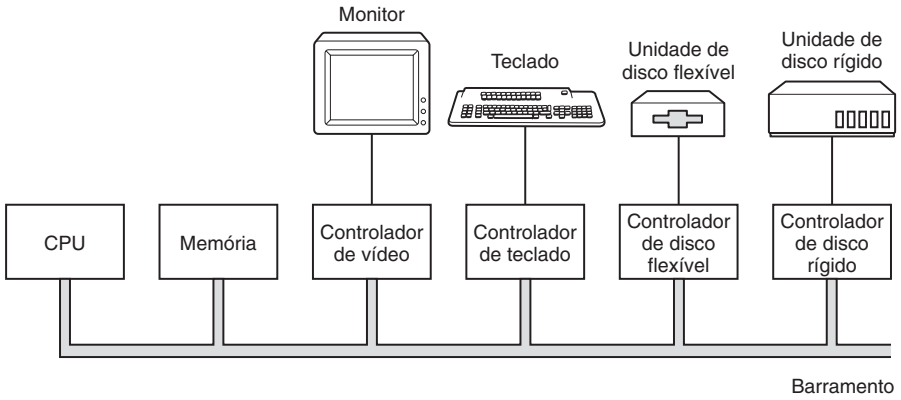
Chamada	Descrição
s = chdir(dirname)	Altere o diretório de trabalho
s = chmod(name, mode)	Altere os bits de proteção do arquivo
s = kill(pid, signal)	Envie um sinal a um processo

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

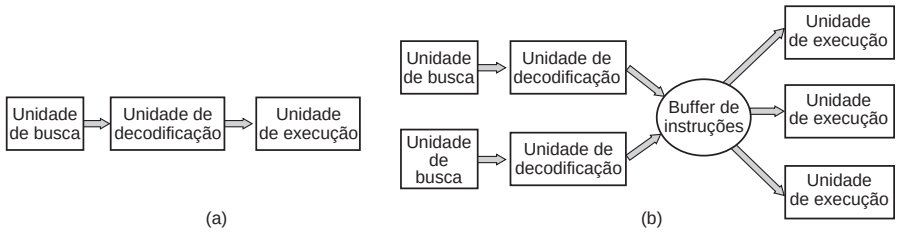
Sumário

Componentes de um computador



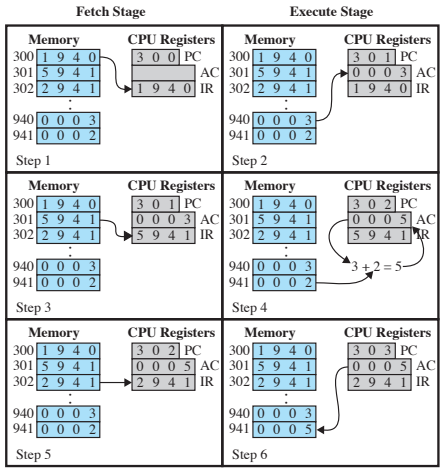
Processador: organização básica

- 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 três estágios (b) Uma CPU superescalar

Exemplo de busca-decodificação-execução

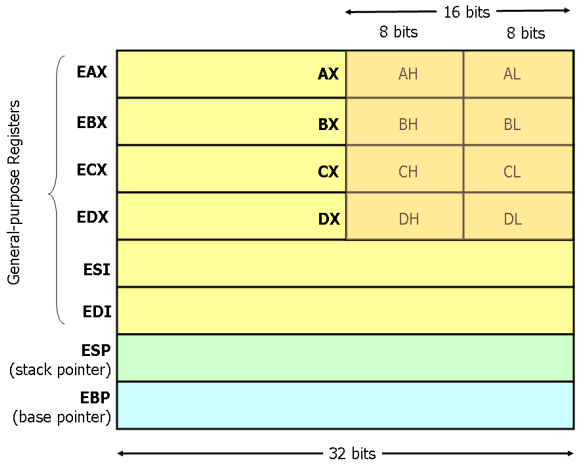


Opcodes 1: AC ← mem 2: mem ← AC 5: AC ← AC+mem

Registradores na arquitetura IA-32 (x86)

- Registradores de propósito geral (ou nem tanto)
  - EAX: acumulador para operandos e resultados
  - EBX: ponteiro para dados no segmento DS
  - ECX: contador para laços e operações com strings
  - EDX: ponteiro de E/S
  - ESI: ponteiro de origem para operações com strings
  - EDI: ponteiro de destino para operações com strings
  - EBP: ponteiro para a base da pilha atual (no segmento SS)
  - ESP: ponteiro para o topo da pilha atual (no segmento SS)
- Registradores de segmento
  - CS: segmento de código
  - SS: segmento de pilha
  - DS, ES, FS, GS: segmentos de dados
- Registradores de status e controle
  - EIP: ponteiro de instrução
  - EFLAGS: flags indicando o resultado de operações lógicas e aritméticas
    - zero, vai um (*carry*), overflow, paridade, sinal

Registradores de propósito geral no x86



Busca e execução no x86

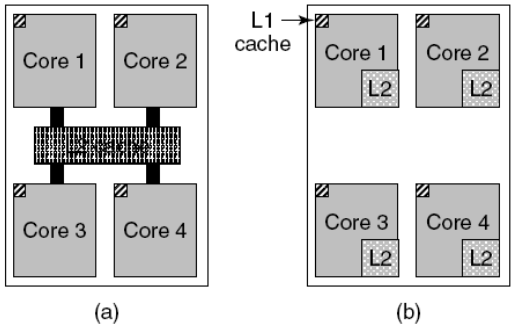
- CPU busca instrução apontada por EIP
  - EIP é incrementado automaticamente
    - instruções x86 têm tamanhos diferentes
- EIP é modificado por desvios (JMP/Jxx) e chamadas de sub-rotinas (CALL/RET)
  - não é possível manipular EIP diretamente

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)

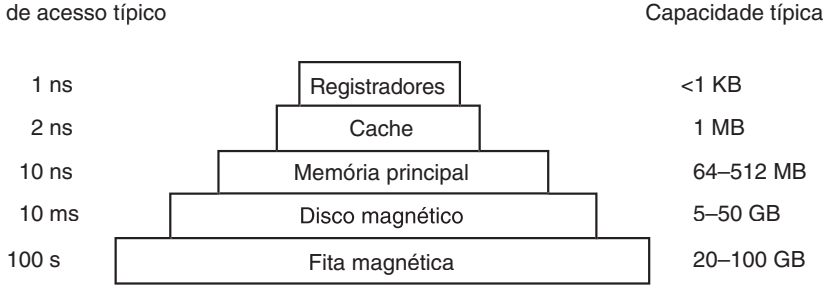
Processador: modos de operação

- 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

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

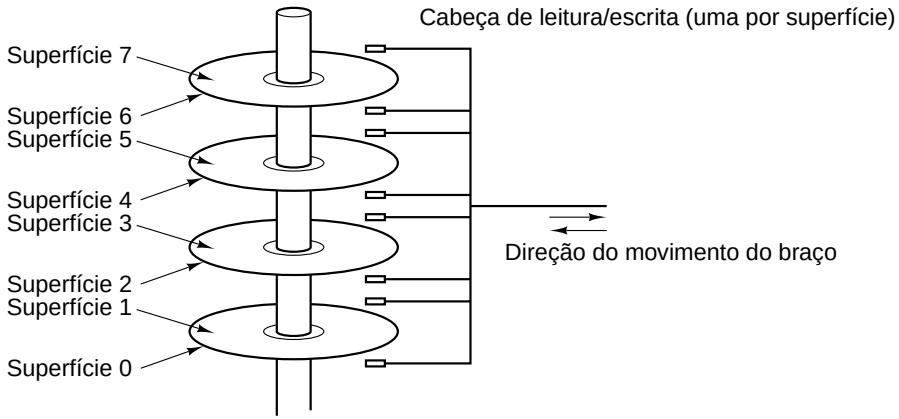


números mostrados são apenas aproximações

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

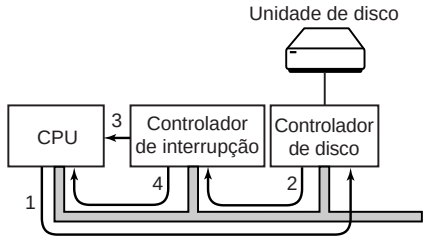
Estrutura de uma unidade de disco



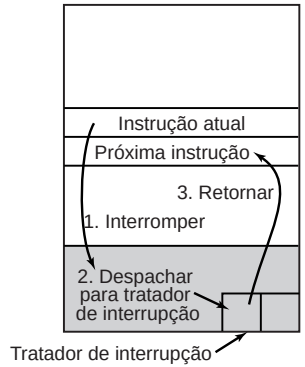
Dispositivos de E/S (1/2)

- 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)



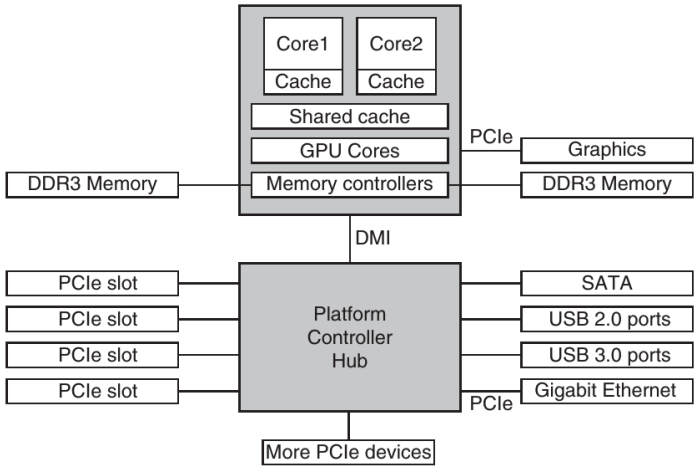
(a)



(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

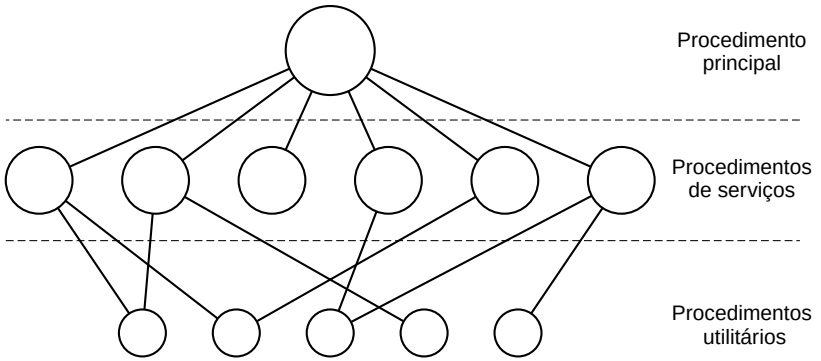


Sumário

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

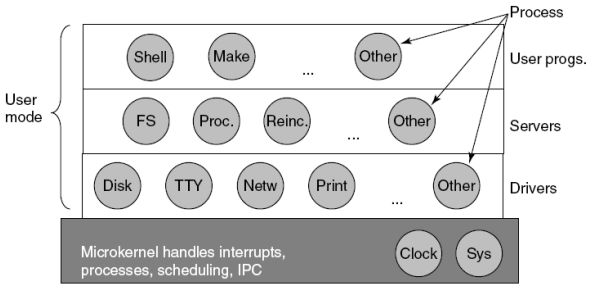
Sistemas em camadas

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

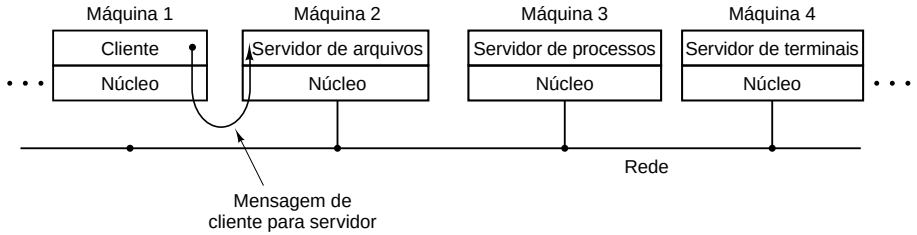
Micronúcleo



Arquitetura do MINIX 3

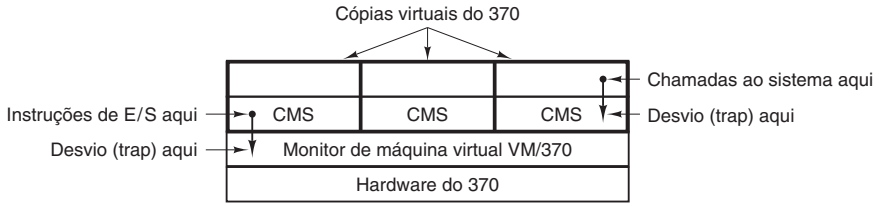
- Funcionalidades do núcleo particionadas
  - micronúcleo (*microkernel*) em modo supervisor
  - drivers e servidores em modo usuário
- Comunicação por troca de mensagens
- Algumas funções exigem modo núcleo
- Confiabilidade vs desempenho

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

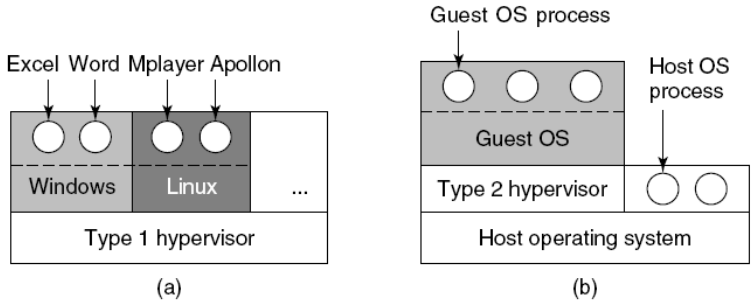
Máquinas virtuais



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

Bibliografia Básica

Andrew S. Tanenbaum e Herbert Bos.
Sistemas Operacionais Modernos, 4ª Edição. Capítulo 1.
Pearson, 2016.

William Stallings.
Operating Systems: Internals and Design Principles, 6th Ed.
Capítulos 1 e 2.
Pearson Prentice Hall, 2009.

Abraham Silberchatz, Greg Gagne e Peter Baer Galvin.
Fundamentos de Sistemas Operacionais, 6ª Edição.
LTC – Livros Técnicos e Científicos Editora, 2004.