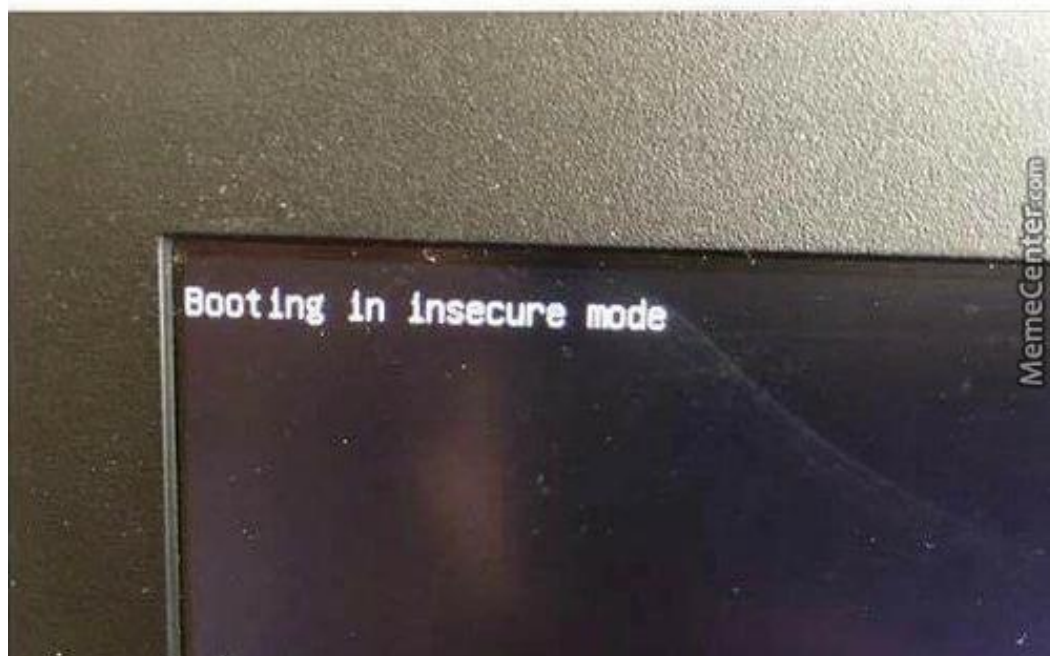


SISTEMAS OPERACIONAIS 1

21270 A

when you wake up in the morning



Apresentação baseada nos slides
do Prof. Dr. Antônio Carlos Sementille e Prof.
Kalinka C. Branco e nas transparências
fornecidas no site de compra do livro
“Sistemas Operacionais Modernos”

Componente básicos

- Componentes básicos de hardware:
 - CPU;
 - Memória;
 - Controladoras;
 - Dispositivos de Entrada/Saída e Unidades de armazenamento.

CPU (*Central Processing Unit*)

- É o “cérebro” do computador responsável por executar instruções;
- CPU busca instruções na memória, decodifica essas instruções e as executa até sua finalização;
- Durante a execução de instruções, a CPU utiliza-se de registradores para armazenar variáveis e resultados temporários;
- Instruções são executadas por ciclos de relógio;

CPU - registradores

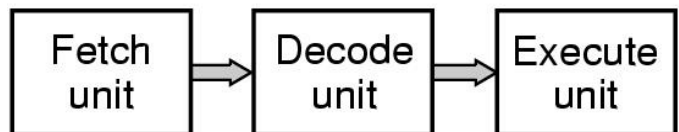
- **Registradores:**

- Contador de programa (*program counter - PC*): contém o endereço de memória da próxima instrução a ser lida e executada;
- Ponteiro da pilha (*stack pointer - SP*): aponta para o topo da pilha corrente na memória (estrutura para cada procedimento);
 - Informações que não são mantidas nos registradores:
 - Parâmetros de entrada;
 - Variáveis locais e temporárias;
- Registrador de instrução (*instruction register - IR*) = instrução que está sendo atualmente executada;
- PSW (*program status word*): bits de controle;

CPU – modos de execução

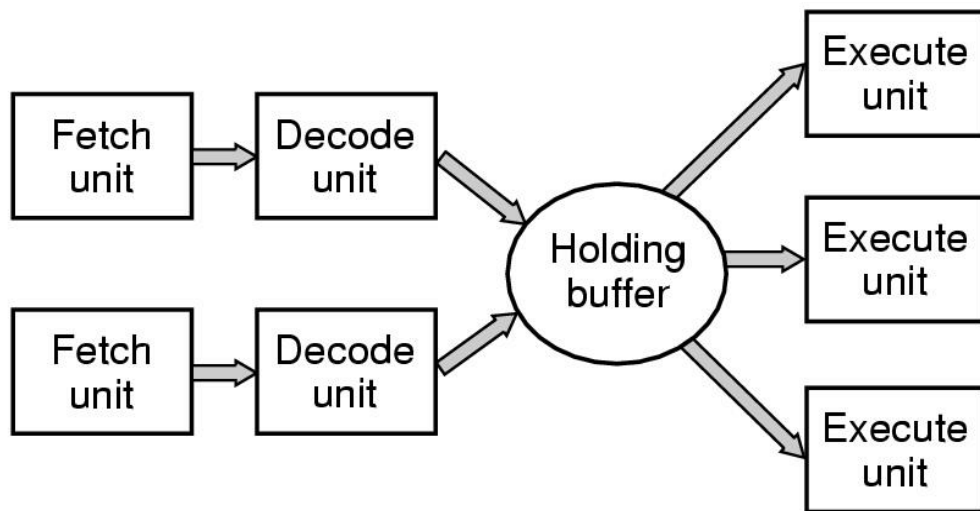
- As CPUs mais modernas, ao contrário das mais antigas, executam mais de uma instrução por ciclo de relógio:
 - *Pipeline*;
 - *Superscalar*;

CPU – modos de execução



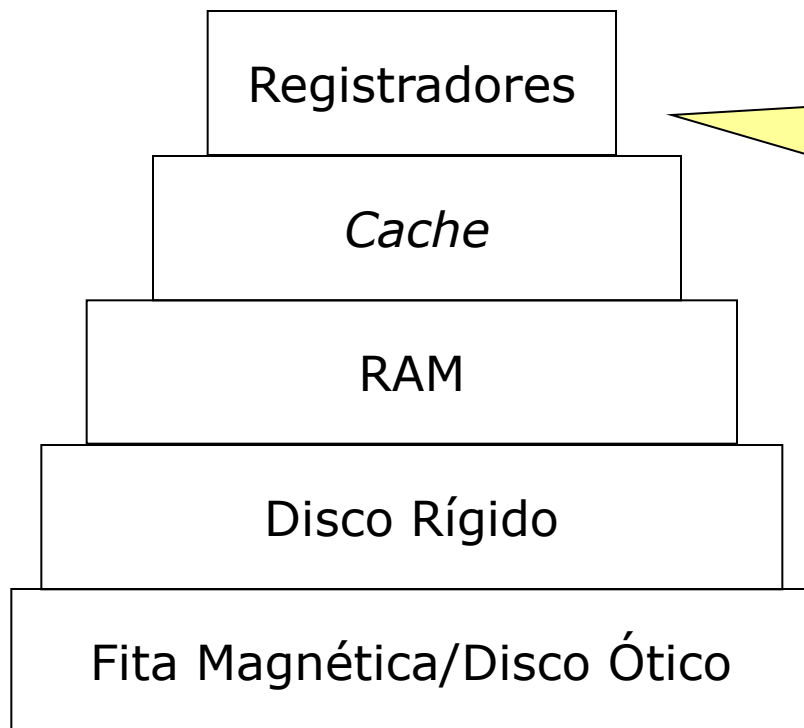
(a)

Pipeline

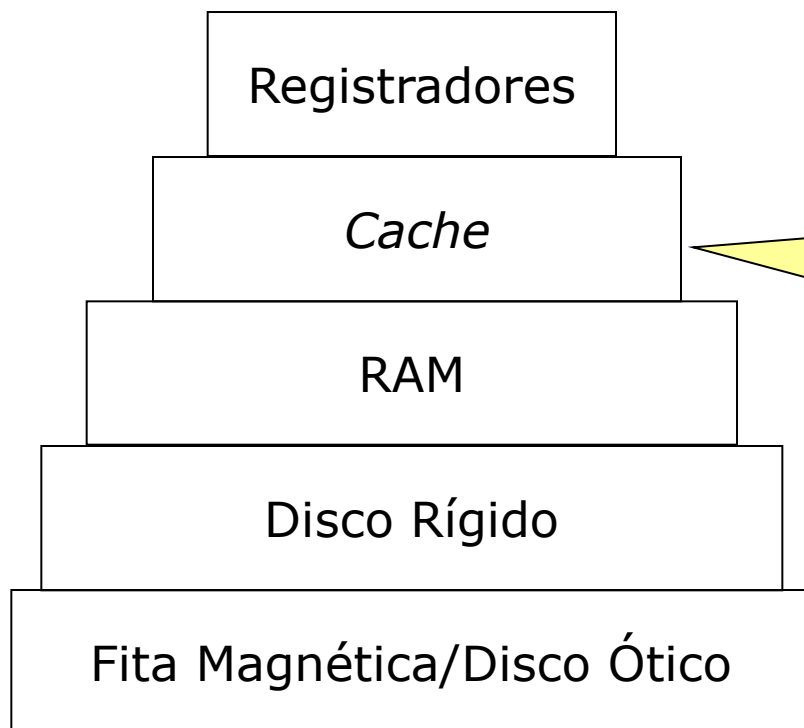


(b)

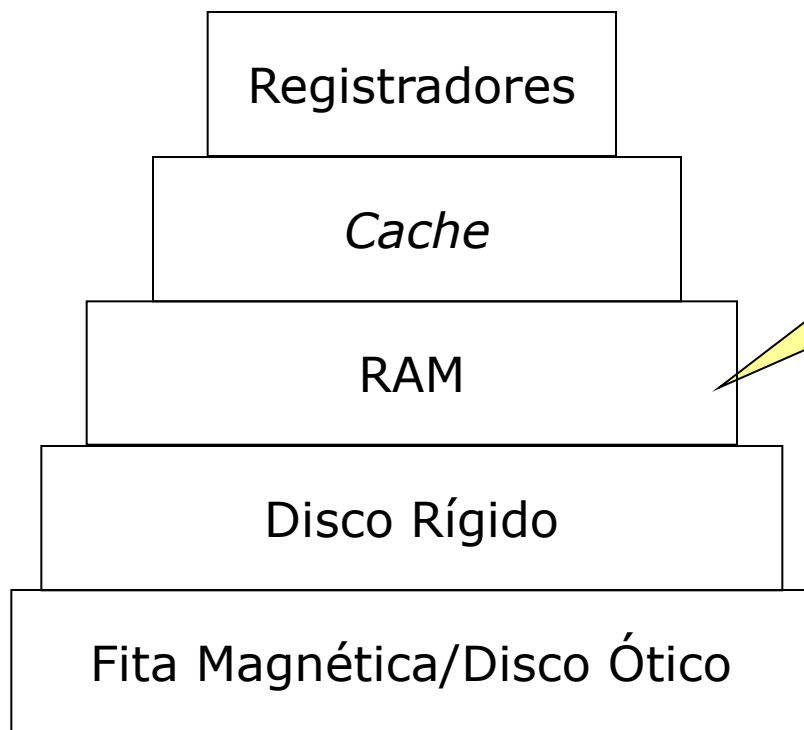
*Superscalar
ou Superpipeline*



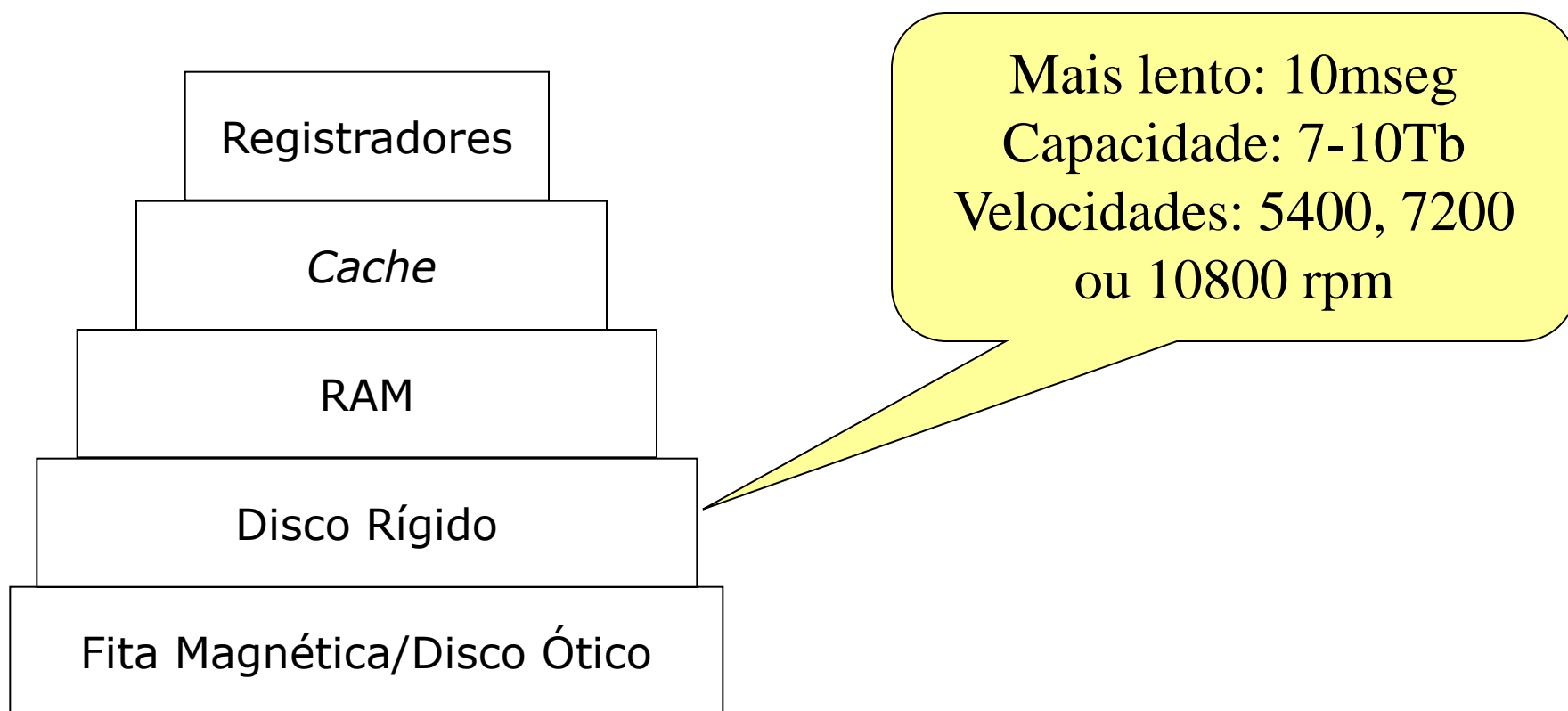
Rápidos: 1nseg.
Componentes internos à
CPU
Capacidade: 32 ou 64 bits
Controlados por software

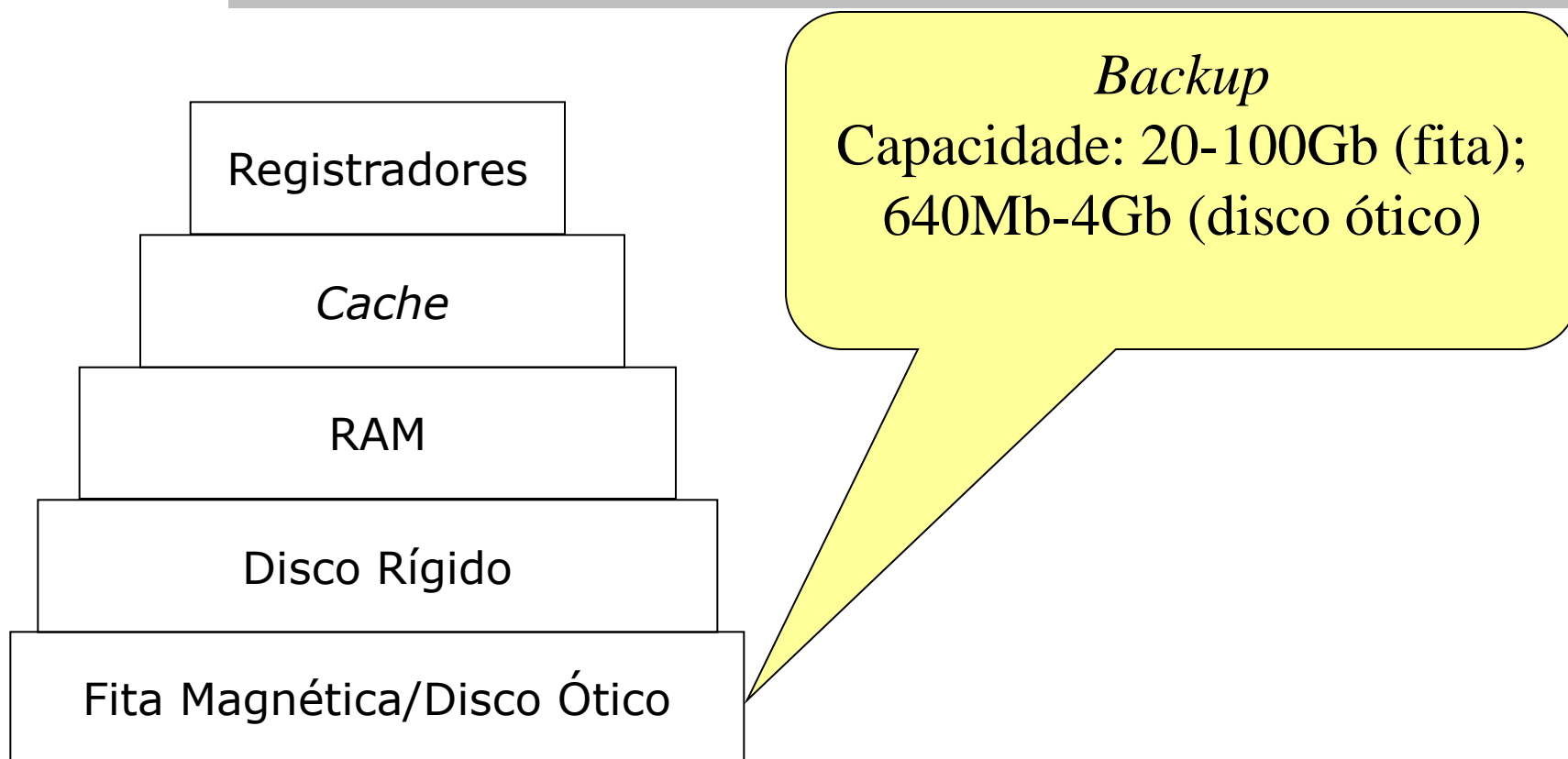


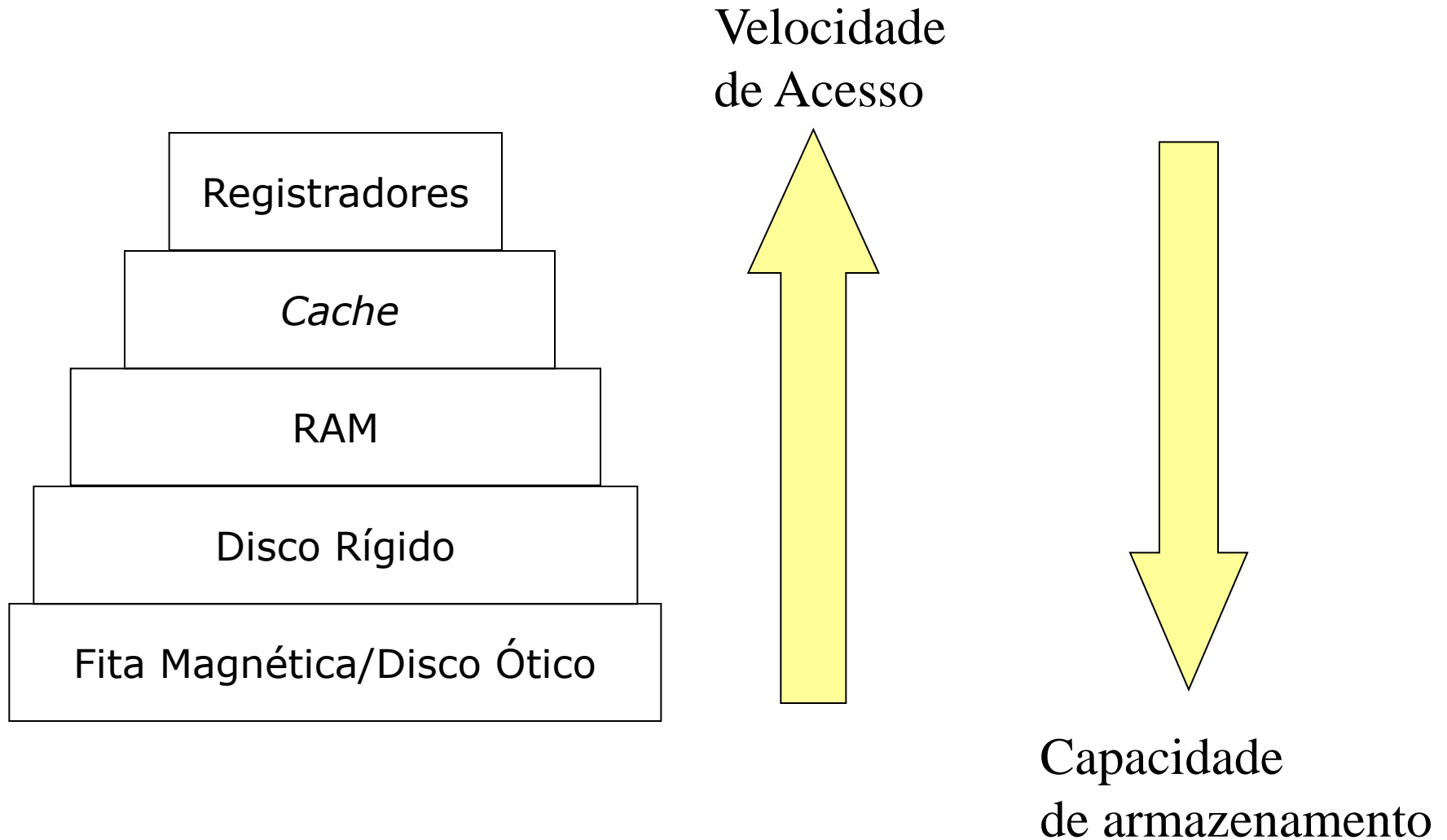
Rápida: 2nseg
Capacidade: 8 ou 16 kb;
128, 256, 512 Kb, 1Mb ou
2Mb
Controlada por hardware



Random Access Memory
Rápida: 10nseg
Capacidade: até gigabytes
Volátil







- ROM (*Read Only Memory*):
 - Programável;
 - Somente leitura;
 - Rápida (mais lenta que a RAM) e barata;
 - Não volátil;
 - Inicializa os circuitos da placa-mãe;
 - Programas armazenados na ROM da placa-mãe:
 - BIOS: configurações de hardware;
 - POST (*Power-on Self-Test*): auto teste;
 - *Setup*: altera configurações na CMOS;

- Variações:
 - *Flash ROM*:
 - Pode ser apagada e reescrita;
 - Todo conteúdo é apagado;
 - Utilizada em telefones celulares digitais, câmeras digitais, PC Cards;
 - *PROM (Programmable ROM)*;
 - *EPROM (Erasable PROM)*:
 - Pode ser apagada e reescrita;
 - Exposição à luz ultra violeta apaga o conteúdo;
 - *EEPROM (Electrically EPROM)*:
 - Pode ser apagada e reescrita;
 - Impulsos elétricos são utilizados para apagar o conteúdo;
 - É possível reescrever um único endereço;

- CMOS (*Complementary Metal Oxide Semiconductor*)
 - Volátil;
 - Utilizada para gravar hora/data correntes
 - Possui uma bateria;
 - Também grava parâmetros de configuração do sistema, como por exemplo qual é o disco de *boot*.

Dispositivos de E/S

- Interagem com o sistema operacional;
- Controladoras
 - *Chip* (conjunto de *chips*) conectado à placa mãe que fisicamente controla os dispositivos físicos aceitando comandos do SO;
 - Controle é feito por meio de uma interface: *driver*

Dispositivos de E/S

- Sistema Operacional manipula as controladoras (parte eletrônica dos dispositivos)
 - Comandos especiais são carregados nos registradores das controladoras;
 - Sinais elétricos acionam os dispositivos;

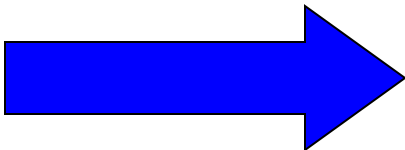
Dispositivos de E/S

- *Driver*
 - Diferentes controladoras e diferentes sistemas operacionais utilizam diferentes *drivers*;
 - Rodam em modo *kernel*;
 - Podem ser carregados dinamicamente – depende do sistema operacional e do dispositivo físico;

Dispositivos de E/S

- A cada dispositivo físico são atribuídos:
 - Uma interrupção;
 - Um endereço (em hexadecimal) de E/S;
- Atualmente, a interrupção e o endereço são atribuídos automaticamente pelo sistema operacional;

- BIOS (*Basic Input Output System*)
 - Presente na placa mãe;
 - Contém configurações de software de E/S de baixo nível (*Flash Rom*);
 - Seqüência Básica:
 - Checa memória RAM;
 - Checa teclado, monitor, mouse;
 - Checa barramentos para detectar outros dispositivos conectados;
 - Checa disco de *boot* na CMOS – lista de *boot*;
 - Lê o primeiro setor de *boot* na memória e o executa. Esse setor normalmente contém um programa que examina a tabela de partições para saber qual partição está ativa;
 - Checa informações de configuração;
 - Checa os *drivers* disponíveis;
 - SO é iniciado e carrega *drivers*.

- Sistemas de computadores modernos são compostos por diversos dispositivos:
 - Processadores;
 - Memória;
 - Controladoras;
 - Monitor;
 - Teclado;
 - Mouse;
 - Impressoras;
 - Etc...
- 
- Alta Complexidade**

Por que?

- Com tantos dispositivos, surge a necessidade de gerenciamento e manipulação desses diversos dispositivos;
- **Sistema Operacional**: Software responsável por gerenciar dispositivos que compõem um sistema computacional e realizar a interação entre o usuário e esses dispositivos;

Arquitetura do Sistema



Arquitetura do Sistema

- **Hardware:** Diversas camadas
 - Dispositivos físicos:
 - Circuitos (*chips*)
 - Cabos
 - Transistores
 - Capacitores
 - Memória
 - Disco rígido
 - etc...

Arquitetura do Sistema

- **Micro Arquitetura:** dispositivos físicos são agrupados para formar unidades funcionais
 - CPU – processamento;
 - ULA (Unidade Lógica Aritmética) – operações aritméticas. Essas operações podem ser controladas por software (micro programas) ou por circuitos de hardware;

Arquitetura do Sistema

- **Linguagem de Máquina:** conjunto de instruções interpretadas pelos dispositivos que compõem a micro arquitetura;
 - Possui entre 50 e 300 instruções;
 - Realiza operações por meio de registradores;
 - Baixo nível de abstração;
 - Ex.: **Assembler**.

Sistema Operacional

- Pode atuar de duas maneiras diferentes:
 - Como máquina estendida (*top-down*) – tornar uma tarefa de baixo nível mais fácil de ser realizada pelo usuário;
 - Como gerenciador de recursos (*bottom-up*) – gerenciar os dispositivos que compõem o computador;

Sistema Operacional como Máquina Estendida

- Ex.: como é feita a entrada/saída de um disco flexível
 - tarefa: Leitura e Escrita
 - SO: baixo nível de detalhes
 - Número de parâmetros;
 - Endereço de bloco a ser lido;
 - Número de setores por trilha;
 - Modo de gravação;
 - Usuário: alto nível – abstração simples
 - Visualização do arquivo a ser lido e escrito;
 - Arquivo é lido e escrito;
 - Arquivo é fechado.

Sistema Operacional como Gerenciador de Recursos

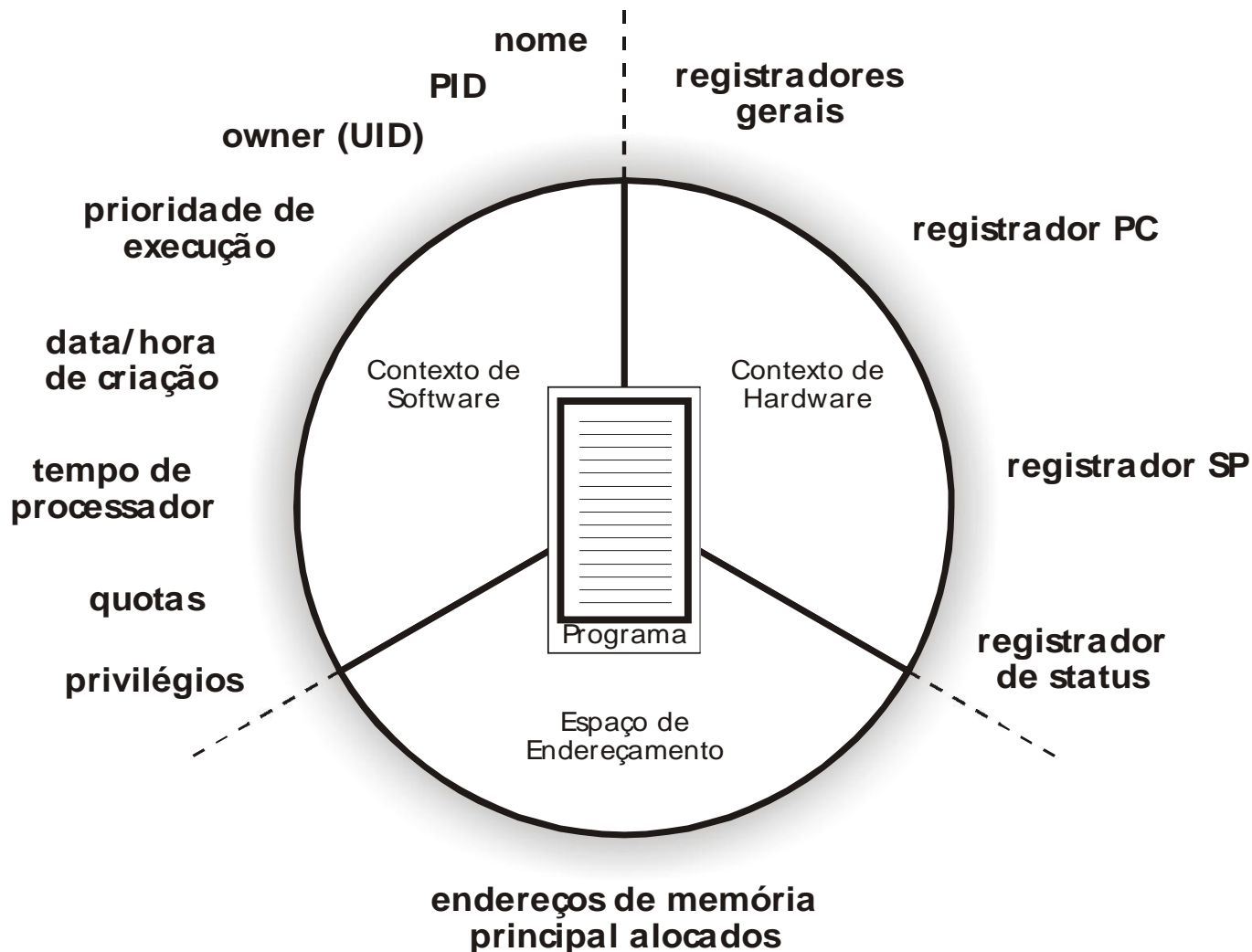
- Gerenciar todos os dispositivos e recursos disponíveis no computador
 - Ex.: se dois processos querem acessar um mesmo recurso, por exemplo, uma impressora, o SO é responsável por estabelecer uma ordem para que ambos os processos possam realizar sua tarefa de utilizar a impressora.
 - Uso do HD;
 - Uso da memória;
- Coordena a alocação controlada e ordenada dos recursos;

Conceitos Básicos Processos

- Processo: chave do SO;
 - Caracterizado por programas em execução;
 - Cada processo possui:
 - Programa (instruções que serão executadas);
 - Um espaço de endereço de memória (mínimo, máximo);
 - Contextos de hardware: informações de registradores;
 - Contextos de software: atributos;
- O Sistema Operacional gerencia todos os processos → bloco de controle de processo;

Conceitos Básicos

Processos - Contextos



Conceitos Básicos

Processos - BCP

ponteiros
Estado do processo
Nome do processo
Prioridade do processo
Registradores
Limites de memória
Lista de arquivos abertos
⋮

Bloco de Controle de Processo: Contém informações sobre o estado do processo

Conceitos Básicos

Processos

- Basicamente, um processo possui três segmentos:
 - **Texto**: código do(s) programa(s);
 - **Dados**: as variáveis;
 - **Pilha de Execução**: controla a execução do processo;
- Um processo possui três estados básicos: executando, bloqueado e pronto;

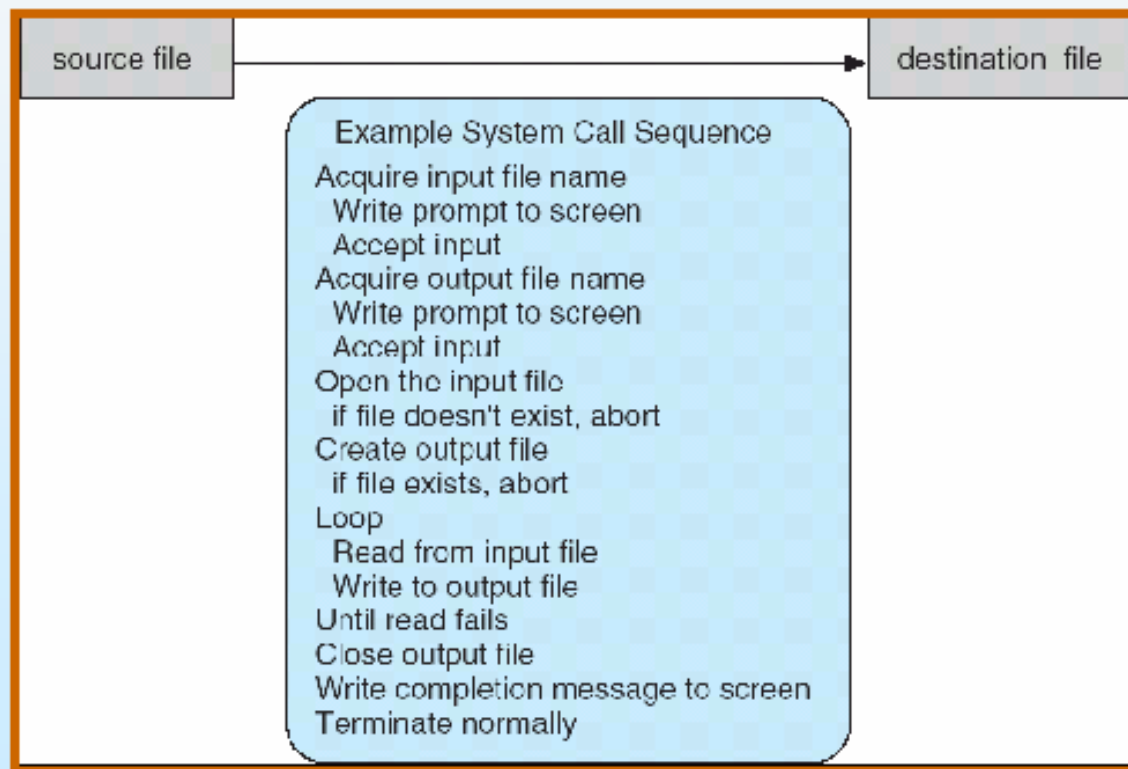
- Um processo pode resultar na execução de outros processos, chamados de processos-filhos:
 - Características para a hierarquia de processos:
 - Comunicação (Interação) e Sincronização;
 - Segurança e proteção;
 - Uma árvore de no máximo três níveis;
- Escalonadores de processos – processo que escolhe qual será o próximo processo a ser executado;
 - Diversas técnicas para escalonamento de processos;
- Comunicação e sincronismo entre processos;

Chamadas de Sistema

- Interface de programação fornecida pelo SO
- Normalmente escrita em linguagem de alto nível (C, C++ ou Java)
- Normalmente as aplicações utilizam uma Application Program Interface (API) que encapsula o acesso directo aos system calls
- As APIs mais utilizadas são a Win32 API para Windows, a POSIX API para praticamente todas as versões de UNIX, e a Java API para a Java *Virtual Machine* (JVM).
- Motivos para utilizar APIs em vez dos system calls directamente
 - Portabilidade – independência da plataforma
 - Esconder complexidade inerente aos *system calls*
 - Acréscimo de funcionalidades que optimizam o desempenho
- O acesso aos *system calls* está implementada em bibliotecas que são carregadas com as aplicações

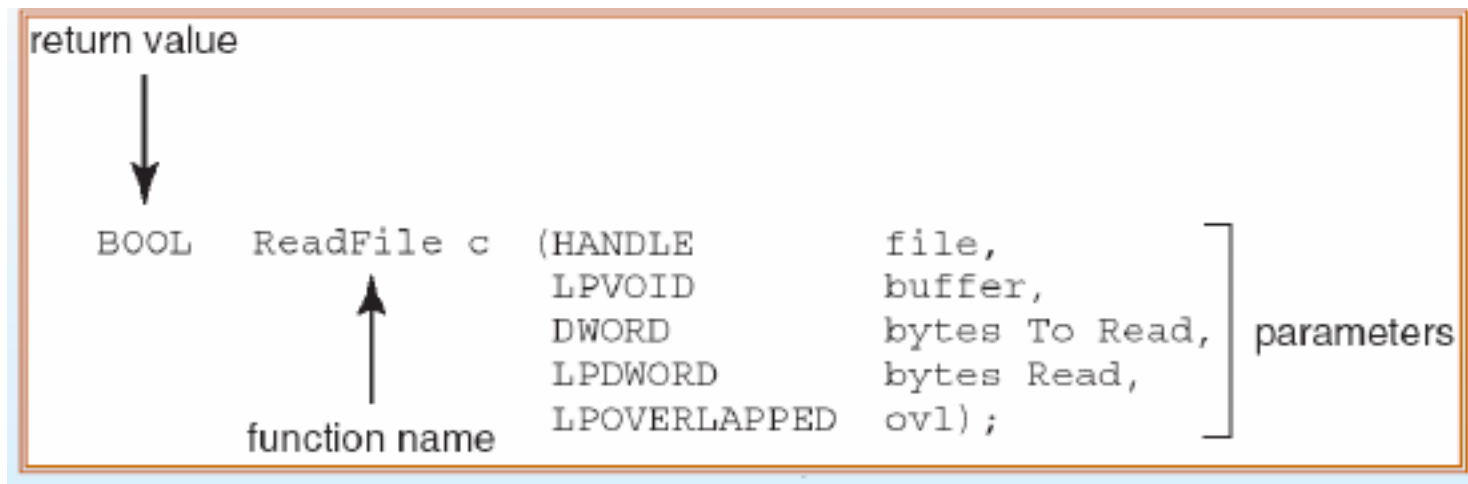
Chamadas de Sistema

- Sequência de *System Calls* para copiar o conteúdo de um arquivo para outro



Exemplo da API - Windows Standard

- A função ReadFile() da Win32 API
 - Uma função para ler o conteúdo de um arquivo



Exemplo da API - Windows Standard

- Descrição dos parâmetros de ReadFile()
 - HANDLE file - the file to be read
 - LPVOID buffer - a buffer where the data will be read into and written from
 - DWORD bytesToRead - the number of bytes to be read into the buffer
 - LPDWORD bytesRead - the number of bytes read during the last read
 - LPOVERLAPPED ovl - indicates if overlapped I/O is being used

Exemplo da API UNIX Standard

NAME

`read` - read from a file descriptor

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

int read(int fd, char *buf, size_t count);
```

DESCRIPTION

`read` reads up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

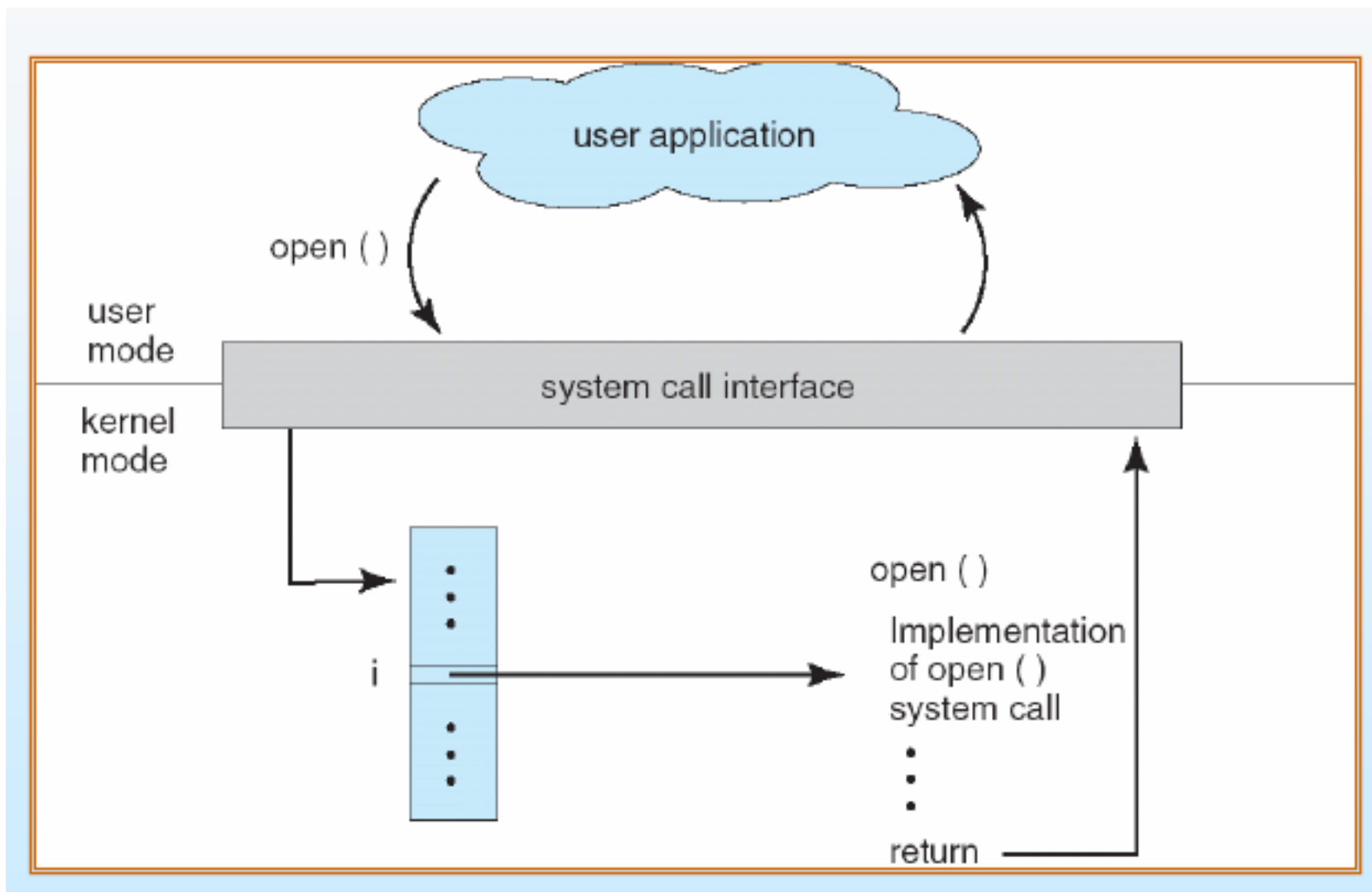
RETURN VALUE

On success, the number of bytes read are returned (zero indicates end of file). On error, -1 is returned, and *errno* is set appropriately.

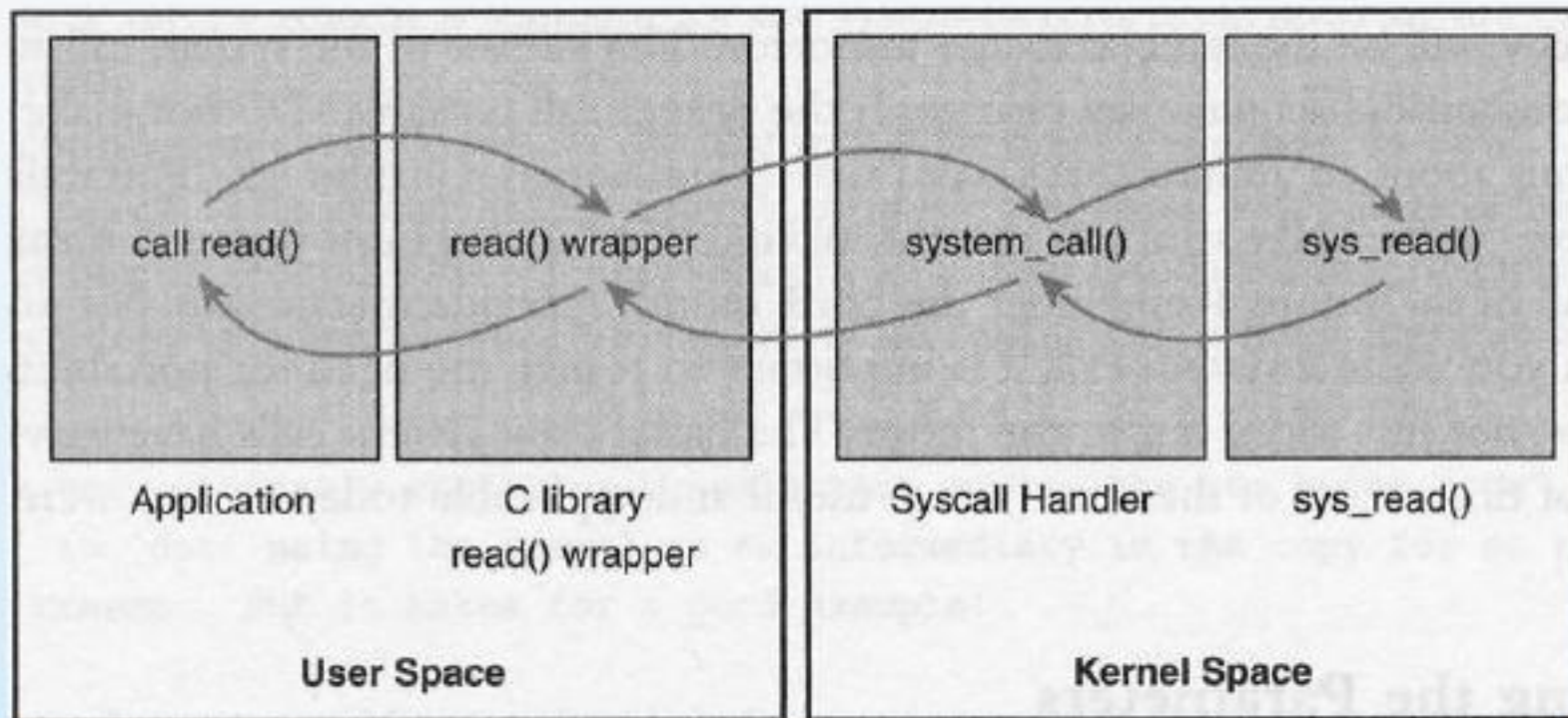
Implementação

- A cada *system call* está associado um número
 - A interface mantém uma tabela com o endereço de cada *system call* handler que é indexada pelo número do *system call*
- Através desta tabela, o respectivo handler é invocado no kernel
 - Os parâmetros do *system call* são transferidos para o kernel
 - Uma vez executado, o resultado e os parâmetros de retorno são transferidos para o programa utilizador, como se tivesse havido uma invocação de uma função normal
- A aplicação que invoca o *system call* não precisa saber como este é implementado
 - Só precisa de obedecer à sintaxe da API (assinatura do método) e estar à espera dos resultados da invocação
 - Precisa de conhecer o comportamento associado ao *system call*
 - Os detalhes da interface sistema são escondidos pela API
 - São geridos pela biblioteca de run-time – camada de funções de biblioteca que são incluídas na aplicação quando da compilação e carregamento do ficheiro executável

API – System Call – OS Relationship



Transição para os Syscalls em Linux



Linux System Calls Numbers

```
/*  
 * This file contains the system call numbers.  
 */
```

```
#define _NR_restart_syscall    0  
#define _NR_exit               1  
#define _NR_fork               2  
#define _NR_read               3  
#define _NR_write              4  
#define _NR_open               5  
#define _NR_close              6  
#define _NR_waitpid            7  
#define _NR_creat              8  
#define _NR_link               9  
#define _NR_unlink             10  
#define _NR_execve             11  
#define _NR_chdir              12  
#define _NR_time               13  
#define _NR_mknod              14  
#define _NR_chmod              15  
#define _NR_lchown             16  
#define _NR_break              17  
#define _NR_oldstat            18  
#define _NR_lseek              19  
#define _NR_getpid             20  
#define _NR_mount              21  
#define _NR_umount             22  
#define _NR_setuid             23  
#define _NR_getuid             24  
#define _NR_stime              25  
#define _NR_ptrace             26  
#define _NR_alarm              27  
#define _NR_oldfstat           28  
#define _NR_pause              29  
#define _NR_utime              30  
#define _NRatty                31  
#define _NR_gtty               32  
#define _NR_access              33  
#define _NR_nice                34  
#define _NR_ftime              35  
#define _NR_sync               36  
#define _NR_kill               37  
#define _NR_rename             38  
#define _NR_mkdir              39  
#define _NR_rmdir              40  
#define _NR_dup                41  
#define _NR_pipe               42
```

/usr/src/linux/include/asm-i386/unistd.h

Linux Syscall Table

```
.data
ENTRY(sys_call_table)
    .long sys_restart_syscall      /* 0 - old "setup()" system call, used for restarting */
    .long sys_exit
    .long sys_fork
    .long sys_read
    .long sys_write
    .long sys_open                /* 5 */
    .long sys_close
    .long sys_waitpid
    .long sys_creat
    .long sys_link
    .long sys_unlink              /* 10 */
    .long sys_execve
    .long sys_chdir
    .long sys_time
    .long sys_mknod
    .long sys_chmod               /* 15 */
    .long sys_lchown16
    .long sys_ni_syscall          /* old break syscall holder */
    .long sys_stat
    .long sys_lseek
    .long sys_getpid              /* 20 */
    .long sys_mount
    .long sys_oldumount
    .long sys_setuid16
    .long sys_getuid16
    .long sys_stime               /* 25 */
    .long sys_ptrace
    .long sys_alarm
    .long sys_fstat
    .long sys_pause
    .long sys_utime               /* 30 */
    .long sys_ni_syscall          /* old tty syscall holder */
    .long sys_ni_syscall          /* old gtty syscall holder */
    .long sys_access
    .long sys_nice
    .long sys_ni_syscall          /* 35 - old ftime syscall holder */
    .long sys_sync
    .long sys_kill
    .long sys_rename
    .long sys_mkdir
    .long sys_rmdir               /* 40 */
    .long sys_dup
    .long sys_pipe
```

/usr/src/linux/arch/i386/kernel/entry.S

Invocação direta de Syscalls

- Programa em Assembler que invoca os system calls `write()` e `exit()` através da instrução `int 0x80`

```
section .data                                ;section declaration
msg      db      "Hello, world!",0xa         ;our dear string
len      equ     $ - msg                     ;length of our dear string

section .text                                ;section declaration

        global _start                       ;we must export the entry point to the ELF linker or
                                           ;loader. They conventionally recognize _start as their
                                           ;entry point. Use ld -e foo to override the default.

_start:

;write our string to stdout

        mov     edx,len ;third argument: message length
        mov     ecx,msg ;second argument: pointer to message to write
        mov     ebx,1   ;first argument: file handle (stdout)
        mov     eax,4   ;system call number (sys_write)
        int     0x80    ;call kernel

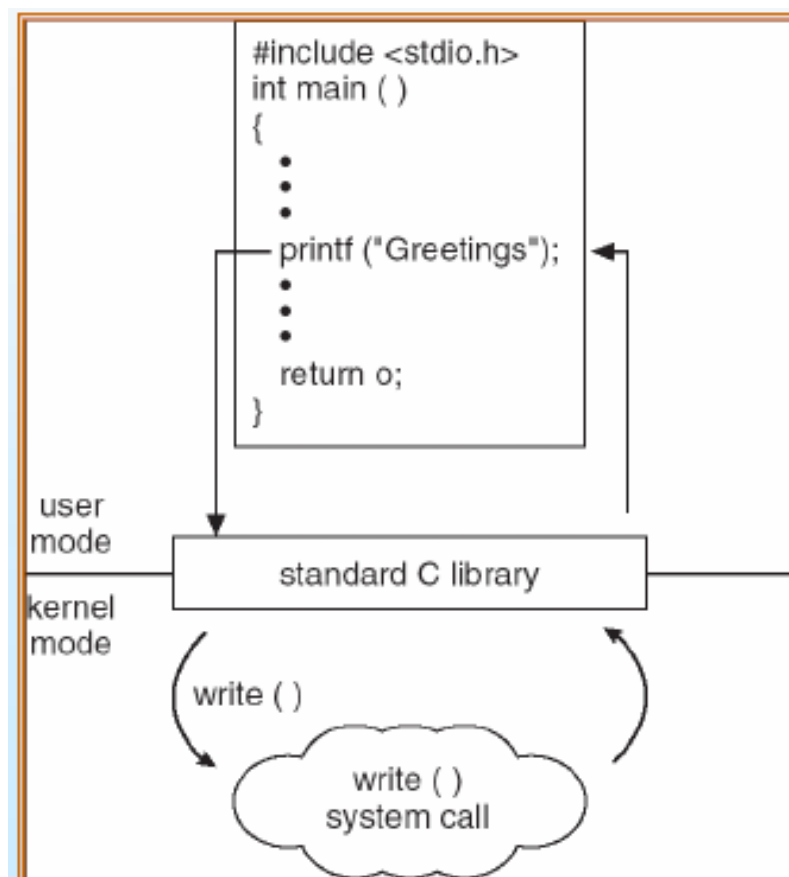
;and exit

        mov     ebx,0   ;first syscall argument: exit code
        mov     eax,1   ;system call number (sys_exit)
        int     0x80    ;call kernel
```

- Na arquitetura Intel a partir do Pentium II, duas novas instruções permitem a realização de *system calls* mais rapidamente
 - `sysentry` permite a entrada no sistema sem passar por uma interrupção software
 - `sysexit` permite a saída do kernel pelo mesmo mecanismo
 - O kernel linux utiliza estas instruções preferencialmente a partir da versão 2.6
- A invocação destas instruções faz-se pela invocação direta de código assembler colocado pelo kernel numa página específica de todos os processos (virtual dynamic shared object - vdso)
 - O ponto de entrada é designado por `kernel_vsycall`
 - Endereço pode variar por distribuição e formato executável
- Ver referência
 - http://manugarg.googlepages.com/systemcallinlinux2_6.html

Invocação através da Libc

- Programa em C que invoca a função de biblioteca printf(), que por sua vez chama o system call write()



Conceitos Básicos Chamadas de Sistema

- **Modos de Acesso:**
 - Modo usuário;
 - Modo *kernel* ou Supervisor ou Núcleo;
 - São determinados por um conjunto de bits localizados no registrador de status do processador: PSW (*program status word*);
 - Por meio desse registrador, o hardware verifica se a instrução pode ou não ser executada pela aplicação;
 - Protege o próprio *kernel* do Sistema Operacional na RAM contra acessos indevidos;

Conceitos Básicos Chamadas de Sistema

- Modo usuário:
 - Aplicações não têm acesso direto aos recursos da máquina, ou seja, ao hardware;
 - Quando o processador trabalha no modo usuário, a aplicação só pode executar **instruções sem privilégios, com um acesso reduzido de instruções**;
 - Por que?
 - Para garantir a **segurança** e a **integridade do sistema**;

Conceitos Básicos Chamadas de Sistema

- Modo *Kernel*:
 - Aplicações têm acesso direto aos recursos da máquina, ou seja, ao hardware;
 - **Operações com privilégios;**
 - Quando o processador trabalha no modo *kernel*, a aplicação tem **acesso ao conjunto total de instruções;**
 - Apenas o SO tem acesso às instruções privilegiadas;

Conceitos Básicos Chamadas de Sistema

- Se uma aplicação precisa realizar alguma instrução privilegiada, ela realiza uma **chamada de sistema**, que altera do modo usuário para o modo *kernel*;
- Chamadas de sistemas são a **porta de entrada** para o modo *Kernel*;
 - São a interface entre os programas do usuário no modo usuário e o Sistema Operacional no modo kernel;
 - As chamadas se diferem de SO para SO, no entanto, os conceitos relacionados às chamadas são similares independentemente do SO;

Conceitos Básicos Chamadas de Sistema

- **TRAP**: instrução que permite o acesso ao modo *kernel*;
- Exemplo:
 - Instrução do UNIX:
count = read(fd,buffer,nbytes) ;



Arquivo a ser lido

Conceitos Básicos Chamadas de Sistema

- **TRAP**: instrução que permite o acesso ao modo *kernel*;
- Exemplo:
 - Instrução do UNIX:
count = `read(fd,buffer,nbytes)` ;



Ponteiro para o Buffer

Conceitos Básicos Chamadas de Sistema

- **TRAP**: instrução que permite o acesso ao modo *kernel*;
- Exemplo:
 - Instrução do UNIX:
count = `read(fd,buffer,nbytes)` ;

Bytes a serem lidos

O programa sempre deve checar o retorno da chamada de sistema para saber se algum erro ocorreu!!!

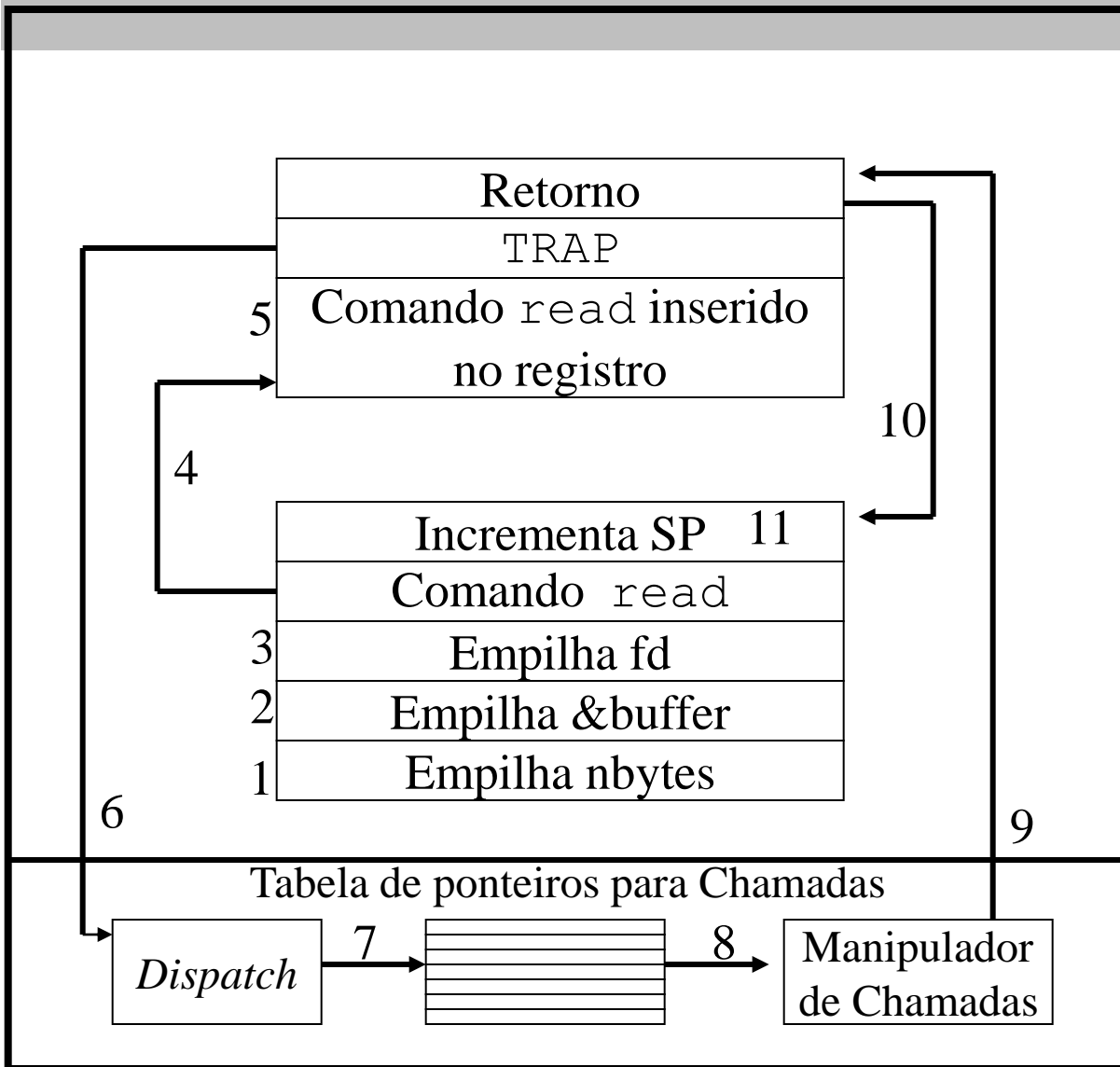
Chamadas de Sistema

Endereço
0xFFFFFFFF

Espaço
do
Usuário

Kernel
SO

Endereço 0



Biblioteca do
Procedimento
READ

Chamada ao
Procedimento
READ

Conceitos Básicos

Chamadas de Sistema

- Exemplos de chamadas de sistema:
 - Chamadas para gerenciamento de processos:
 - `Fork` (`CreateProcess` – WIN32) – cria um processo;
 - Chamadas para gerenciamento de diretórios:
 - `Mount` – monta um diretório;
 - Chamadas para gerenciamento de arquivos:
 - `Close` (`CloseHandle` – WIN32) – fechar um arquivo;
 - Outros tipos de chamadas:
 - `Chmod`: modifica permissões;

Referência do Código Kernel

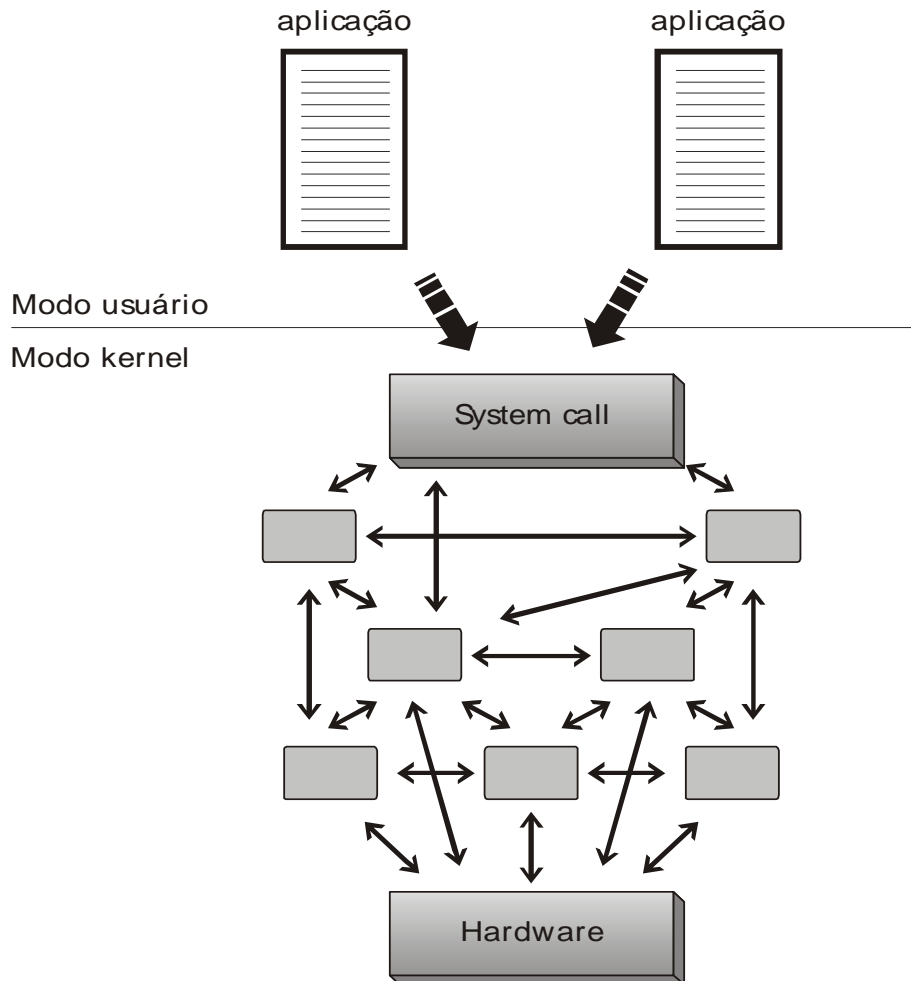
- Para aprofundar estes conceitos aconselha-se a leitura do código do kernel:
 - Definição dos números de System Calls
 - <http://lxr.linux.no/source/include/asm-i386/unistd.h>
 - Tabela de System Calls
 - http://lxr.linux.no/source/arch/i386/kernel/syscall_table.S
 - Pontos de entrada no Kernel
 - `int $80 -> system_call`
 - <http://lxr.linux.no/source/arch/i386/kernel/entry.S#L364>
 - `sysentry`
 - <http://lxr.linux.no/source/arch/i386/kernel/entry.S#L285>
 - `vsyscall-int80`
 - <http://lxr.linux.no/source/arch/i386/kernel/vsyscall-int80.S>
 - `vsyscall-sysenter`
 - <http://lxr.linux.no/source/arch/i386/kernel/vsyscall-sysenter.S>
 - Cópia de argumentos in/out
 - <http://lxr.linux.no/source/arch/i386/lib/usercopy.c>

- Principais tipos de estruturas:
 - Monolíticos;
 - Em camadas;
 - Máquinas Virtuais;
 - Arquitetura *Micro-kernel*;
 - Cliente-Servidor;

Estrutura dos Sistemas Operacionais - Monolítico

- Todos os módulos do sistema são compilados individualmente e depois ligados uns aos outros em um único **arquivo-objeto**;
- O Sistema Operacional é um conjunto de processos que podem interagir entre si a qualquer momento sempre que necessário;
- Cada processo possui uma interface bem definida com relação aos parâmetros e resultados para facilitar a comunicação com os outros processos;
- Simples;
- Primeiros sistemas UNIX e MS-DOS;

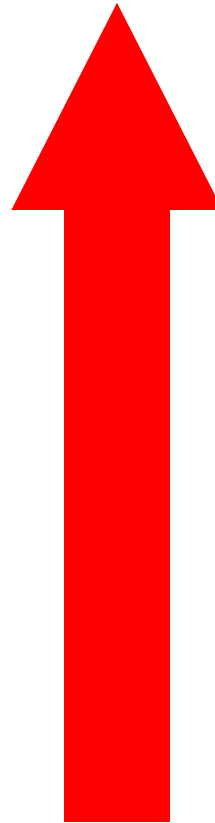
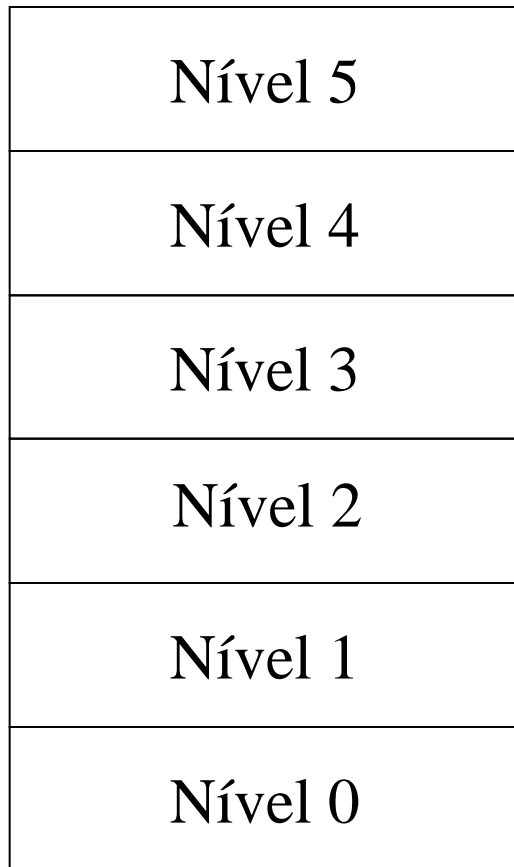
Estrutura dos Sistemas Operacionais - Monolítico



Estrutura dos Sistemas Operacionais – Em camadas

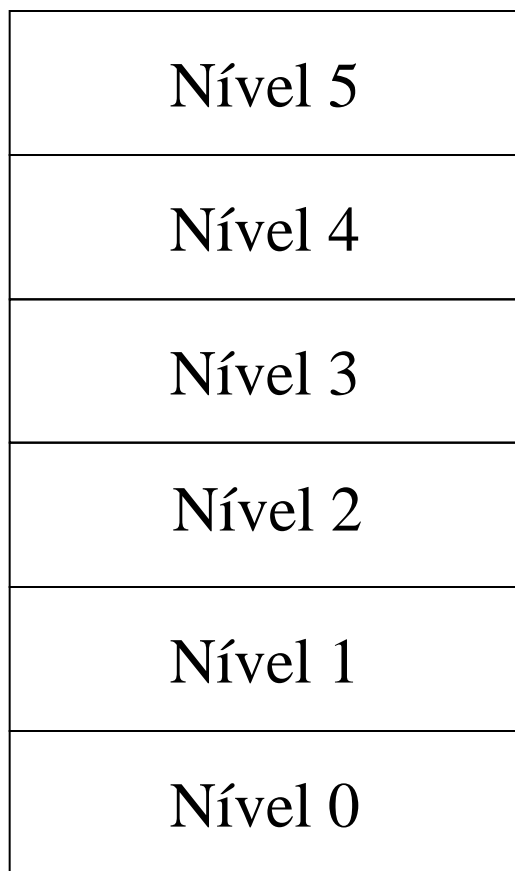
- Possui uma hierarquia de níveis;
- Primeiro sistema em camadas: THE (idealizado por E.W. Dijkstra);
 - Possuía 6 camadas, cada qual com uma função diferente;
 - Sistema em *batch* simples;
- Vantagem: isolar as funções do sistema operacional, facilitando manutenção e depuração
- Desvantagem: cada nova camada implica uma mudança no modo de acesso
- Atualmente: modelo de 2 camadas

Estrutura dos Sistemas Operacionais – Em camadas



Fornecimento de Serviços

Estrutura dos Sistemas Operacionais – Em camadas



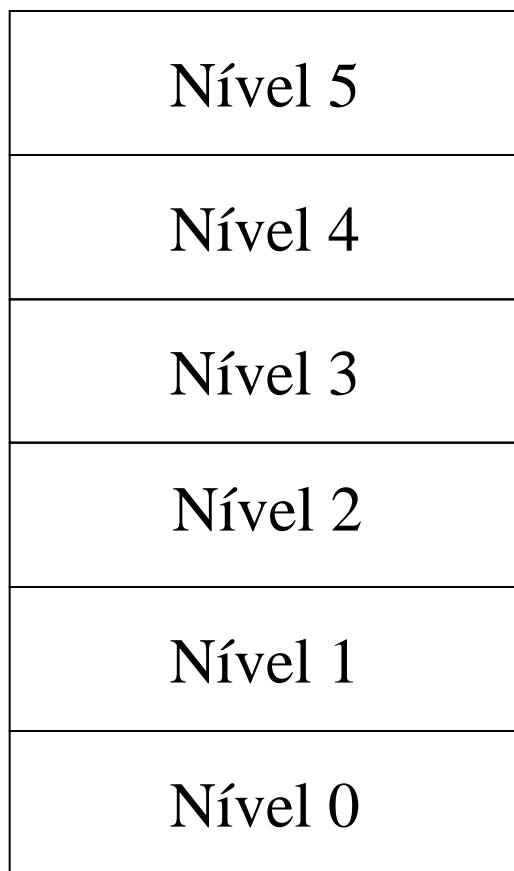
- Alocação do processador;
- Chaveamento entre os processos em execução – multiprogramação;

Estrutura dos Sistemas Operacionais – Em camadas

Nível 5
Nível 4
Nível 3
Nível 2
Nível 1
Nível 0

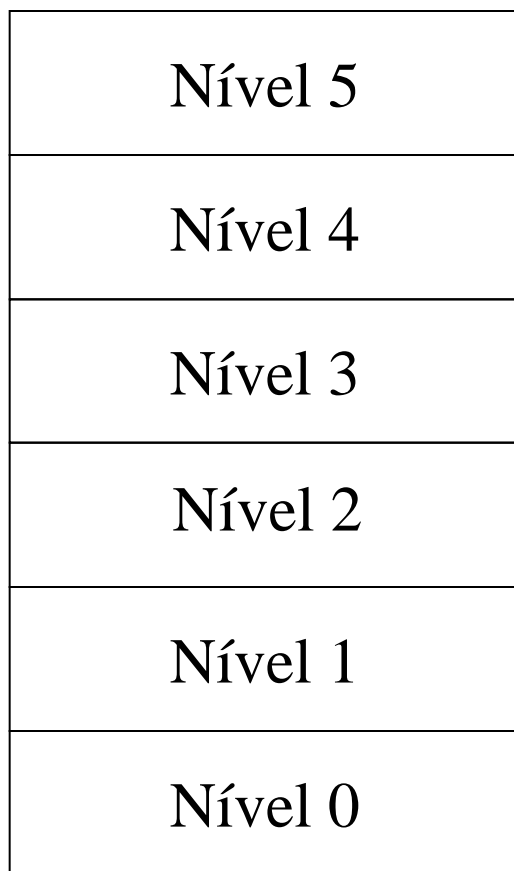
- Gerenciamento da memória;
- Alocação de espaço para processos na memória e no disco:
 - Processo dividido em partes (páginas) para ficarem no disco;

Estrutura dos Sistemas Operacionais – Em camadas



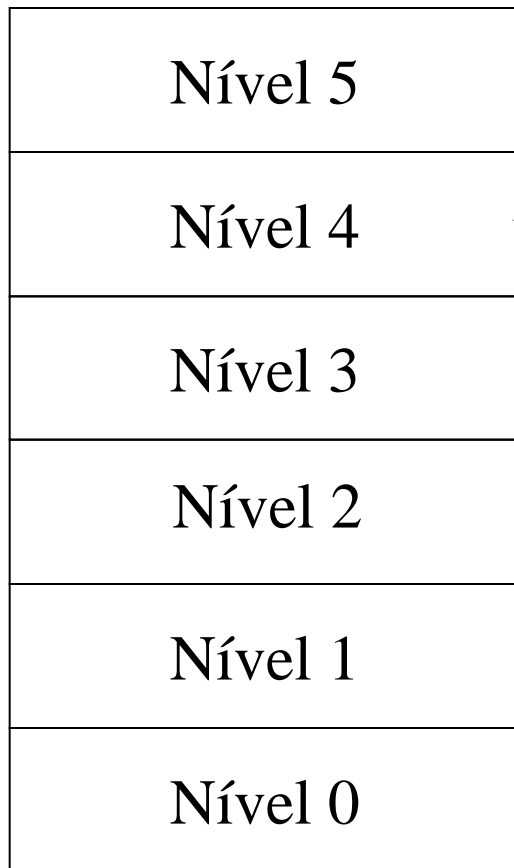
- Comunicação entre os processos;

Estrutura dos Sistemas Operacionais – Em camadas



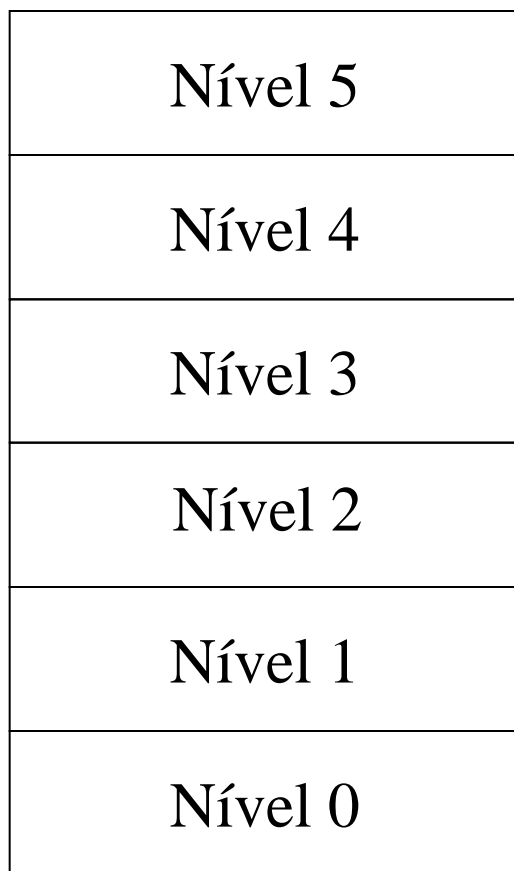
- Gerenciamento dos dispositivos de entrada/saída – armazenamento de informações de/para tais dispositivos;

Estrutura dos Sistemas Operacionais – Em camadas



- Programas dos usuários;
- Alto nível de abstração;

Estrutura dos Sistemas Operacionais – Em camadas

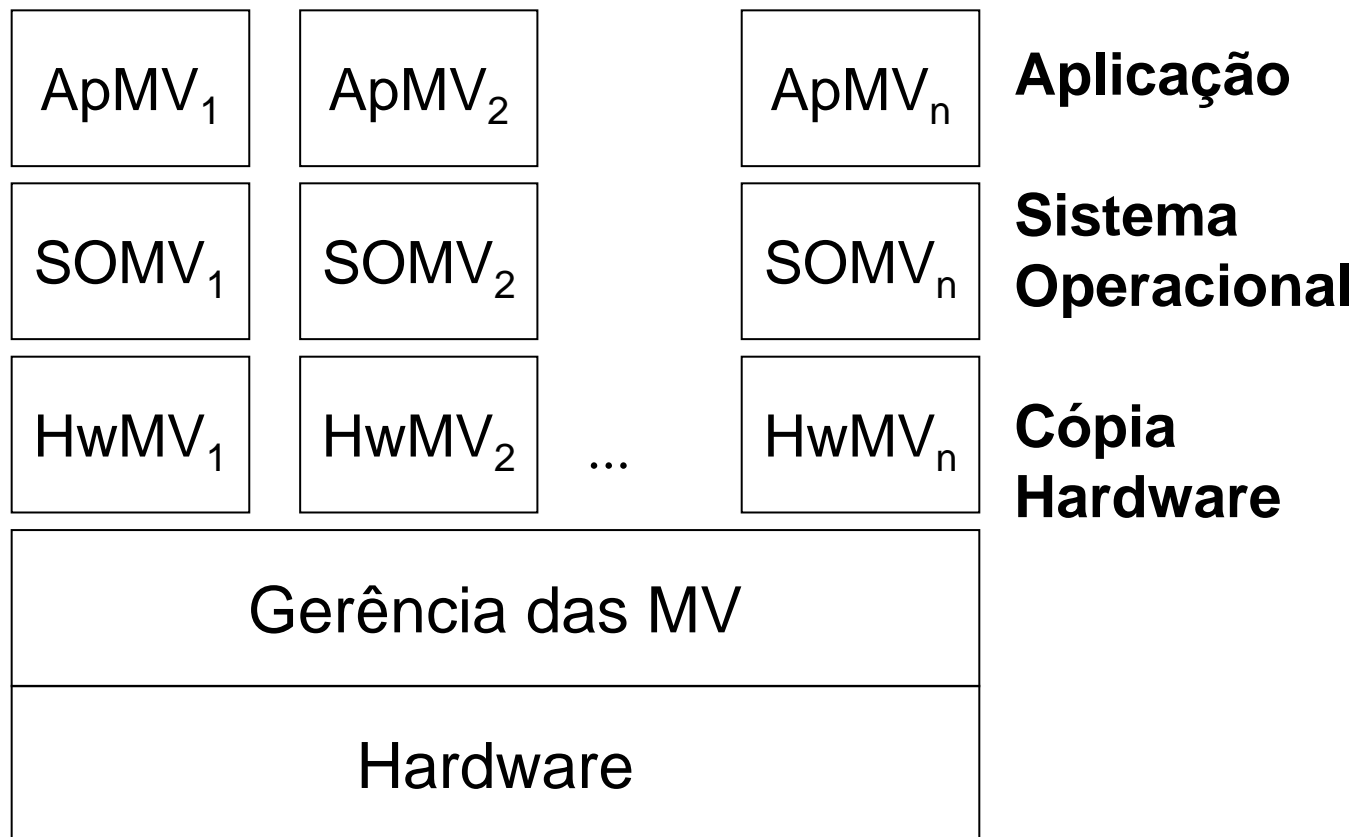


- Processo do operador do sistema;

Estrutura dos Sistemas Operacionais – Máquina Virtual

- Idéia em 1960 com a IBM → VM/370;
- Modelo de máquina virtual cria um nível intermediário entre o SO e o Hardware;
- Esse nível cria diversas **máquinas virtuais independentes e isoladas**, onde cada máquina oferece um cópia virtual do hardware, incluindo modos de acesso, interrupções, dispositivos de E/S, etc.;
- Cada máquina virtual pode ter seu próprio SO;

Estrutura dos Sistemas Operacionais – Máquina Virtual



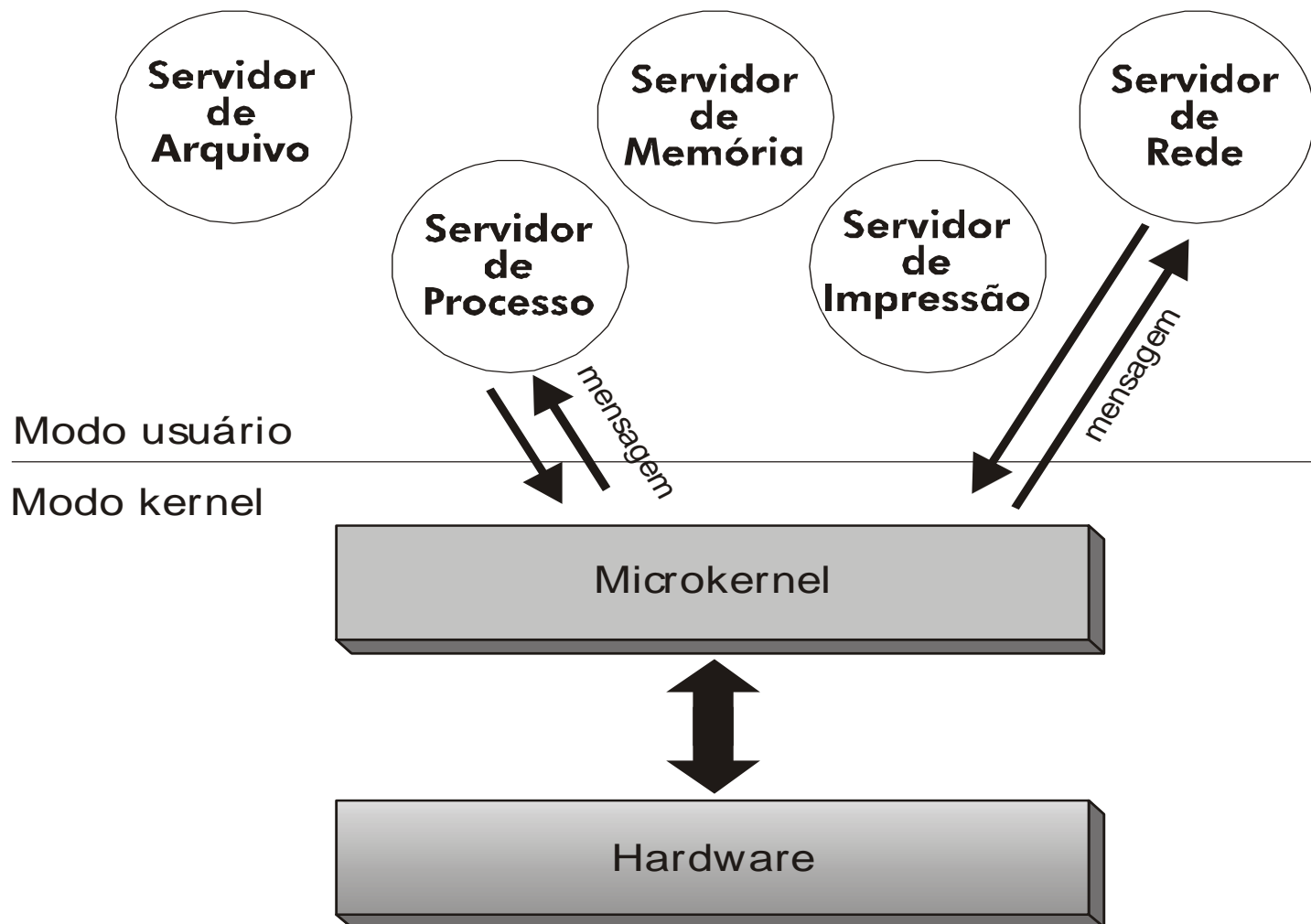
Estrutura dos Sistemas Operacionais – Máquina Virtual

- Atualmente, a idéia de máquina virtual é utilizada em contextos diferentes:
 - Programas MS-DOS: rodam em computadores 32bits;
 - As chamadas feitas pelo MS-DOS ao Sistema Operacional são realizadas e monitoradas pelo monitor da máquina virtual (VMM);
 - Virtual 8086;
 - Programas JAVA (Máquina Virtual Java-JVM): o compilador Java produz código para a JVM (*bytecode*). Esse código é executado pelo interpretador Java:
 - Programas Java rodam em qualquer plataforma, independentemente do Sistema Operacional;

Estrutura dos Sistemas Operacionais – Máquina Virtual

- Vantagens
 - Flexibilidade;
- Desvantagem:
 - Simular diversas máquinas virtuais não é uma tarefa simples → sobrecarga;

Estrutura dos Sistemas Operacionais – *Micro-Kernel*

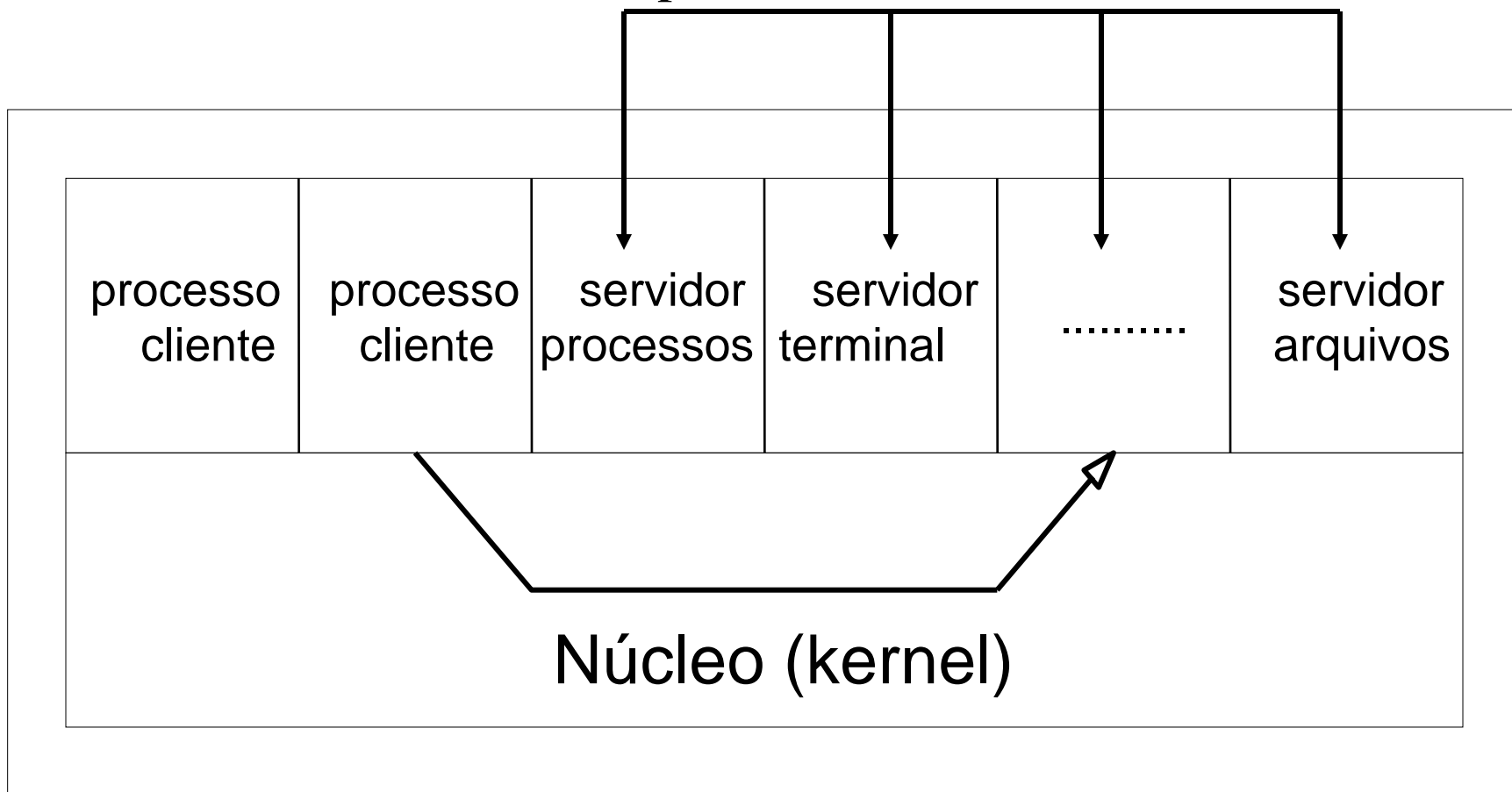


Estrutura dos Sistemas Operacionais – Cliente/Servidor

- Reduzir o Sistema Operacional a um nível mais simples:
 - **Kernel**: implementa a comunicação entre processos clientes e processos servidores → Núcleo mínimo;
 - Maior parte do Sistema Operacional está implementado como processos de usuários (nível mais alto de abstração);
 - Sistemas Operacionais Modernos;

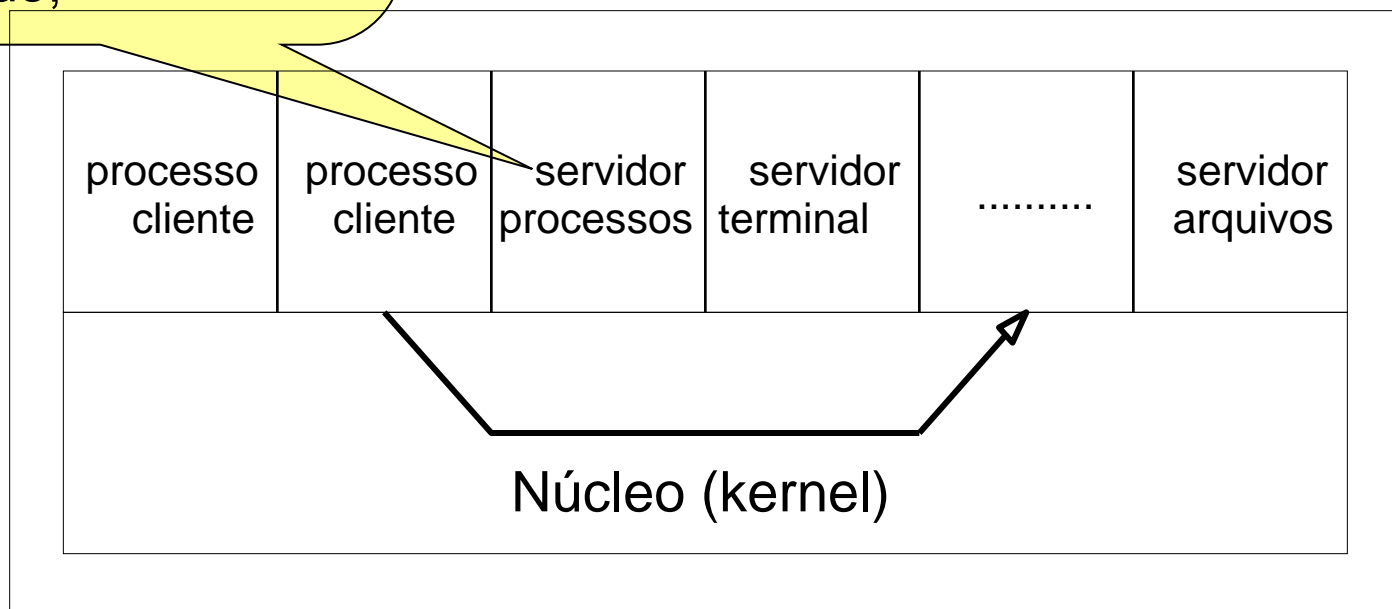
Estrutura dos Sistemas Operacionais – Cliente/Servidor

Cada processo servidor trata de uma tarefa



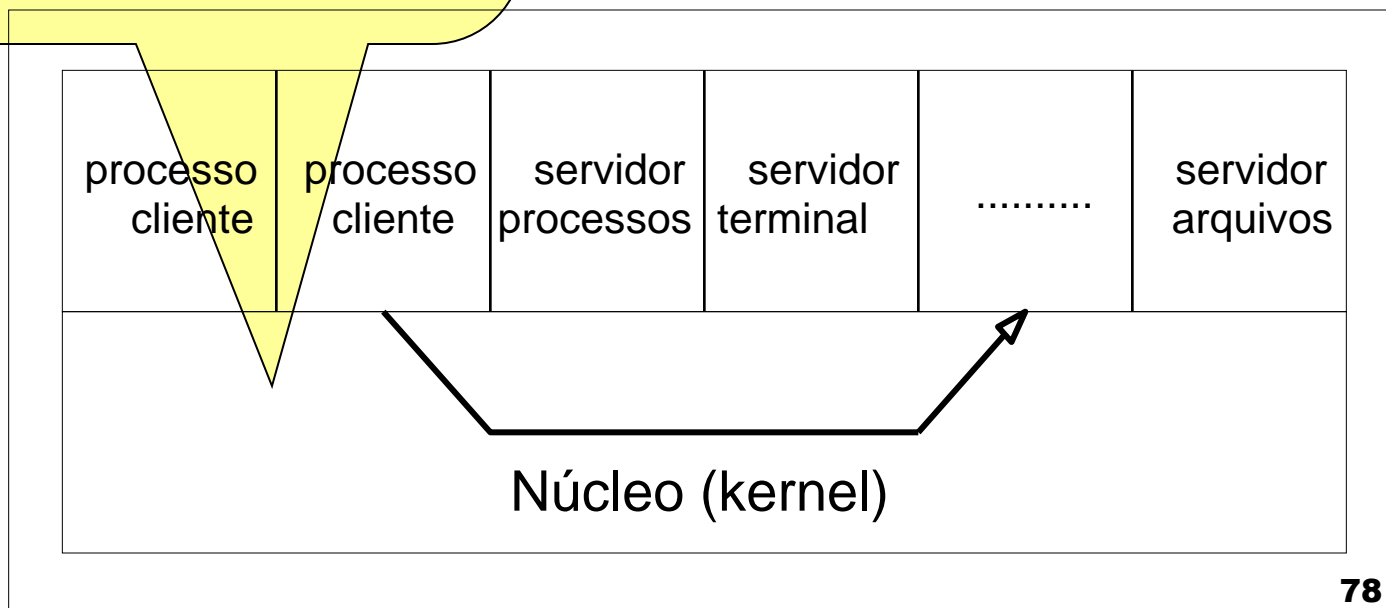
Estrutura dos Sistemas Operacionais – Cliente/Servidor

Os processos servidores não têm acesso direto ao hardware. Assim, se algum problema ocorrer com algum desses servidores, o hardware não é afetado;



Estrutura dos Sistemas Operacionais – Cliente/Servidor

O mesmo não se aplica aos serviços que controlam os dispositivos de E/S, pois essa é uma tarefa difícil de ser realizada no modo usuário devido à limitação de endereçamento. Sendo assim, essa tarefa ainda é feita no *kernel*.



Estrutura dos Sistemas Operacionais – Cliente/Servidor

- Adaptável para Sistemas Distribuídos;

