

Sistemas Operacionais

Estrutura de um S.O.

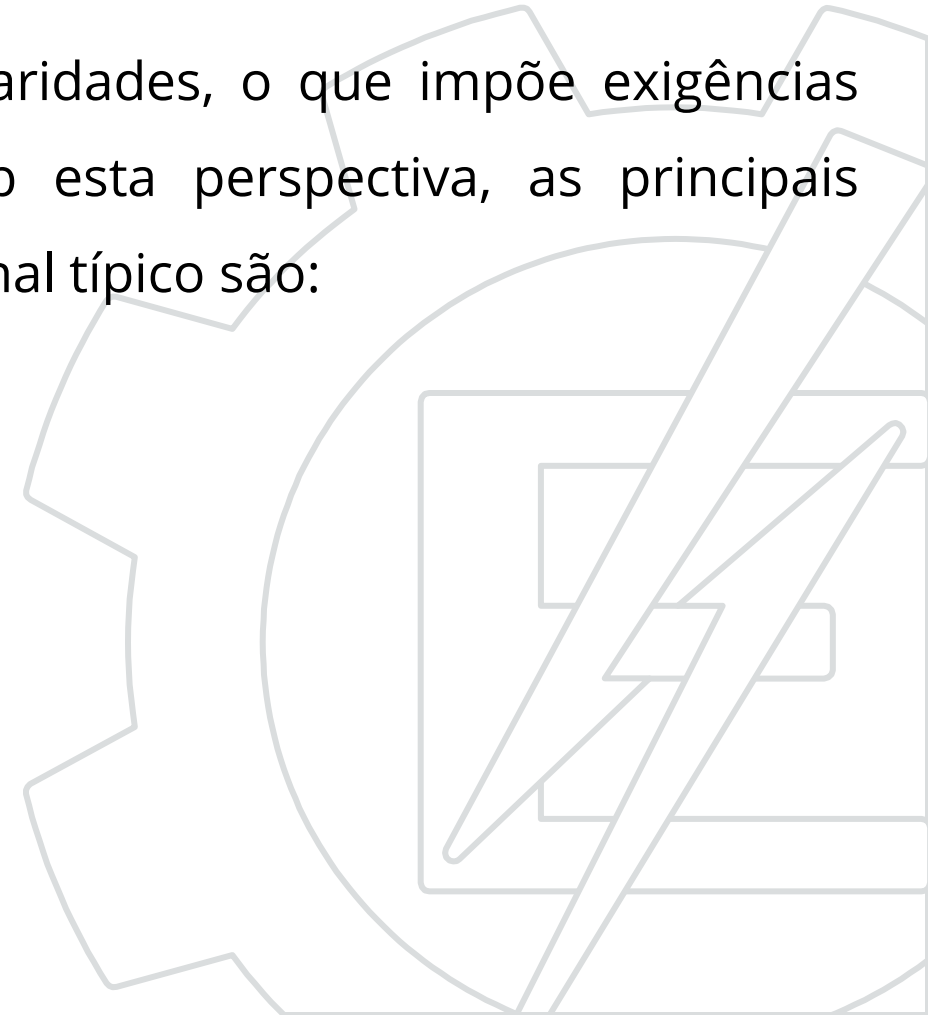
(Parte 2)

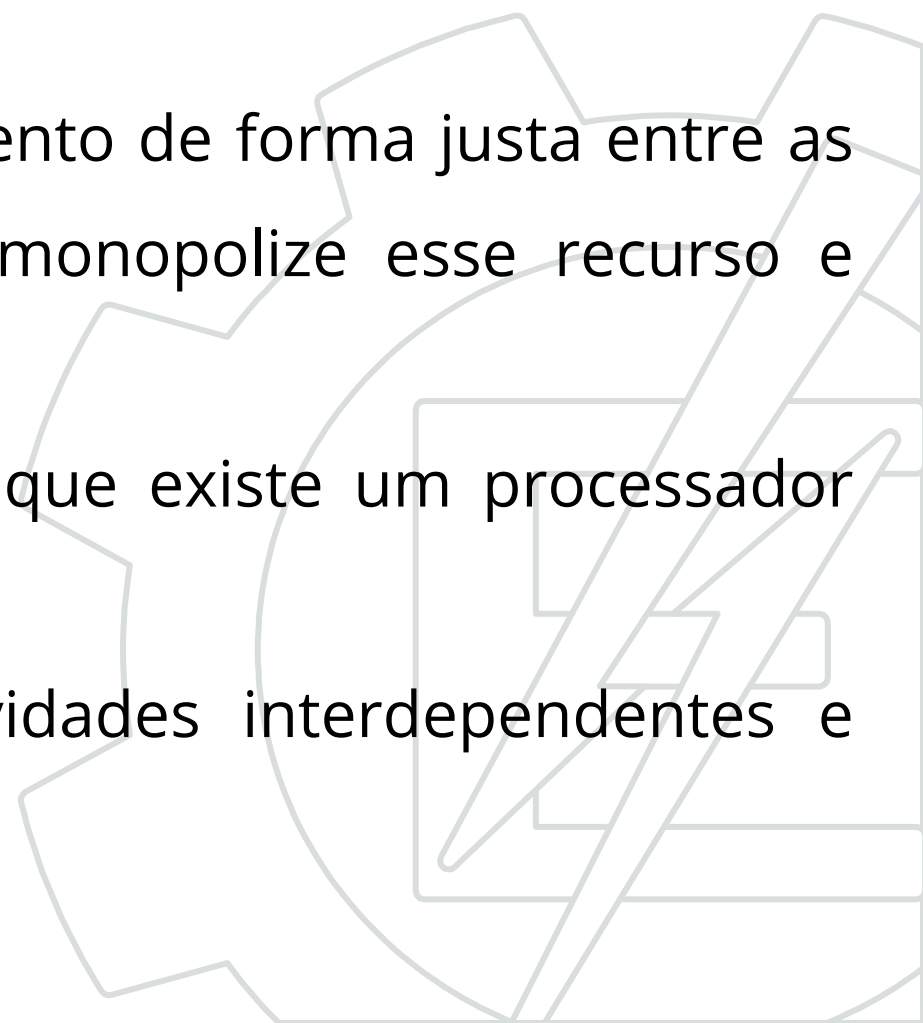
Prof. Otávio Gomes

otavio.gomes@unifei.edu.br



- Para cumprir seus objetivos de **abstração** e **gerência**, o sistema operacional deve atuar em várias frentes.
- Cada um dos recursos do sistema possui suas particularidades, o que impõe exigências específicas para gerenciar e abstrair os mesmos. Sob esta perspectiva, as principais funcionalidades implementadas por um sistema operacional típico são:
 - Gerência do processador;
 - Gerência de memória;
 - Gerência de dispositivos;
 - Gerência de arquivos;
 - Gerência de proteção.



- **Gerência do processador:** também conhecida como gerência de processos ou de atividades.
 - Visa distribuir a capacidade de processamento de forma justa entre as aplicações, evitando que uma aplicação monopolize esse recurso e respeitando as prioridades dos usuários;
 - O sistema operacional provê a ilusão de que existe um processador independente para cada tarefa;
 - Fornece abstrações para sincronizar atividades interdependentes e prover formas de comunicação entre elas.
- 

- **Gerência de memória:** tem como objetivo fornecer a cada aplicação uma área de memória própria, independente e isolada das demais aplicações e inclusive do núcleo do sistema.
 - Melhora a estabilidade e segurança do sistema como um todo;
 - Caso a memória RAM existente seja insuficiente para as aplicações, o sistema operacional pode aumentá-la de forma transparente às aplicações;
 - Uma importante abstração construída pela gerência de memória é a noção de memória virtual, que desvincula os endereços de memória vistos por cada aplicação dos endereços acessados pelo processador na memória RAM.

- **Gerência de dispositivos:** cada periférico do computador possui suas peculiaridades; assim, o procedimento de interação com uma placa de rede é completamente diferente da interação com um disco rígido SCSI.
 - Existem muitos problemas e abordagens em comum para o acesso aos periféricos.
 - A função da gerência de dispositivos (também conhecida como gerência de entrada/saída) é implementar a interação com cada dispositivo por meio de *drivers* e criar modelos abstratos que permitam agrupar vários dispositivos distintos sob a mesma interface de acesso.

- **Gerência de arquivos:** esta funcionalidade é construída sobre a gerência de dispositivos e visa criar arquivos e diretórios, definindo sua interface de acesso e as regras para seu uso.
 - Os conceitos abstratos de arquivo e diretório são tão importantes e difundidos que muitos sistemas operacionais os usam para permitir o acesso a recursos que nada tem a ver com armazenamento.
 - No sistema operacional experimental Plan 9 [Pike et al., 1993], todos os recursos do sistema operacional são vistos como arquivos.

- **Gerência de proteção:** com computadores conectados em rede e compartilhados por vários usuários, é importante definir claramente os recursos que cada usuário pode acessar, as formas de acesso permitidas (leitura, escrita, etc) e garantir que essas definições sejam cumpridas.
 - Para proteger os recursos do sistema contra acessos indevidos, é necessário: **a)** definir usuários e grupos de usuários; **b)** disponibilizar procedimentos de autenticação; **c)** definir e aplicar regras de controle de acesso aos recursos, relacionando todos os usuários, recursos e formas de acesso; **d)** registrar o uso dos recursos pelos usuários, para fins de auditoria e contabilização.

- Além dessas funcionalidades básicas oferecidas pela maioria dos sistemas operacionais, várias outras vêm se agregar aos sistemas modernos, para cobrir aspectos complementares, como a **interface gráfica, suporte de rede, fluxos multimídia, gerência de energia**, etc.
- As funcionalidades do sistema operacional geralmente são interdependentes: por exemplo, a gerência do processador depende de aspectos da gerência de memória, assim como a gerência de memória depende da gerência de dispositivos e da gerência de proteção.

- Um sistema operacional deve gerenciar os recursos do *hardware*, fornecendo-os às aplicações conforme suas necessidades.
- Para assegurar a integridade dessa gerência, é essencial garantir que as aplicações não consigam acessar o *hardware* diretamente, mas sempre através de pedidos ao sistema operacional, que avalia e realiza o controle de todos os acessos ao *hardware*.
- Mas como impedir as aplicações de acessar o *hardware* diretamente?

- Núcleo, *drivers*, utilitários e aplicações são constituídos basicamente de código de máquina. Todavia, devem ser diferenciados em sua capacidade de interagir com o *hardware*.
 - Enquanto o núcleo e os *drivers* devem ter pleno acesso ao *hardware*, para poder configurá-lo e gerenciá-lo, os utilitários e os aplicativos devem ter acesso mais restrito a ele, para não interferir nas configurações e na gerência, o que acabaria desestabilizando o sistema inteiro. Além disso, aplicações com acesso pleno ao *hardware* tornariam inúteis os mecanismos de segurança e controle de acesso aos recursos (tais como arquivos, diretórios e áreas de memória).

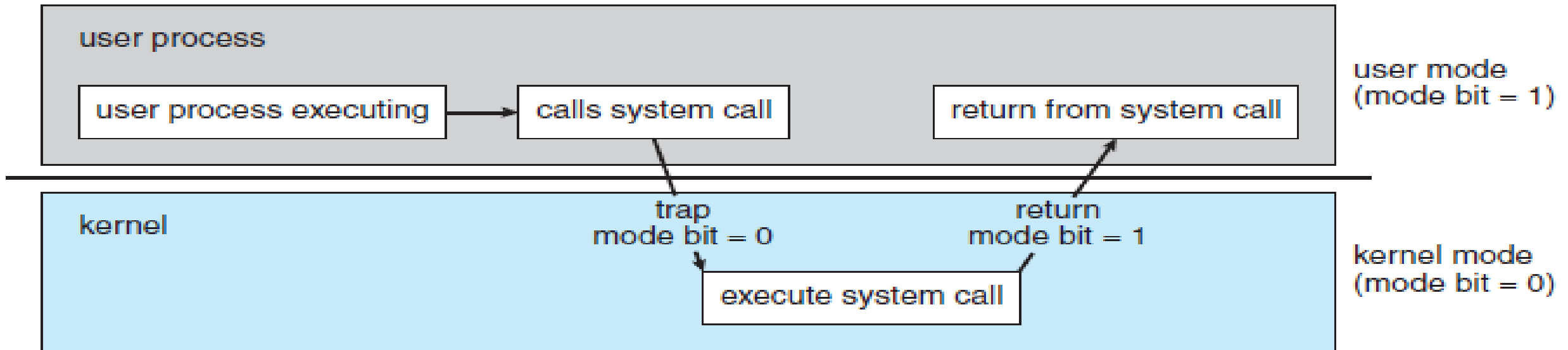
- Para permitir diferenciar os privilégios de execução dos diferentes tipos de *software*, os processadores modernos contam com dois ou mais níveis de privilégio de execução.
- Esses níveis são controlados por *flags* especiais nos processadores, e as formas de mudança de um nível de execução para outro são controladas estritamente pelo processador.
 - O processador Pentium, por exemplo, conta com 4 níveis de privilégio (sendo 0 o nível mais privilegiado), embora a maioria dos sistemas operacionais construídos para esse processador só use os níveis extremos (0 para o núcleo e drivers do sistema operacional e 3 para utilitários e aplicações).

- **Nível núcleo:** também denominado nível supervisor, sistema, monitor ou ainda *kernel space*.
 - Para um código executando nesse nível, todo o processador está acessível: todos os recursos internos do processador (registradores e portas de entrada/saída) e áreas de memória podem ser acessados. Além disso, todas as instruções do processador podem ser executadas.
 - Ao ser ligado, o processador entra em operação neste nível.

- **Nível usuário** (ou *userspace*): neste nível, somente um subconjunto das instruções do processador, registradores e portas de entrada/saída estão disponíveis.
 - Instruções “perigosas” como **HALT** (parar o processador) e **RESET** (reiniciar o processador) são proibidas para todo código executando neste nível.
 - O *hardware* restringe o uso da memória, permitindo o acesso somente a áreas previamente definidas.
 - Caso o código em execução tente executar uma instrução proibida ou acessar uma área de memória inacessível, o *hardware* irá gerar uma exceção, desviando a execução para uma rotina de tratamento dentro do núcleo, que provavelmente irá abortar o programa em execução (e também gerar a famosa frase “*este programa executou uma instrução ilegal e será finalizado*”, no caso do Windows).

Sistema Operacional

Chamadas de Sistema - *Operações privilegiadas*



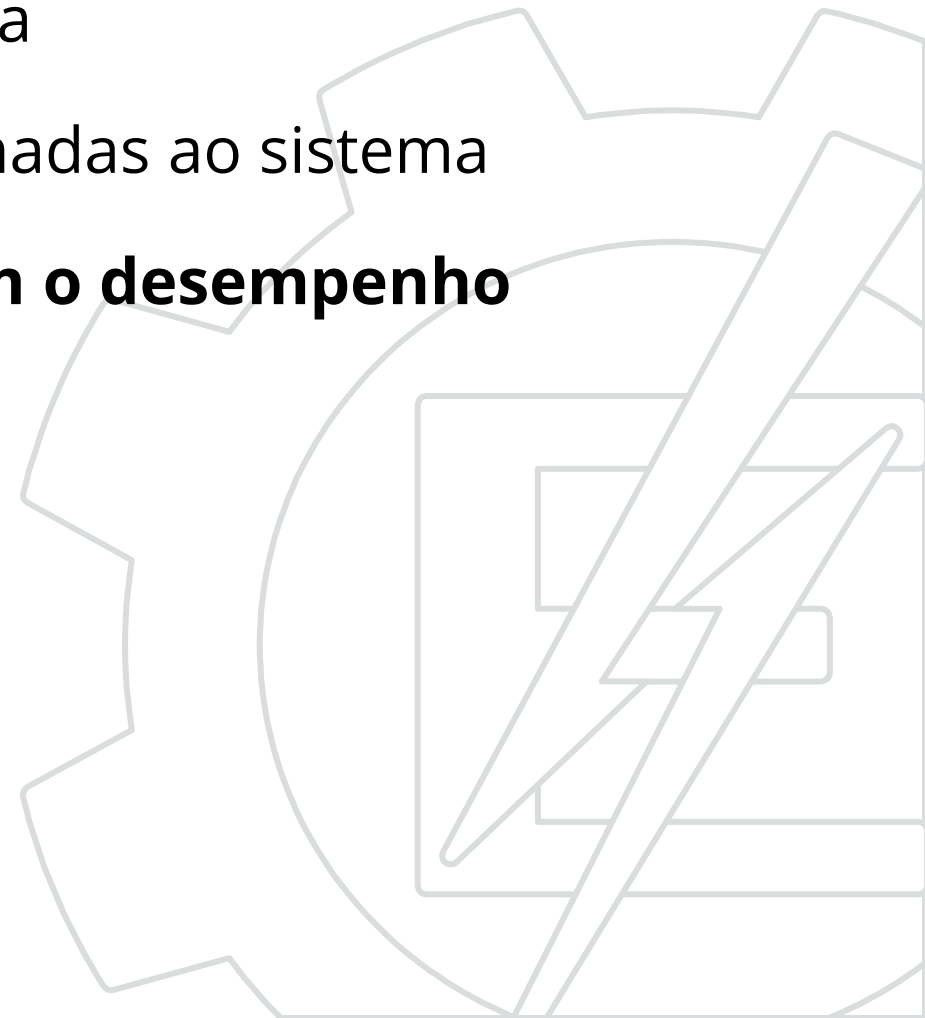
- Interface de programação para serviços fornecidos pelo S.O.
- Normalmente escrita em uma linguagem de alto-nível (C ou C++)
- Na maioria das vezes acessada por programas via uma API (*Application Program Interface*) de alto nível em vez de uma chamada direta ao sistema
- As três APIs mais comuns são **Win32 API** para Windows, **POSIX API** para sistemas baseados em POSIX (incluindo virtualmente todas as versões de UNIX, Linux e Mac OS X), e **Java API** para a máquina virtual Java (JVM)
- Por que usar APIs em vez de chamadas de sistema?

Chamadas de Sistema

Exemplos em Windows e Unix

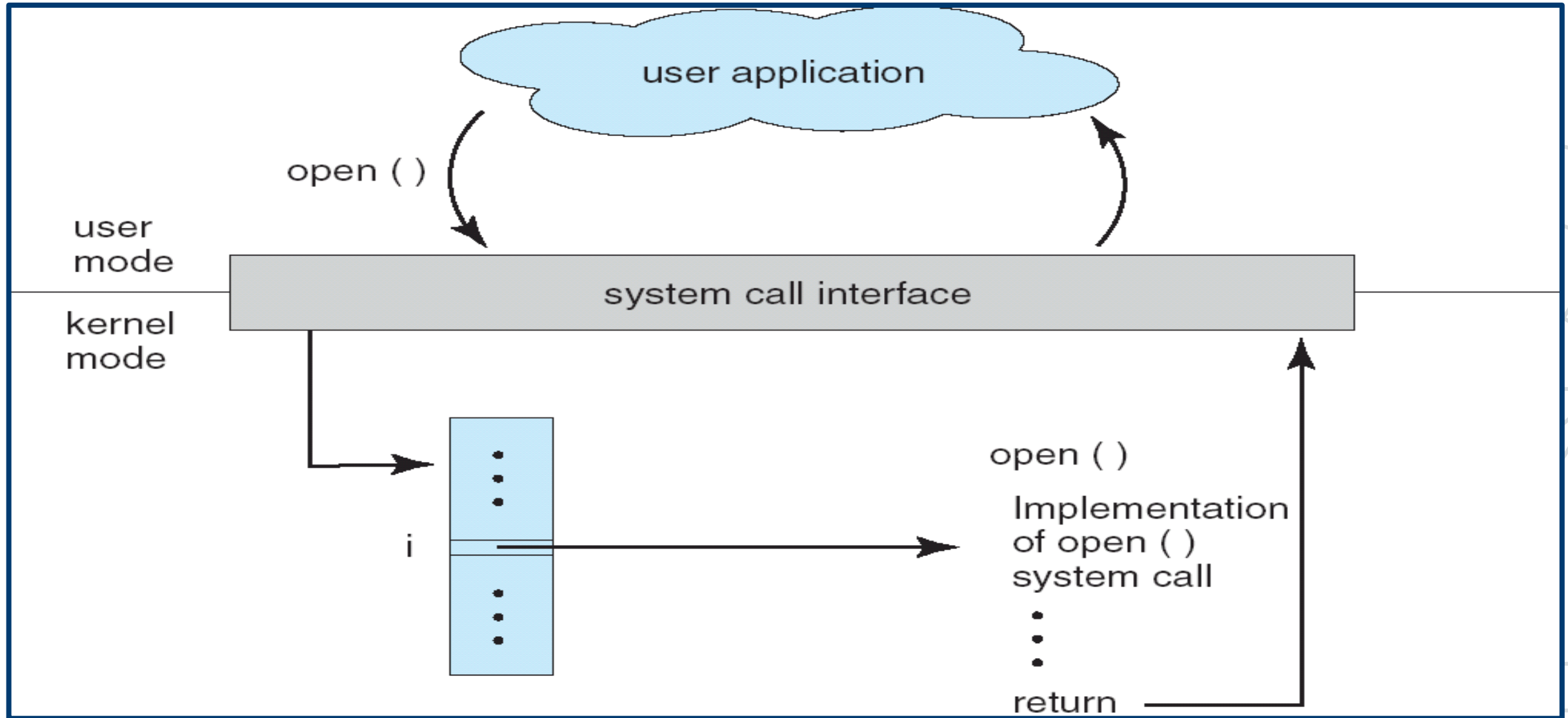
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information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

- Por que usar APIs em vez de chamadas de sistema?
 - **Portabilidade**: independência da plataforma
 - Esconder a **complexidade** inerente às chamadas ao sistema
 - Acréscimo de funcionalidades que **otimizam o desempenho**

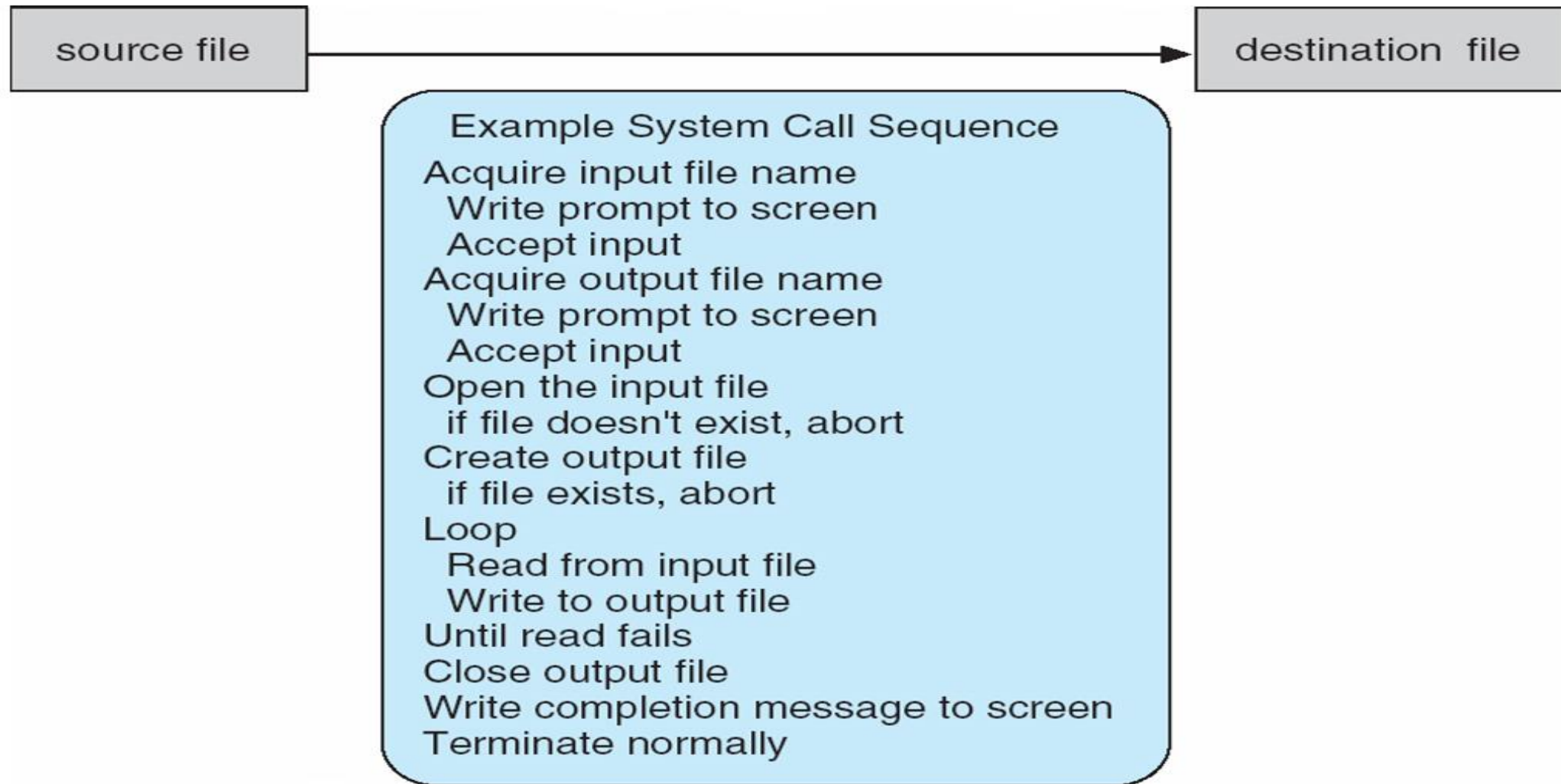


Chamadas de Sistema

Relacionamento: API – Chamada de Sistema - SO



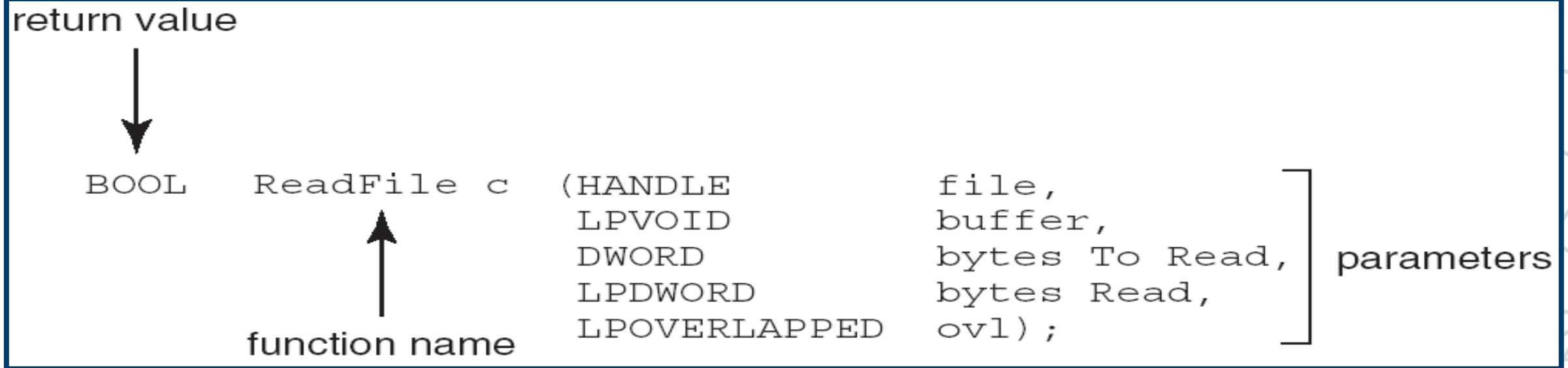
- Sequência de chamadas de sistema para copiar o conteúdo de um arquivo para outro:



Chamadas de Sistema

Exemplo de API Padrão

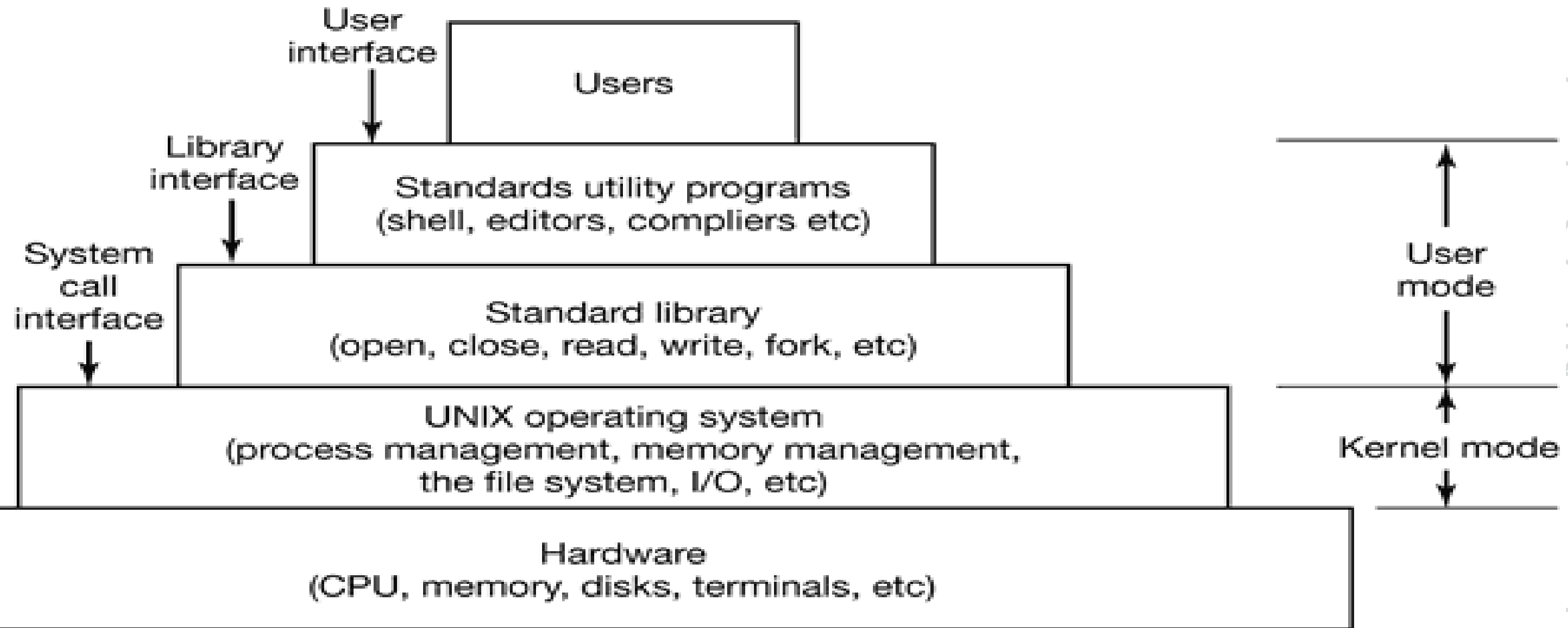
- Considere a função **ReadFile()** na API Win32 —uma função para leitura de um arquivo:



- Uma descrição dos parâmetros passados para **ReadFile()**
 - `HANDLE file`— o arquivo a ser lido
 - `LPVOID buffer`—um buffer de onde os dados serão lidos e gravados
 - `DWORD bytesToRead`—o número de bytes a ser lido no buffer
 - `LPDWORD bytesRead`— o número de bytes lidos durante a última leitura
 - `LPOVERLAPPED ovl`—indica se E/S **sobreposta** está sendo usada

Sistema Operacional

Chamadas de Sistema



Chamadas de Sistema

Implementação

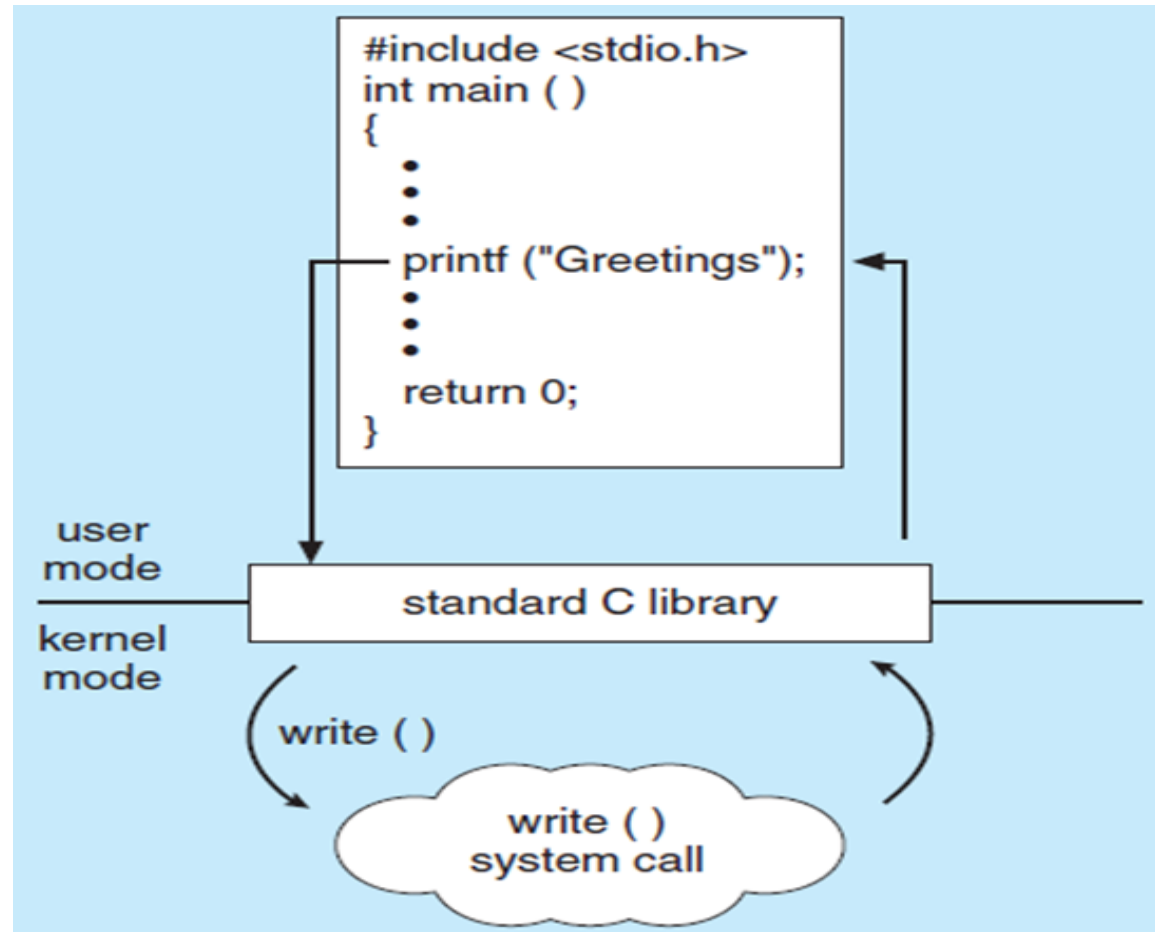
- Normalmente, um número associado com cada chamada de sistema. A interface de chamada de sistema mantém uma tabela indexada de acordo com estes números;
- A interface de chamada de sistema invoca a chamada de sistema solicitada no *kernel* do SO e retorna o *status* da chamada do sistema e quaisquer valores de retorno;
- O código cliente não precisa saber como a chamada de sistema é implementada, apenas precisa obedecer a API e entender o que o SO faz como resultado da chamada. A maioria dos detalhes da interface com o SO é escondida do programador pela API .

Gerenciada por biblioteca de suporte em tempo de execução (conjunto de funções incorporadas em bibliotecas, incluídas com o compilador)

Chamadas de Sistema

Exemplo de uma biblioteca padrão C

- Programa C invocando a chamada de biblioteca ***printf()***, que invoca a chamada de sistema ***write()***



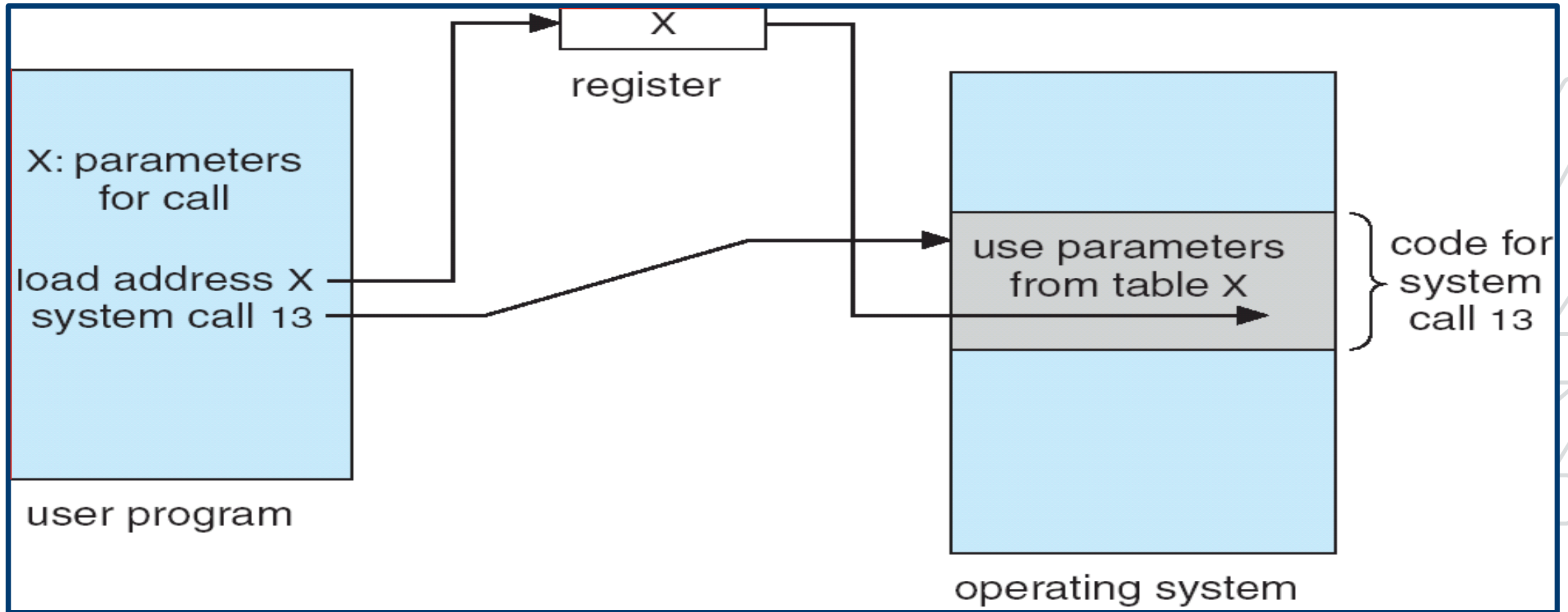
Chamadas de Sistema

Passagem de parâmetros

- Três métodos em geral são usados para passar parâmetros para o SO
 - **Registradores** (Mais simples): passar os parâmetros em *registradores*. Em alguns casos, podem existir mais parâmetros do que registradores
 - **Parâmetros armazenados em um *bloco*, ou *tabela*, em memória**, e endereço do bloco passado como parâmetro em um registrador. Esta é a abordagem utilizada pelo Linux e Solaris.
 - **Parâmetros localizados ou *inseridos na stack*** pelo programa e *retirados da stack* pelo sistema operacional.
- Os métodos de bloco e *stack* não limitam o número ou tamanho dos parâmetros sendo passados

Chamadas de Sistema

Passagem de parâmetros via tabela



Chamadas de Sistema

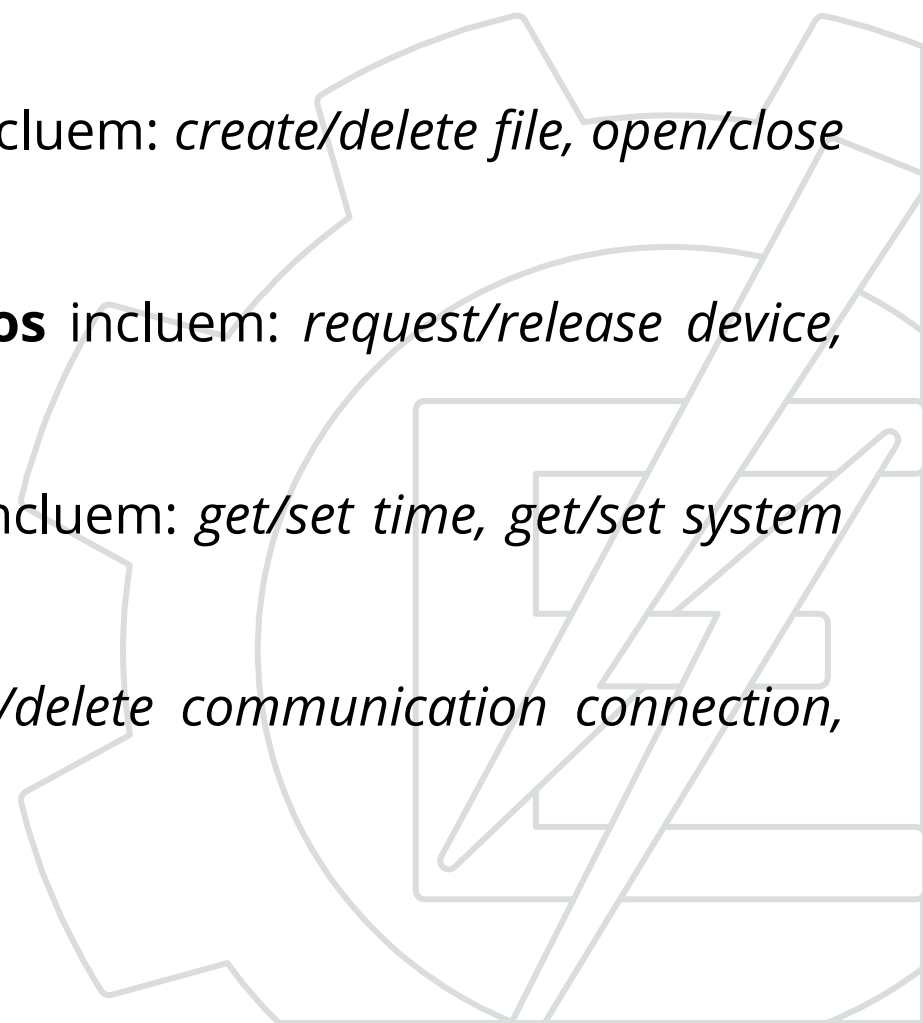
Tipos

- Controle de processo
- Gerenciamento de arquivos
- Gerenciamento de dispositivo
- Manutenção da informação
- Comunicação
- Proteção



Chamadas de Sistema

Exemplos em Windows e Unix

- As chamadas de sistema de **processos** incluem: *end, abort, load, execute, create/terminate process, wait, allocate/free memory*
 - As chamadas de sistemas de **gerenciamento de arquivos** incluem: *create/delete file, open/close file, read, write, get/set attributes*
 - As chamadas de sistema de **gerenciamento de dispositivos** incluem: *request/release device, read, write, logically attach/detach devices*
 - As chamadas de sistema de **manutenção da informação** incluem: *get/set time, get/set system data, get/set process/file/device attributes*
 - As chamadas de sistema de **comunicação** incluem: *create/delete communication connection, send/receive, transfer status information*
- 

Programas de Sistema

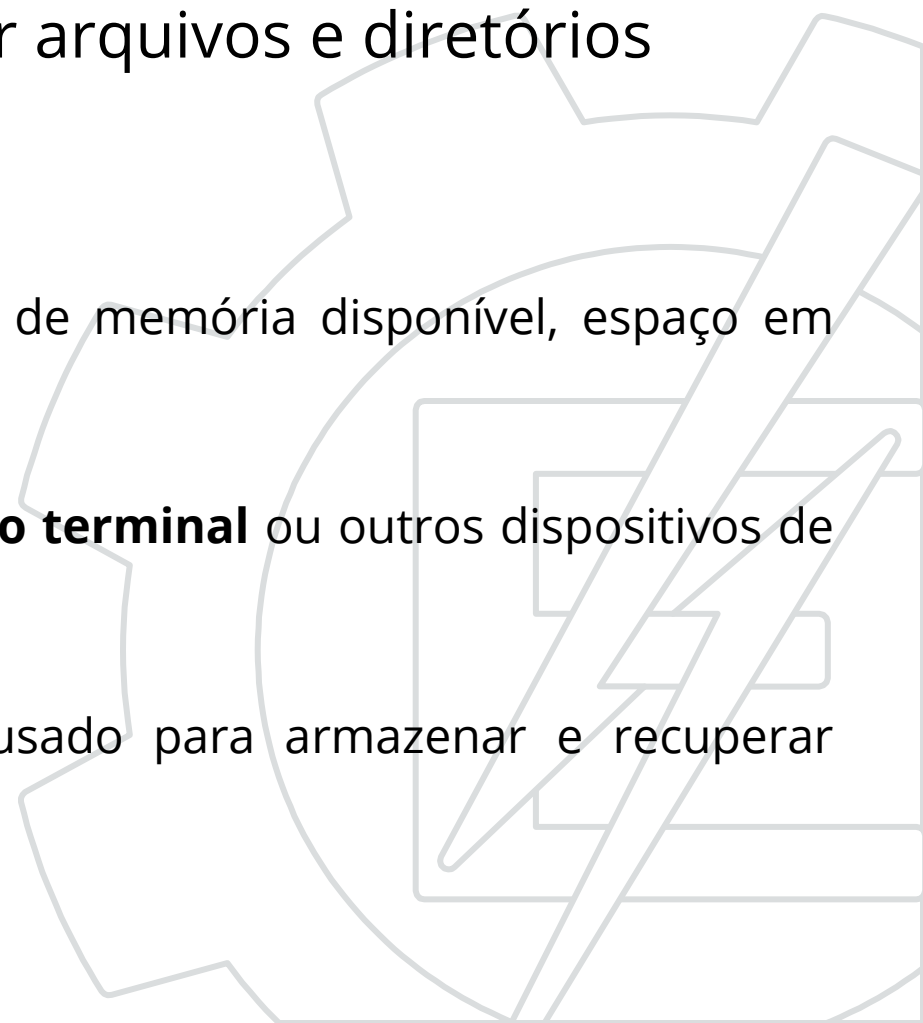
- Programas de sistema fornecem um ambiente conveniente para o desenvolvimento e execução de programas. Podem ser divididos em:
 - Manipulação de arquivos
 - Informação de status
 - Modificação de arquivos
 - Suporte à linguagens de programação
 - Carregamento e execução de programas
 - Comunicação
 - Programas aplicativos



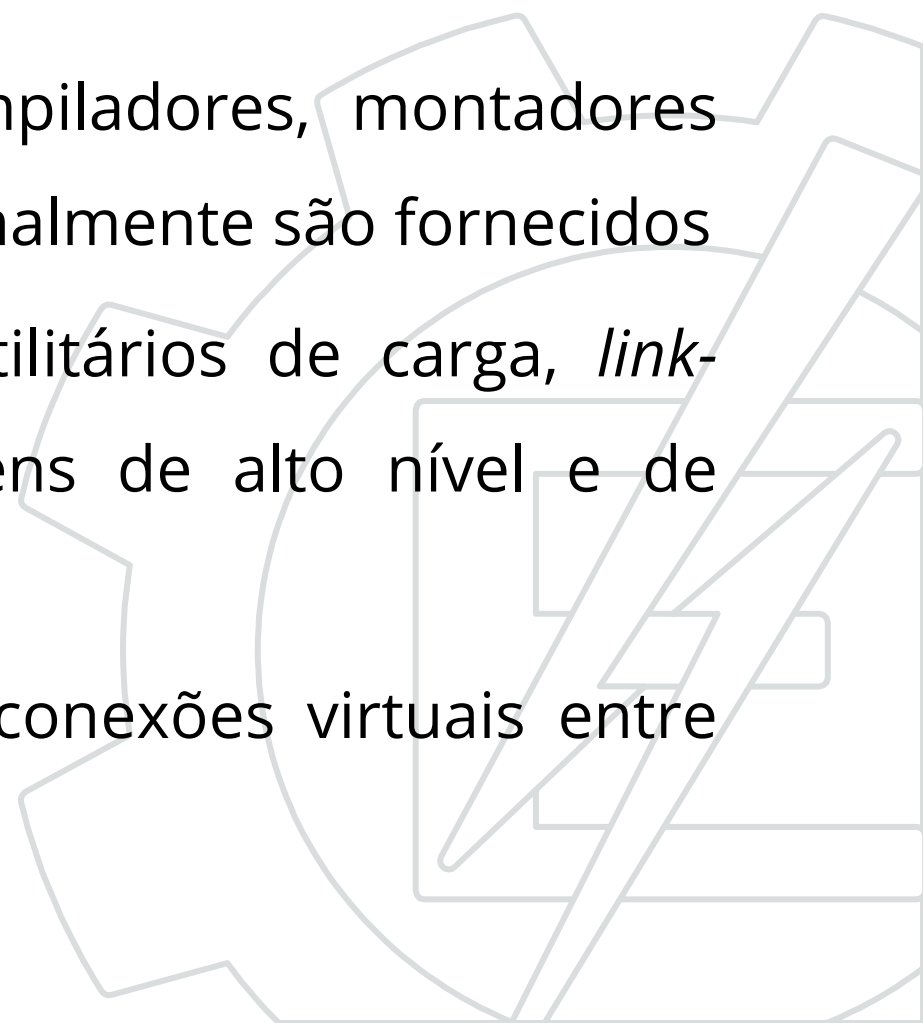
Programas de Sistema

- Programas de sistema fornecem um ambiente conveniente para o desenvolvimento e execução de programas.
- Fornecem ambiente conveniente para desenvolvimento e execução de programas. Alguns deles são simplesmente interfaces entre o usuário e chamadas do sistema; outros são consideravelmente mais complexos
- A **visão da maioria dos usuários** do sistema operacional é **definida pelos programas do sistema**, não pelas chamadas de sistema.

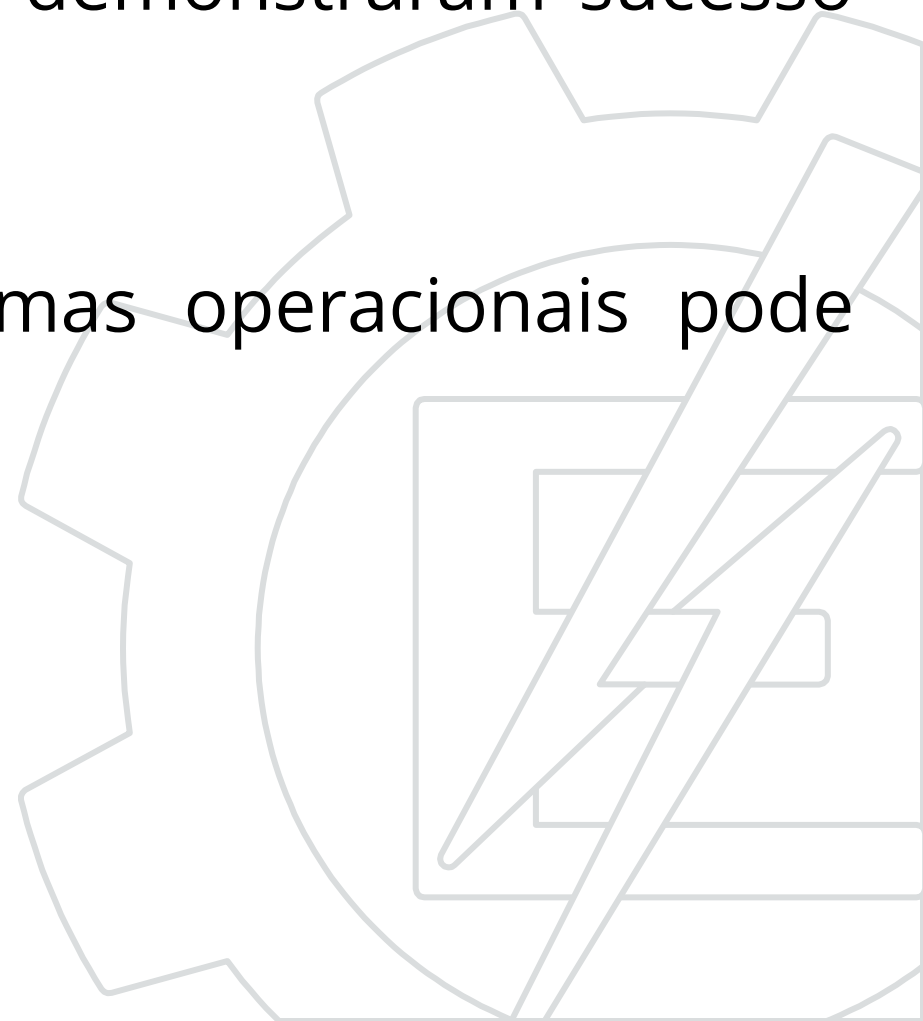
Programas de Sistema

- **Gerenciamento de arquivos** - Criar, remover, copiar, renomear, imprimir, efetivação de *dump*, listar e geralmente manipular arquivos e diretórios
 - **Informação de status**
 - O sistema **fornece informações** - data, hora, quantidade de memória disponível, espaço em disco, número de usuário
 - Tipicamente, estes programas **formatam e geram a saída no terminal** ou outros dispositivos de saída
 - Alguns sistemas implementam um **registro (registry)** - usado para armazenar e recuperar informação de configuração
- 

Programas de Sistema

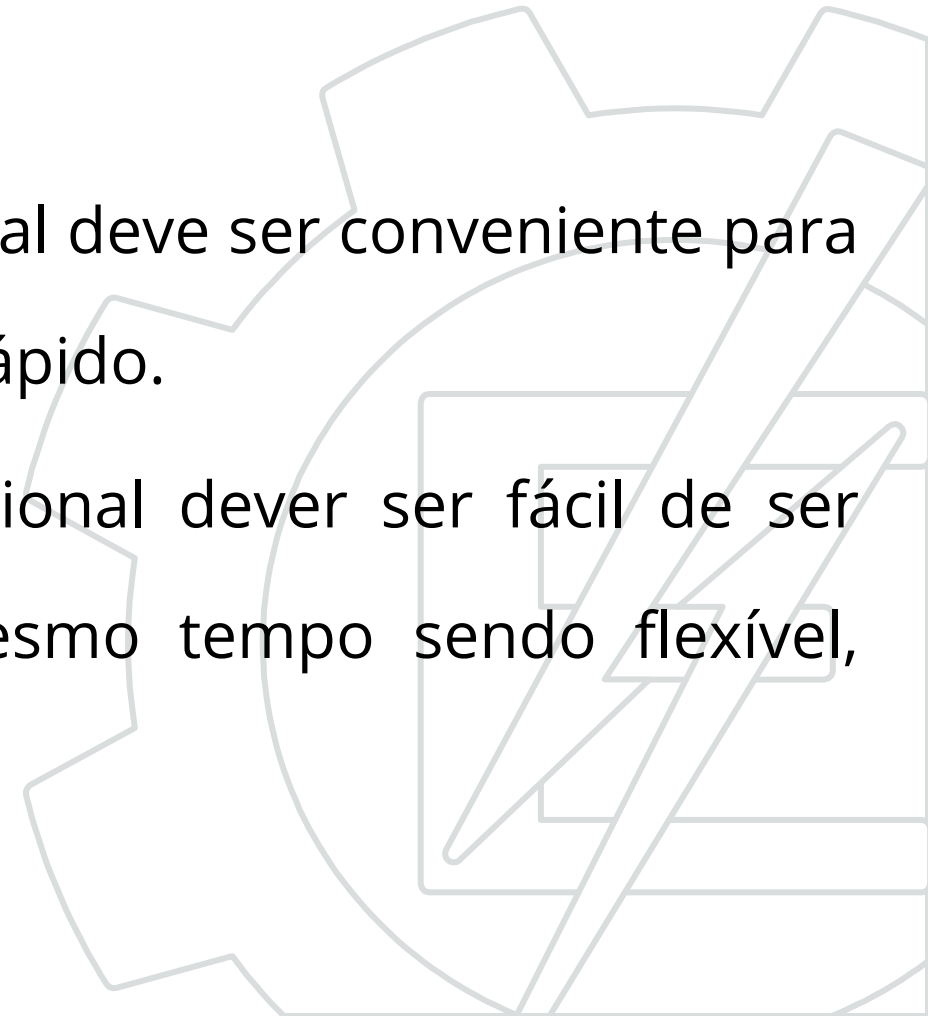
- **Modificação de arquivos:** Editores de texto para criar e modificar arquivos.
 - **Suporte à linguagens de programação:** Compiladores, montadores (*assemblers*), depuradores e interpretadores normalmente são fornecidos
 - **Carregamento e execução de programas:** utilitários de carga, *link-editores*, sistemas de depuração para linguagens de alto nível e de máquina
 - **Comunicação:** Mecanismos para a criação de conexões virtuais entre processos, usuários e entre computadores.
- 

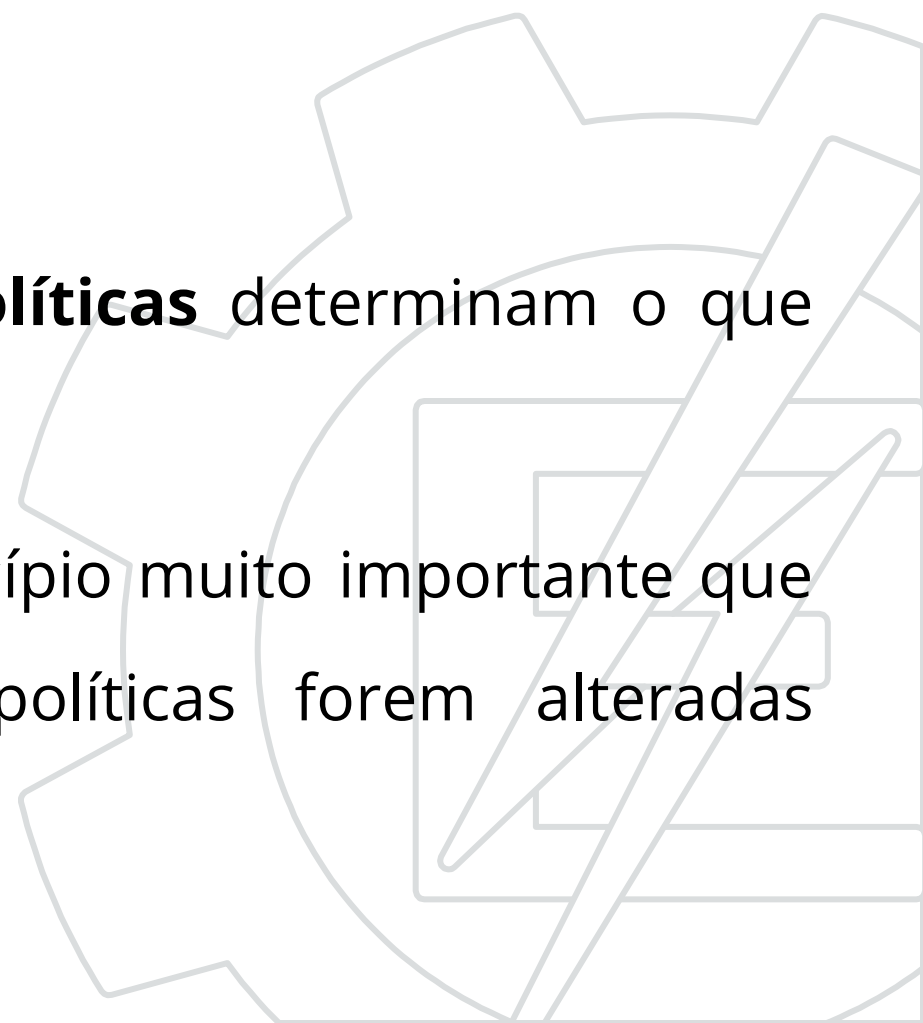
- O projeto e implementação de sistemas operacionais não possui solução única, mas algumas abordagens demonstraram sucesso com o passar do tempo.
- A estrutura interna de diferentes sistemas operacionais pode variar drasticamente
- Inicie definindo objetivos e especificações



• O projeto e implementação são afetados pela escolha do *hardware* e do tipo do sistema:

- **Objetivos do usuário** – um sistema operacional deve ser conveniente para uso, de fácil aprendizado, confiável, seguro e rápido.
- **Objetivos do sistema** – um sistema operacional deve ser fácil de ser projetado, implementado e mantido, ao mesmo tempo sendo flexível, confiável, livre de erros e eficiente.

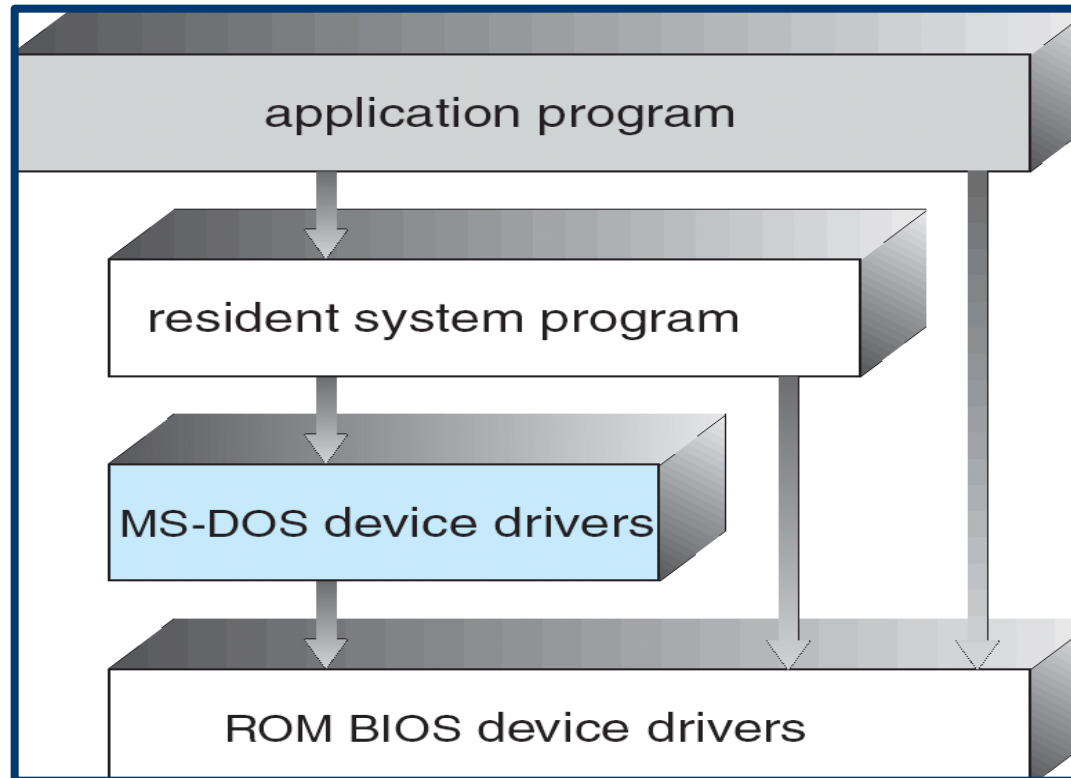


- Princípios:
 - **Política:** O que será feito?
 - **Mecanismo:** Como fazer?
 - **Mecanismos** determinam como fazer algo e **políticas** determinam o que será feito.
 - A separação de política e mecanismo é um princípio muito importante que permite máxima **flexibilidade** se decisões políticas forem alteradas posteriormente.
- 

Sistemas Operacionais

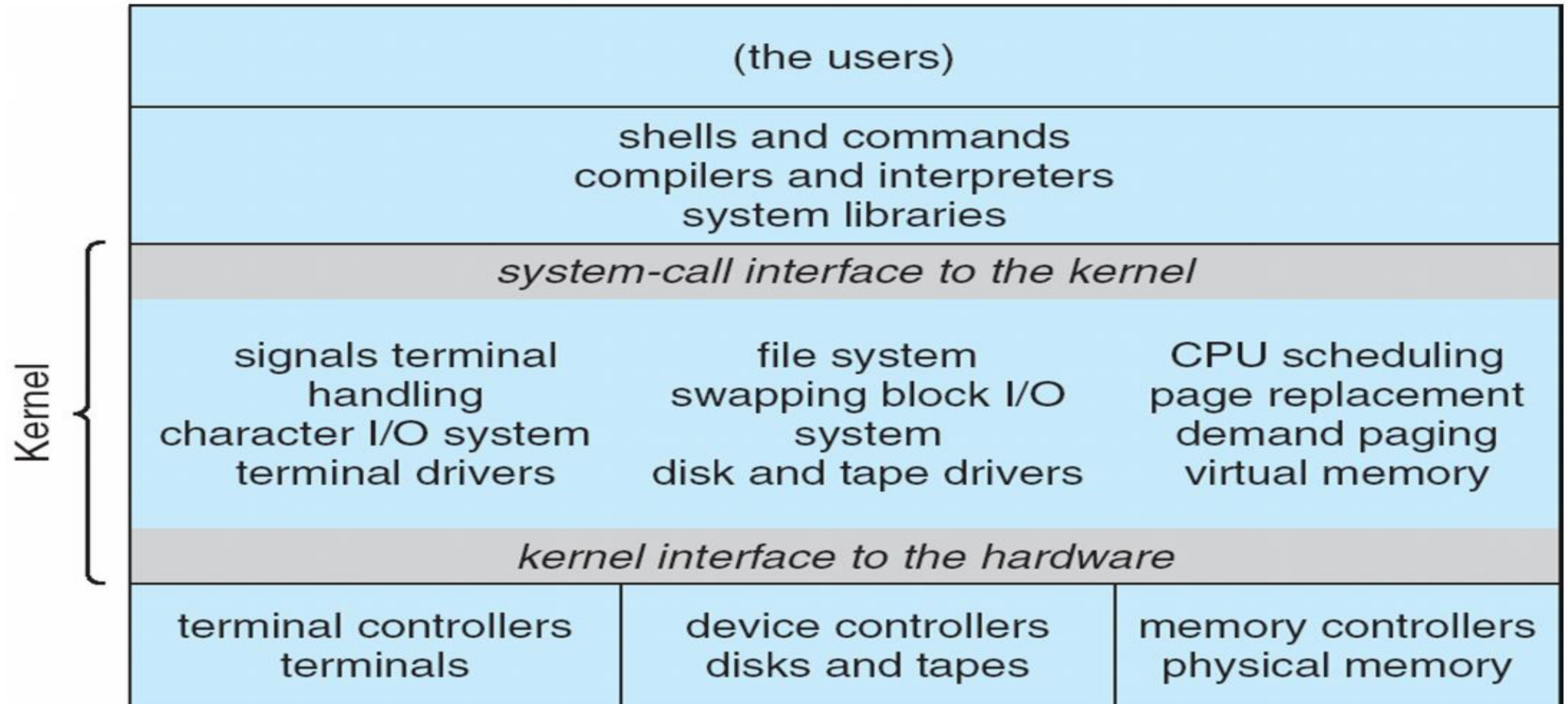
MS-DOS - Estrutura simples

- Escrito para fornecer mais funcionalidade em menos espaço.
- Não foi dividido em módulos. Estrutura em camadas.



Sistemas Operacionais

Estrutura de um UNIX tradicional



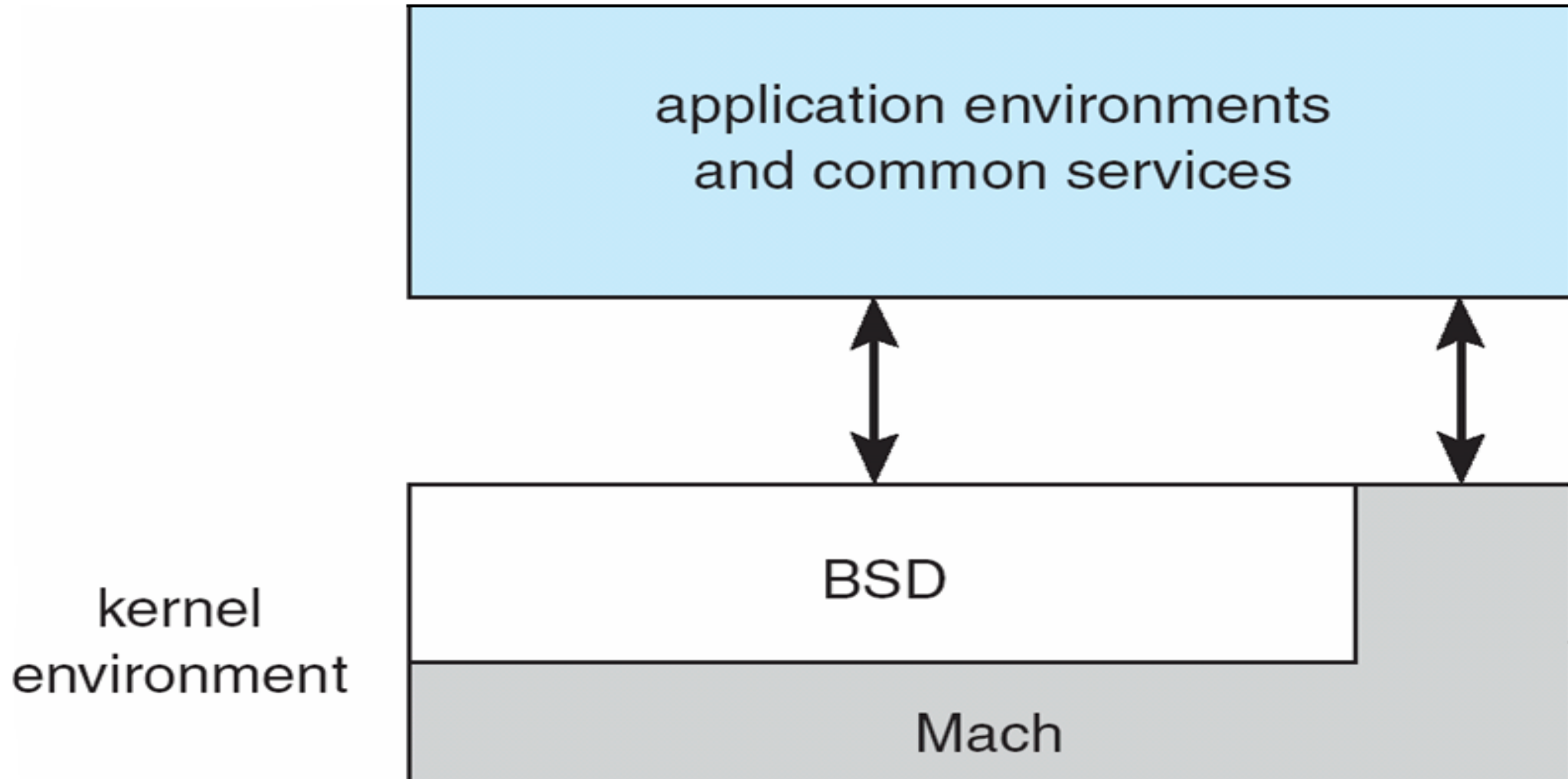
- UNIX – limitado pela funcionalidade do hardware, o **sistema UNIX original** possuía uma estruturação limitada.
- O sistema operacional UNIX consiste de duas partes separadas:
Programas do sistema e o kernel.

Consiste de tudo abaixo da interface de chamadas do sistema e acima do *hardware*

Fornece o sistema de arquivos, escalonamento de CPU, gerenciamento de memória, e outras funções de um sistema operacional; um extenso conjunto de funções para um nível único

Sistemas Operacionais

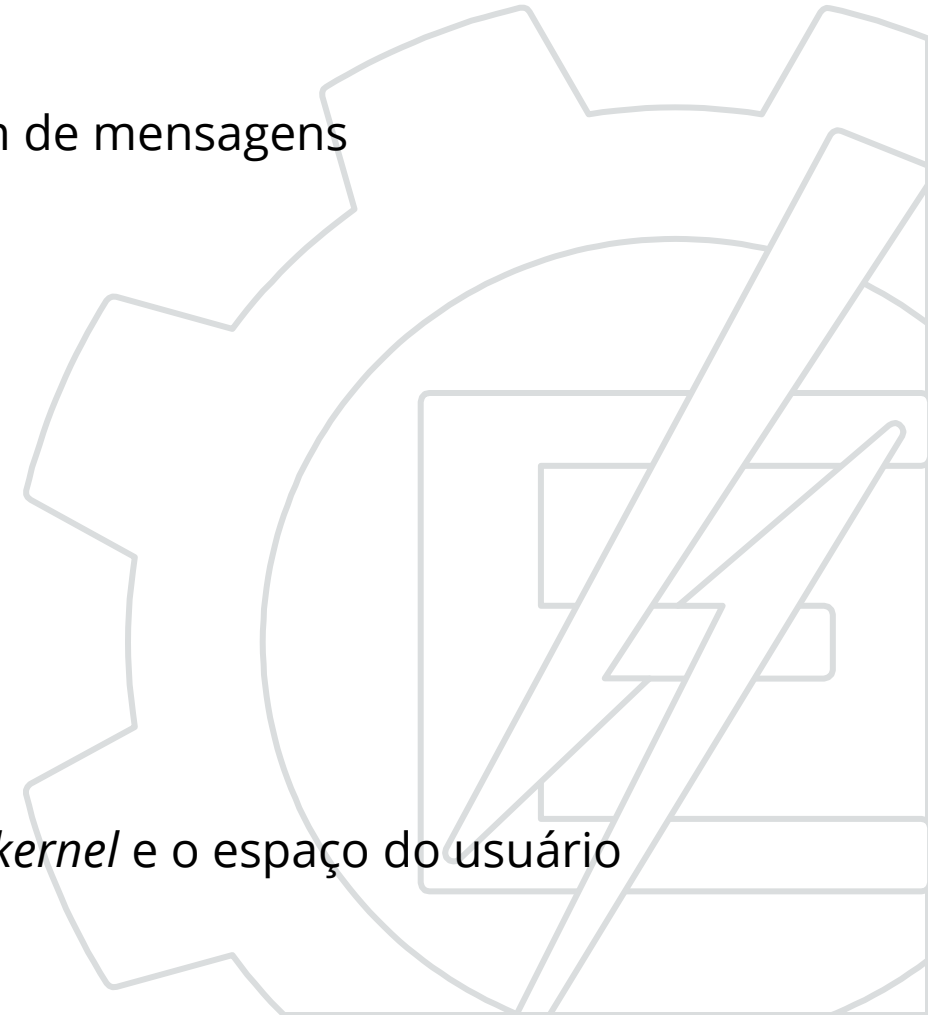
Estrutura de um MAC OS X



Sistemas Operacionais

Estrutura de um sistema com *microkernel*

- Remove os componentes não-essenciais do *kernel* implementando-os como programas de sistema e de nível de usuário
- A comunicação ocorre entre módulos do usuário usando passagem de mensagens
- Benefícios:
 - Mais fácil estender o *microkernel*
 - Mais fácil portar o sistema operacional para novas arquiteturas
 - Mais confiável (menos código executando no modo *kernel*)
 - Mais seguro
- Desvantagem:
 - **Overhead** de desempenho na comunicação entre o espaço do *kernel* e o espaço do usuário

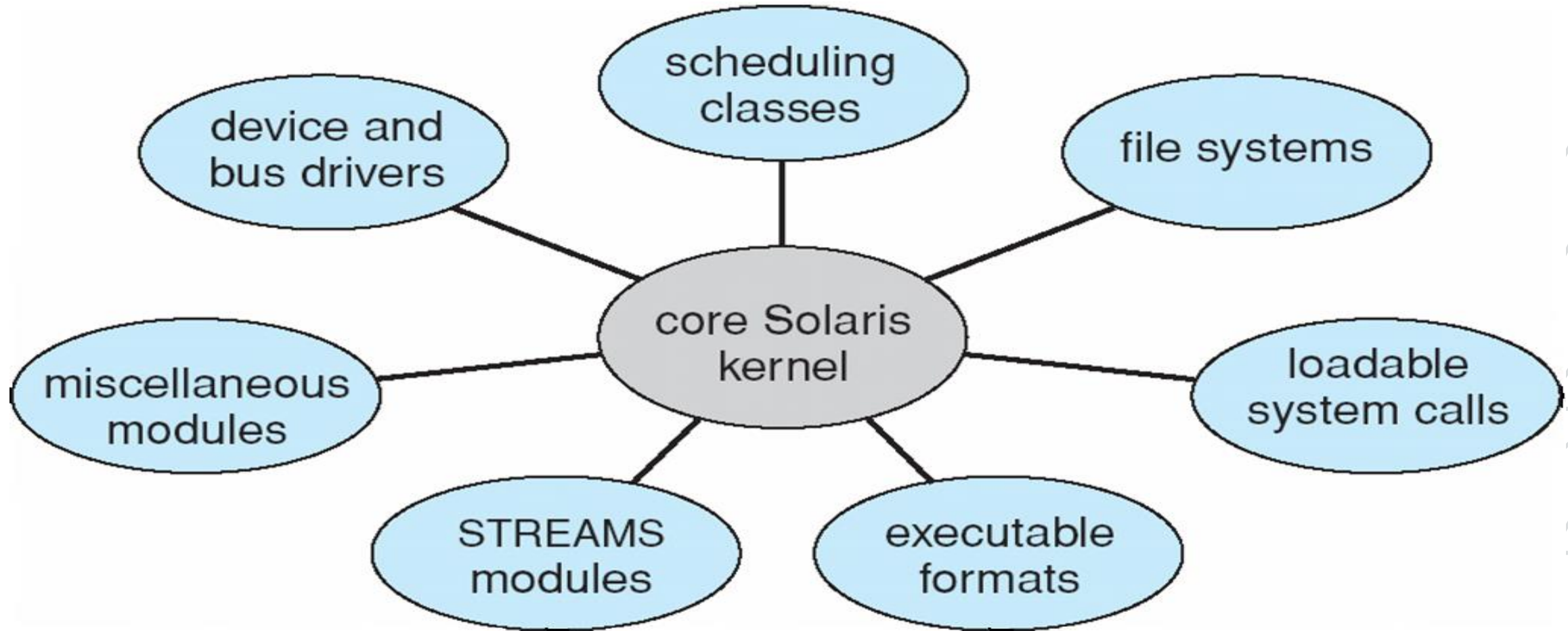


- A maioria dos sistemas operacionais modernos implementam a abordagem de microkernels:
 - Utiliza a abordagem orientada a objetos
 - Cada componente principal é separado
 - Comunicação através de interface conhecidas
 - Cada módulo é carregado quando necessário dentro do kernel
- De forma geral, similar à camadas mas com mais flexibilidade

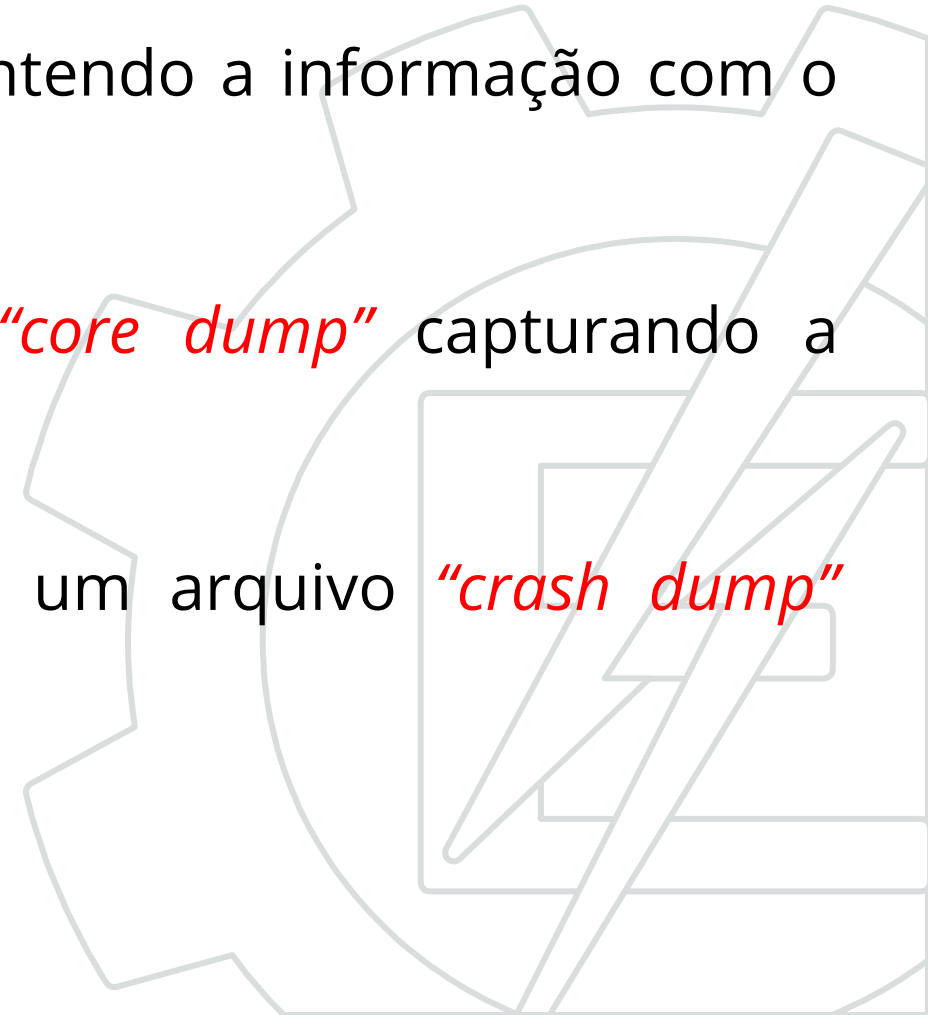


Sistemas Operacionais

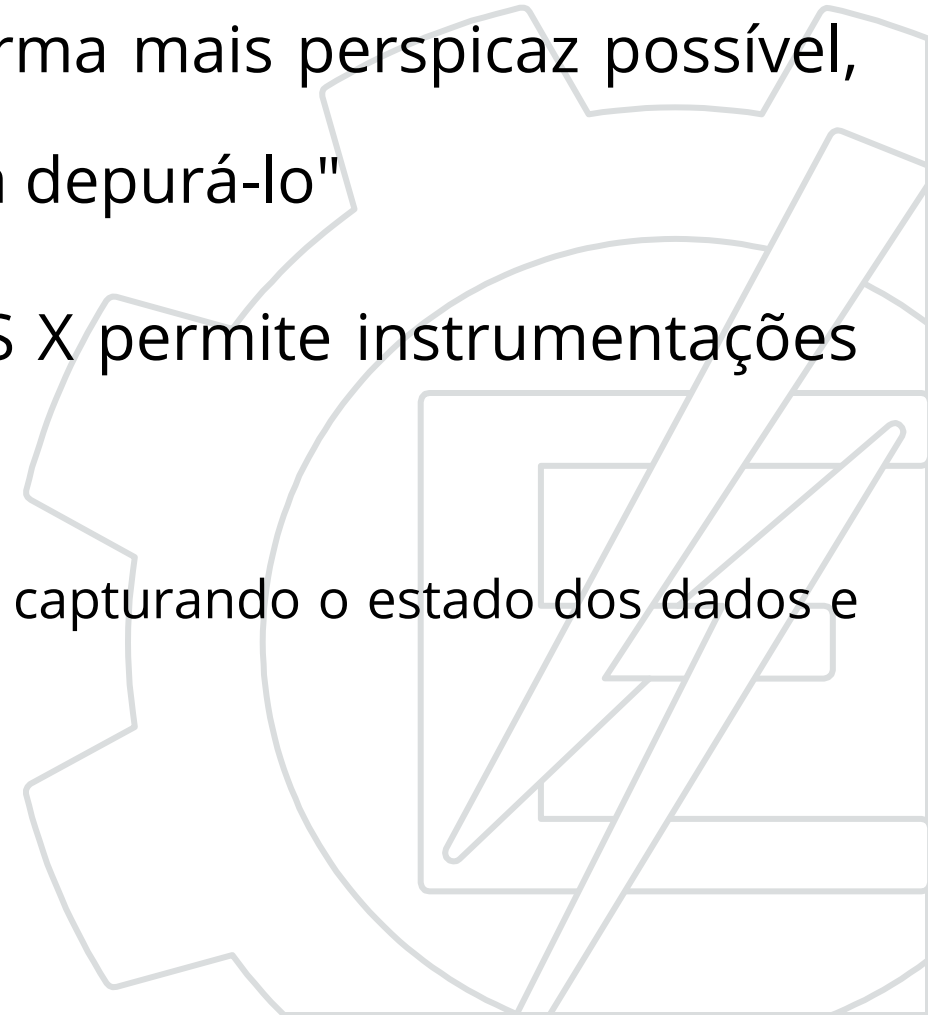
Abordagem modular do Solaris



- Depuração é encontrar e corrigir erros ou *bugs*
- Sistemas operacionais geram *arquivos de log* contendo a informação com o erro
- Falha na aplicação pode gerar um arquivo *"core dump"* capturando a memória do processo
- Uma falha no sistema operacional pode gerar um arquivo *"crash dump"* contendo a memória do kernel



- **Lei de Kernighan's** : "A depuração é duas vezes mais difícil do que escrever código. Portanto, se você escrever o código da forma mais perspicaz possível, você é, por definição, não esperto o suficiente para depurá-lo"
- A ferramenta **DTrace** no Solaris, FreeBSD, Mac OS X permite instrumentações em tempo real em sistemas em produção
 - **Consultas** são disparadas quando o código é executado, capturando o estado dos dados e enviando-os para os clientes dessas consultas



Sistemas Operacionais

para uma configuração específica de máquina

- Sistemas operacionais são projetados para execução em qualquer máquina de uma determinada classe; o sistema deve ser configurado ou gerado para cada local de instalação específico, um processo às vezes denominado **geração de sistema** (SYSGEN)
- O programa SYSGEN obtém informação sobre a configuração específica do sistema de *hardware*
- Uma vez gerado o sistema, ele deve ser iniciado através do **processo de boot**– iniciar um computador carregando o *kernel*
- **Programa Bootstrap** – código armazenado em **ROM** que é capaz de localizar o *kernel*, carregá-lo em memória e iniciar sua execução

- O sistema operacional deve estar disponível para que o hardware possa iniciá-lo
 - Um pequeno pedaço de código – **carregador de *bootstrap***, localiza o kernel, o carrega em memória e o inicia.
 - Algumas vezes esse processo se dá em dois passos com um **bloco de boot** em uma localização fixa inicia o carregador de bootstrap
 - Quando o computador é ligado, a execução inicia em uma localização fixa da memória.

Um *firmware* é usado para manter o código inicial de boot

Sistemas Operacionais

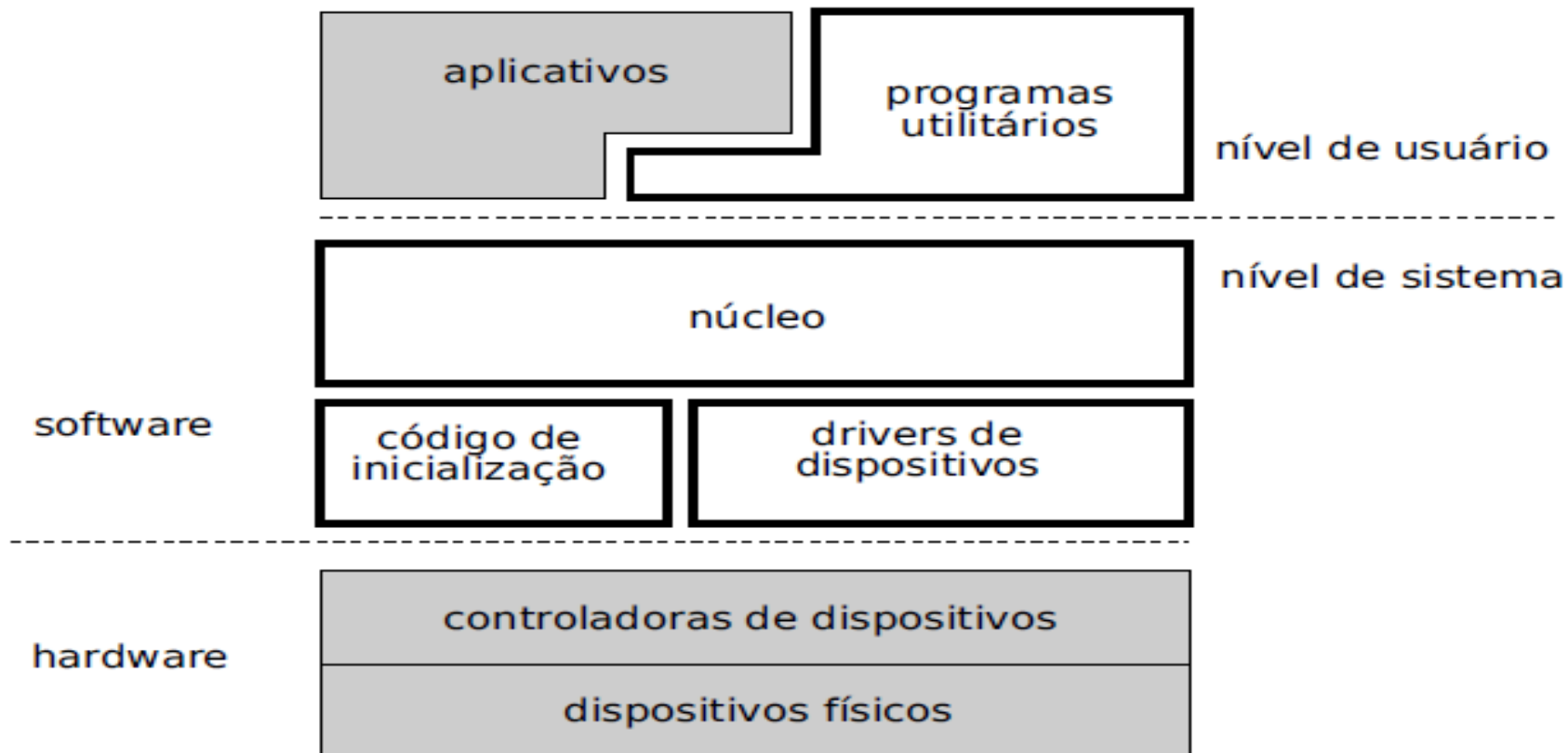
Boot do sistema



Sistemas Operacionais

Síntese

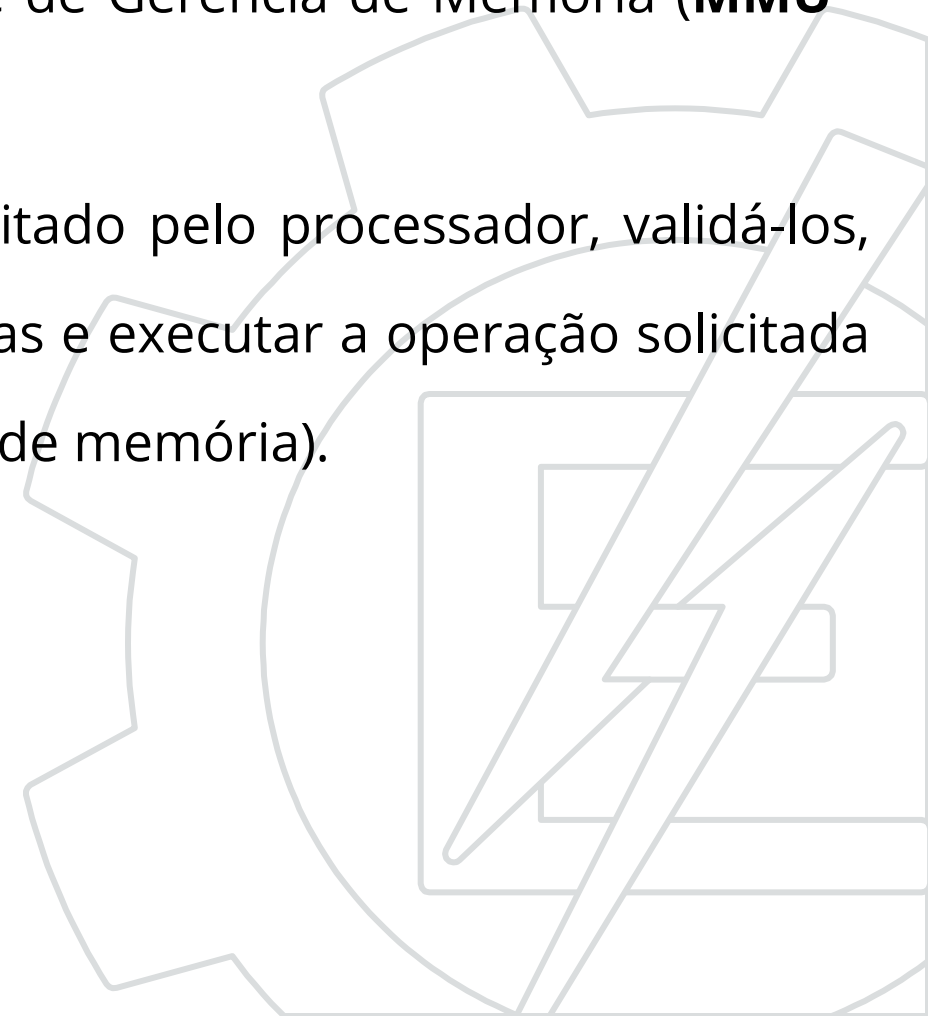
- Um sistema operacional não é um bloco único e fechado de *software* executando sobre o hardware. Na verdade, ele é composto de diversos componentes com objetivos e funcionalidades complementares.



- **Núcleo (kernel):** é o coração do sistema operacional, responsável pela gerência dos recursos do hardware usados pelas aplicações. Ele também implementa as principais abstrações utilizadas pelos programas aplicativos.
- **Drivers:** módulos de código específicos para acessar os dispositivos físicos. Existe um *driver* para cada tipo de dispositivo, como discos rígidos IDE, SCSI, portas USB, placas de vídeo, etc. Muitas vezes o *driver* é construído pelo próprio fabricante do *hardware* e fornecido em forma compilada (em linguagem de máquina) para ser acoplado ao restante do sistema operacional.
- **Código de inicialização:** a inicialização do *hardware* requer uma série de tarefas complexas, como reconhecer os dispositivos instalados, testá-los e configurá-los adequadamente para seu uso posterior. Outra tarefa importante é carregar o núcleo do sistema operacional em memória e iniciar sua execução.

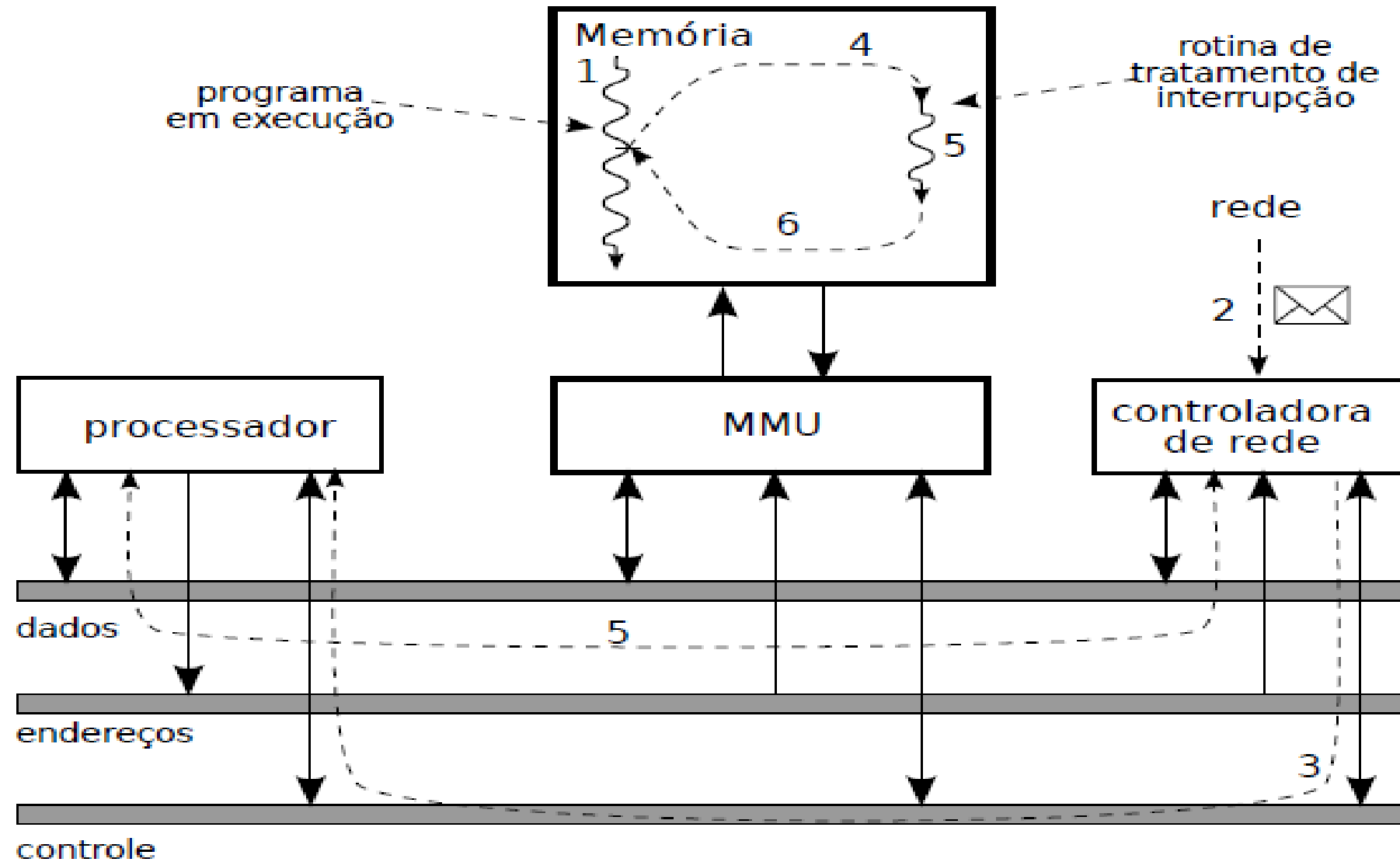
- **Programas utilitários:** são programas que facilitam o uso do sistema computacional, fornecendo funcionalidades complementares ao núcleo, como formatação de discos e mídias, configuração de dispositivos, manipulação de arquivos (mover, copiar, apagar), interpretador de comandos, terminal, interface gráfica, gerência de janelas, etc.
- O núcleo do sistema de computação é o **processador**. Todas as transferências de dados entre processador, memória e periféricos são feitas através dos barramentos: o barramento de endereços indica a posição de memória (ou o dispositivo) a acessar, o barramento de controle indica a operação a efetuar (leitura ou escrita) e o barramento de dados transporta a informação indicada entre o processador e a memória ou um controlador de dispositivo.

- O acesso à memória é geralmente mediado por um controlador específico (que pode estar fisicamente dentro do próprio processador): a Unidade de Gerência de Memória (**MMU** - *Memory Management Unit*).
 - Ela é responsável por analisar cada endereço solicitado pelo processador, validá-los, efetuar as conversões de endereçamento necessárias e executar a operação solicitada pelo processador (leitura ou escrita de uma posição de memória).



Sistemas Operacionais

Síntese



Bibliografia

- TANENBAUM, Andrew S; BOS, Herbert. Sistemas operacionais modernos. 4a ed. São Paulo: Pearson Education do Brasil, 2016.

Capítulo 1.

<https://plataforma.bvirtual.com.br/Acervo/Publicacao/1233>

- DEITEL, H.M; DEITEL, P.J; CHOFFNES,D.R. Sistemas Operacionais. 3a ed. São Paulo: Pearson Prentice Hall, 2005. **Capítulo 1.**

<https://plataforma.bvirtual.com.br/Acervo/Publicacao/315>



Sistemas Operacionais

Prof. Otávio Gomes

otavio.gomes@unifei.edu.br

