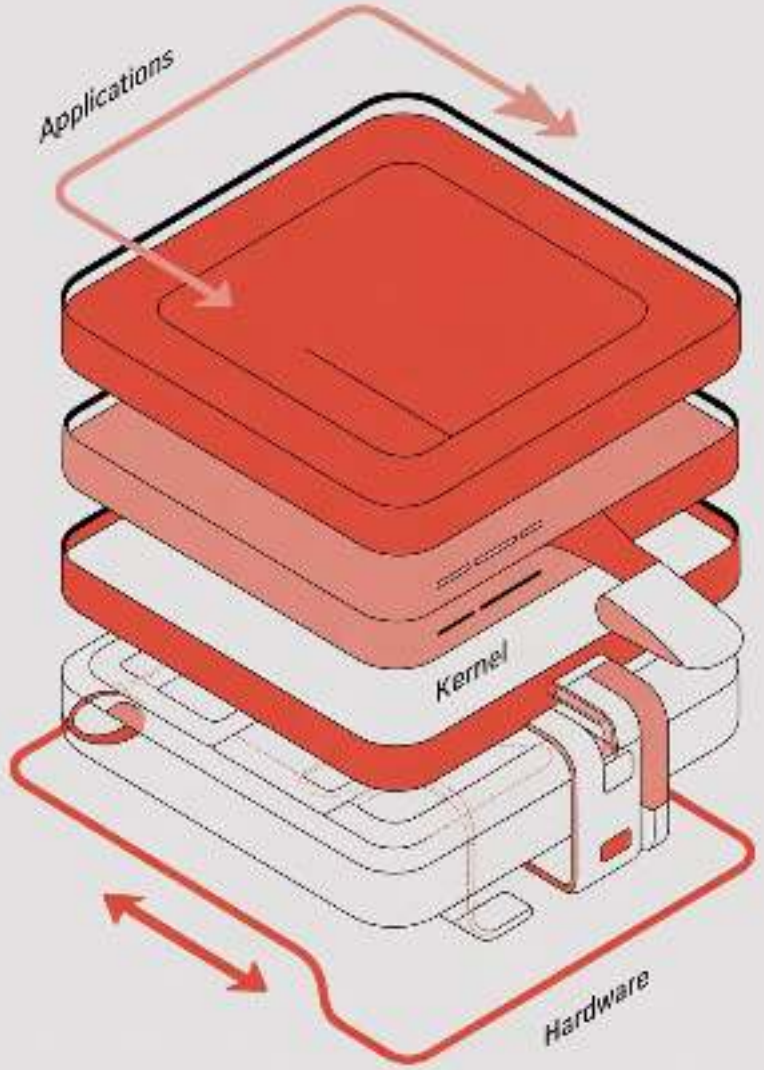
# Estrutura de Sistemas Operacionais

Este curso apresenta uma visão detalhada sobre a estrutura e funcionamento dos sistemas operacionais modernos, explorando seus componentes essenciais e como eles interagem com o hardware para proporcionar serviços às aplicações.



# Conteúdo Programático

1

## Introdução à Estrutura de SOs

Componentes básicos e organização geral

2

## Elementos de Hardware

Arquitetura, interrupções, exceções e níveis de privilégio

3

## Chamadas de Sistema

Mecanismos de comunicação entre aplicações e núcleo

4

## Implementação Prática

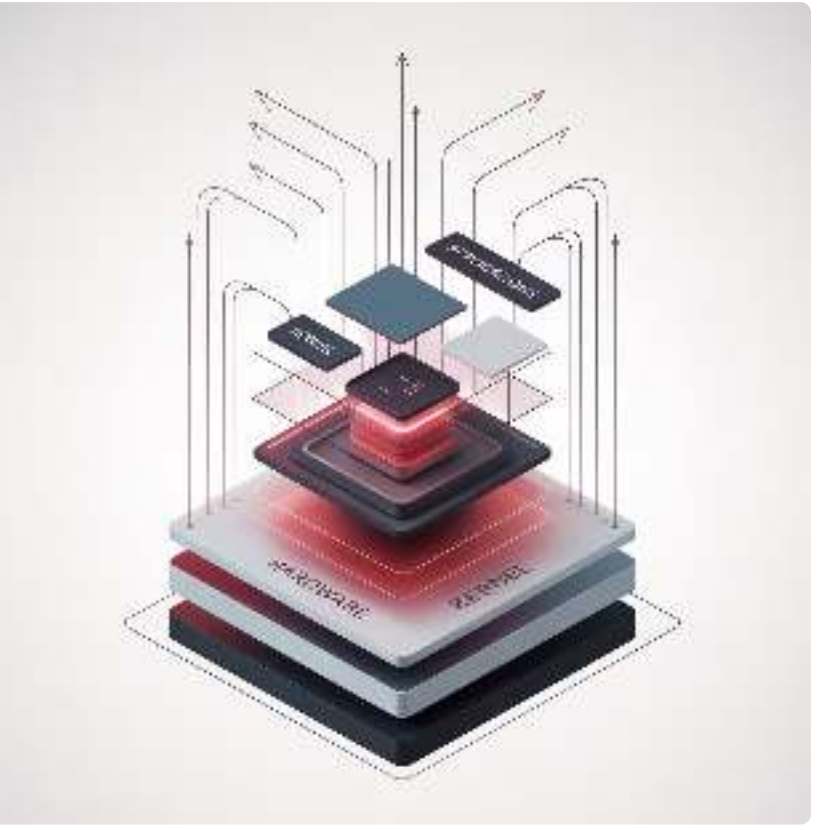
Exemplos reais e casos de estudo

Ao final deste curso, você compreenderá como os sistemas operacionais são estruturados e como seus componentes interagem para fornecer uma camada de abstração sobre o hardware.

# O Que é um Sistema Operacional?

Um sistema operacional **não é um bloco único e fechado** de software executando sobre o hardware, mas sim um conjunto de componentes com objetivos complementares que trabalham juntos para:

- Fornecer uma interface entre o hardware e as aplicações
- Gerenciar recursos do computador (processador, memória, dispositivos)
- Oferecer serviços que facilitam o desenvolvimento de aplicações
- Garantir a segurança e isolamento entre diferentes programas



Os sistemas operacionais modernos são altamente complexos, com milhões de linhas de código organizadas em diversos módulos que interagem entre si.

# Principais Componentes de um SO

## Núcleo (Kernel)

Coração do sistema operacional, responsável pela gerência dos recursos do hardware usados pelas aplicações. Implementa as principais abstrações utilizadas pelos aplicativos.

## Código de Inicialização

Responsável por reconhecer dispositivos instalados, testá-los, configurá-los e carregar o núcleo do sistema operacional em memória para iniciar sua execução.

## Drivers

Módulos específicos para acessar os dispositivos físicos. Existe um driver para cada tipo de dispositivo (discos, USB, placas gráficas). Frequentemente são construídos pelos fabricantes.

## Utilitários

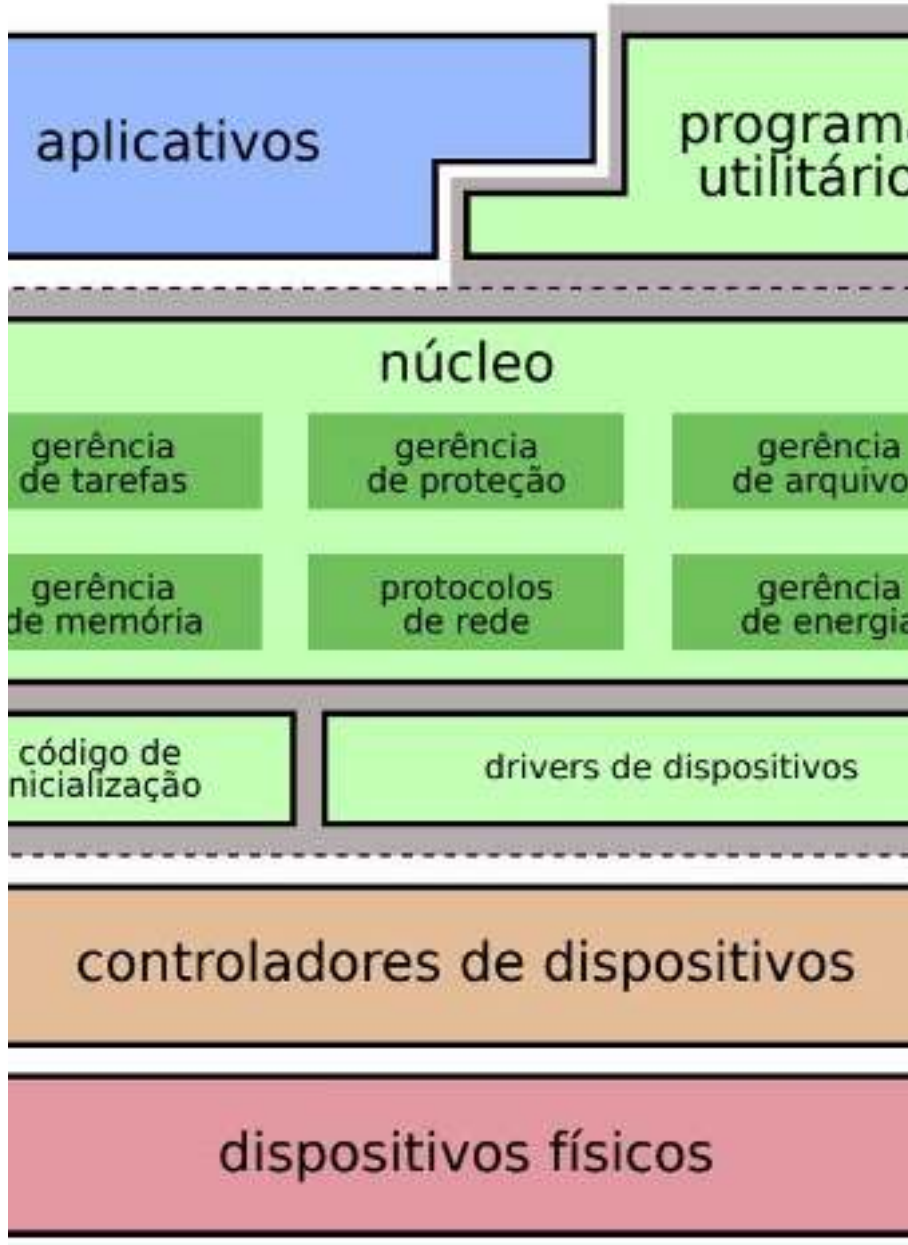
Programas que facilitam o uso do sistema, fornecendo funcionalidades complementares como formatação de discos, manipulação de arquivos, terminal e interface gráfica.

Estes componentes interagem de forma coordenada para fornecer um ambiente completo para execução de aplicações.

# Estrutura de um SO Típico

Na figura acima, a região cinza indica o sistema operacional propriamente dito. A forma como os diversos componentes do SO são interligados varia de sistema para sistema.

O **núcleo** (kernel) executa em modo privilegiado (modo sistema), enquanto programas e aplicações executam em modo usuário, com acesso limitado ao hardware.



# Exemplo de Estrutura Real: Android

## Camadas de Alto Nível

- **System Apps:** Aplicações como discador, email, câmera
- **Java API Framework:** Fornece blocos de construção para aplicativos
- **Content Providers:** Gerenciamento de dados compartilhados
- **Managers:** Gerenciadores de recursos e funcionalidades

## Camadas de Baixo Nível

- **Native Libraries:** Código em C/C++ para performance
- **Android Runtime:** Ambiente de execução (ART)
- **HAL:** Camada de abstração de hardware
- **Linux Kernel:** Base do sistema (gerenciamento de processos, memória, drivers)

O Android é um exemplo de sistema operacional complexo baseado no Linux, com uma rica estrutura de bibliotecas e serviços no nível de usuário para facilitar o desenvolvimento de aplicações.

# Elementos de Hardware

Para que o sistema operacional possa cumprir suas funções com eficiência e confiabilidade, o hardware deve fornecer algumas funcionalidades básicas:

### Arquitetura do Computador

Organização física dos componentes (processador, memória, periféricos) e como se comunicam

### Mecanismos de Interrupção

Permitem a comunicação eficiente entre o processador e os dispositivos periféricos

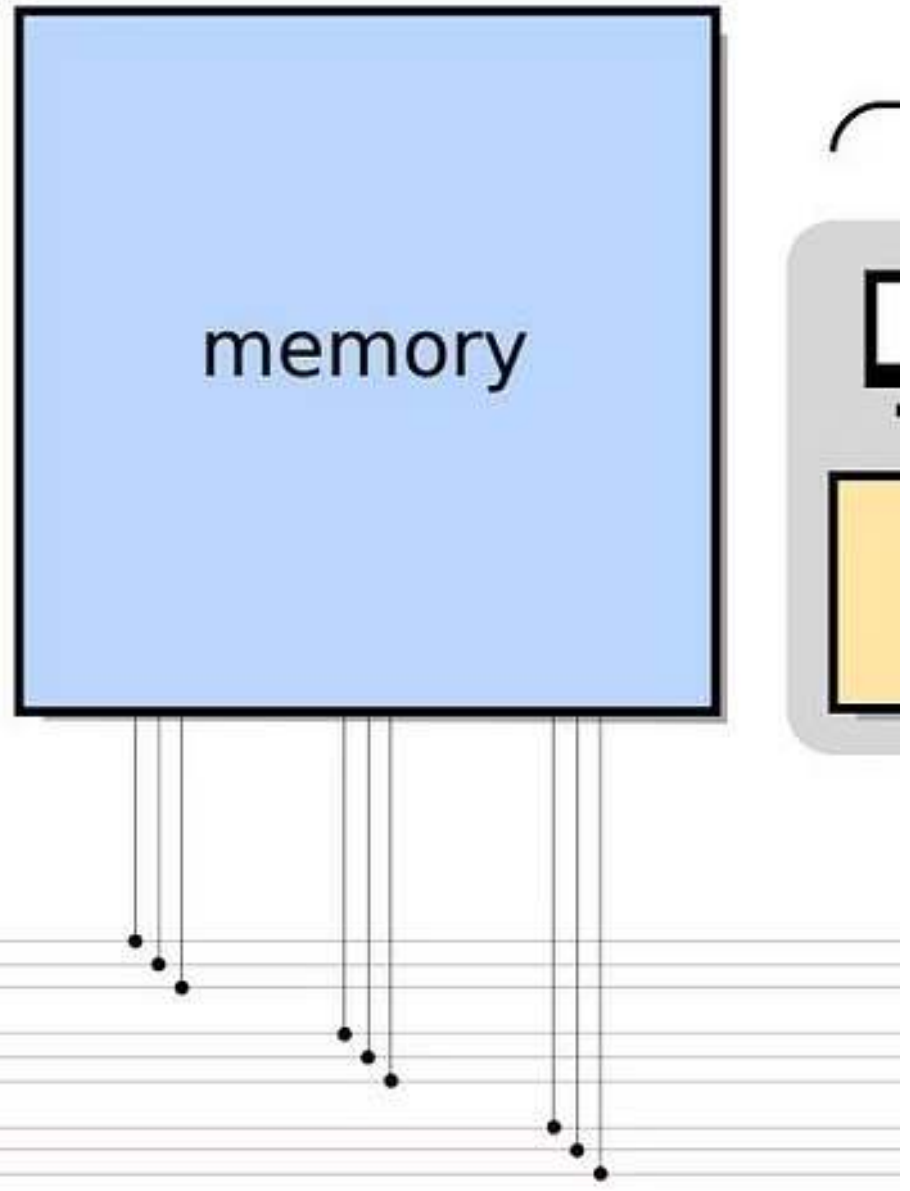
### Níveis de Privilégio

Separam o código do sistema operacional das aplicações de usuário

### Unidade de Gerência de Memória (MMU)

Controla o acesso à memória e possibilita isolamento entre processos

Estas características de hardware são fundamentais para a implementação de um sistema operacional moderno e seguro.




# Arquitetura de um Computador Típico

A maioria dos computadores monoprocessados atuais segue uma arquitetura básica definida nos anos 40 por János (John) Von Neumann, conhecida como "arquitetura Von Neumann".

A principal característica desse modelo é o "programa armazenado", ou seja, o programa a ser executado reside na memória junto com os dados, diferente de sistemas anteriores onde o programa era fixo no hardware.

# Componentes da Arquitetura

## Processador (CPU)

- Centro do sistema de computação
- Contém a Unidade Lógica e Aritmética (ULA)
- Possui registradores para armazenar dados de trabalho
- Registradores especiais: contador de programa, ponteiro de pilha, flags de status

## Barramentos

- **Barramento de endereços:** indica a posição de memória a acessar
- **Barramento de controle:** indica a operação a efetuar
- **Barramento de dados:** transporta a informação

## Memória

- Armazena programas e dados
- Organizada em posições endereçáveis
- Hierarquia de memória (cache, RAM, armazenamento)

## Controladores de Dispositivos

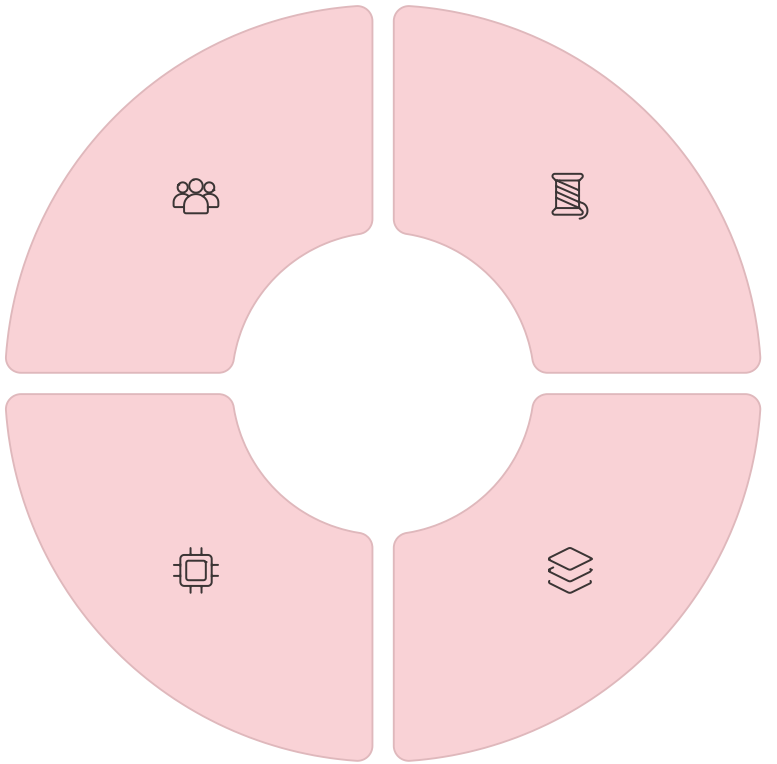
- Circuitos específicos para cada tipo de periférico
- Acessados através de portas de entrada/saída endereçáveis
- Exemplos: controlador de vídeo, Ethernet, USB

A integração desses componentes forma a base sobre a qual o sistema operacional é construído.

# Processadores Modernos

### Múltiplos Núcleos

Processadores com vários núcleos de processamento (cores) permitem executar várias tarefas simultaneamente



### Hyperthreading

Tecnologia que permite que um único núcleo físico apareça como dois núcleos lógicos para o sistema operacional

### Arquiteturas Avançadas

Execução fora de ordem, previsão de desvios e outras otimizações para aumento de desempenho

### Memória Cache

Vários níveis de cache (L1, L2, L3) para reduzir a latência de acesso à memória principal

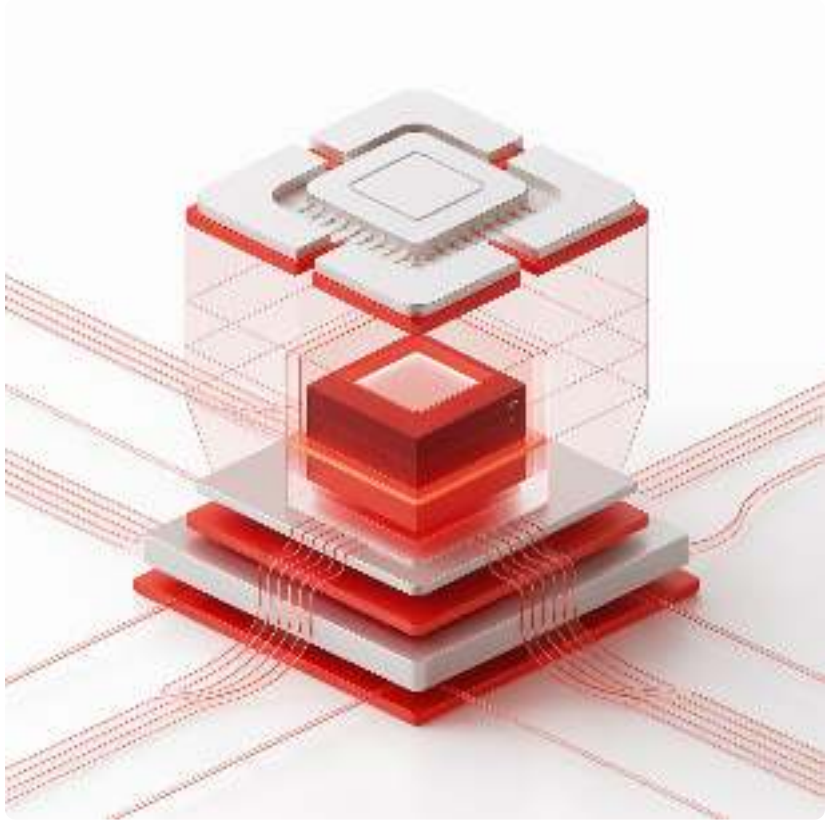
Processadores modernos são incrivelmente mais complexos que os modelos originais, apresentando recursos avançados que demandam sistemas operacionais igualmente sofisticados para explorá-los adequadamente.

# A Unidade de Gerência de Memória (MMU)

A MMU (Memory Management Unit) é um componente crítico que:

- Media o acesso do processador à memória
- Analisa cada endereço de memória acessado pelo processador
- Valida os endereços solicitados
- Efetua conversões de endereçamento quando necessárias
- Implementa a proteção de memória entre processos
- Possibilita a implementação de memória virtual

A MMU pode estar fisicamente dentro do próprio chip do processador nas arquiteturas modernas.



# Controladores de Dispositivos

Os periféricos do computador são acessados através de circuitos eletrônicos específicos denominados **controladores**:

| Dispositivo | Endereços de acesso (hex) |
|---|---|
| Temporizador | 0040-0043 |
| Teclado | 0060-006F |
| Porta serial COM1 | 02F8-02FF |
| Controlador SATA | 30BC-30BF |
| Controlador Ethernet | 3080-309F |

Cada controlador é acessado através de uma faixa específica de endereços de portas de entrada/saída, que podem variar dependendo da configuração do sistema.

# Interrupções e Exceções

**Problema: Como dispositivos periféricos comunicam eventos ao processador?**

## Método de Espera (Polling)

O processador verifica periodicamente o estado dos dispositivos para detectar eventos.

**Desvantagem:** Ineficiente, pois o processador desperdiça tempo verificando dispositivos que podem não ter eventos para reportar.

## Método de Interrupção

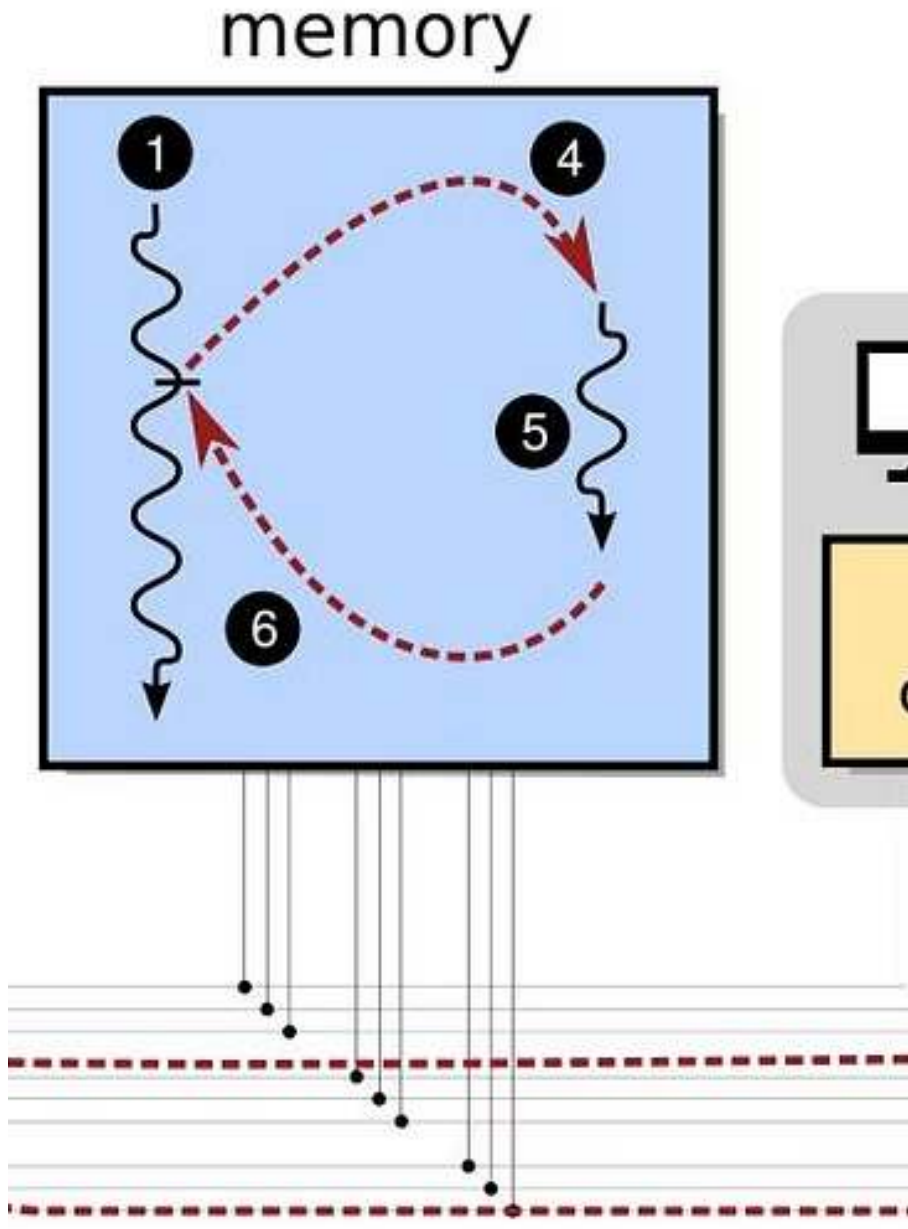
O dispositivo envia um sinal (IRQ - Interrupt Request) ao processador quando tem um evento para reportar.

**Vantagem:** Eficiente, pois o processador só é notificado quando realmente necessário, podendo dedicar seu tempo a outras tarefas.

O mecanismo de interrupções é fundamental para a eficiência dos sistemas operacionais modernos.

# Funcionamento das Interrupções

1. O processador está executando um programa qualquer
2. Um evento ocorre em um dispositivo (ex: pacote recebido na rede)
3. O controlador do dispositivo envia uma solicitação de interrupção (IRQ)
4. O processador suspende a execução atual e desvia para a rotina de tratamento
5. A rotina de tratamento interage com o dispositivo para resolver o evento
6. Ao finalizar, o processador retorna à execução do programa interrompido



# Tabela de Interrupções

Para distinguir interrupções geradas por dispositivos distintos, cada interrupção é identificada por um número inteiro:

### Interrupções de Hardware

- Geradas por dispositivos externos
- Exemplos: teclado, mouse, disco, rede
- Números geralmente acima de 32

### Exceções (Interrupções de Software)

- Geradas pelo próprio processador
- Exemplos: divisão por zero, instrução ilegal
- Números geralmente abaixo de 32

A Tabela de Interrupções (IVT - Interrupt Vector Table) contém os endereços das rotinas de tratamento correspondentes a cada número de interrupção.

# Exemplos de Interrupções do Intel Pentium

## Exceções

- 0: Erro de divisão
- 6: Opcode inválido
- 13: Proteção geral
- 14: Falha de página
- 16: Erro de ponto flutuante

## Benefícios das Interrupções

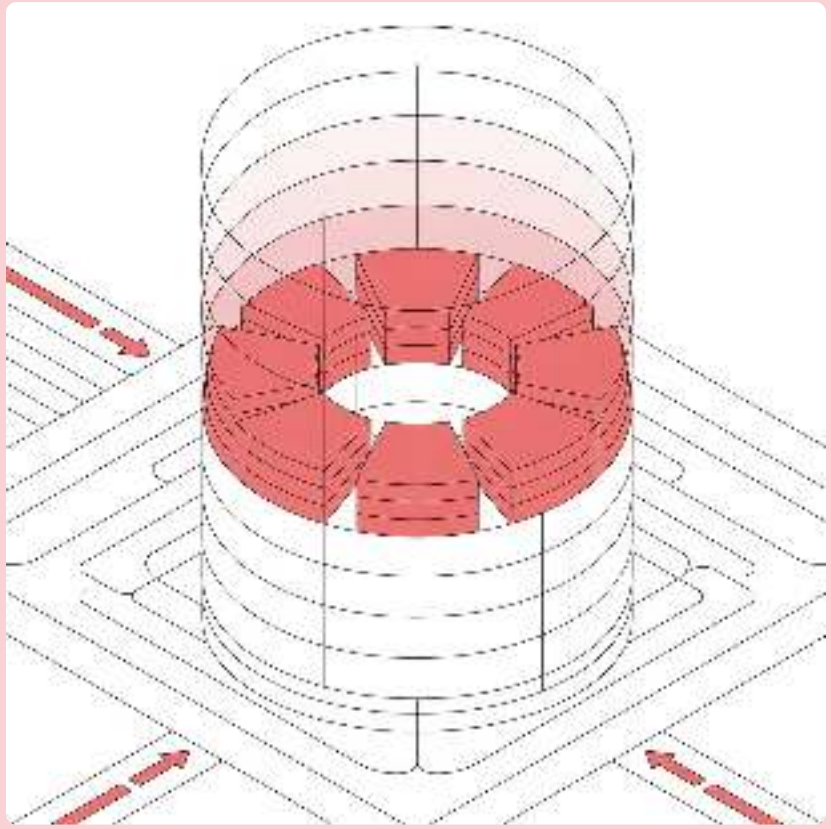
- Tornam eficiente a interação do processador com os dispositivos
- Evitam que o processador perca tempo "varrendo" todos os dispositivos
- Permitem operações de entrada/saída assíncronas
- O processador pode realizar outras tarefas enquanto aguarda a conclusão de operações em dispositivos
- Garantem resposta rápida a eventos críticos

Interrupções não são eventos raros - centenas ou milhares podem ocorrer por segundo, dependendo da carga do sistema.

# Níveis de Privilégio

## Por que diferenciar privilégios de execução?

- O núcleo e drivers precisam de acesso pleno ao hardware
- Aplicativos devem ter acesso restrito para não interferirem nas configurações
- Aplicações com acesso pleno representariam risco à segurança
- Necessário controlar quem pode acessar recursos críticos



Os processadores modernos implementam níveis de privilégio de execução para permitir essa diferenciação, controlados por flags especiais na CPU.

# Níveis de Privilégio do x86

Os processadores x86 definem 4 níveis de privilégio (0 a 3), embora a maioria dos sistemas operacionais use apenas os níveis extremos:

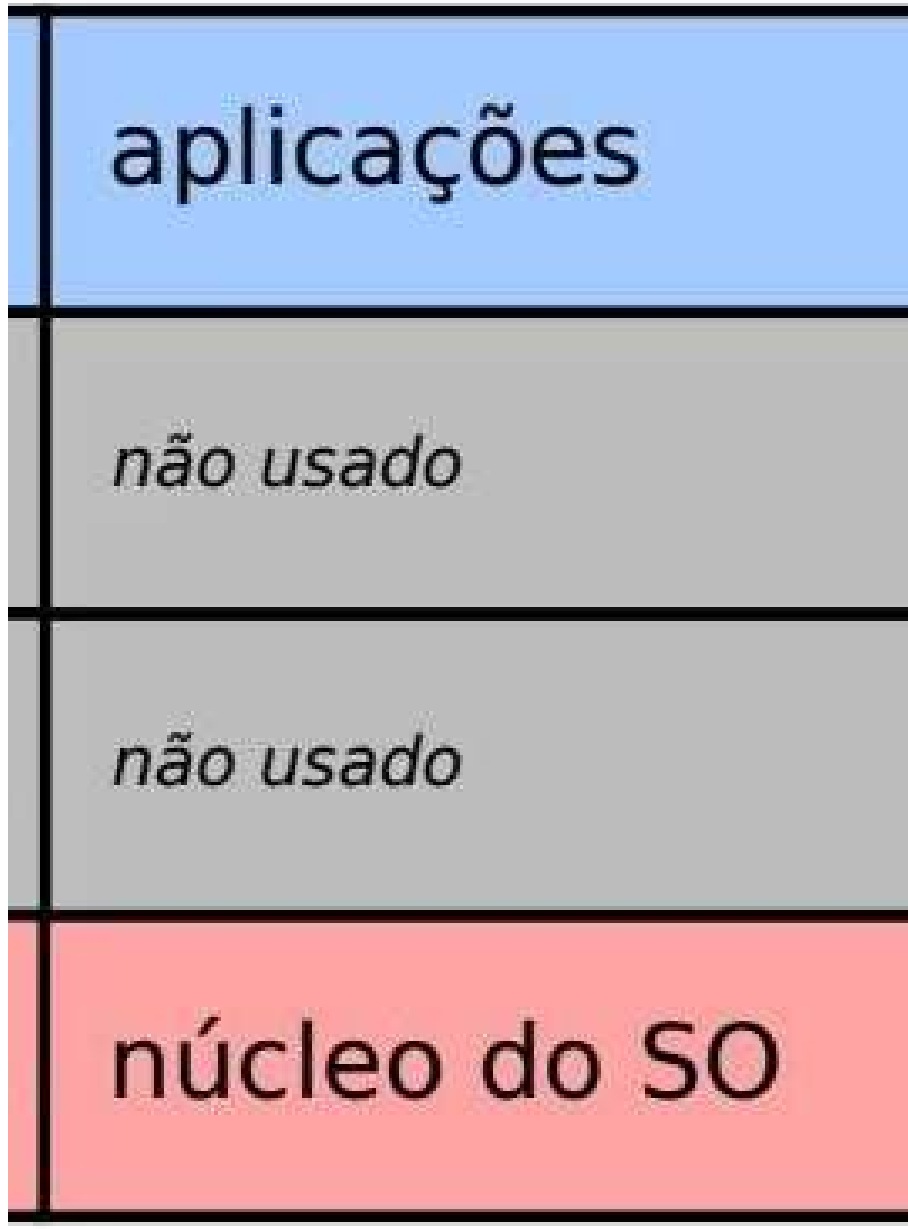
### Nível 0 (Kernel)

Maior privilégio, utilizado pelo núcleo do SO. Todas as instruções do processador estão disponíveis, incluindo operações "perigosas" como acessar portas de E/S.

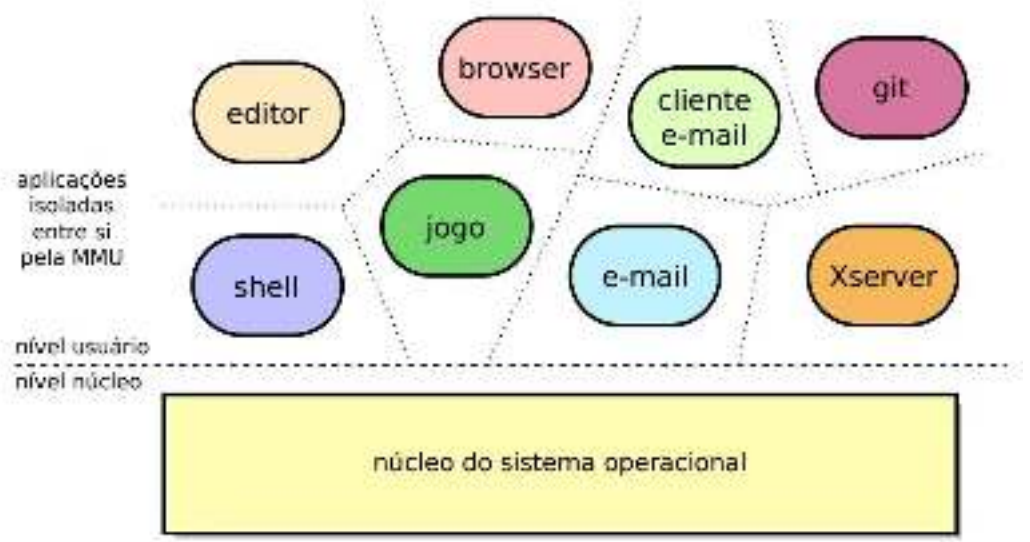
### Nível 3 (Usuário)

Menor privilégio, utilizado por aplicações comuns. Instruções que podem comprometer a estabilidade do sistema são proibidas.

A ideia de níveis de privilégio foi introduzida pelo sistema Multics nos anos 1960, com os "anéis de proteção" (protection rings).



# Separação entre Núcleo e Aplicações




### Benefícios do Modelo de Execução Confinada:

- Isolamento entre aplicações
- Erros em uma aplicação não afetam outras
- Maior estabilidade do sistema
- Aumento da segurança
- Melhor gerenciamento de recursos

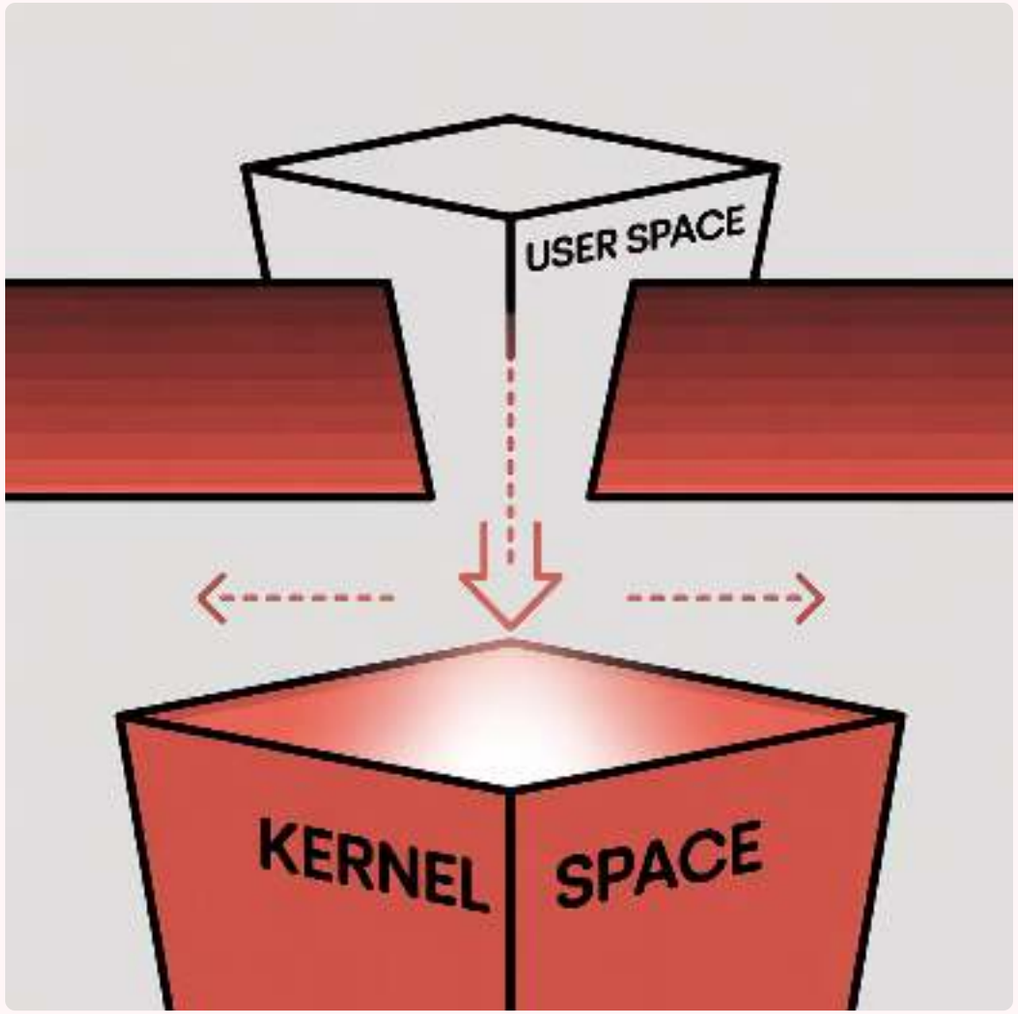
Com a proteção provida pelos níveis de privilégio e pelas áreas de memória exclusivas configuradas pela MMU, obtém-se um modelo de execução confinada, no qual as aplicações executam isoladas umas das outras e do núcleo.

# O Problema da Comunicação

## Como as aplicações podem acessar serviços do núcleo?

**Desafio:** O confinamento de cada aplicação em sua área de memória traz robustez e confiabilidade, mas cria um novo problema:

- O código do núcleo reside em área inacessível à aplicação
- Operações de desvio como `jump` e `call` não funcionariam
- É necessário um mecanismo seguro para "cruzar a fronteira"




A solução desse problema está no mecanismo de chamadas de sistema, que permite que aplicações solicitem serviços do núcleo de forma controlada.

# Chamadas de Sistema

## System Calls: A Ponte entre Aplicações e o Núcleo

As chamadas de sistema são o mecanismo que permite que programas em modo usuário acessem serviços fornecidos pelo núcleo do sistema operacional. Elas funcionam através de uma interrupção de software (trap).

### Aplicação

Solicita um serviço do SO, como abrir um arquivo ou alocar memória

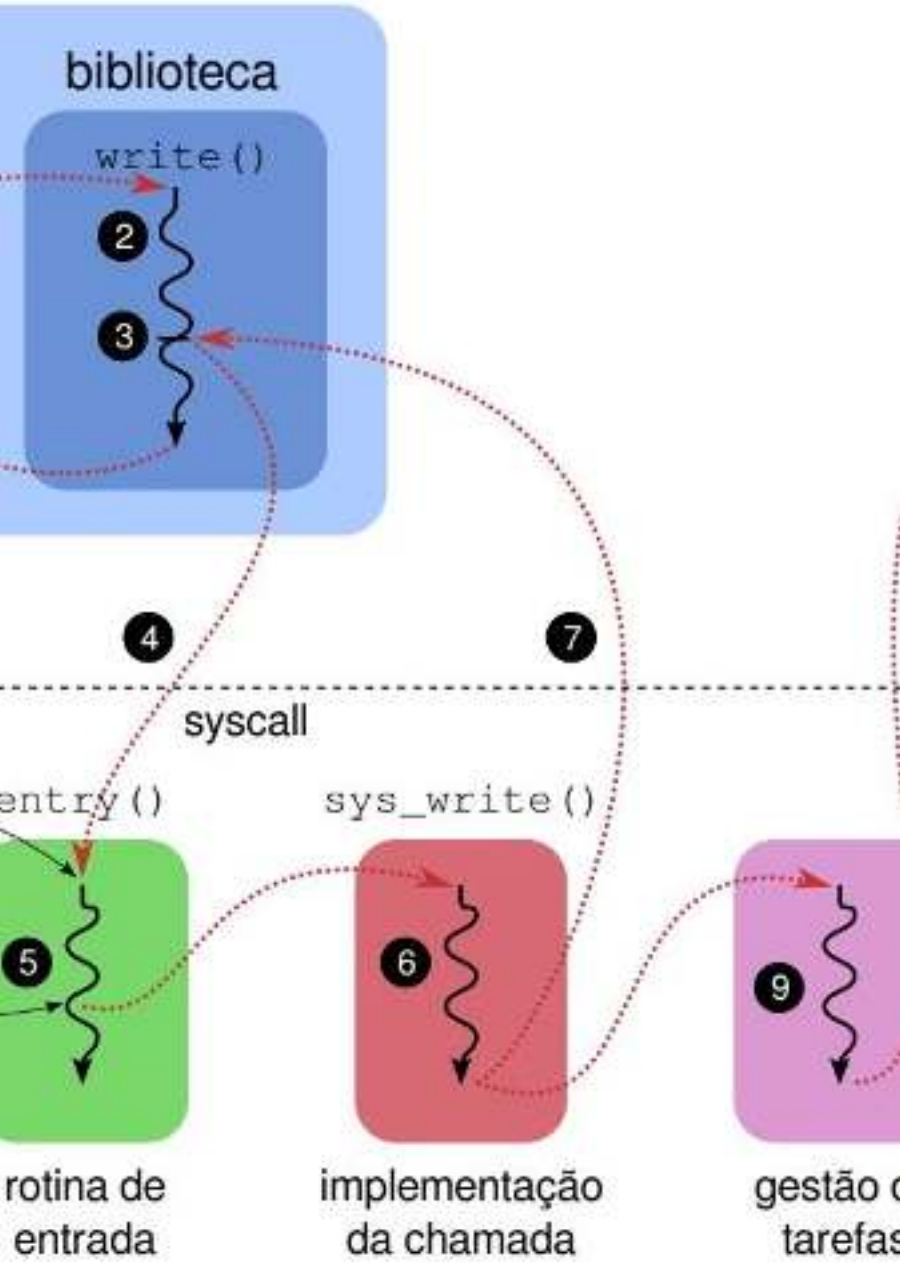
### Biblioteca do Sistema

Prepara parâmetros e invoca a instrução especial de interrupção

### Núcleo

Executa o serviço solicitado com seus privilégios elevados e retorna o resultado

Este mecanismo permite acessar recursos do sistema de forma controlada e segura.




# Funcionamento das Chamadas de Sistema

As chamadas de sistema utilizam instruções especiais do processador como `sysenter/sysexit` (x86 32 bits) ou `syscall/sysret` (x86 64 bits).

Antes de invocar a chamada, registradores são preparados com valores específicos, como o número da operação desejada (geralmente no registrador AX).

# Categorias de Chamadas de Sistema

## Gestão de Processos

Criar, terminar, carregar código, esperar, gerenciar atributos de processos

## Gestão da Memória

Alocar, liberar e modificar áreas de memória

## Gestão de Arquivos

Criar, remover, abrir, fechar, ler, escrever e modificar atributos de arquivos

## Comunicação

Criar/destruir canais de comunicação, receber/enviar dados

## Gestão de Dispositivos

Ler/mudar configurações, ler/escrever dados em dispositivos

## Gestão do Sistema

Ler/mudar data e hora, desligar/suspender/reiniciar o sistema

Os sistemas operacionais modernos implementam centenas de chamadas de sistema distintas para as mais diversas finalidades.

# Exemplo de Chamada de Sistema em C

```
#include

int main (int argc, char *argv[])
{
  write (1, "Hello World!\n", 13) ; /* write string to stdout */
  _exit (0) ; /* exit with no error */
}
```

Este programa simples em C usa duas chamadas de sistema:

- `write`: escreve uma string na saída padrão (stdout)
- `_exit`: encerra o processo com um código de status

A biblioteca C (`libc`) facilita o uso de chamadas de sistema, abstraindo os detalhes de baixo nível.

# Exemplo em Assembly

```
section .data
msg db 'Hello World!', 0xA ; output string (0xA = "\n")
len equ 13 ; string size

section .text

global _start

_start:
  mov rax, 1 ; syscall opcode (1: write)
  mov rdi, 1 ; file descriptor (1: stdout)
  mov rsi, msg ; data to write
  mov rdx, len ; number of bytes to write
  syscall ; make syscall

  mov rax, 60 ; syscall opcode (60: _exit)
  mov rdi, 0 ; exit status (0: no error)
  syscall ; make syscall
```

Em Assembly, podemos ver claramente a preparação dos registradores e a invocação direta das chamadas de sistema.

# APIs de Sistema Operacional

### Win32 API (Windows)

- Interface para sistemas Microsoft
- Milhares de funções documentadas
- Suporte para aplicações desktop e serviços
- Organizadas em DLLs do sistema

### POSIX API (Unix/Linux)

- Padrão para sistemas compatíveis com Unix
- Define interface comum entre diferentes sistemas
- Facilita a portabilidade de aplicações
- Implementada pelo Linux, macOS, FreeBSD, etc.

O conjunto de chamadas de sistema oferecidas por um núcleo define a API (*Application Programming Interface*) do sistema operacional, que serve como contrato entre o SO e as aplicações.

# Caso Prático: Análise da Chamada write()

## Passo a passo da execução da chamada de sistema write()

1. A aplicação invoca a função `write(fd, &buffer, bytes)` da biblioteca do sistema
2. A função preenche registradores da CPU com os parâmetros e o opcode da chamada
3. A instrução `syscall` é executada, comutando para modo privilegiado
4. O núcleo recebe o controle e identifica a operação solicitada através do opcode
5. A função `sys_write` do núcleo é invocada para realizar a operação
6. A função verifica os parâmetros e executa a operação de escrita
7. O controle retorna à aplicação com o resultado da operação

Este processo ocorre milhares de vezes por segundo em um sistema em funcionamento.

# Quando a Operação Não Pode ser Concluída

### Aplicação faz chamada de sistema

Por exemplo, solicitando leitura de dados do teclado

### Kernel verifica disponibilidade

Constata que não há dados disponíveis (usuário não digitou)

### Gerência de tarefas suspende a aplicação

Ao invés de retornar, o kernel suspende a aplicação até que os dados estejam disponíveis

### Outra aplicação recebe controle

O processador é cedido a outra aplicação que esteja pronta para executar

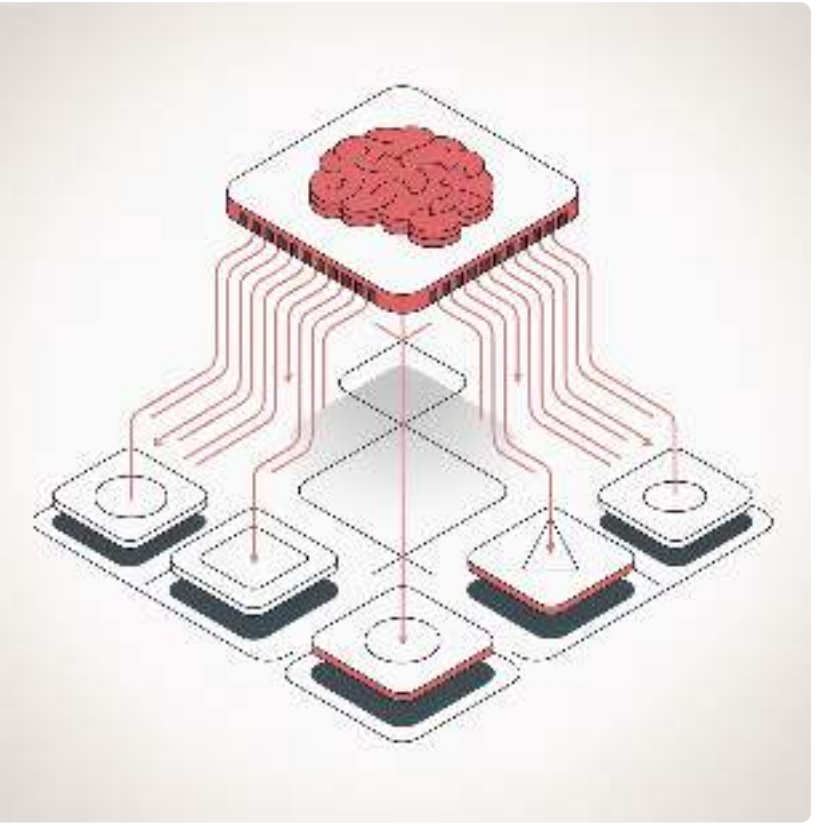
Este mecanismo é fundamental para o multitasking e para a eficiência do sistema, evitando que o processador fique ocioso aguardando operações lentas.

# Chamadas de Sistema e Multitarefa

As chamadas de sistema são pontos ideais para implementação de multitarefa por várias razões:

- São momentos em que o núcleo já tem o controle do processador
- Muitas operações envolvem espera por dispositivos lentos
- A aplicação não pode observar a troca de contexto
- O núcleo tem todas as informações necessárias para decidir qual aplicação deve executar

Este modelo de multitarefa é chamado de preemptivo cooperativo, pois a aplicação "coopera" cedendo o controle nas chamadas de sistema.



# Visão Completa: Hardware e SO

Ao entender a relação entre os componentes de hardware e o sistema operacional, podemos compreender como eles trabalham juntos:


Essa estrutura fundamental é similar em praticamente todos os sistemas operacionais modernos.

# Diferenças Entre Tipos de SOs

### Desktop/Laptop

- Foco em interface de usuário
- Multitarefa intensiva
- Suporte a diversos periféricos
- Exemplos: Windows, macOS, Linux

### Servidores

- Foco em desempenho e estabilidade
- Otimizados para rede e armazenamento
- Geralmente sem interface gráfica
- Exemplos: Linux Server, Windows Server

### Embarcados/Mobile

- Otimizados para eficiência energética
- Foco em recursos limitados
- Interface adaptada para toque
- Exemplos: Android, iOS, RTOS

Apesar das diferenças de foco e otimização, todos compartilham os mesmos princípios fundamentais de estrutura e organização.

# Evolução dos Sistemas Operacionais

1

## Anos 1950-60

Sistemas de lote (batch) simples, sem multitarefa real. Foco em maximizar o uso do processador com jobs sequenciais.

2

## Anos 1970-80

Surgimento de sistemas multiusuário (Unix) e introdução de conceitos como proteção de memória e multitarefa preemptiva.

3

## Anos 1990

Popularização de interfaces gráficas, sistemas operacionais pessoais (Windows, MacOS) e refinamento dos mecanismos de proteção.

4

## Anos 2000

Adaptação para arquiteturas multicore, virtualização e consolidação dos sistemas de código aberto (Linux).

5

## 2010 até hoje

Foco em mobilidade, cloud computing, segurança avançada e otimização para novos tipos de hardware.

A estrutura básica dos SOs evoluiu para acomodar novos requisitos, mas manteve seus princípios fundamentais.

# Exercício Prático: Análise de Chamadas de Sistema

## Ferramentas para Análise

- **strace (Linux):** Monitora chamadas de sistema feitas por um processo
- **ltrace (Linux):** Monitora chamadas de biblioteca
- **Process Monitor (Windows):** Registra atividade do sistema, incluindo chamadas de API

Estas ferramentas permitem observar a interação entre aplicações e o sistema operacional em tempo real.

```
$ strace ls
execve("/bin/ls", ["ls"], [/* 21 vars */]) = 0
brk(NULL)                               = 0x55d932e85000
access("/etc/ld.so.nohwcap", F_OK) = -1
mmap(NULL, 8192, PROT_READ|PROT_WRITE, ...) = 0x7
...
write(1, "Desktop  Documents  Downloads  "..., 54) = 54
close(1)                                = 0
exit_group(0)                           = ?
```

Experimente analisar aplicações simples para entender como interagem com o sistema operacional.

# Considerações de Segurança

### Vulnerabilidades em Chamadas de Sistema

As chamadas de sistema são pontos críticos de segurança, pois representam a fronteira entre código não confiável (aplicações) e código privilegiado (núcleo).

### Verificação de Parâmetros

O núcleo deve validar rigorosamente todos os parâmetros recebidos das aplicações para evitar ataques como buffer overflow.

### Sandbox e Contenção

Tecnologias modernas implementam camadas adicionais de isolamento, como contêineres e máquinas virtuais.

### Minimização de Superfície de Ataque

Sistemas modernos tendem a reduzir o número de chamadas de sistema disponíveis e limitar seus privilégios.

A segurança da estrutura do SO é fundamental para a proteção do sistema como um todo.

# Tendências Recentes

## Microkernels

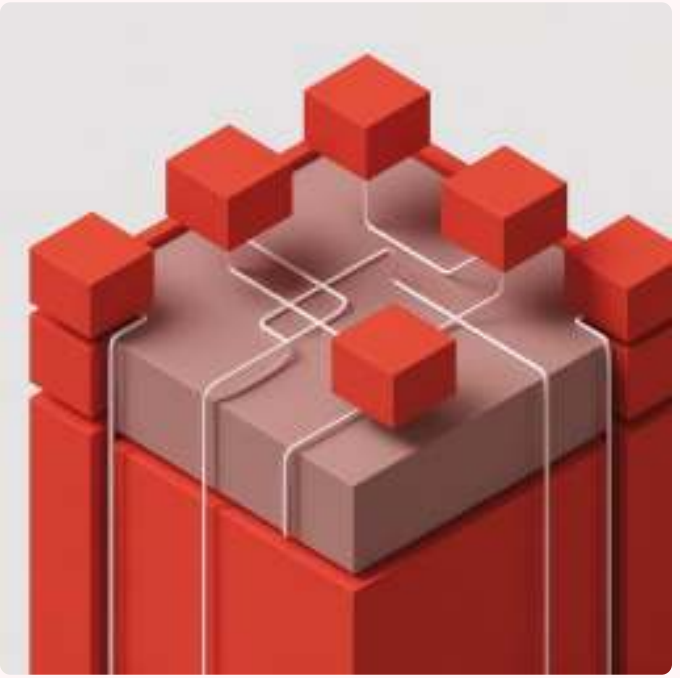
Mínimo possível de código no núcleo, com a maioria dos serviços executando em modo usuário. Maior robustez, mas potencialmente menor desempenho.

## Unikernels

Biblioteca de sistema operacional especializada que compila junto com a aplicação, eliminando camadas e abstrações desnecessárias.


## Núcleos Exokernels

Foco em expor o hardware diretamente para as aplicações, com mínima abstração, permitindo otimizações específicas.



Estas tendências representam diferentes compromissos entre desempenho, segurança e facilidade de desenvolvimento.

# Virtualização

## O que é Virtualização?

Tecnologia que permite executar múltiplos sistemas operacionais completos em uma única máquina física, cada um "pensando" que tem acesso exclusivo ao hardware.

## Tipos de Virtualização:

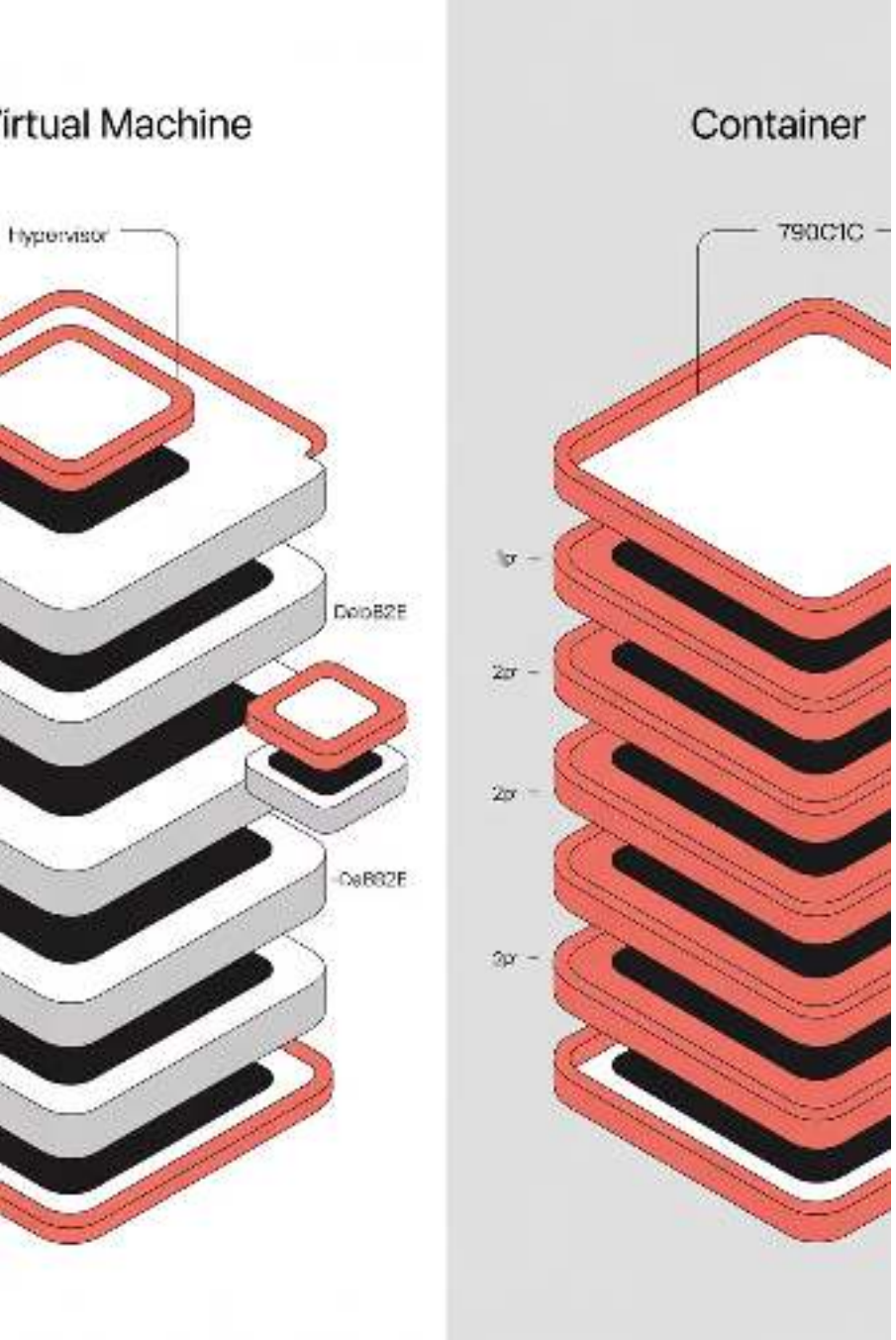
- **Tipo 1:** Hypervisor executa diretamente no hardware
- **Tipo 2:** Hypervisor executa sobre um SO hospedeiro

## Impacto na Estrutura do SO:

- Introduz uma nova camada entre o SO e o hardware
- Requer mecanismos adicionais para tradução de endereços
- Necessita interceptar operações privilegiadas
- Exige suporte de hardware para virtualização eficiente

Extensões de CPU como Intel VT-x e AMD-V facilitam a virtualização eficiente.

A virtualização adiciona complexidade à estrutura tradicional do SO, mas traz flexibilidade e melhor utilização de recursos.




# Contêineres vs. Máquinas Virtuais

## Máquinas Virtuais

Virtualizam hardware completo, exigindo um SO completo em cada instância.

- Isolamento forte entre VMs
- Maior sobrecarga de recursos
- Inicialização mais lenta
- Flexibilidade para executar diferentes SOs

## Contêineres

Virtualizam apenas o ambiente de usuário, compartilhando o kernel do host.

- Isolamento mais leve
- Menor sobrecarga de recursos
- Inicialização muito rápida
- Limitado ao SO do host

# Recursos para Aprofundamento

## Livros

- "Sistemas Operacionais Modernos" - Tanenbaum
- "Sistemas Operacionais: Conceitos e Mecanismos" - Maziero
- "Operating System Concepts" - Silberschatz et al.

## Projetos Práticos

- xv6: sistema didático baseado em Unix
- Minix: microkernel educacional
- OSDev Wiki: recursos para desenvolvimento de SOs

## Cursos Online

- CS162 (Berkeley)
- 6.828 (MIT)
- Cursos na Coursera e edX sobre sistemas operacionais

## Comunidades

- Fóruns Linux e BSD
- Grupos de desenvolvimento de kernel
- Stack Overflow - tags de sistemas operacionais

O estudo de sistemas operacionais é um campo vasto que combina teoria e prática.

# Projetos Práticos Recomendados

## Nível Iniciante

1. Escrever e analisar programas simples usando chamadas de sistema diretamente
2. Monitorar chamadas de sistema de aplicações com ferramentas como strace
3. Escrever um driver simples para Linux (módulo de kernel)
4. Modificar o código-fonte de um kernel educacional como xv6

## Nível Avançado

1. Implementar um escalonador de processos personalizado
2. Criar um sistema de arquivos simples
3. Desenvolver um hypervisor básico
4. Contribuir para projetos de código aberto como Linux ou FreeBSD

A experiência prática é essencial para compreender verdadeiramente o funcionamento interno dos sistemas operacionais.

# Recapitulação e Conclusão

1

## Estrutura do SO

O sistema operacional é composto por núcleo, código de inicialização, drivers e programas utilitários, cada um com funções específicas.

2

## Elementos de Hardware

O hardware fornece mecanismos essenciais como níveis de privilégio, MMU e interrupções, que possibilitam a implementação segura e eficiente do SO.

3

## Chamadas de Sistema

As chamadas de sistema formam a interface entre aplicações e núcleo, permitindo acesso controlado aos recursos do sistema.

4

## Evolução Contínua

Novas arquiteturas e requisitos continuam impulsionando inovações na estrutura dos sistemas operacionais.

Compreender a estrutura dos sistemas operacionais é fundamental para o desenvolvimento de software eficiente, seguro e confiável, além de ser a base para áreas avançadas como virtualização, sistemas distribuídos e segurança de sistemas.