

Sistemas Operacionais

Cap.2 - Estruturas do Sistema Operacional



Prof. MSc. Renzo P. Mesquita
renzo@inatel.br

Objetivos

- Descrever serviços mais específicos que um Sistema Operacional fornece para usuários, processos e outros sistemas;
- Discutir as diversas maneiras de estruturar um Sistema Operacional;
- Explicar como os Sistemas Operacionais são instalados, personalizados e como são inicializados;



Capítulo 2

Estruturas do Sistema Operacional

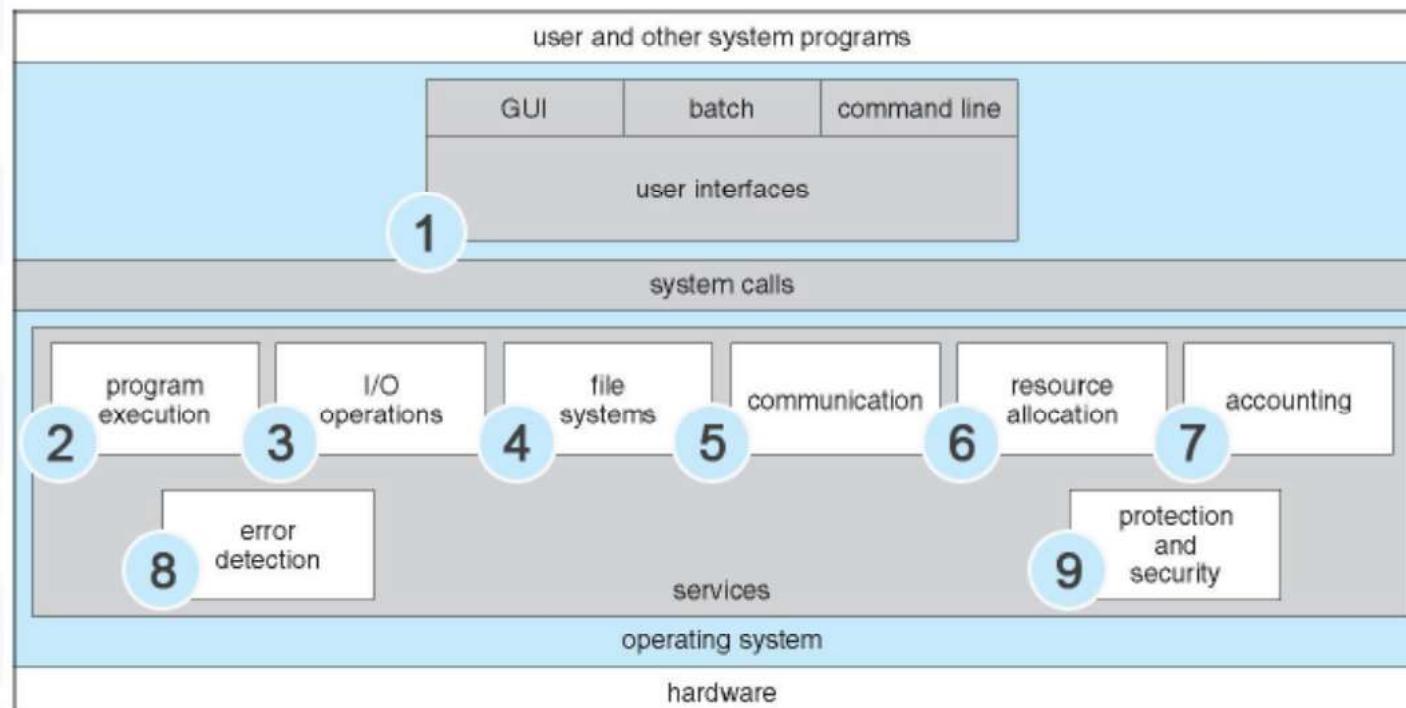
- 2.1. Serviços do Sistema Operacional;*
- 2.2. Interface entre Usuário e SO;*
- 2.3. Chamadas de Sistema (System Calls);*
- 2.4. Programas de Sistema;*
- 2.5. Projeto e Implementação de SO's;*
- 2.6. Estrutura do Sistema Operacional;*
- 2.7. Máquinas Virtuais;*
- 2.8. Depuração e Geração do Sistema Operacional;*
- 2.9. Inicialização do Sistema;*



2.1. Serviços do Sistema Operacional

Como já discutimos superficialmente, sabemos que os Sistemas Operacionais fornecem serviços para programas e para usuários de programas, visando tornar mais fácil a tarefa de programar;

Os serviços específicos fornecidos diferem de um Sistema Operacional para o outro, mas podemos identificar algumas classes de serviços comuns, são eles:



2.1. Serviços do Sistema Operacional

1) Interface de Usuário

Quase todos Sistemas Operacionais possuem uma UI (user interface). Essa interface pode assumir várias formas:

- Command-line Interface (CLI) - Interface de linha de comando;
- Batch Interface - Comandos e diretivas inseridas em arquivos que podem ser executados automaticamente pelo SO. Ex: arquivos .bat;
- Graphical User Interface (GUI) - Sistema de janelas;

2) Execução de Programas

O SO deve ser capaz de carregar um programa em memória, executá-lo e terminá-lo de forma normal ou anormal;

3) Operações de I/O

O SO deve ser capaz de prover serviços de I/O em benefícios dos usuários, pois eles não podem ter acesso direto a estes serviços;

2.1. Serviços do Sistema Operacional

4) Manipulação do Sistema de Arquivos

Ler, escrever, criar e remover arquivos na memória secundária. Algumas vezes também oferecem controle de permissões sobre eles;

5) Comunicações

Deve permitir a troca de informações entre processos, seja na mesma máquina ou em máquinas diferentes em uma rede;

6) Alocação de Recursos

Gerenciar ciclos de CPU, Memória Principal, Armazenamento de Arquivos, I/O entre outros;

7) Contabilização

Criação de estatísticas para melhorar os serviços de processamento e utilização de recursos em sistemas multiusuário;

2.1. Serviços do Sistema Operacional

8) Detecção de Erros

O Sistema Operacional deve ser capaz de detectar erros de Hardware (CPU, Memória etc..) e dos Programas. Para cada tipo de erro, ele deve tomar uma medida adequada para assegurar um processamento consistente;

9) Proteção e Segurança

Proteção significa garantir que um processo não interfira na operação de outro quando executados concorrentemente e garantir que o acesso a recursos do sistema seja controlado;

O Sistema também deve fornecer mecanismos que evitem invasores no sistema, por meio de mecanismos de autenticação.



2.2. Interface entre Usuário e SO

Existem várias maneiras dos Usuários se comunicarem com os Sistemas Operacionais. Duas abordagens populares são:

1) *Command Line Interface (CLI)*

- O Interpretador de Comandos **pode fazer parte do kernel**, mas usualmente, ele é um programa do sistema que serve como interface entre Usuário e SO;
- Quando o SO oferece vários **Interpretadores de Comandos**, estes são conhecidos como **Shells**;
- Exemplo: shell Bourne, shell C, Interface CLI do DOS, entre muitos outros;

2) *Graphical User Interface (GUI)*

- São interfaces amigáveis, que fazem uso do **mouse e ícones**;
- Exemplo: Windows Explorer, X-Window (Unix), KDE (Linux), GNOME (Linux);



```
C:\Windows\system32\cmd.exe
20.815 Forum post.txt
11.381 hijackthis.log
3.215.858 Lestezio delloro.mp3
6.243.965 Lux Reterna.mp3
232.501 Minecraft.exe
203 My Withings.url
4.857.594 Organ donor.mp3
3.510.489 stateofmind.mp3
4.368.164 Time.mp3
944.640 WinAudit.exe
world
_inc
23 File(s) 86.754.747 bytes
7 Dir(s) 212.093.128.784 bytes free
C:\Users\Mrhope\Desktop>
```



2.3. Chamadas de Sistema (System Calls)

2.3.1. API (Application Programming Interface)

Antes de adentrarmos um pouco mais a fundo nos detalhes das System Calls, veremos o que são API's.

API's especificam um conjunto de funções que estão disponíveis para o programador de aplicações, inclusive os parâmetros que são passados para cada função e os valores de retorno que o programador pode esperar.

As API's permitem que programadores usem funções pré-definidas para interagir por exemplo com Sistemas Operacionais, em vez de escrevê-las desde o início.



2.3. Chamadas de Sistema (System Calls)

2.3.2 Definição

System Calls fornecem uma interface para acessar os serviços oferecidos pelo Sistema Operacional.

Geralmente são rotinas escritas em Assembly (Quando o hardware precisa ser acessado diretamente) ou linguagem de alto nível como C ou C++.

Um programa Aplicativo pode chamar diretamente as System Calls ou chamá-las por intermédio de uma API (Application Programming Interface).

Exemplo de API's para Sistemas Operacionais: Win32 (Windows), POSIX (UNIX,LINUX,MAC OS X), entre outros.

Ex: BOOL ReadFile(c (
 HANDLE file,
 LPVOID buffer,
 DWORD bytes To Read,
 LPDWORD bytes Read,
 LPOVERLAPPED ovl);

(Função ReadFile() - função para
leitura de um arquivo - da API Win32)

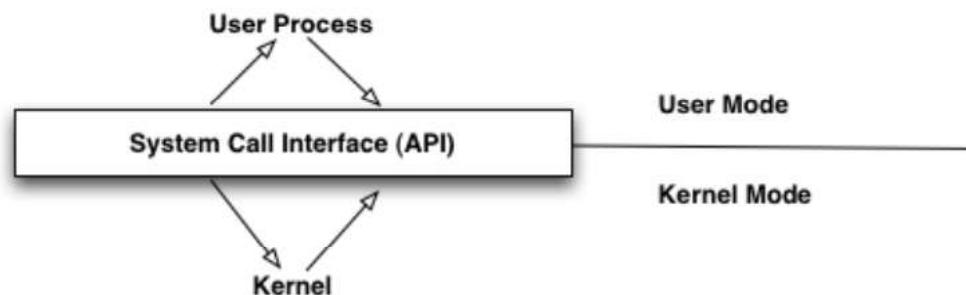
2.3. Chamadas de Sistema (System Calls)

As System Calls na maioria dos casos são chamadas por meio de API's ao invés de serem acessadas diretamente.

Vantagens:

- Portabilidade. O Programa Aplicativo pode ser compilado e executado em qualquer sistema que ofereça a mesma API;
- Systems Calls de acesso direto são mais completas, porém, mais complexas de serem implementadas;

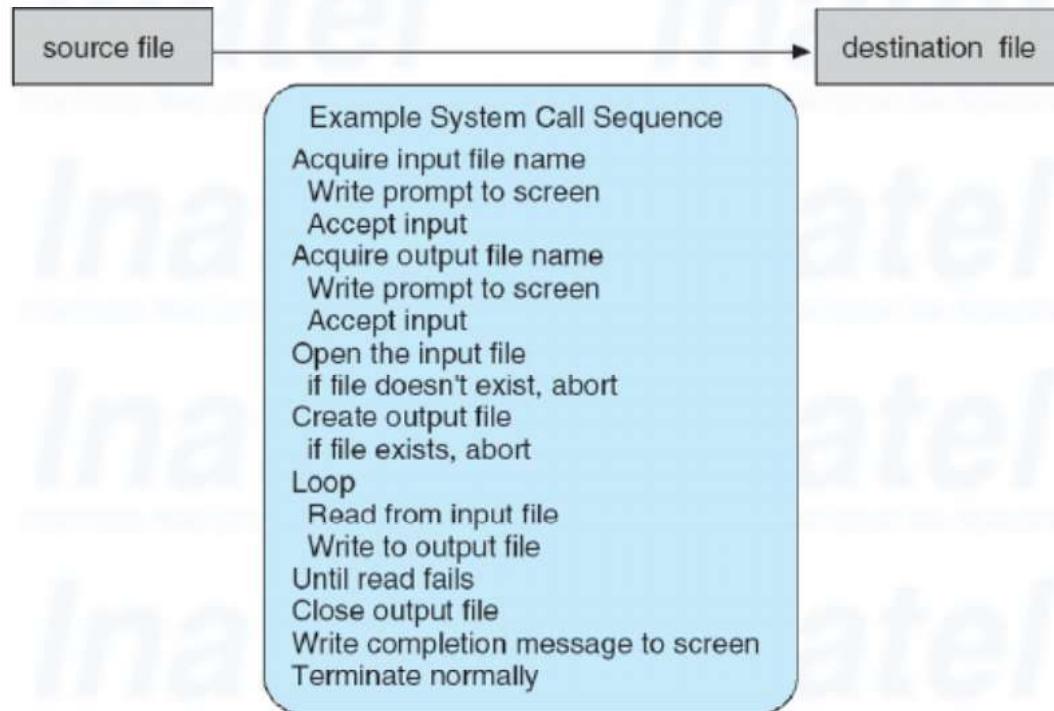
Até mesmo programas simples, como ler dados de um arquivo e copiá-los para outro arquivo, usam bastante os serviços do Sistema Operacional. Geralmente, os sistemas executam milhares de System Calls por segundo.



2.3. Chamadas de Sistema (System Calls)

Até mesmo programas e rotinas simples podem usar bastante os serviços dos Sistemas Operacionais.

Ex:



Exemplos de sequências de System Calls necessárias para copiar conteúdos de um arquivo para outro.

2.3. Chamadas de Sistema (System Calls)

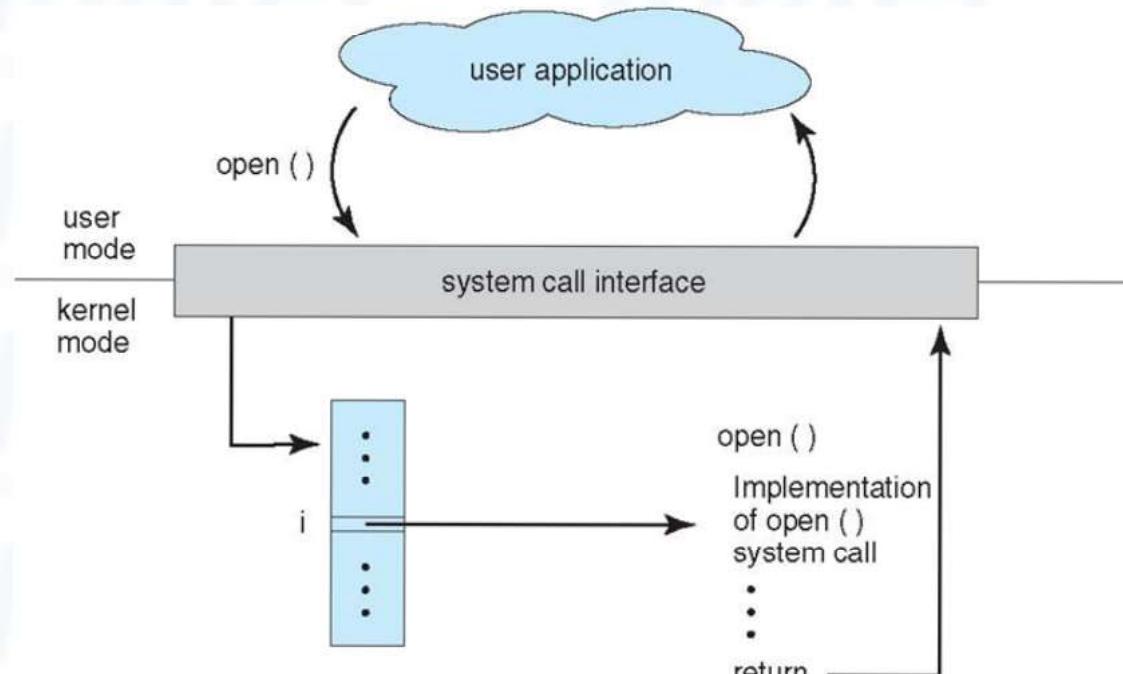
2.3.3 Tipos de System Calls

Geralmente, as System Calls nos SO's são agrupadas em seis categorias:

	Windows	Unix
1	Controle de Processos CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
2	Manipulação de Arquivos CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
3	Manipulação de Dispositivos SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
4	Manutenção de Informações GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
5	Comunicação CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
6	Proteção SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

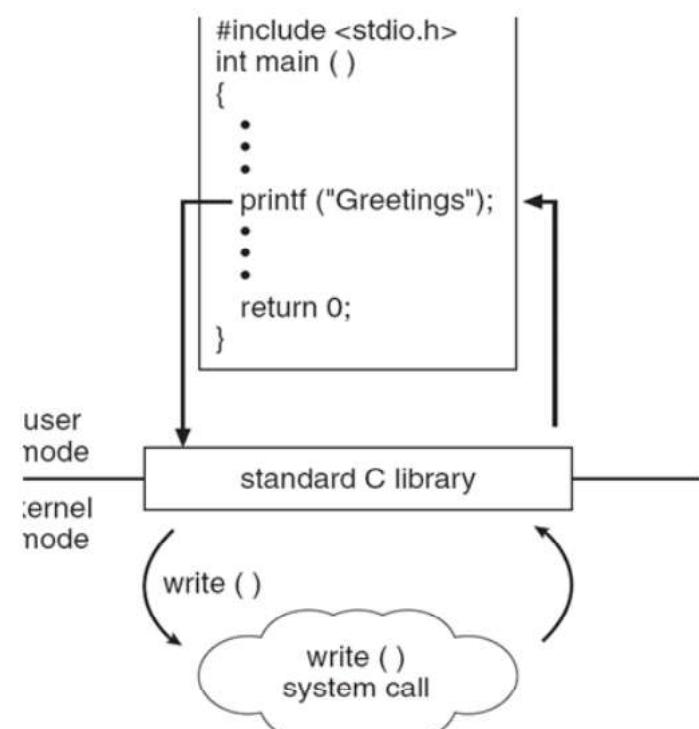
Alguns Exemplos de System Calls populares de cada categoria nos SO's Windows e Linux.

2.3. Chamadas de Sistema (System Calls)



Exemplo da chamada da System Call write() no UNIX:

Exemplo do processo de chamada da System Call open() no UNIX:



2.4. Programas de Sistema

Outra característica de um sistema operacional moderno é a presença de um conjunto de programas de sistema.

Os Programas de Sistema, também conhecidos como utilitários de sistema, fornecem um ambiente amigável para o desenvolvimento e a execução de programas.



Geralmente, são subdivididos em 7 (sete) categorias.

Vamos dar uma rápida olhada em alguns exemplos de cada uma destas categorias?

2.4. Programas de Sistema

1) Gerenciamento de Arquivos

Ex: Programas de manipulação de arquivos no Linux: cp,mv,rm,lp, etc..

2) Informações de Status

Ex: date, time, ps, df, etc..

3) Modificação de Arquivos

Ex: Bloco de Notas do Windows;

4) Utilitários

Ex: Browsers, planilhas, jogos etc..

5) Carga e Execução de Programas

Ex: loaders, linkers etc..

6) Comunicações

Ex: FTP, Telnet etc..

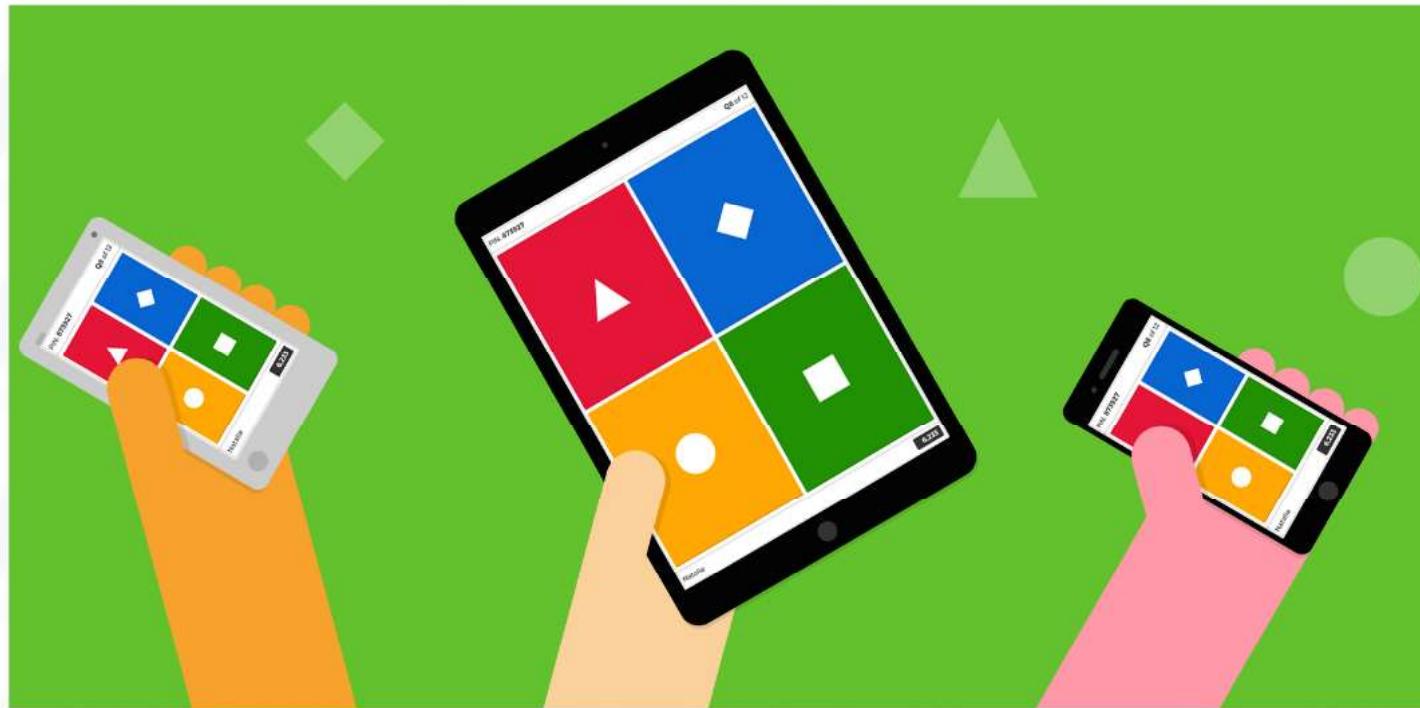
7) Suporte a linguagens de Programação

Ex: Compiladores, montadores, depuradores e interpretadores de programação comuns (C, C++, Java, Perl);



HORA DO

Kahoot!



ACESSE:

<https://kahoot.it>

2.5 Projeto e Implementação de SO's

Sucintamente, quais os passos necessários para criarmos um Sistema Operacional?

1) Definir Objetivos

- O SO será monousuário, multiusuário, multitarefa, tempo-real, para PC ou dispositivos móveis? Etc..

2) Definir Políticas (o que fazer) e Mecanismos (como fazer)

- É bom que mecanismos sejam implementados de forma independente de políticas, de modo que um SO permita mudar as políticas facilmente. Ex: modificação do quantum de CPU, configuração e recompilação do kernel, etc..

3) Definir a linguagem de Programação

- Hoje em dia, os SO's são implementados em linguagens de alto nível (C, C++...), e não mais em Assembly;



2.6 Estrutura do Sistema Operacional

Um ambiente tão grande e complexo como um Sistema Operacional, ao ser projetado, pode ser organizado utilizando de 4 (quatro) formas diferentes de organização, são elas:

- 1) Arquitetura Simples;*
- 2) Arquitetura em Camadas;*
- 3) Arquitetura de Microkernel;*
- 4) Arquitetura em Módulos.*



Vamos dar um olhada como funciona cada uma delas?

2.6 Estrutura do Sistema Operacional

2.6.1 Arquitetura Simples

- As interfaces e níveis de funcionalidades oferecidas pelo SO não estão bem separadas (Tudo colocado em um grande programa executável);
- Trabalha como se fosse um só módulo;

Vantagens

- Simplicidade e Desempenho (rapidez);

Desvantagens

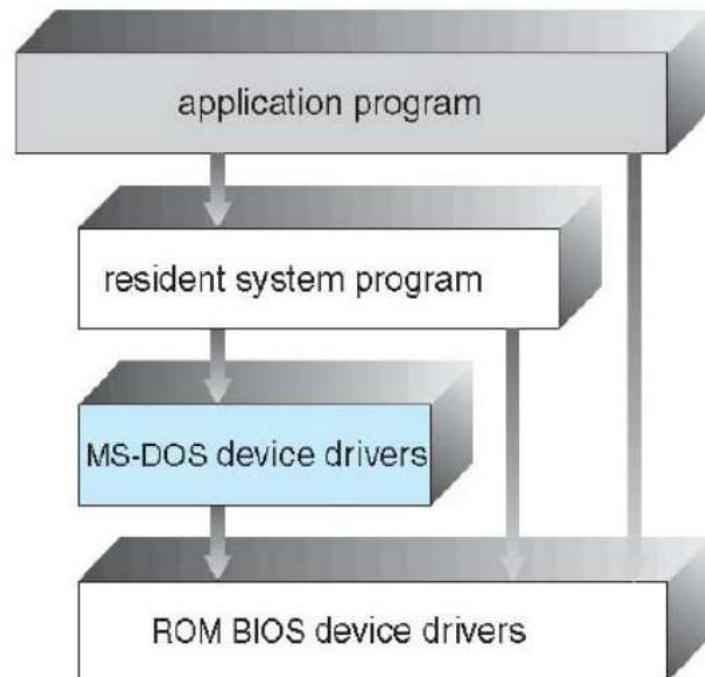
- Dificuldade de implementação e manutenção;
- Muito vulnerável a programas oportunistas. O sistema inteiro é interrompido quando programas de usuário falham;

Exemplos?

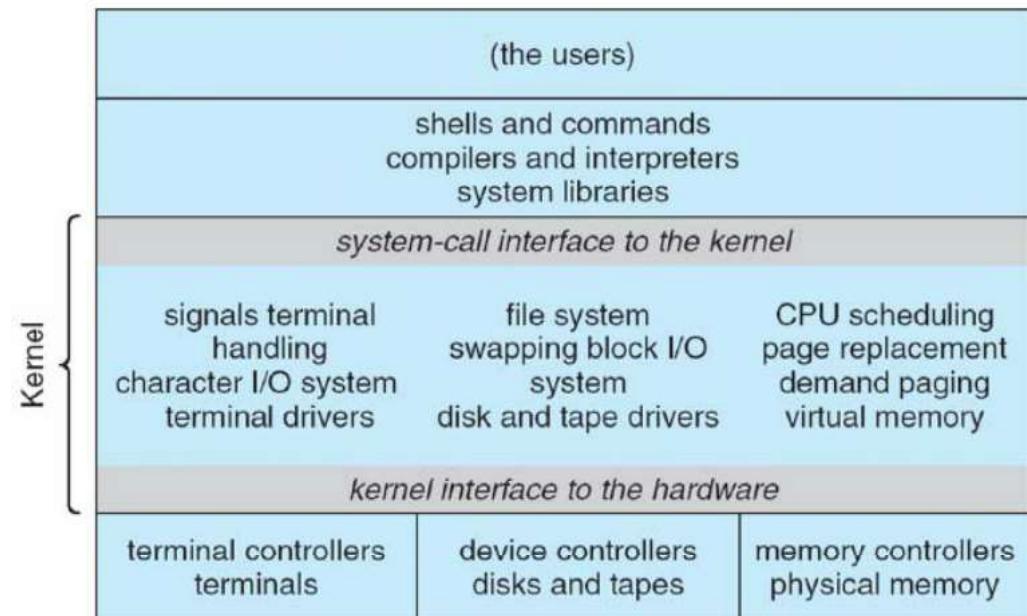
2.6 Estrutura do Sistema Operacional

2.6.1 Arquitetura Simples

Ex:



MS-DOS



UNIX (Tradicional)

2.6 Estrutura do Sistema Operacional

2.6.2 Arquitetura em Camadas

- Possui várias camadas com funções específicas;
- Cada camada usa somente as funções e serviços disponibilizados pelas camadas inferiores;

Vantagens

- Modularidade, facilidade de depuração e modificação;
- Cada Camada oculta das camadas de nível superior a existência de certas estruturas de dados e operações;

Desvantagens

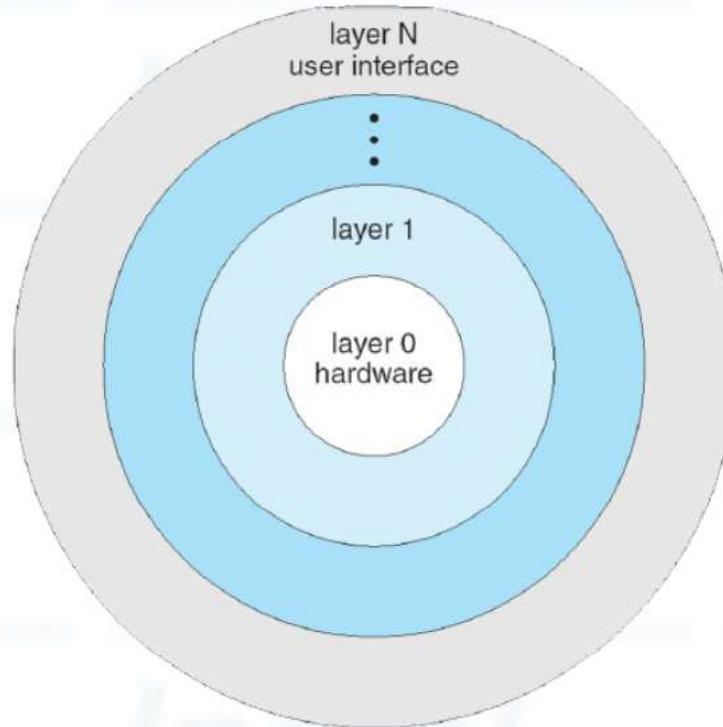
- Menor desempenho. Cada camada adiciona um overhead à uma chamada de sistema;
- Difícil divisão de camadas;

Exemplos?

2.6 Estrutura do Sistema Operacional

2.6.2 Arquitetura em Camadas

Ex:



THE OS, IBM OS/2 e MULTICS

Obs: Muitos dos Sistemas Operacionais utilizam a estruturação em camadas de forma parcial hoje em dia, ou seja, não são SO's desenvolvidos utilizando estritamente desta arquitetura.

2.6 Estrutura do Sistema Operacional

2.6.3 Arquitetura de Microkernel

- O Kernel do sistema é **enxuto**, possuindo somente os componentes essenciais (**Primitivas do núcleo**);
- Os **componentes essenciais** normalmente são os **códigos de baixo nível** necessários para interagir com o hardware e abstrações fundamentais;
- Todos os **outros aspectos de alto nível** como políticas de gerenciamento de recursos são implementados **fora do núcleo (Espaço do Usuário)**.
- Uma função importante do microkernel é fornecer uma **ponte de comunicação entre as aplicações e serviços** do espaço do usuário por meio de troca de mensagens;

Vantagens

- Kernel mais seguro e confiável;
- **Facilidade de extensão do SO;**

Desvantagens

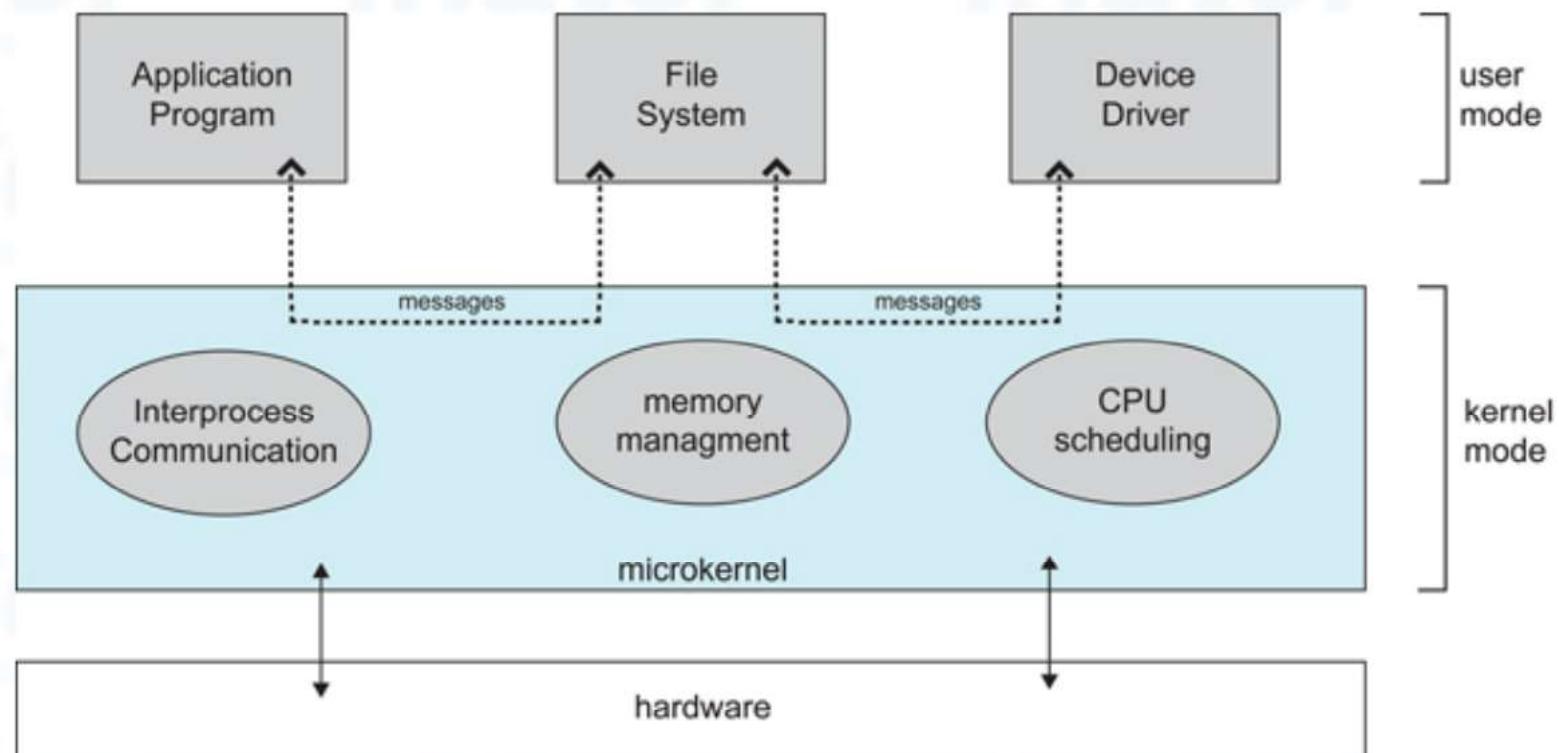
- **Desempenho comprometido** devido ao overhead de funções do sistema;

Exemplos?

2.6 Estrutura do Sistema Operacional

2.6.3 Arquitetura de Microkernel

Ex:



QNX, Mac OS X, Windows NT (Tradicional)

2.6 Estrutura do Sistema Operacional

2.6.4 Arquitetura em Módulos

- É o melhor tipo de arquitetura;
- O Kernel é enxuto, possuindo somente os componentes essenciais;

Vantagens

- Os demais módulos que compõem o SO são carregados, conforme a necessidade, durante o boot ou dinamicamente durante a execução do SO;
- Ao contrário da arquitetura em camadas, cada módulo pode chamar serviços oferecidos por outros módulos independentemente, pois todos os módulos estão no mesmo nível hierárquico;
- Ao contrário da arquitetura de Microkernel, cada módulo pode se comunicar diretamente com os demais, não mais através de passagem de mensagens no Kernel;

Desvantagens

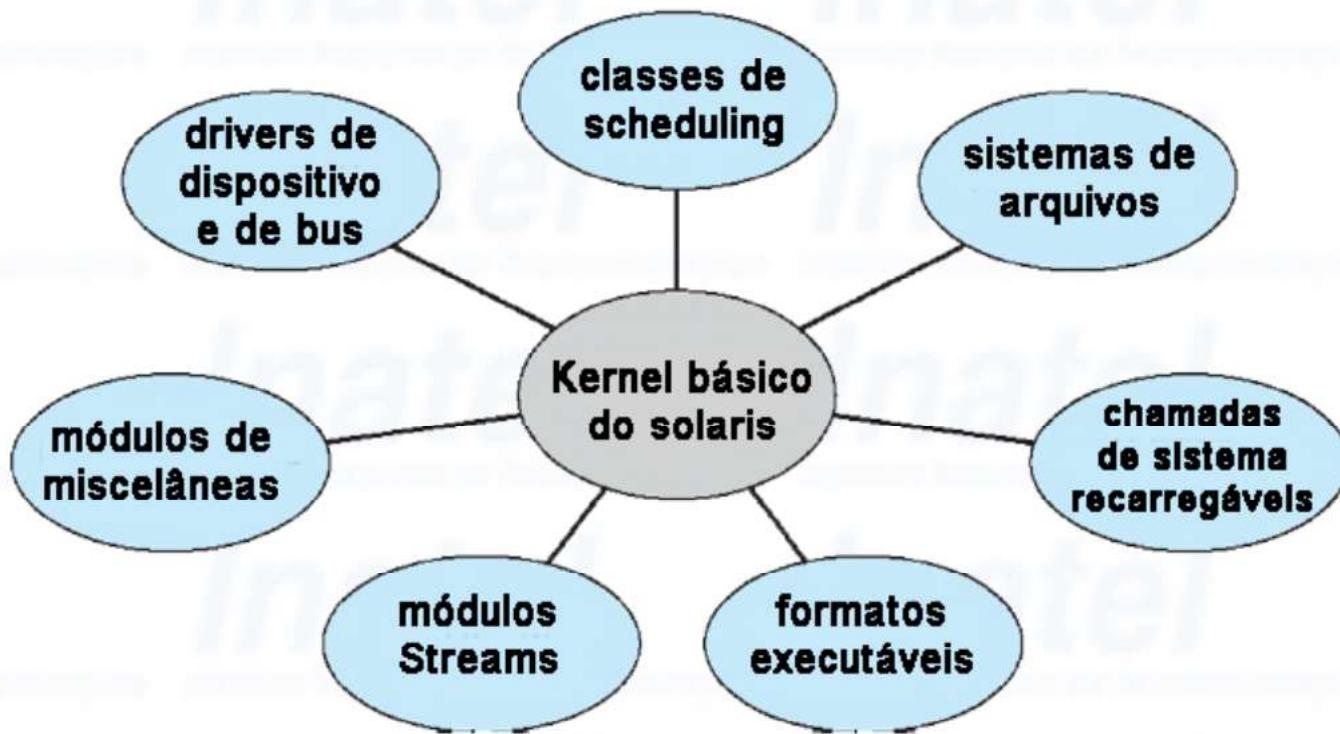
- Maior complexidade de desenvolvimento;

Exemplos?

2.6 Estrutura do Sistema Operacional

2.6.4 Arquitetura em Módulos

Ex:

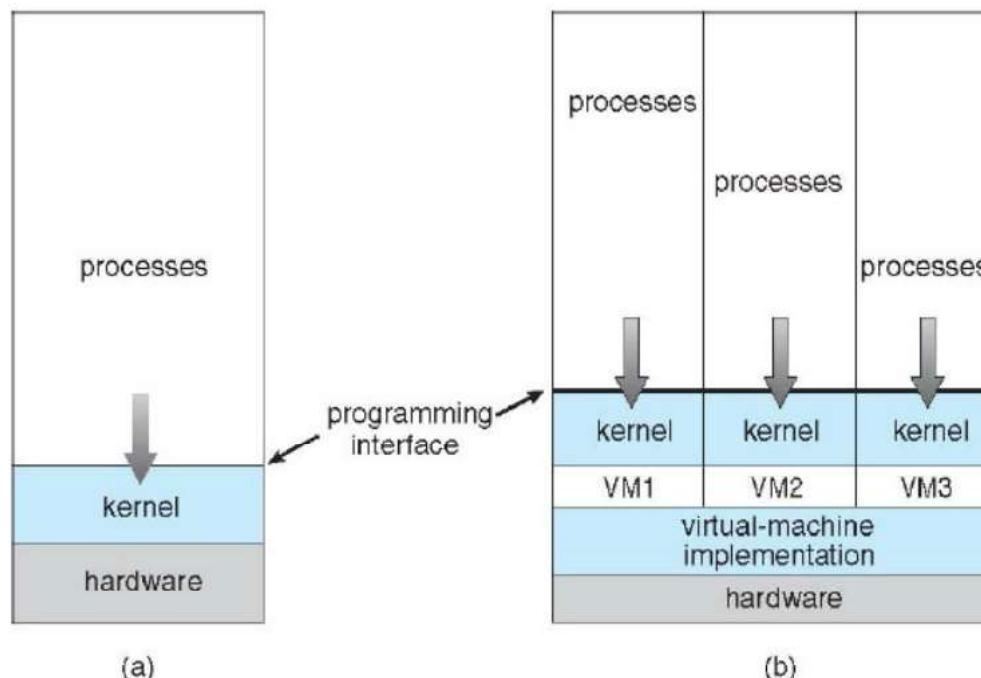


Versões mais recentes do Solaris, Linux e MacOS

2.7. Máquinas Virtuais

A idéia de Máquinas Virtuais surgiu com a IBM em 1972, quando ela desenvolveu seu sistema VM.

A idéia é criar vários ambientes de execução convidados (Guests) sobre um mesmo hardware, ou seja, criar várias Máquinas Virtuais convidadas com seus respectivos Sistemas Operacionais.

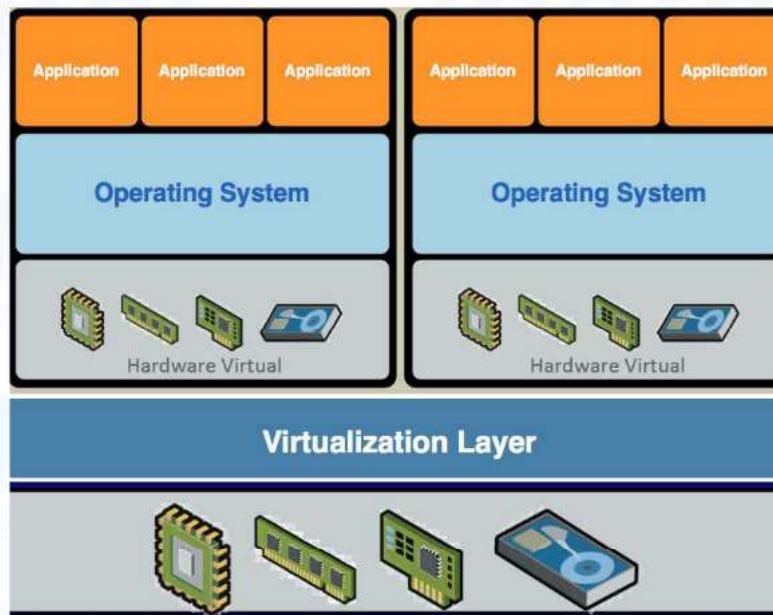


(a) Máquina Não Virtualizada (b) Máquina Virtualizada

2.7. Máquinas Virtuais

2.7.1 Vantagens

- Cada Máquina Virtual executa em um ambiente isolado, protegido do acesso das demais máquinas virtuais;
- Vários Sistemas Operacionais podem executar sobre um mesmo Hardware;
- Uma Virtual Machine (e seu SO), pode ser ajustada para executar uma determinada aplicação;



2.7. Máquinas Virtuais

A Virtualização de sistemas como discutimos até agora é apenas uma das muitas metodologias de emulação de sistemas. Existem três tipos de virtualização, são elas:

1) Virtualização Total (Full Virtualization)

A Máquina Virtual simula todo o hardware do sistema para permitir que um Sistema Operacional convidado seja executado de maneira isolada. O SO convidado não 'sabe' que está sendo emulado; Ex: Uso do VMWare, VirtualBox etc.

2) Emulação (Emulation)

Um sistema computacional virtual (Hardware e SO) é emulado sobre um sistema computacional real; Ex: JVM, .NET, Emuladores de Consoles de Games, etc.

3) Paravirtualização (Paravirtualization)

Apesar de ser mais complexa, a paravirtualização oferece maiores vantagens, visto que nesta modalidade, o sistema operacional a ser virtualizado 'sabe' que está sendo virtualizado e é modificado para tirar a maior vantagem possível dos recursos de hardware do computador hospedeiro. Ex: Xen Hypervisor;

2.8. Depuração e Geração do Sistema Operacional

2.8.1. Depuração

Depuração (Debugging) é a atividade de descobrir e corrigir erros (bugs) em um Sistema.

- Depuração envolve também a identificação de erros de Hardware e de problemas de desempenho;
- A maioria dos Sistemas Operacionais gravam as informações de erros (tanto de processos quanto de Kernel) em **arquivos de Logs** e guardam Capturas de Memória (**Memory Dumps**) para alertar aos operadores ou usuários que um problema ocorreu;
- Uma **falha no Kernel** é chamada de **Desastre**;
- Depuradores (Debuggers) são ferramentas que analisam arquivos de logs e "dumps" de memórias para a identificação de erros e desastres;



2.8. Depuração e Geração do Sistema Operacional

2.8.2. Geração do Sistema Operacional

SYSGEN (Geração de Sistema) é o processo de configuração ou geração de um SO para um Hardware específico.

- Um programa especial SYSGEN verifica o Hardware ou pergunta ao usuário informações para definir os parâmetros e os componentes da máquina;
- O programa SYSGEN executa durante a instalação de um SO em uma máquina. Ele copia do DVD ou ISO de instalação somente os módulos necessários do SO, de acordo com o Hardware, e os instala no disco;

Que tipos de informações o SYSGEN deve obter para instalar o SO com sucesso?

2.8. Depuração e Geração do Sistema Operacional

Que tipos de informações o SYSGEN deve obter para instalar o SO com sucesso?

- Qual ou quais CPU's serão usadas?
- Como o disco de inicialização será formatado (Quantas seções ou partições ele será dividido, e o que haverá em cada uma)?
- Qual a quantidade de memória disponível?
- Que dispositivos de I/O estão disponíveis?
- Que opções do SO são desejadas? (Tamanho de buffers, Algoritmos de Scheduling, Quantidade máxima de processos etc.)

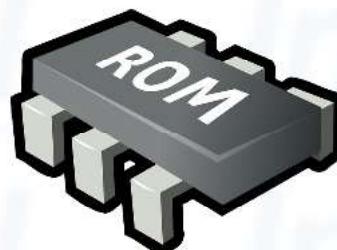


2.9 Inicialização do Sistema

Após um Sistema Operacional ser instalado, ele deve ser disponibilizado para uso pelo Hardware.

Mas como o Hardware sabe onde está o Kernel ou como carregar este Kernel?

- Quando a CPU é resetada, o registro de instrução é carregado com um endereço pré-definido de memória;
- Este endereço aponta para uma posição de Memória ROM, que contém o endereço inicial do programa bootstrap;
- O Programa bootstrap pode rodar um programa-diagnóstico do Hardware, carregar um segundo programa de bootstrap do disco na memória, e finalmente, carregar o SO na memória;
- Em Sistemas Operacionais de Sistemas Portáteis ([Ex: Smartphones](#)), o **SO reside integralmente em ROM/EPROM**, não havendo necessidade de um programa bootstrap;



HORA DO

Kahoot!



ACESSE:

<https://kahoot.it>

Fim do Capítulo 2



EXERCÍCIOS