



Universidade Federal de Uberlândia  
Faculdade de Computação  
Sistemas Operacionais



# **Introdução aos Sistemas Operacionais**

Prof. Dr. Marcelo Zanchetta do Nascimento

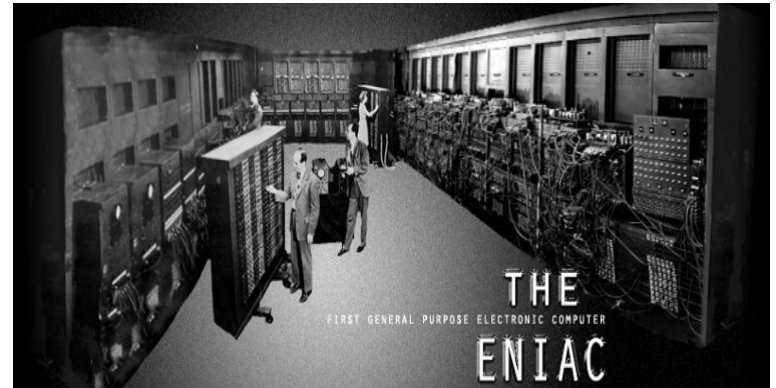
# Sumário

- História
- Sistemas Operacionais (SO)
- Interpretador (usuário)
- Chamada de Sistema (System Call)
- Estrutura do SO
- Monitoramento de um SO
- Exemplos de SO
- Leituras Sugeridas

# História: Como começou esse software?

## Primeira Geração (1945-1955): Tubos

- Relés mecânicos e tubos de válvulas;
- Operação via conexão de cabos para codificação;
- O SO ainda não existia.



**Exemplo: ENIAC construído por William Mauchley e seu estudante J. Presper Eckert na Universidade da Pensilvânia**

1950: aprimoramento com a introdução das perfuradoras de cartões.

# História

## Segunda Geração (1955-1965): Transistores e sistema batch

- Introdução dos transistores: computadores mais confiáveis;
- Separação entre as “funções” de projetista, fabricante e programadores;
- Essas máquinas de grande porte (denominada de mainframes) em salas especiais;
- A codificação de programas em papel (FORTRAN ou linguagem de máquina) e perfuração de cartões;
- Início do **sistema em lote (batch)** para redução de tempo: gravar as tarefas em fita magnética.

# História

## Segunda Geração (1955-1965): Transistores



Figura 1: Fase para processamento em sistema batch (Fortran Monitor System)

### Fases:

- Programador leva os cartões para o IBM 1401;
- O equipamento lê os cartões para a fita;
- Programador coloca a fita no IBM 7094 para o processamento;
- Programador coloca a fita no IBM 1401 que imprime a saída.

# História

## Segunda Geração (1955-1965): Transistores

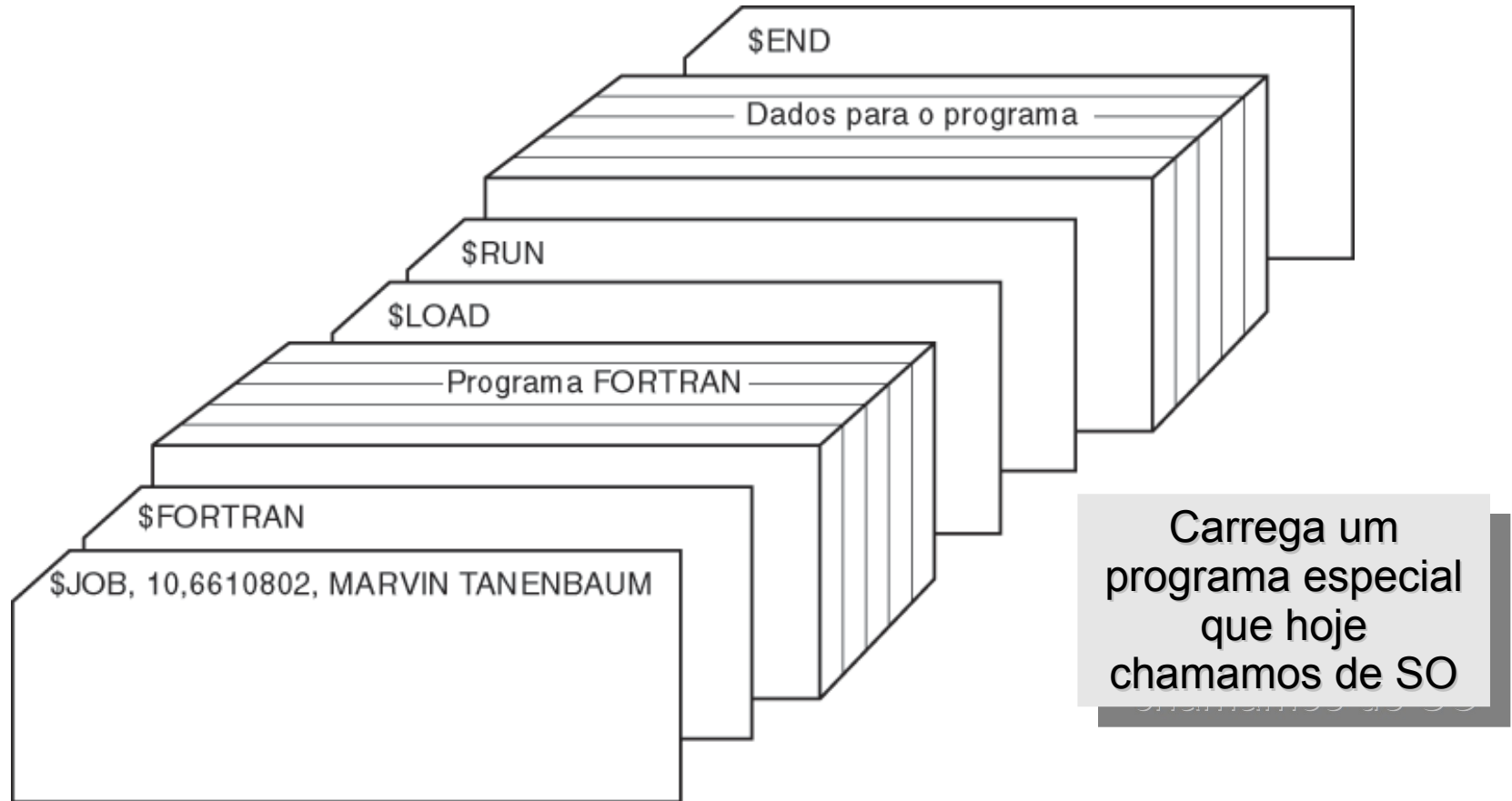


Figura 2: Estrutura de um job em um sistema operacional Fortran Monitor System.

# História

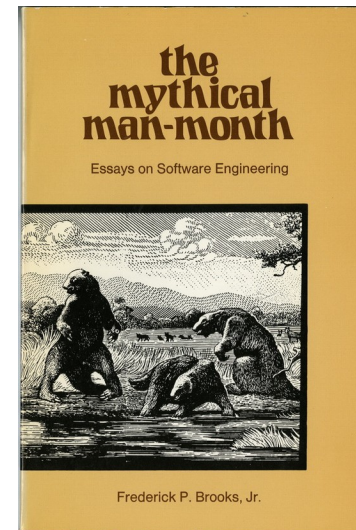
## Terceira Geração (1965-1980): CI e Multiprogramação

- Fabricantes ofereciam computadores: a) orientado as palavras (IBM 7094); b) orientado aos caracteres (IBM1401);
- Com o surgimento do Circuitos Integrados (Cis) - (IBM/360) : “Família de máquinas”;
- IBM inova com o software **System/360**;
  - Computação Científica e Processamento de Dados;
  - Diferentes máquinas (melhor poder computacional): 370, 4300, 3080 e 3090.

# História

## Terceira Geração (1965-1980): CI

- OS/360 executado em qualquer modelo da IBM, torna-se o **SO genérico**;
  - Enorme, complexo e desenvolvido em linguagem de montagem;
- *Fred Brooks, projetista, escreveu um livro sobre experiências com o SO em que a capa mostra um rebanho de animais pré-históricos presos em um fosso (analogia a capa do SILBERSCHATZ et al. 2000).*
- Introduz a multiprogramação – IBM 7094 (O que é?).

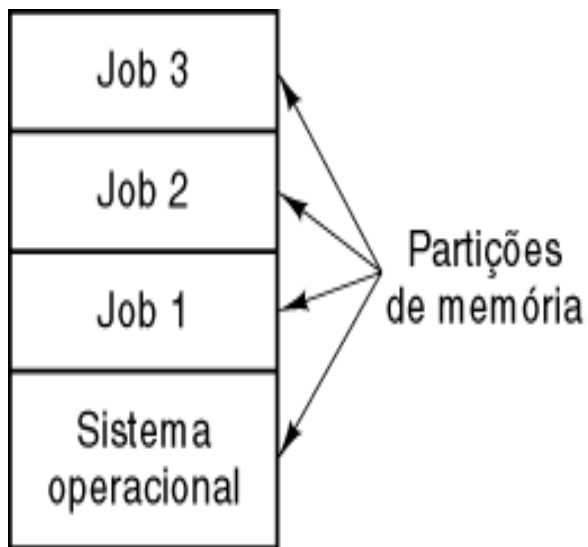




# História

## Terceira Geração (1965-1980): CI

- Multiprogramação: manter vários programas na memória ao mesmo tempo e, enquanto um programa realiza E/S, outro pode ser executado na CPU.



CPU Bound (uso intensivo da CPU)  
I/O Bound (uso intensivo de E/S)

Figura 3: Três jobs na memória RAM

# História

## Terceira Geração (1965-1980): CI

- Técnica spooling (simultaneous peripheral operation online): transferir jobs de cartões para disco magnético sem fitas magnéticas;
- Compatible Time Sharing System (CTSS) - sistema compatível de tempo compartilhado desenvolvido no MIT;
- Computador “utilitário” - MULTICS (Serviço de computação e informação multiplexada):
  - Suportava centenas de usuários;
  - Incorporava a memória virtual.
- Outro importante equipamento foi o minicomputador (DEC PDP-1).

# História

## Terceira Geração (1965-1980): CI

- Ken Thompson usou um PDP-7 para criar uma versão monousuário do MULTICS - posteriormente o UNIX;
  - Código-fonte: Organizações desenvolveram suas próprias versões e as principais são: SYSTEM V (AT&T) e BSD (Distribuição de Berkeley da Universidade da Califórnia);
  - O IEEE desenvolveu o padrão para UNIX: POSIX (portable operating system interface).
  - A. Tanenbaum (1987) desenvolveu o MINIX com objetivo educacional: [www.minix3.org](http://www.minix3.org);
  - Linus Torvalds (1991): Linux foi desenvolvido sobre o MINIX.

# História

## Quarta Geração (1980-Presente): Computadores Pessoais

- O PC com CPU Intel 8080 e SO denominado CP/M (programa de controle para microcomputador) escrito por Gary Kildall (Posteriormente criou a Digital Research para aperfeiçoá-lo);
- A IBM projetou o PC e na demanda de um software:
  - Contato com Bill Gates para licenciar o interpretador de programa Basic e fornecer um SO;
  - Bill Gates comprou a “Seattle Computer Products” e o sistema operacional DOS (monousuário) e contratou o criador Tim Paterson, o qual o revisou e criou o MS-DOS;

Filmes: Piratas do Vale do Silício e Revolution OS

# História

## Quarta Geração (1980-Presente): Computadores Pessoais

- Circuitos Integrados em Larga Escala;
- Sistemas multi-usuários (UNIX);
- Steve Jobs investiu na construção de Apple com interface gráfica - Graphical User Interface (GUI);
- Crescimento de redes de computadores e o surgimento de sistemas operacionais distribuídos;
- Sistemas de tempo real (empregado para controle de procedimentos que devem responder dentro de um certo intervalo de tempo).

# História

## Quinta Geração (1990-Presente): Dispositivos Móveis

- O primeiro smartphone surgiu em 1990 quando a Nokia combinou dois dispositivos: telefone e PDA (Personal Digital Assistant);
- Os sistemas Android e iOS são os mais empregados.

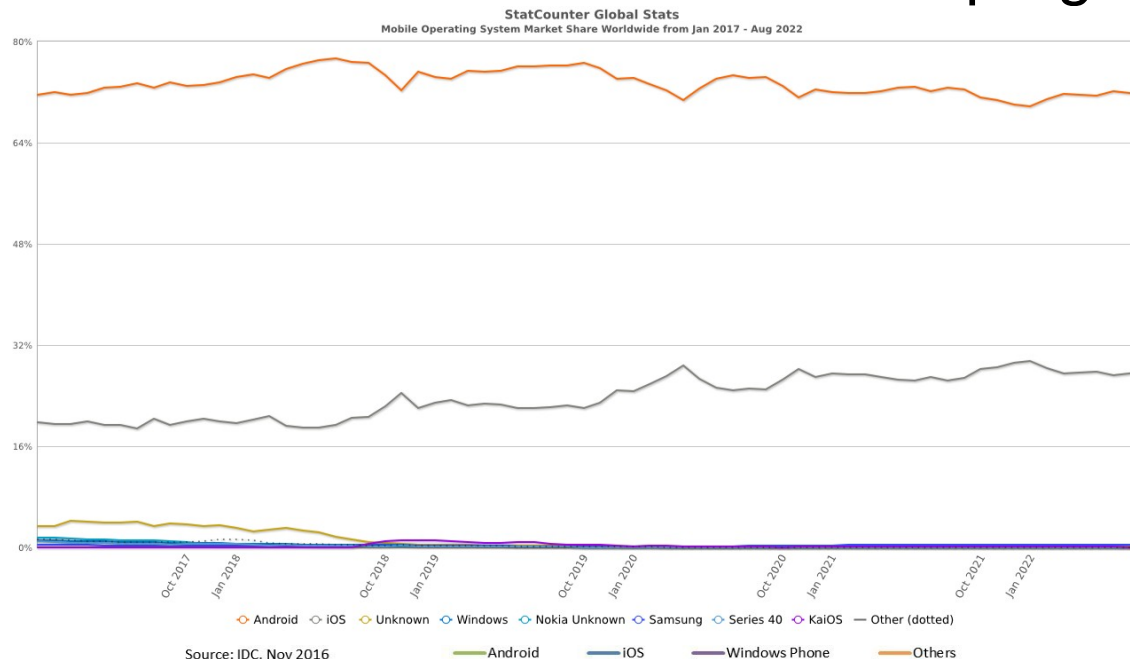


Figura 4: Sistemas operacionais para smartphone (<https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201701-202208>).

# História

## Evolução do UNIX:

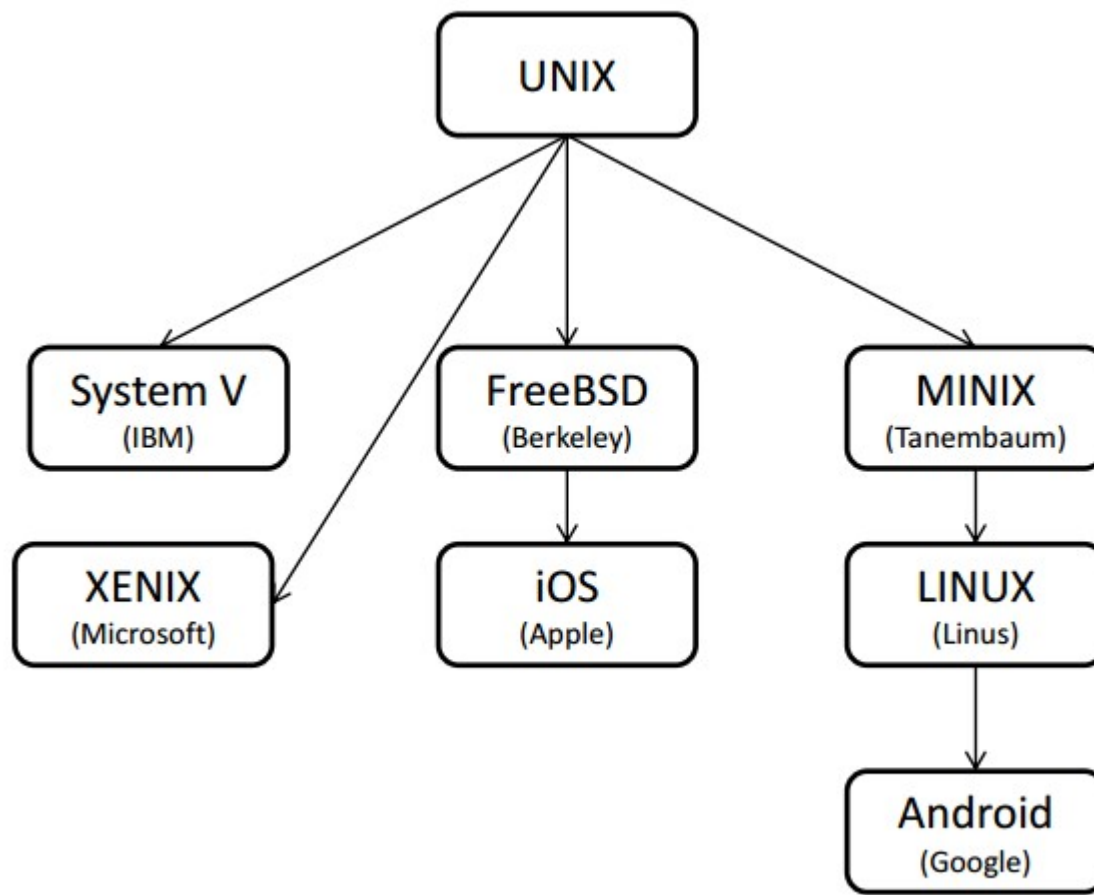


Figura 5: Versões de Sistemas Operacionais Baseados no UNIX.

# Sumário

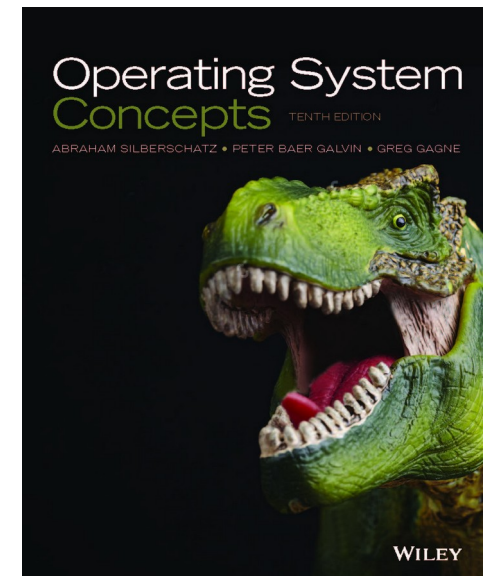
- Sistemas Operacionais (SO)
- Interpretador (usuário)
- Chamada de Sistema (System Call)
- Estrutura do SO
- Monitoramento de um SO
- Exemplos de SO



# Por que estudar Sistemas Operacionais?

Although there are many practitioners of computer science, only a small percentage of them will be involved in the creation or modification of an operating system. Why, then, study operating systems and how they work? Simply because, as almost all code runs on top of an operating system, knowledge of how operating systems work is crucial to proper, efficient, effective, and secure programming. Understanding the fundamentals of operating systems, how they drive computer hardware, and what they provide to applications is not only essential to those who program them but also highly useful to those who write programs on them and use them.

Frase de Avi Silberschatz, Peter Baer Galvin, Greg Gagne em seu livro: "Operating System Concepts", Tenth Edition, John Wiley & Sons, Inc.



# Função do Sistema Operacional (SO)

- O SO é um **alocador de recursos** e um **controlador de programas** que faz controle de hardware e gerencia a execução de programas de usuário.

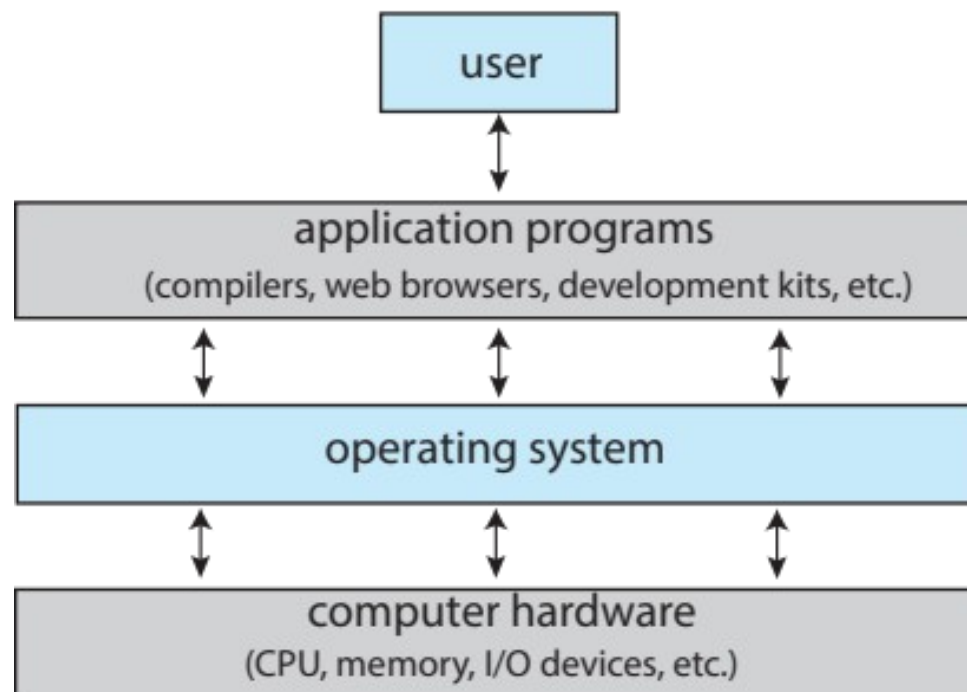


Figura 6: Visão Abstrata dos Componentes de um Sistema

# Serviços do SO

O SO, um software fundamental, executa no modo kernel (modo supervisor) e fornece serviços ao usuário.

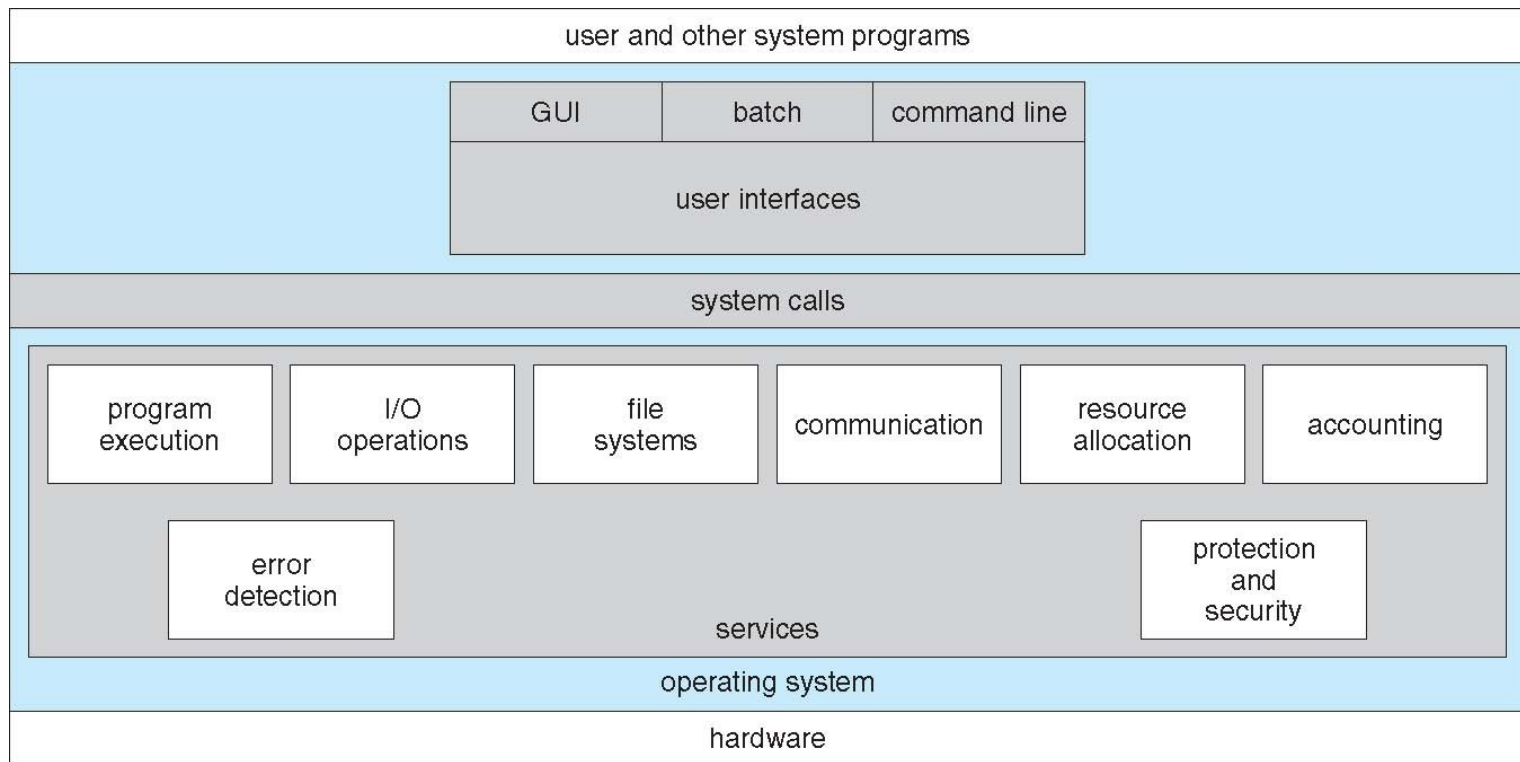


Figura 7: Camadas entre as etapas em um SO

# Sumário

- Interpretador (usuário)
- Chamada de Sistema (System Call)
- Estrutura do SO
- Monitoramento de um SO
- Exemplos de SO

# Interpretador

- **Interface do SO e usuário:**
  - Linha de comando: utiliza comandos em modo texto;
  - Interface de usuário gráfica (graphic user interface – GUI): sistema de janela com apontador para direcionar recursos de E/S;
  - Interface touch-screen: pressionando e deslizando os dedos através da tela.
  - Exemplo: Linux, Windows, Apple IOS, etc.

# Interpretador

- **Interpretador de comando:** recebe e executa o próximo comando especificado pelo usuário;
- **SO** pode incluir no kernel o **interpretador no kernel** ou tratar como **um programa especial**, o qual é executado quando o usuário entra no sistema;
  - **Exemplo:** Shells: Konsole;
- O próprio interpretador contém o código para executar o comando.
  - **Exemplo: Tarefa: Excluir um arquivo:** Ir para uma seção de seu código que define os parâmetros e faz uma chamada (**system call**);

# Interpretador

- Implementação: os comandos são executados por meio de programas do sistema;
- O interpretador não entende o comando. Apenas identifica um arquivo e carrega para memória RAM para execução.
  - **Exemplo:** em Unix o comando “rm aula1.txt”
  - Procura o arquivo **rm** e carrega para memória e executa com os parâmetros de aula1.txt.
- A vantagem que o interpretador não precisa ser modificado para novos comandos no SO.
  - Exemplo de interpretador shell: bash e csh.

# Interpretador

- **Interface gráfica (GUI):** para ativar um programa é necessário apenas clicar em um botão ou ícone com o mouse;
- A grande maioria dos sistemas inclui as interfaces: interpretador de comandos e GUI;
  - **Exemplos:**
    - MAC OS X tem GUI “Aqua” com kernel UNIX e usa interpretador shell;
    - Solaris tem interpretador com GUI (KDE).



# Sumário

- Chamada de Sistema (System Call)
- Estrutura do SO
- Monitoramento de um SO
- Exemplos de SO

# System Call (Chamada de Sistema)

- Fornece uma interface com os serviços disponíveis em um SO (rotinas em C ou C++);
- São utilizados pelos desenvolvedores de aplicação por meio de uma *Application Program Interface (API)*, que permitem o uso de uma chamada ao sistema;
- **Exemplo:**
  - Win32 API para Windows;
  - POSIX API para sistemas baseados em POSIX (*Portable Operating System Interface*), os quais incluem as versões do UNIX, Linux e Mac OS X;
  - Java API para Java Virtual Machine (JVM).

# System Call

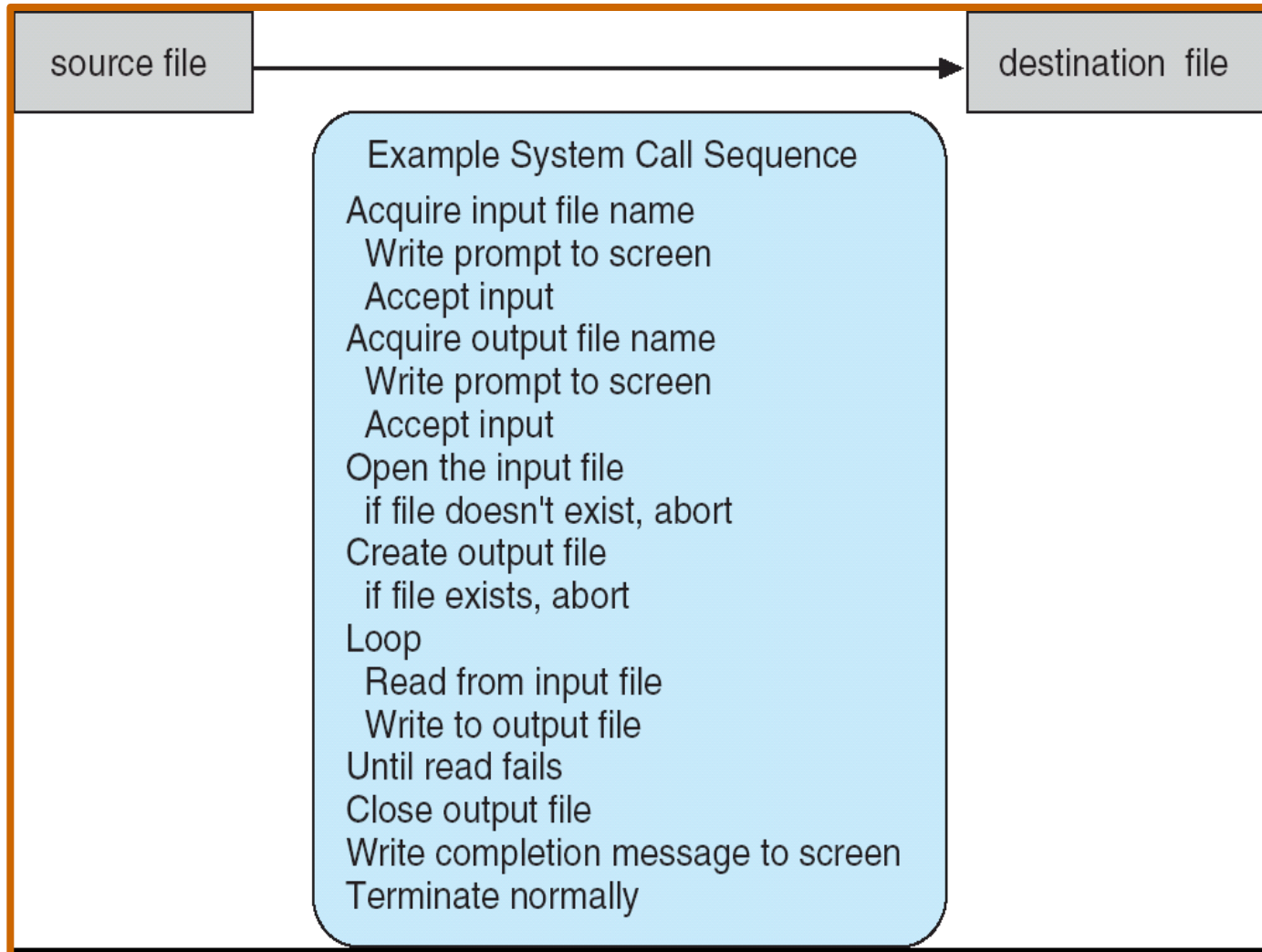


Figura 8: Exemplo de uma chamada de sistema para um arquivo

# System Call

## Implementação

- Um número é associado a cada system call;
  - Uma interface de comando mantém uma tabela indexada de acordo com cada número;
- A interface solicita uma **chamada no kernel do SO** e retorna o estado da chamada e o valor de retorno;
  - A **solicitação não precisa saber** como o system call daquele SO é implementado;
- Os detalhes da interface do SO são escondidos do programador pela API:
  - Controlada pela **biblioteca de suporte** em tempo de execução.

# System Call

## Exemplo 1: Implementação da chamada de sistema **Open**

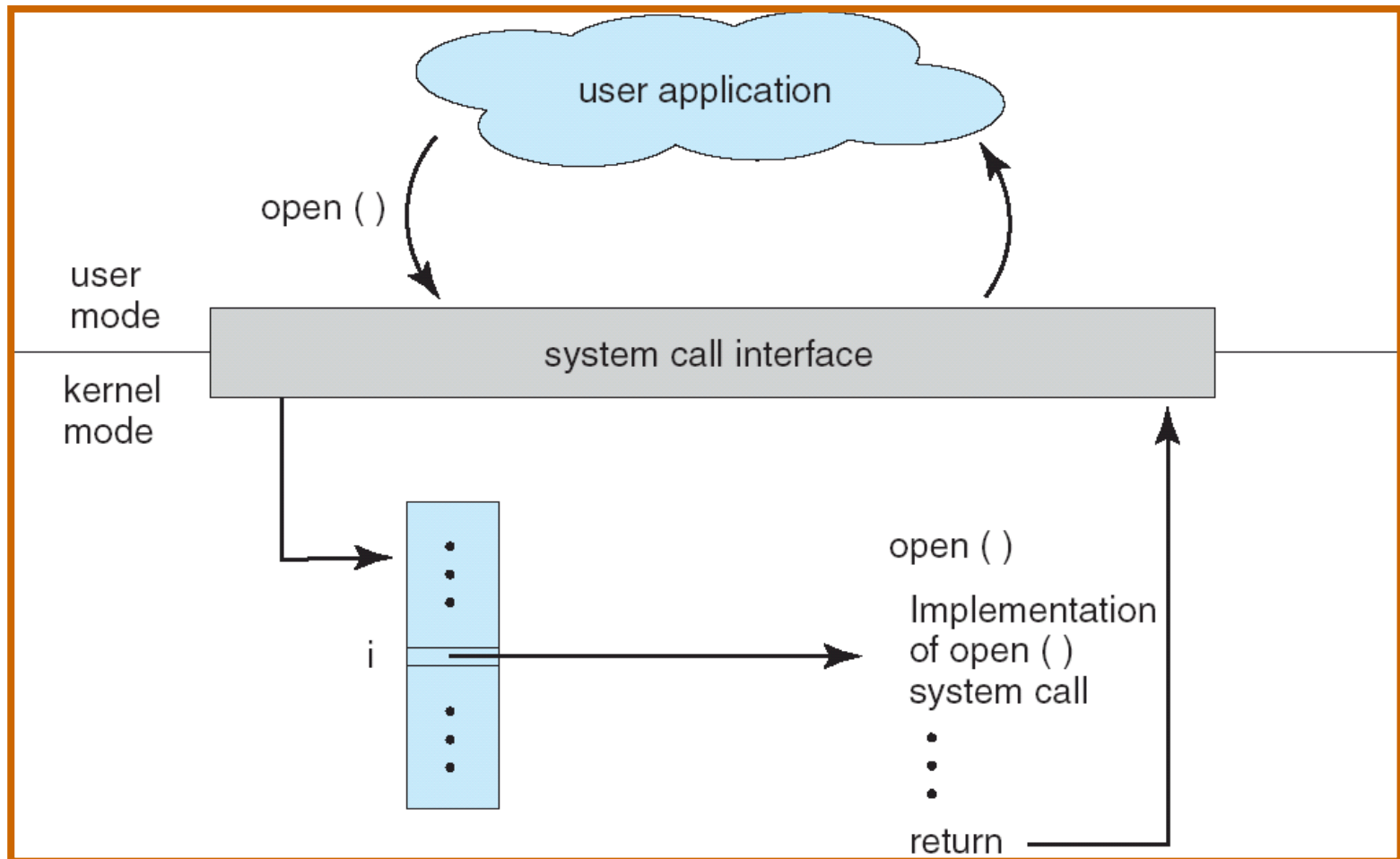


Figura 9: Aplicação de usuário que invoca a chamada de sistema

# System Call

**Exemplo 2:** um programa em C invocando uma chamada a biblioteca **printf**, a qual solicita a system call **write( )** do SO.

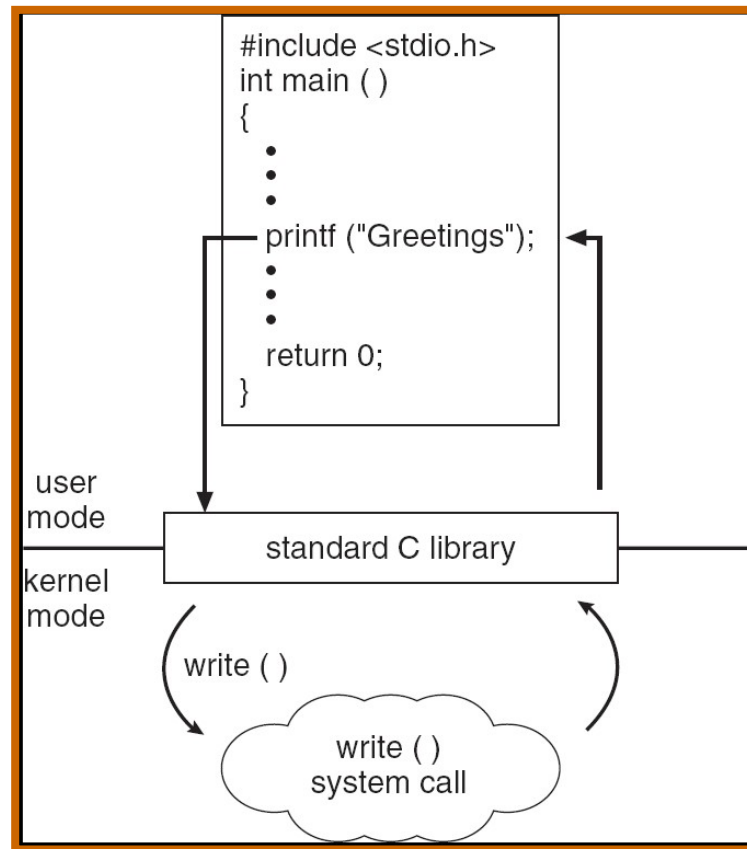


Figura 10: A chamada de sistema da biblioteca usada pela função printf.

# System Call

**Passagem de parâmetros:** Outras informações devem ser passadas:

- Não apenas a identificação que pode variar de acordo com cada SO;
- Há **3 formas de passar os parâmetros** para o SO:
  - ***Simples:*** por meio de registradores;
  - Pode ter mais parâmetros que registradores:
    - Bloco ou tabela na memória e o endereço do bloco é passado como parâmetro em um registrado;
    - **Exemplo:** Aproximação adotada no Linux e Solaris.

# System Call

**Simples:** por meio de registradores

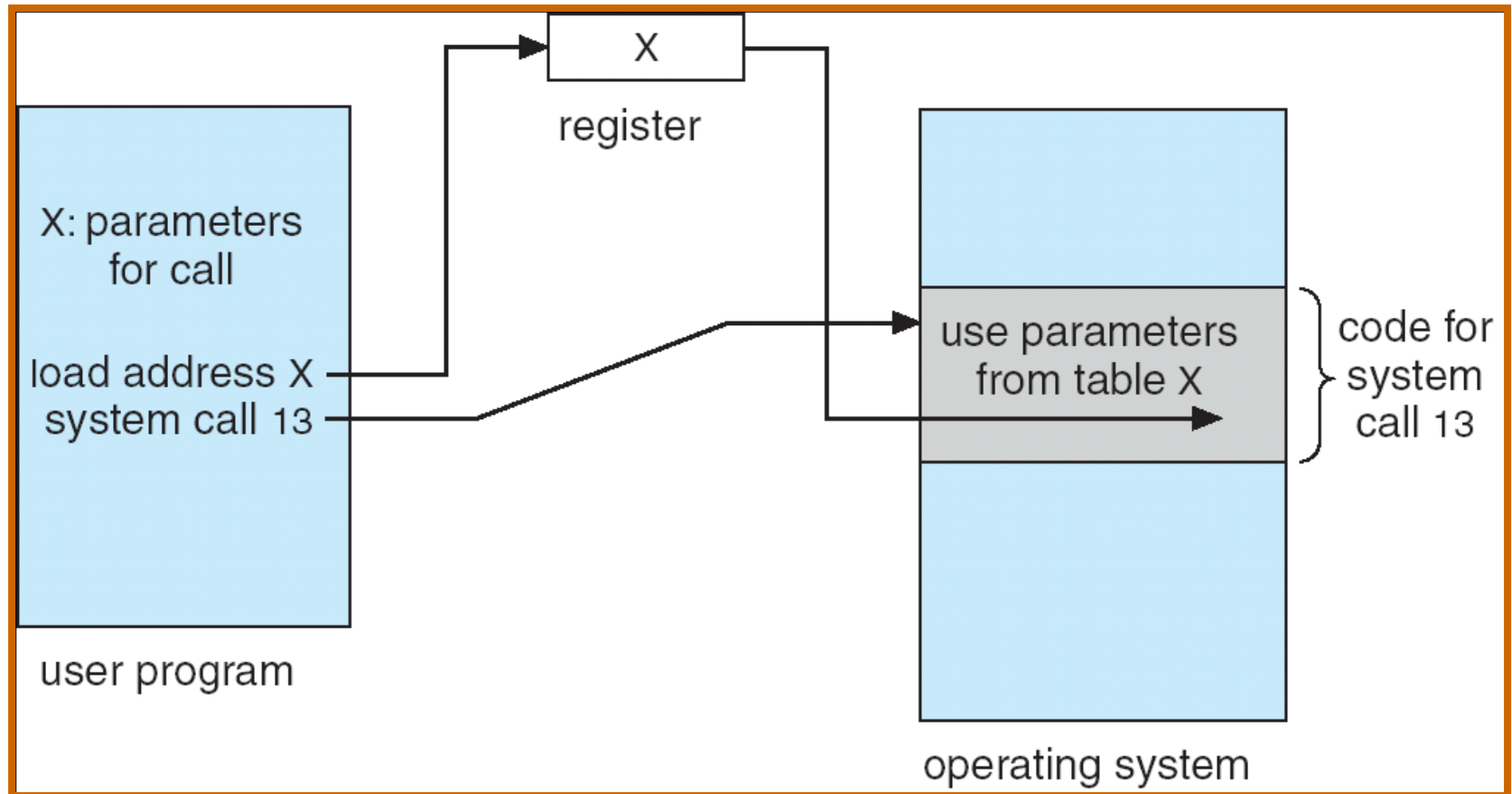


Figura 11: Passagem de parâmetros como uma tabela.



# System Call

## Passagem de parâmetros:

- **Pilha:** programas colocados na pilha e, posteriormente, são retirados pelo SO. Não limita o número ou extensão dos parâmetros.
- **Java:** método java invoca um código em C ou C++, que seja nativo à arquitetura em que o programa está sendo executado;
  - Feito por meio de Java Native Interface (JNI).

# System Call

No PC, em Linux, execute o seguinte comando para acompanhar as chamadas:

```
$ strace -c ls
```

- Mostra um sumário das chamadas solicitadas durante a execução do comando “ls”.

```
$ strace -o teste1.out ls
```

- Mostra o sumário das chamadas solicitadas durante a execução do “ls” e salva no arquivo “teste1.out”

```
$ man strace
```

- Mostra informações (manual) do comando **strace**

# System Call

- **Onde são empregadas as Chamadas:**
  - Controle de Processos (end, abort, load, execute, ...);
  - Administração de Arquivos (open, close, read, write, ...);
  - Administração de Dispositivos (release device, read, write, ...);
  - Manutenção de informação (get ou set time, get process, file ...);
  - Comunicações (conexões, send, receive messages, status information, ...).

# System Call

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

Figura 12: Exemplo de Chamadas de Sistema em SO

# Sumário

- Estrutura do SO
- Monitoramento de um SO
- Exemplos de SO

# Estrutura do SO

- Um SO precisa ser arquitetado com cuidado para funcionar de modo apropriado e ser modificado com facilidade.
- **Como são divididas as estruturas de um SO?**
  - Exemplo:
    - Monolítico (uma camada)
    - Em camadas;
    - MicroKernel;
    - Modular;
    - Híbrida.

# Estrutura do SO

## Monolítico

- MS-DOS escrito para prover o **máximo de funcionalidade** no menor espaço;
- **Não é dividido** em módulos;
- As **interfaces e níveis de funcionalidade** não são bem separadas.

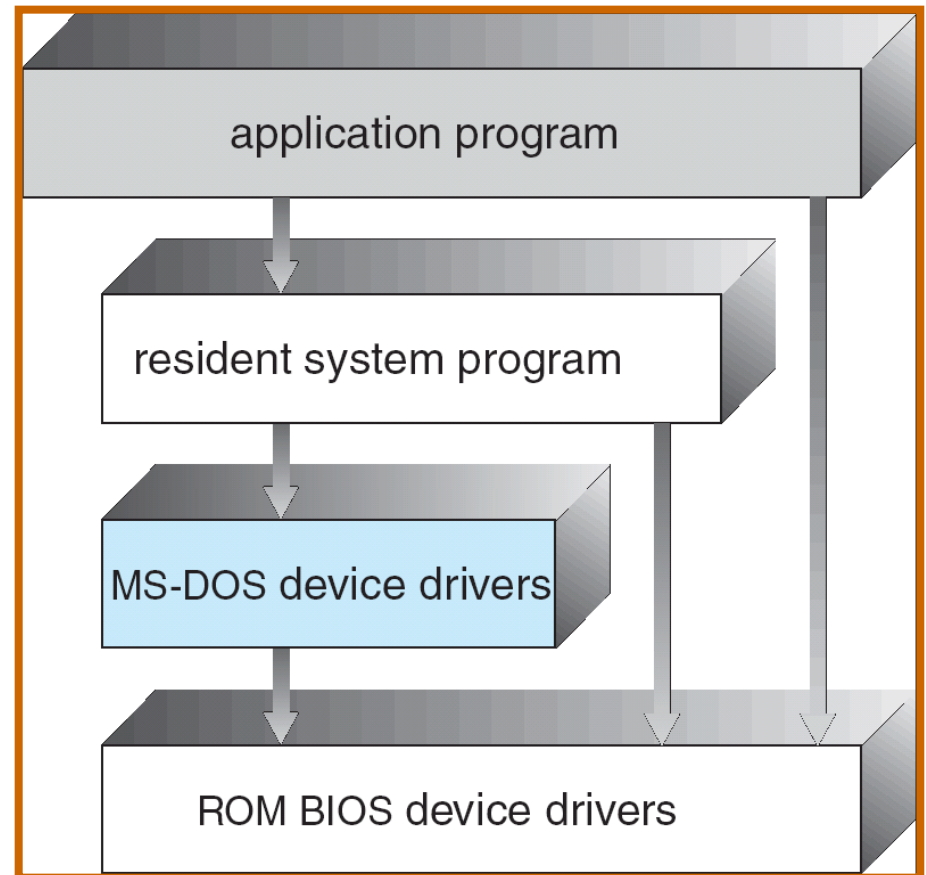


Figura 13: Estrutura de camadas de MS-DOS

# Estrutura do SO

## Abordagem em camadas:

- O SO é dividido em um número de camadas em que cada uma construída no topo de uma camada inferior:
  - A camada mais baixa (camada 0) é o hardware;
  - A camada mais alta (camada N) é a interface com o usuário.
- As camadas construídas em cada uma utiliza funções (operações) e serviços somente de camadas de mais baixo nível;
  - Dificuldade na “definição das camadas”.



# Estrutura do SO

## Abordagem em camadas:

- Exemplo:
  - O software do dispositivo para o **espaço em disco** utilizado pelo algoritmo de **Memória Virtual**, deve estar em nível inferior ao das rotinas de **gerenciamento de memória** (espaço em disco);
  - Exemplo: Windows NT (apenas as primeiras versões).

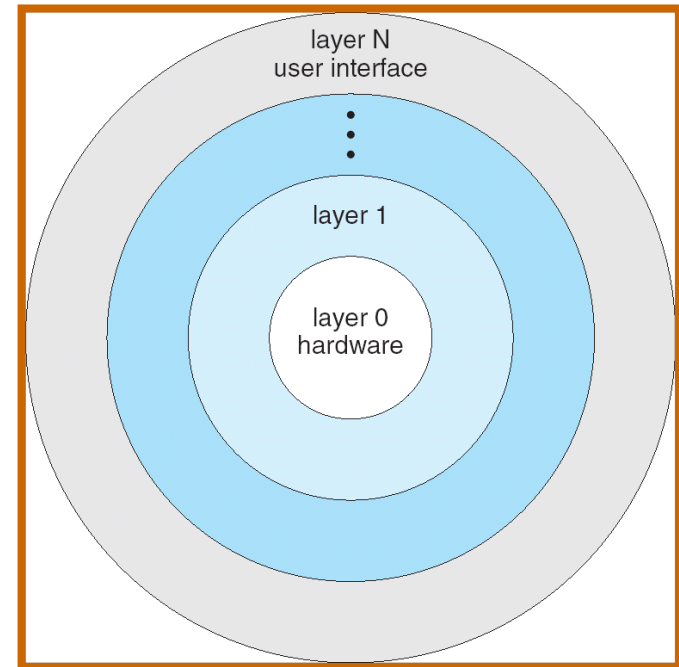


Figura 14: Sistema Operacional em Camadas

# Estrutura do SO

## Microkernels

- Estrutura o SO removendo todos os componentes não essenciais do *kernel*;
- Implementa-os como programas de sistema e de nível “usuário”;
- Pouco consenso em quais serviços devem permanecer no kernel;
- Geralmente, estão alocados os controles: memória e processo;
- Fornecer comunicação (troca de mensagem).

# Estrutura do SO

## Microkernels

- Windows NT (Win32).

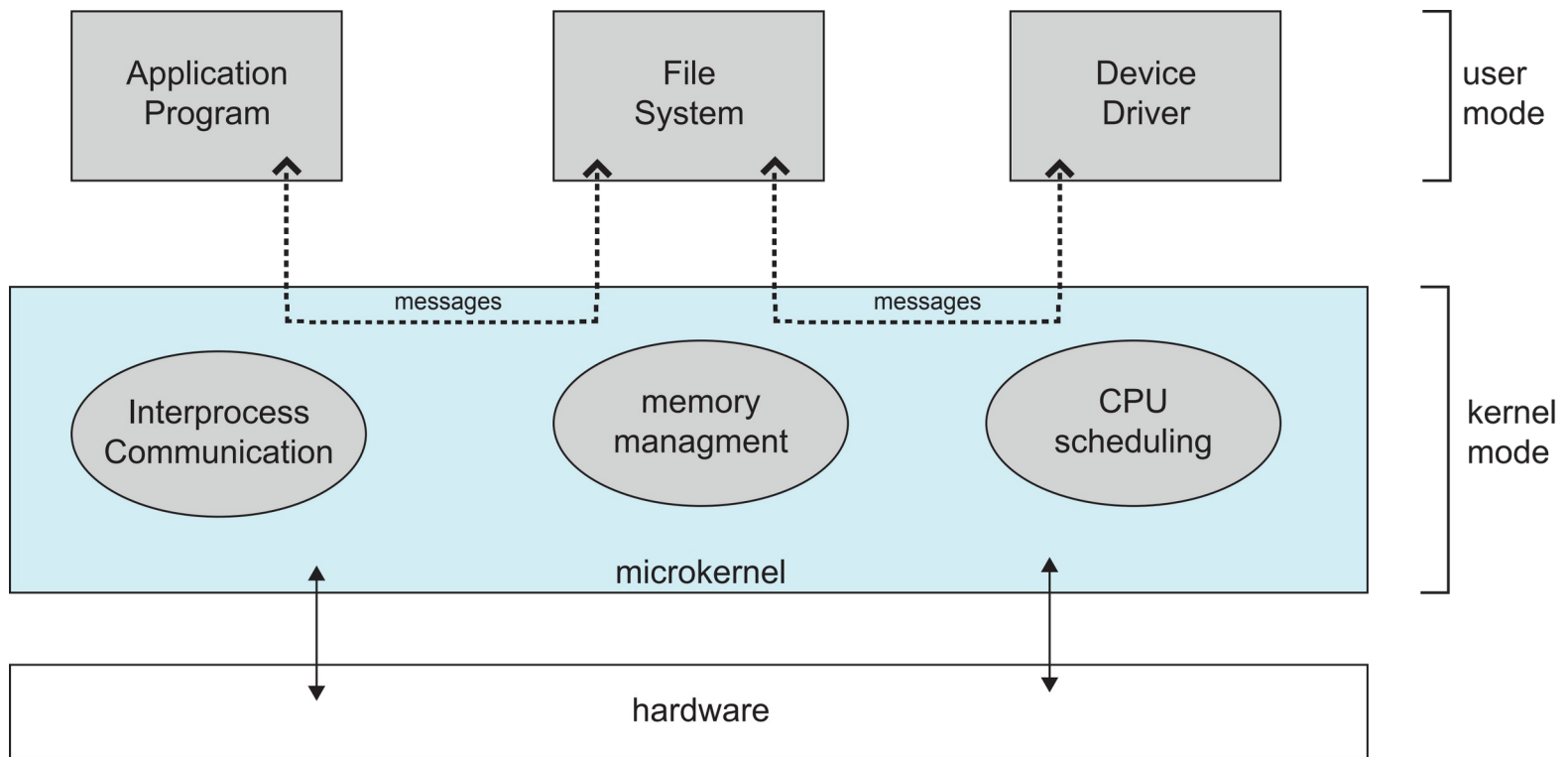


Figura 15: Estrutura em microkernel

# Estrutura do SO

- O SO moderno (Solaris, Linux) implementa os módulos:
  - Usa técnicas de orientação objetos;
  - Cada componente é separado;
  - Kernel possui um conjunto de componentes;
  - Cada um é carregado de acordo com a necessidade.
- Fácil de manter, atualizar e depurar: foca em um módulo de cada vez;
- Eficiente: módulos podem interagir cada um ao outro diretamente.

# Estrutura de módulo: Solaris

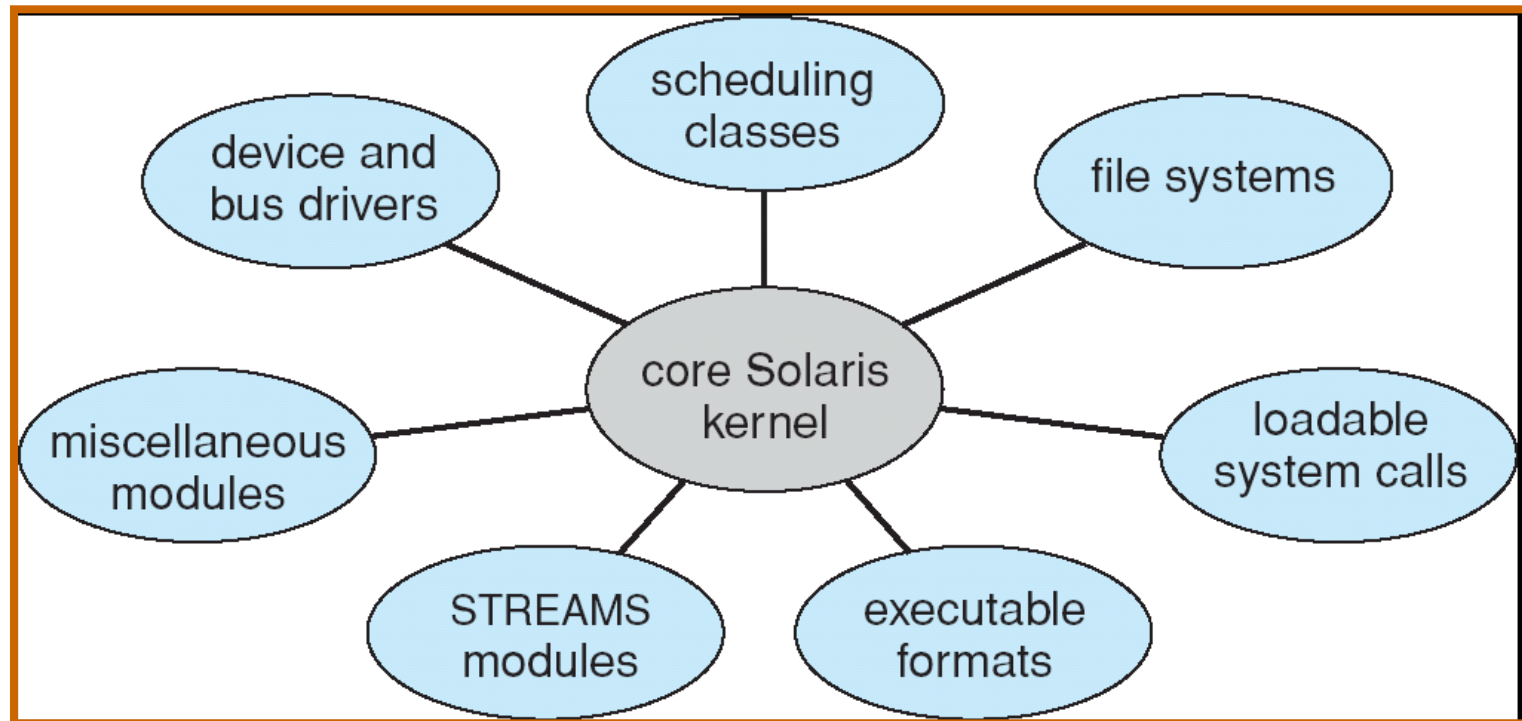


Figura 16: Módulos carregáveis do Solaris

# Estrutura de módulo: Linux

- Execute:
- **\$ /sbin/lsmmod**
  - exibe uma lista dos módulos carregados no momento.
- **\$ /sbin/modprobe**
  - Para carregar um módulo do kernel, use o comando seguido do nome do módulo do kernel.
- **\$ /sbin/modinfo**
  - para exibir informações sobre um módulo do kernel.

# Estrutura Híbrida: Mac OS X

- Aproximação em camada com uma camada com microkernel:
  - Mach: gerência de memória, comunicação interprocesso e chamada de procedimento remoto.
- BSD: sistema de arquivo, interface de comandos, redes e APIs POSIX;

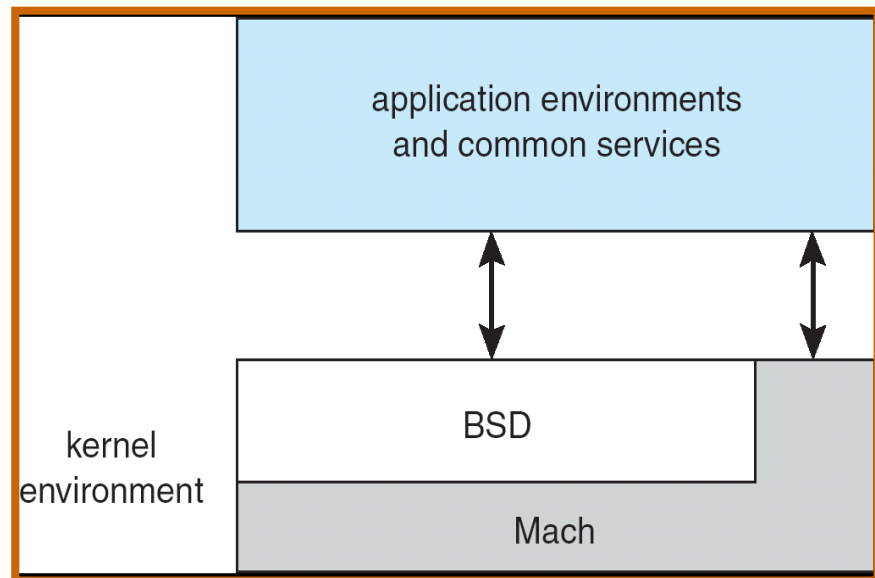


Figura 17: Estrutura do Mac OS X

# Sumário

- Monitoramento de um SO
- Exemplos de SO



# Monitoramento de um SO

- É importante monitorar o desempenho de um sistema operacional;
- Há diversas ferramentas juntos aos SOs:
  - Windows Task Manager
  - top
  - htop
  - ps

# Monitoramento de um SO

A coleção de ferramentas BPF é um toolkit que fornece característica tracing para sistema Linux.

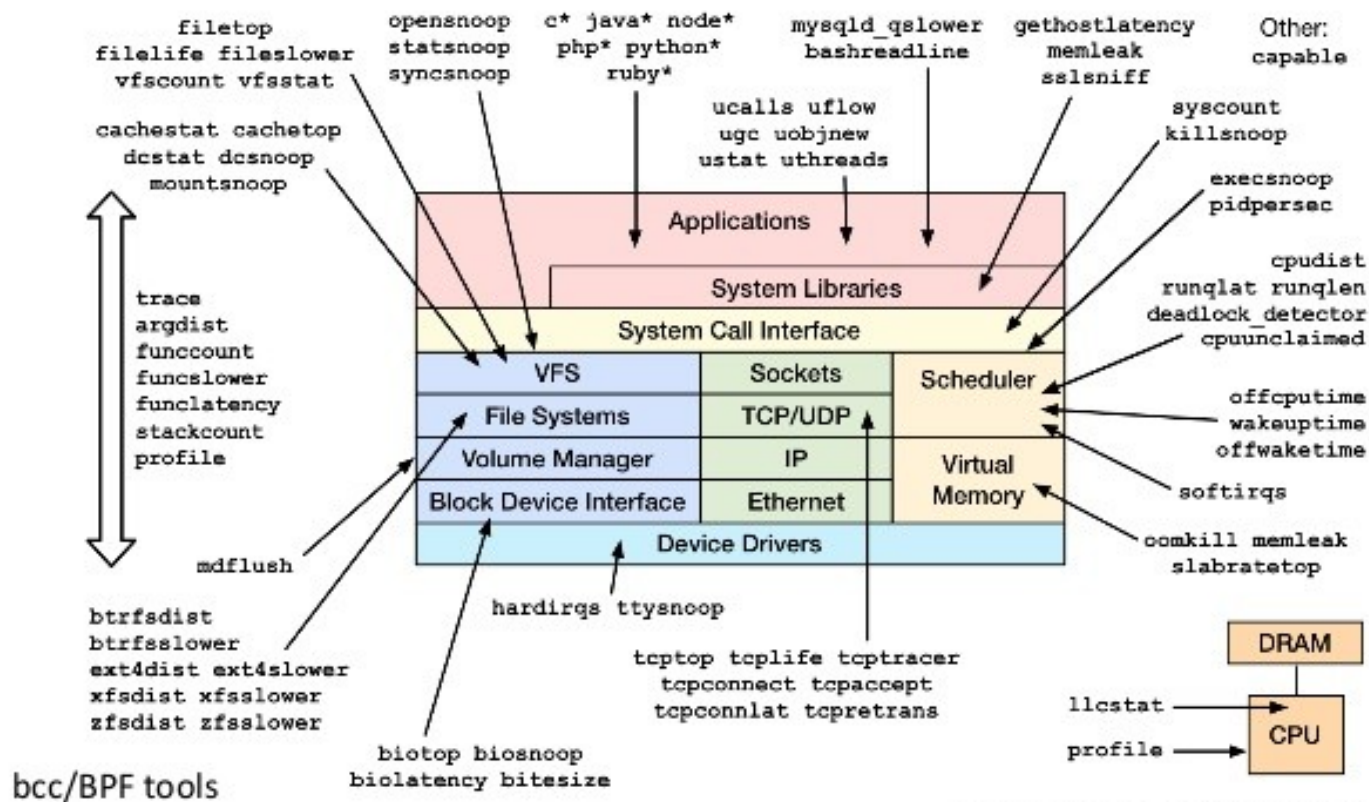


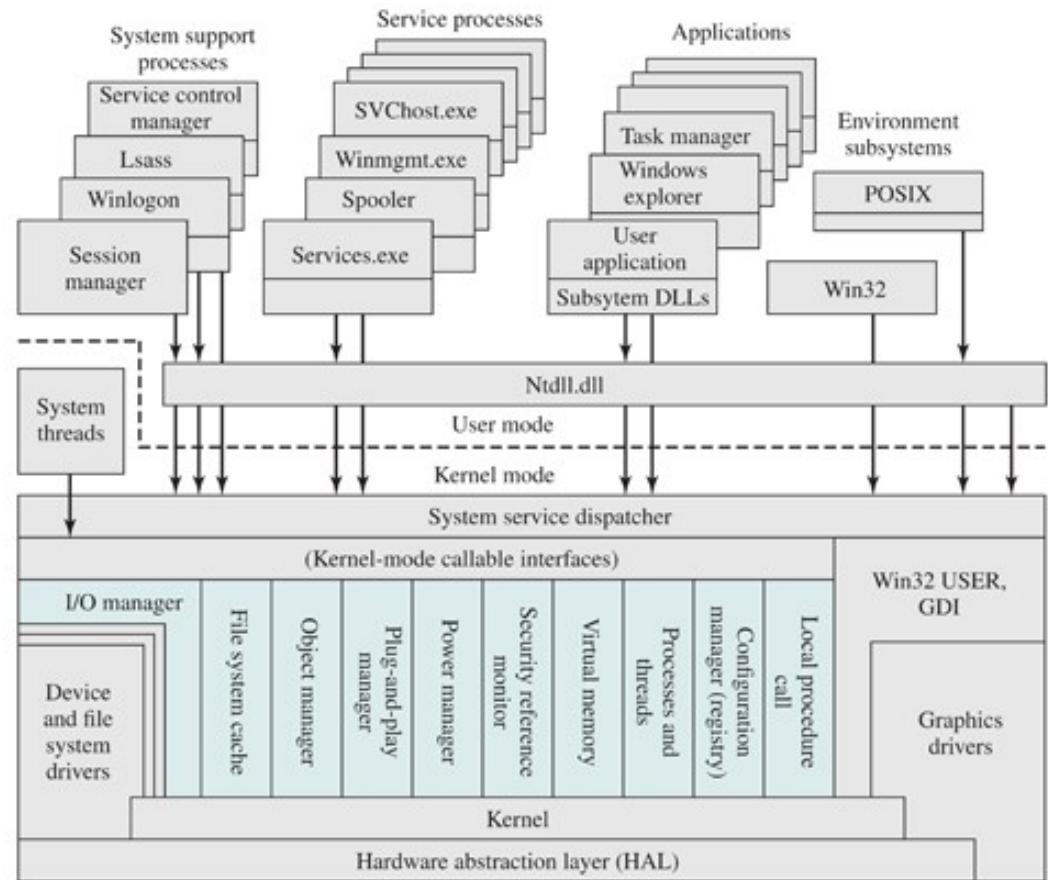
Figura 18: Coleção de ferramentas BPF (Berkeley Packet Filter)

# Sumário

- Exemplos de Sistemas Operacionais

# Windows

- Windows surgiu em 1985.
- Hoje o Windows 10 suporta desktops, laptops, smartphones, tablets e Xbox One.



Lsass = local security authentication server  
POSIX = portable operating system interface  
GDI = graphics device interface  
DLL = dynamic link library

Colored area indicates Executive

Figura 19: Arquitetura do Windows

# Linux

- Linux surgiu em 1991.
- Hoje, o Linux é um sistema semelhante ao Unix que pode ser executado em diversas plataformas.

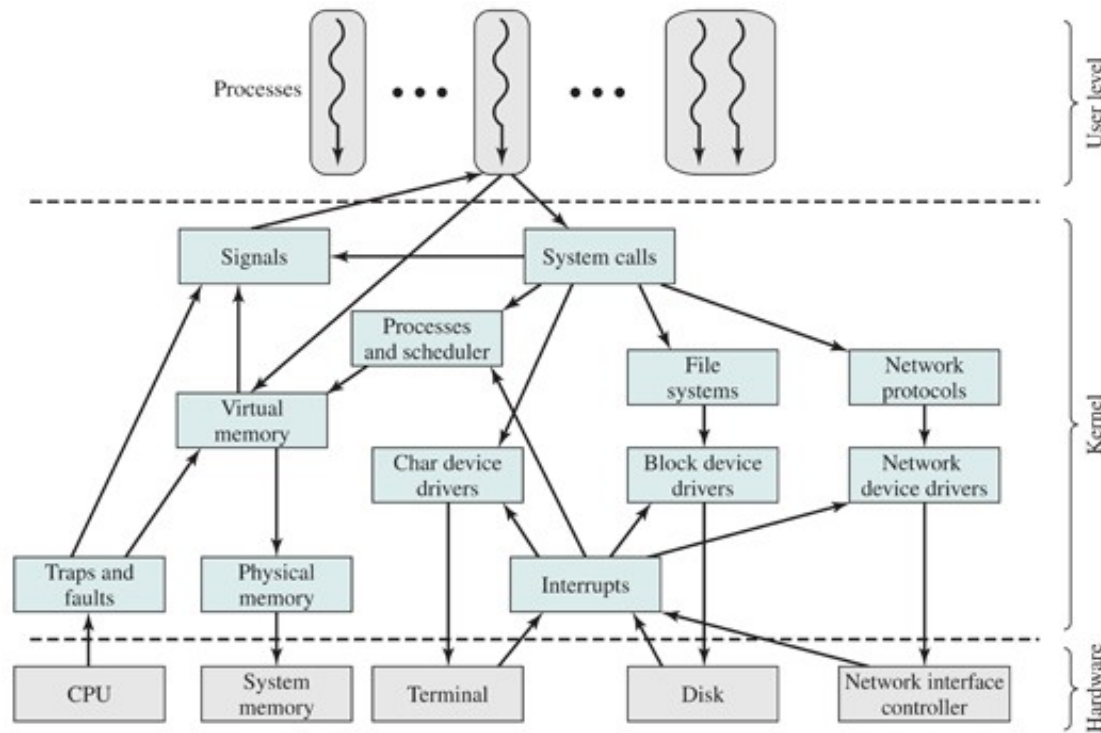


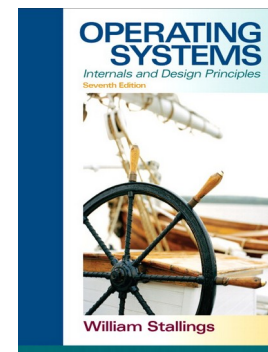
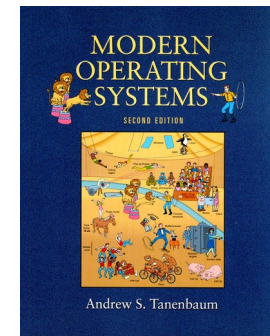
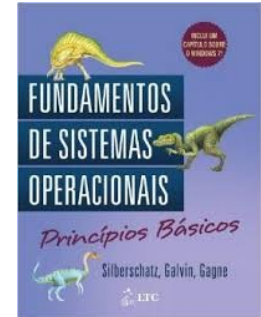
Figura 20: Arquitetura do Linux

# Agenda

- História
- Sistemas Operacionais (SO)
- Interpretador (usuário)
- Chamada de Sistema (System Call)
- Estrutura do SO
- Monitoramento de um SO
- Exemplos de SO

# Leituras Sugeridas

- SILBERSCHATZ, A. & GALVIN, P. B. & GAGNE, G. Fundamentos de Sistemas Operacionais. 8ª Edição, Ed. LTC – Capítulo 1 e 2.
- Andrew S. Tanenbaum. Sistemas Operacionais. Modernos. 2ª Ed. Editora Pearson, 2003. - Capítulo 1
- STALLINGS, W.; Operating Systems – Internals and Design Principles. 7.ed. Pearson Learning Solutions, 2012 – Capítulo 1 e 2



# Leituras Sugeridas

- <https://developer.ibm.com/tutorials/l-system-calls/>