Universidade Federal de Uberlândia
Faculdade de Computação
Sistemas Operacionais

Introdução aos Sistemas Operacionais

Prof. Dr. Marcelo Zanchetta do Nascimento

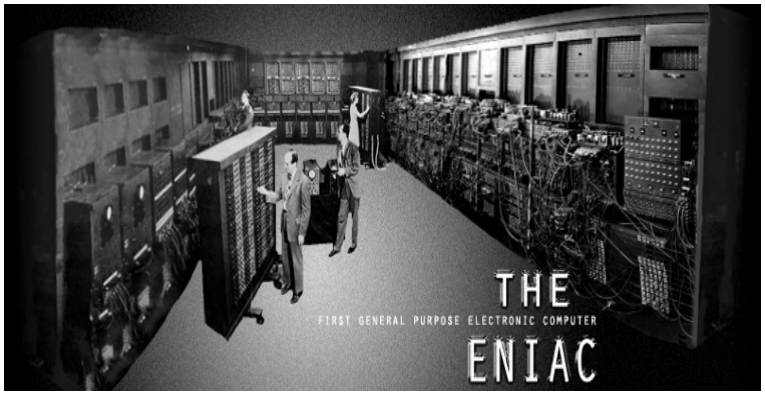
Sumário

- História
- Sistemas Operacionais (SO)
- Serviços do SO e 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.



ENIAC foi construido por *William Mauchley e seu estudante J. Presper Eckert na Universidade da Pennsilvania*

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;
- Máquina de grande porte (Mainframes) em salas especiais;
- Codificação de programas em papel (FORTRAN ou linguagem de máquina) e perfuração de cartões;
- Início do sistema em lote (batch): gravar as tarefas em fita magnética.

História

Segunda Geração (1955-1965): Transistores

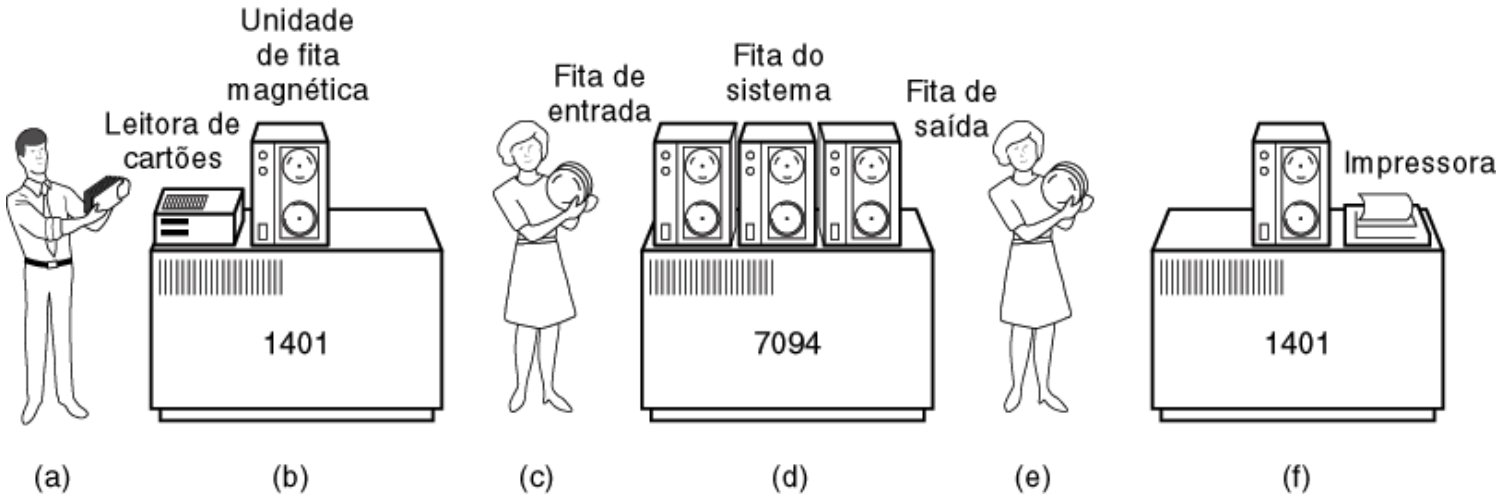


Figura 2: Fase para processamento (Fortran Monitor System)

Sistema em lote:

- 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

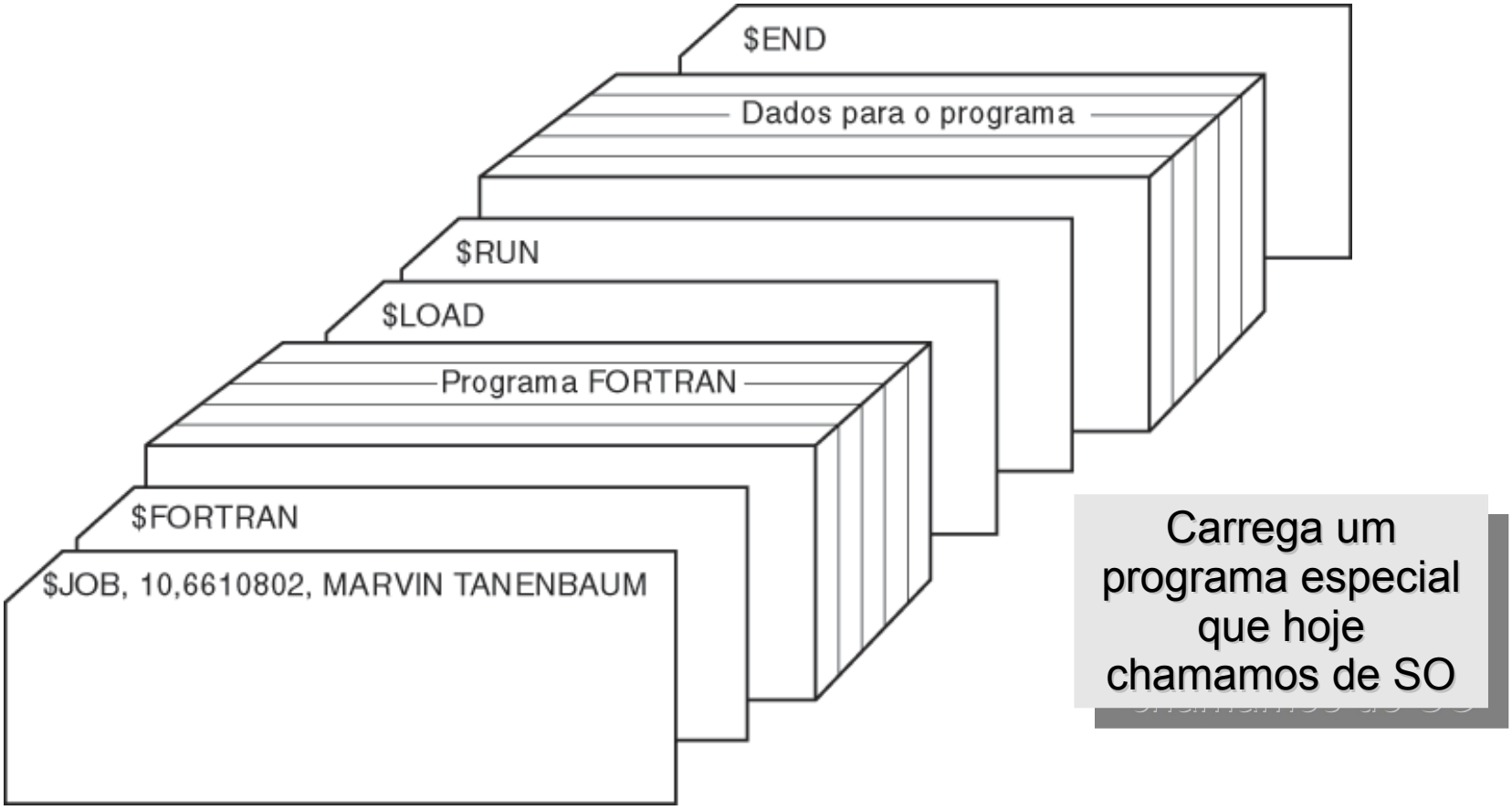


Figura 3: Estrutura de um job em um sistema operacional Fortran Monitor System.

História

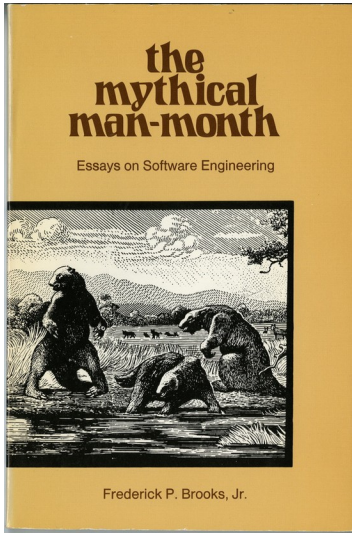
Terceira Geração (1965-1980): CI e Multiprogramação

- Fabricantes ofereciam computadores: orientados a palavras (IBM 7094) e orientados a caracteres (IBM1401);
- Então, 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 (incrementalmente mais potentes): 370, 4300, 3080 e 3090.

História

Terceira Geração (1965-1980): CI

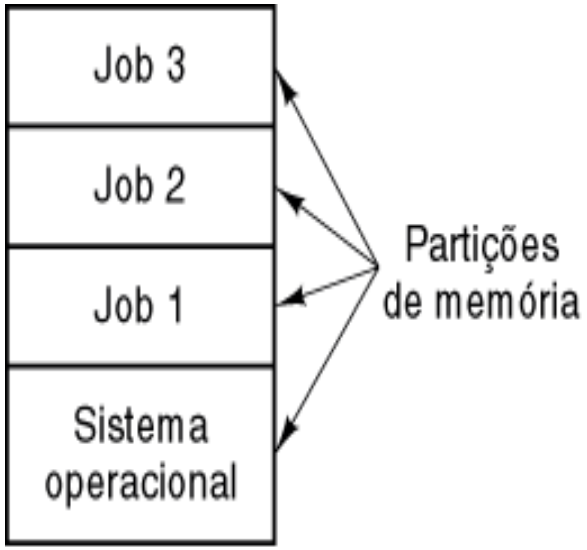
- OS/360 executado em qualquer modelo da IBM, torna-se o SO genérico;
- Enorme, complexo e desenvolvido em linguagem de montagem:
- *Fred Brooks, projetista desse projeto, escreveu um livro sobre experiências com o SO em que a capa mostra um rebanho de animais pré-históricos presos em um fosso (analogia a capa do SILBERSCHATZ et al. 2000).*
- Introduz a multiprogramação (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 4: Existem três jobs na memória RAM

História

Terceira Geração (1965-1980): CI

- Spooling (simultaneous peripheral operation online): transferir jobs de cartões para disco magnético sem fitas magnéticas;
- Timesharing (Conecta por um terminal): o CTSS - sistema compatível de tempo compartilhado desenvolvido no MIT;
- Computador “utilitário”: MULTICS (Serviço de computação e informação multiplexada):
  - Suportava centenas de usuários;
  - Incorporava memória virtual.
- Outro grande 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;
  - Codigo-fonte: Organizações desenvolveram suas próprias versões e as principais são: SYSTEM V (AT&T) e BSD (Distribuição de Berkeley da Universidade da California);
  - O IEEE desenvolveu o padrão para UNIX: POSIX (portable operating system interface).
  - A. Tanenbaum (1987) desenvolveu o MINIX com objetivo educacional: www.minix3.org;
  - Linus Torvalds (1991): Linux foi desenvolvido sobre o MINIX.

História

Terceira Geração (1965-1980): CI

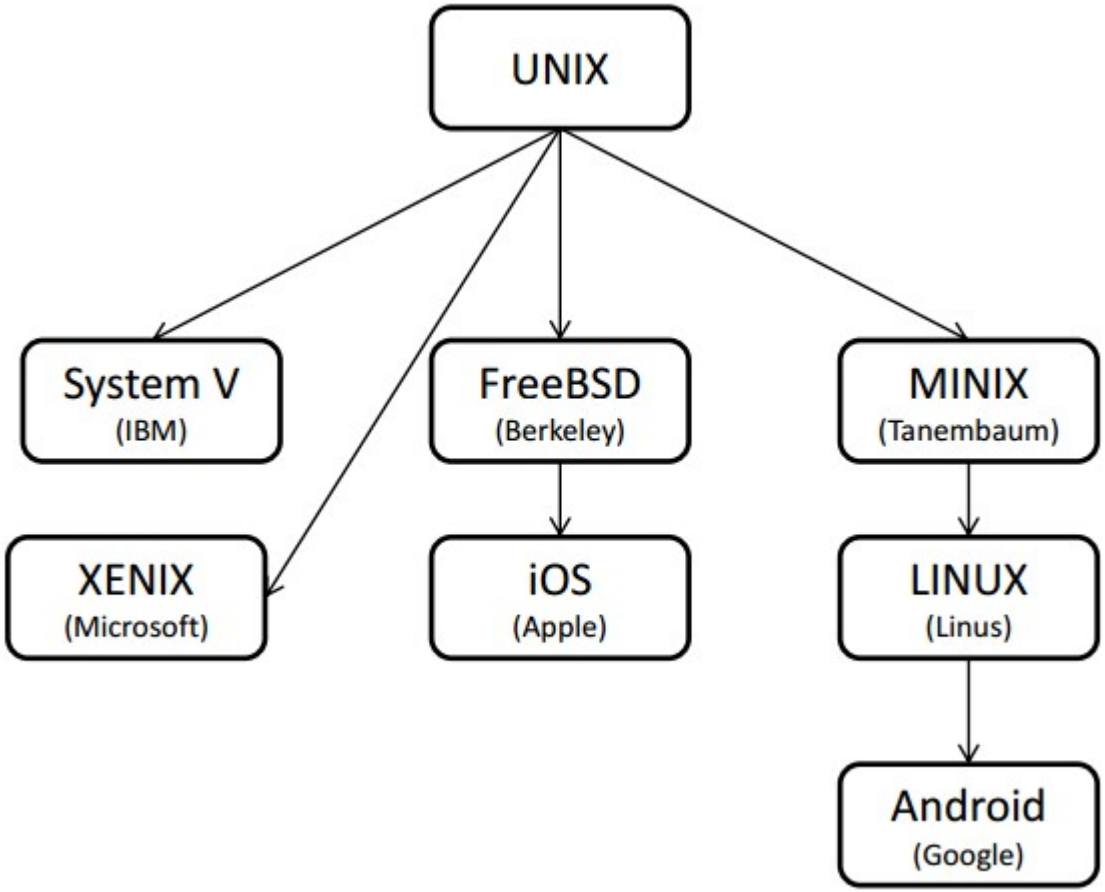


Figura 5: Versões de Sistemas Operacionais Baseados no UNIX.

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 aperfeiçoa-lo);

- IBM projetou IBM PC e para o software:

  – Contato com Bill Gates para licenciar o interpretador de programa Basic e fornecer um SO;

  – Comprou a “Seatlle Computer Products” e o sistema operacional DOS (monousuário) e contratou 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

- Steve Jobs construiu uma interface gráfica GUI - Apple Macintosh.
- Circuitos Integrados em Larga Escala;
- Sistemas multi-usuários (UNIX);
- Sistemas operacionais distribuídos (distribuir a realização de uma tarefa entre vários usuários);
- 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 1990 quando a Nokia (N9000) combinou dois *devices*: um telefone e um PDA (Personal Digital Assistant);

- Os sistemas Android e iOS são os mais empregados em grande parte dos *smartphones*.

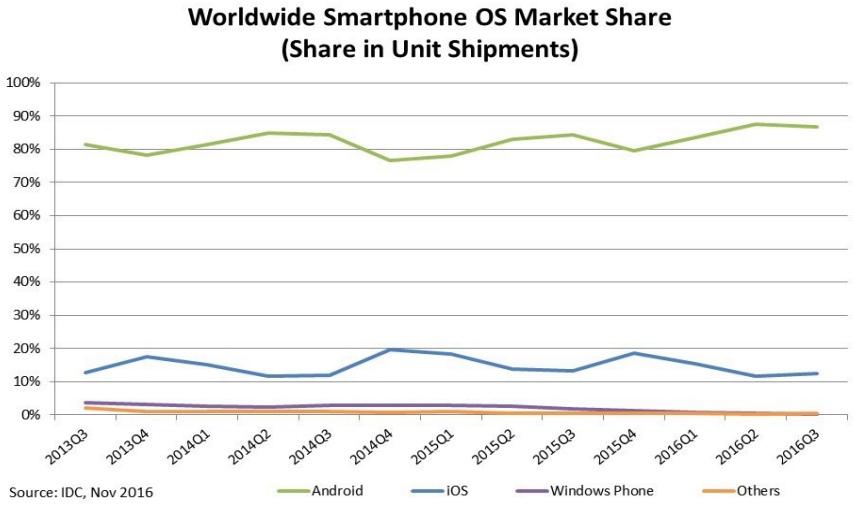


Figura 6: Estatística de uso de sistema em smartphone.

Função do Sistema Operacional (SO)

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

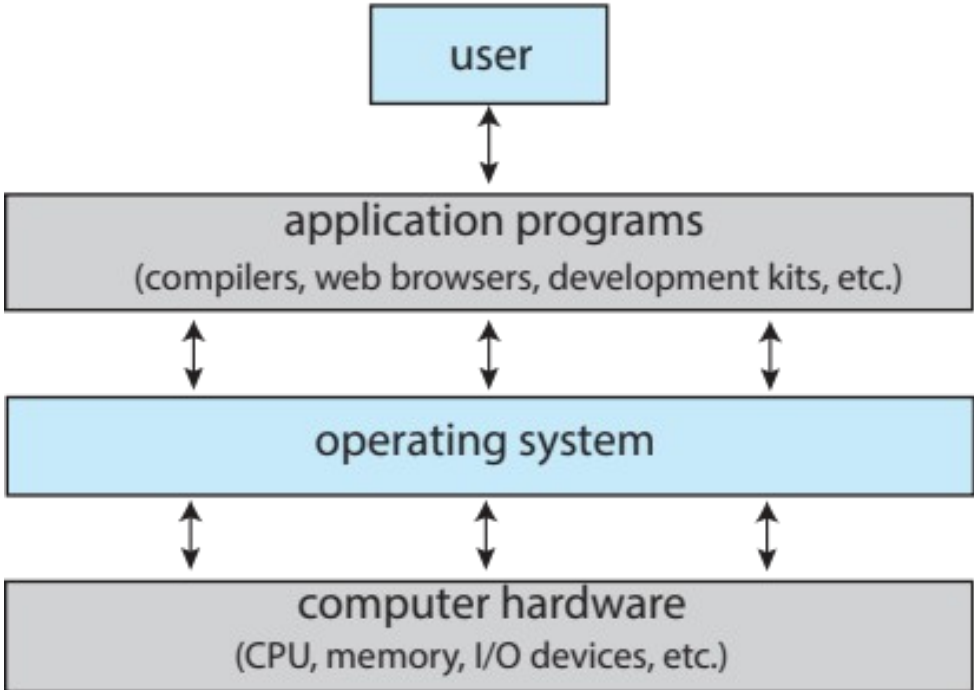
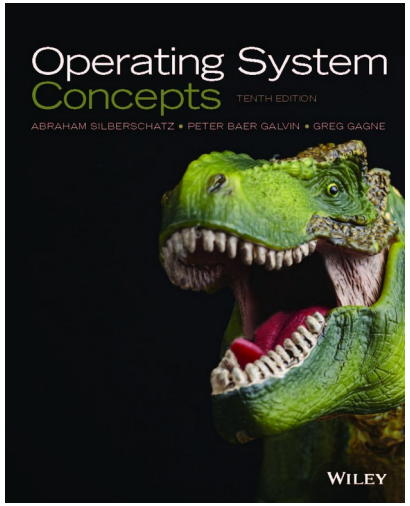


Figura 1: Visão Abstrata dos Componentes de um Sistema

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 - Operating System Concepts, Tenth Edition, John Wiley & Sons, Inc.



Serviços do SO

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

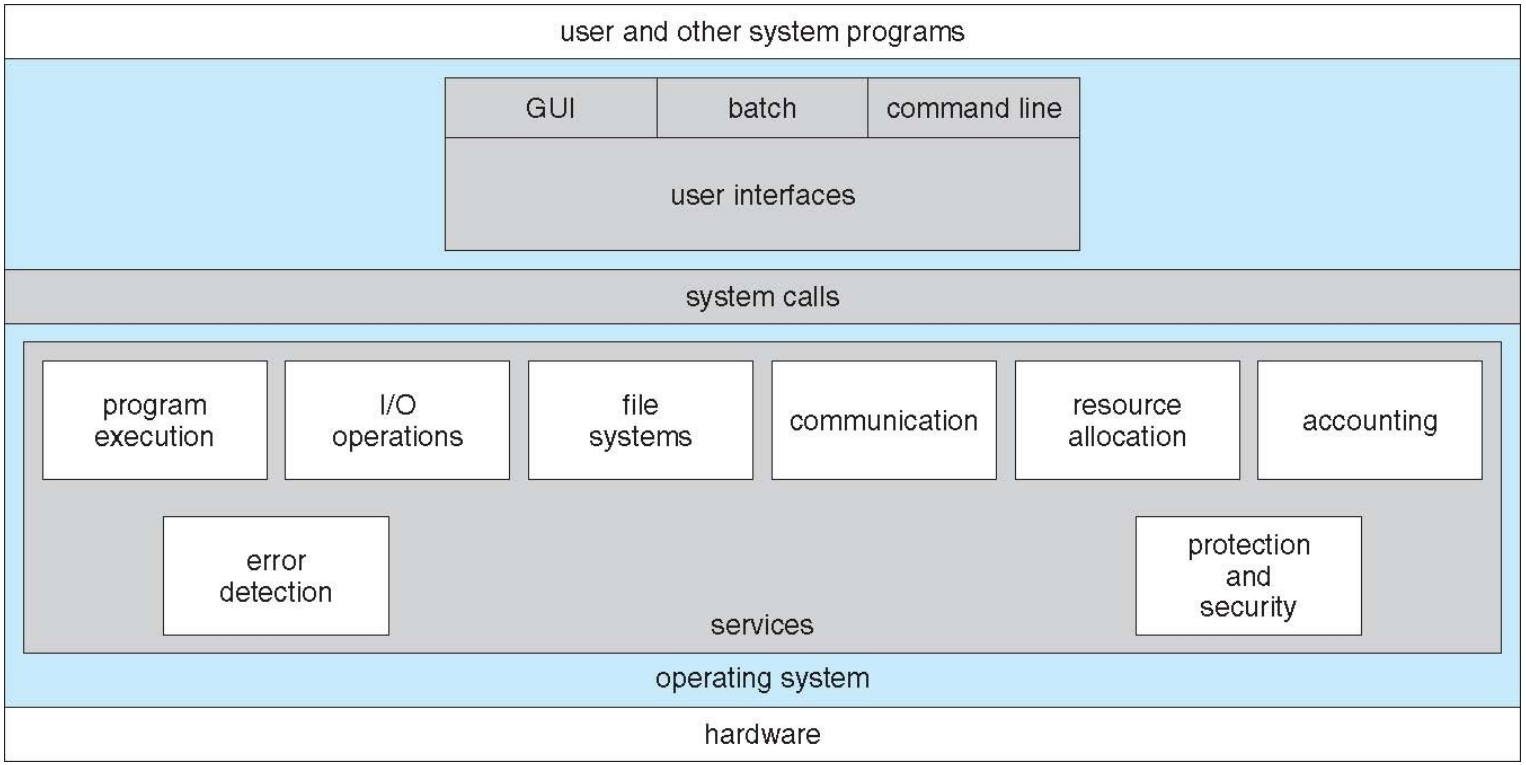


Figura 7: Camadas entre as etapas em um SO

Interpretador

- **Interface de usuário:**
  - **Linha de comando:**
    - utililiza comandos em modo texto;
  - **Interface gráfica** (*graphic user interface* – GUI):
    - sistema de janelas com apontamento para direcionar recursos de E/S.
  - Exemplo: Linux, Windows, etc.

Interpretador

- Interpretador de comando: recebe e executa o próximo comando especificado pelo usuário;
- O 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 interpretador contém o código para executar o comando.
    - Exemplo: Tarefa → Excluir um arquivo
    - Ir para uma seção do código que define os parâmetros e fazer a chamada de sistema (**system call**);

Interpretador

- **Implementação:**
  - os comandos são executados por meio de programas do sistema;
- O **interpretador** não entende o comando. Apenas identifica o arquivo (programa) e carrega para memória RAM para a 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.
- **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).

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), a qual permite 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 a Java Virtual Machine (JVM).

System Call

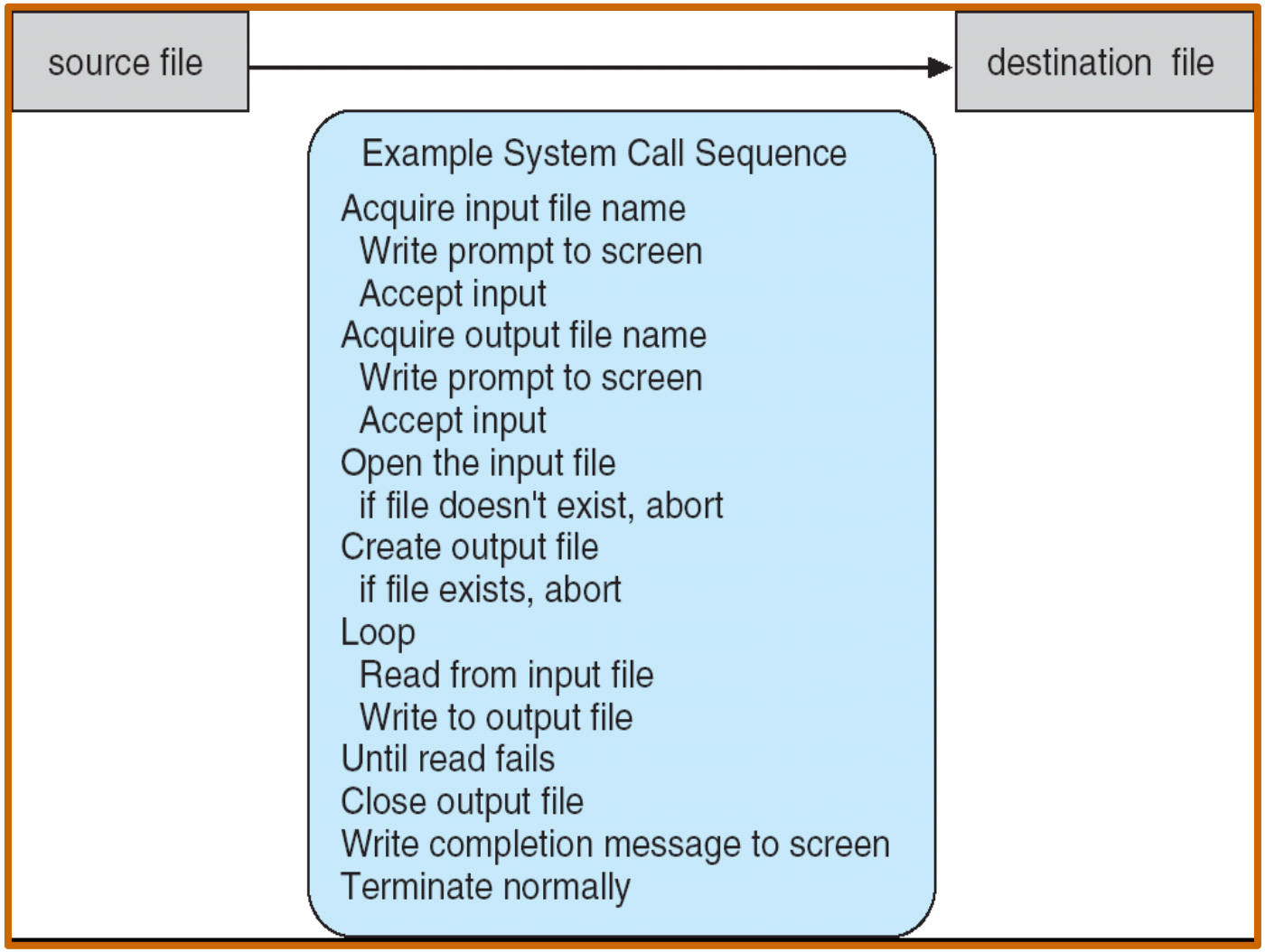


Figura 8: Exemplo de uma chamada de sistema para cópia de um arquivo

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 pelo uso da API:
  - Controlada pela biblioteca de suporte em tempo de execução.

System Call

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

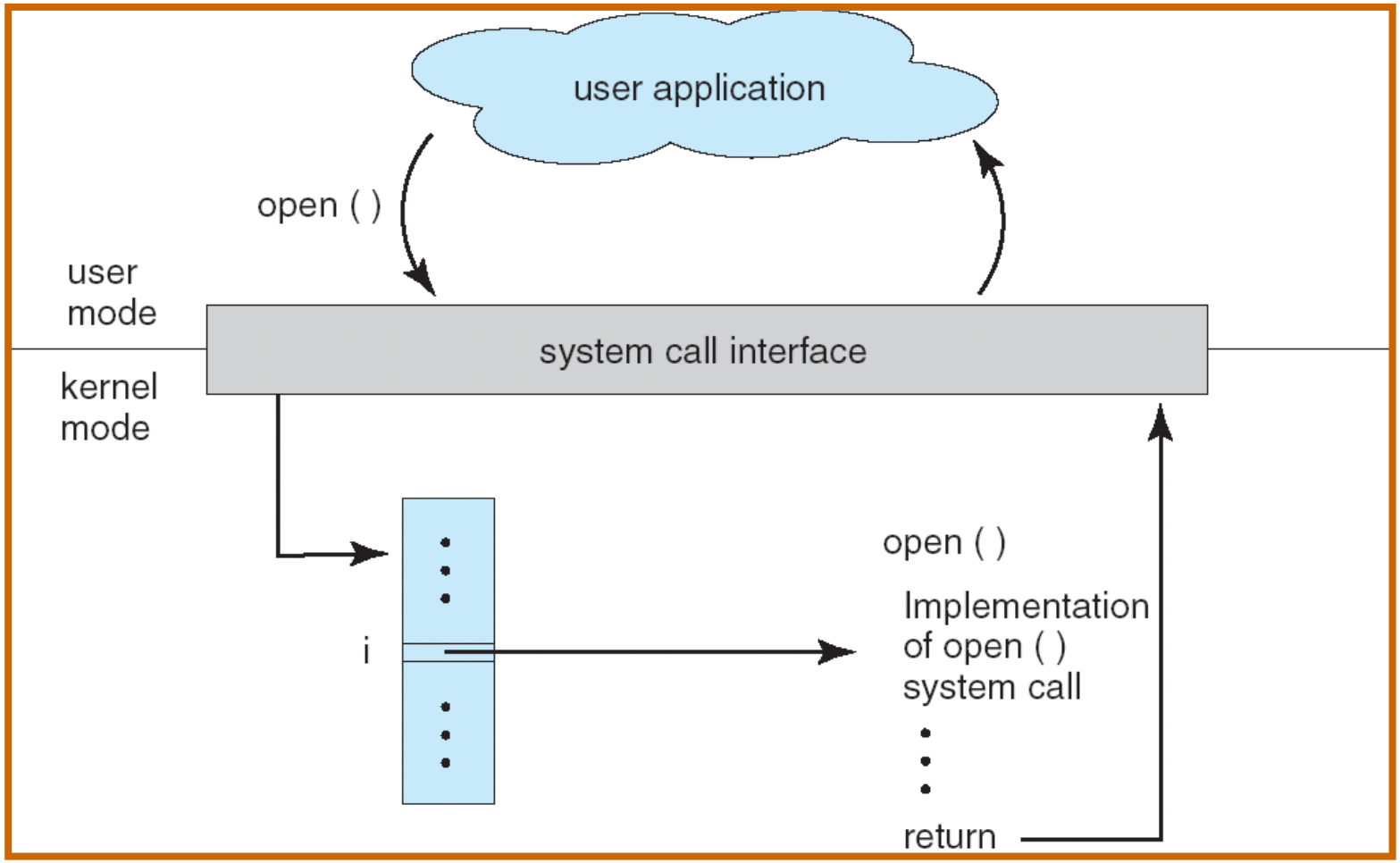


Figura 9: Aplicação do usuário que invoca a chamada de sistema

System Call

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

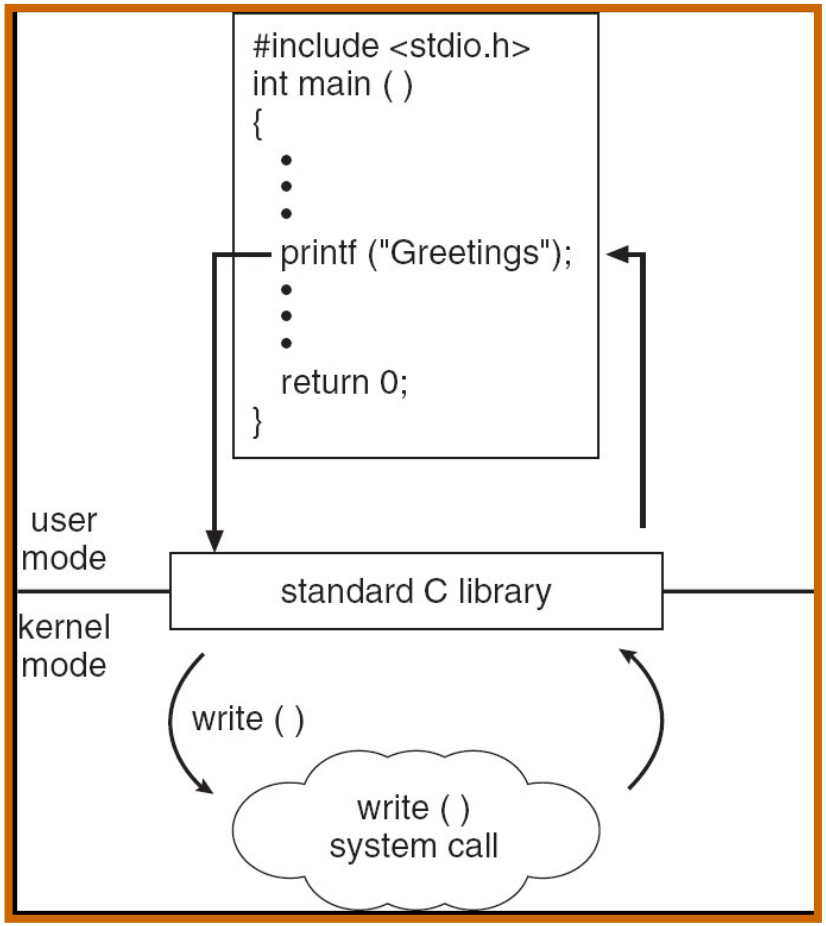


Figura 10: A chamada de sistema da biblioteca usada pela função printf.

System Call

Passagem de parâmetros: Outras informações devem ser enviadas:

- Não apenas a identificação, a qual pode variar de acordo com cada SO;

- Há 3 formas de passagem de parâmetros para o SO:
  - ***Simples:*** por meio de registradores;
  - Pode ter mais parâmetros que os registradores:
    - Bloco ou **tabela na memória** e o **endereço** do bloco são passados como parâmetros em um registrador;
    - Exemplo: Aproximação adotada no Linux e Solaris.

System Call

Simples: por meio de registradores durante uso da system call

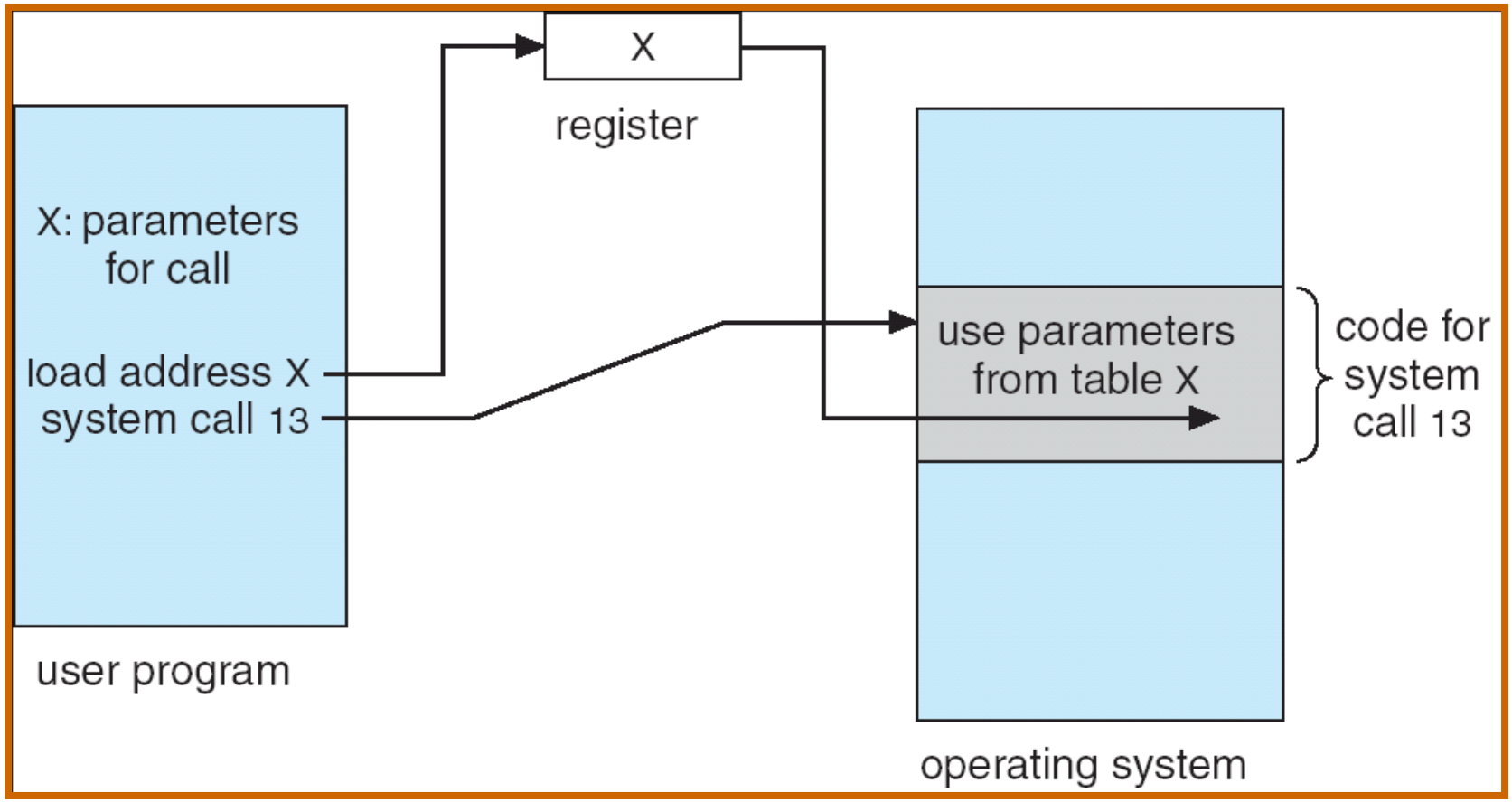


Figura 11: Passagem de parâmetros como uma tabela (bloco).

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 da linguagem Java, que invoca um código em C ou C++ 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 camando **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() 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() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Figura 12: Exemplo de chamadas de sistema em Windows e Linux

Estrutura do SO

- Um SO precisa ser construído com cuidado para funcionar de modo apropriado e permitir modificações com facilidade.
- **Como são divididas as estruturas de um SO?**

Exemplo:

- Monolítico (uma camada);
- Em camadas;
- MicroKernel;
- Modular;
- Híbrida.

Estrutura do SO

Monolítico

- O MS-DOS escrito para prover o máximo de funcionalidade em menor espaço;
- Não é dividido em módulos;
- As interfaces e níveis de funcionalidade não são bem separadas.

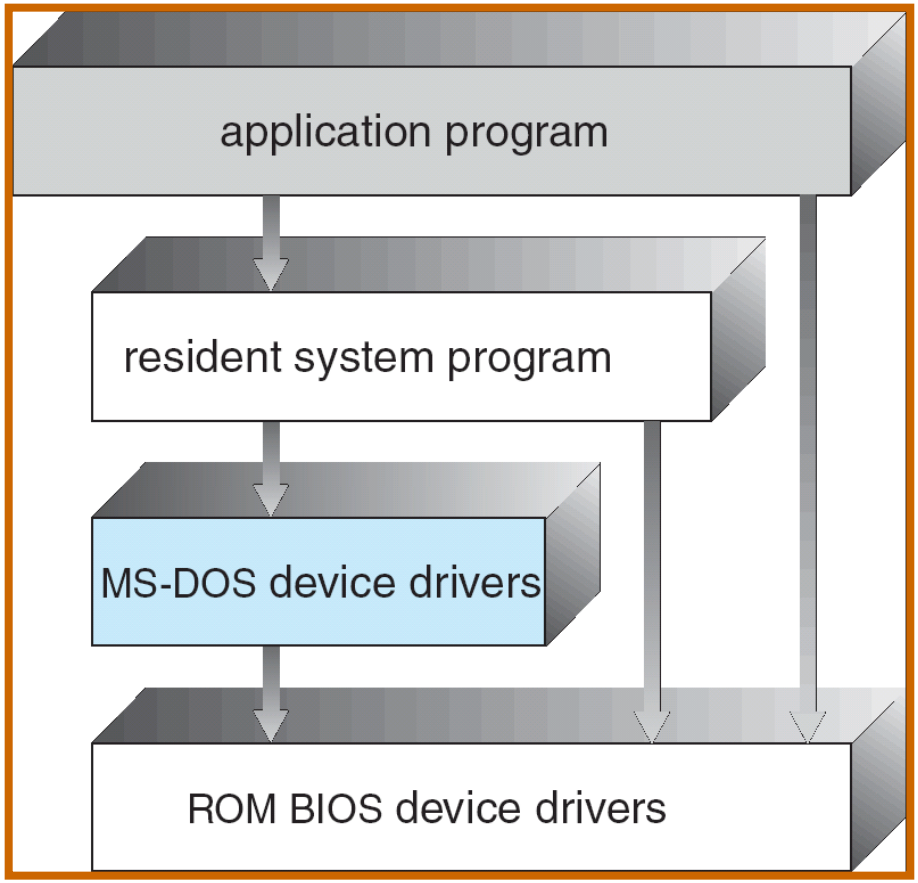


Figura 13: Estrutura de camadas de MS-DOS

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).

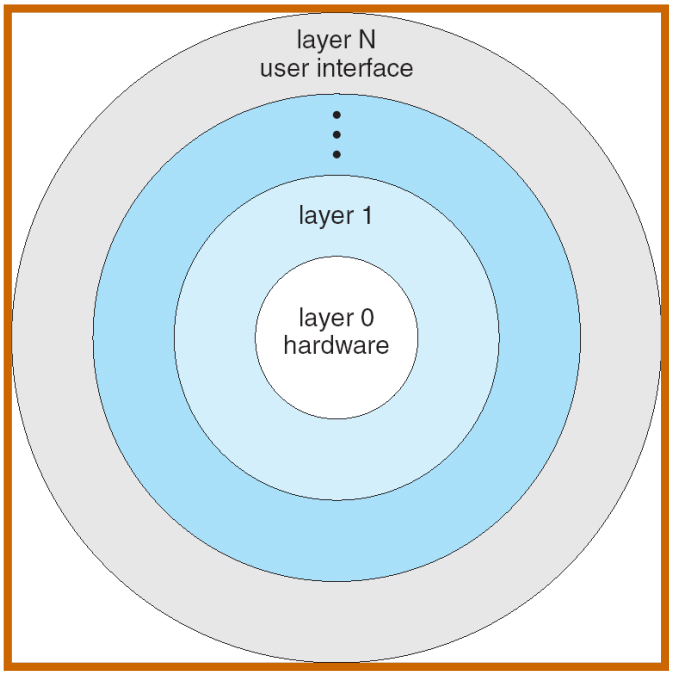


Figura 15: SO em Camadas

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”;
- Há pouco consenso sobre quais serviços devem permanecer no *kernel*;
  - Geralmente, estão alocados os gerenciadores de memória e processo no *kernel*;
- Fornecer a comunicação por troca de mensagens.

Estrutura do SO

Microkernels

- Windows NT (Win32).

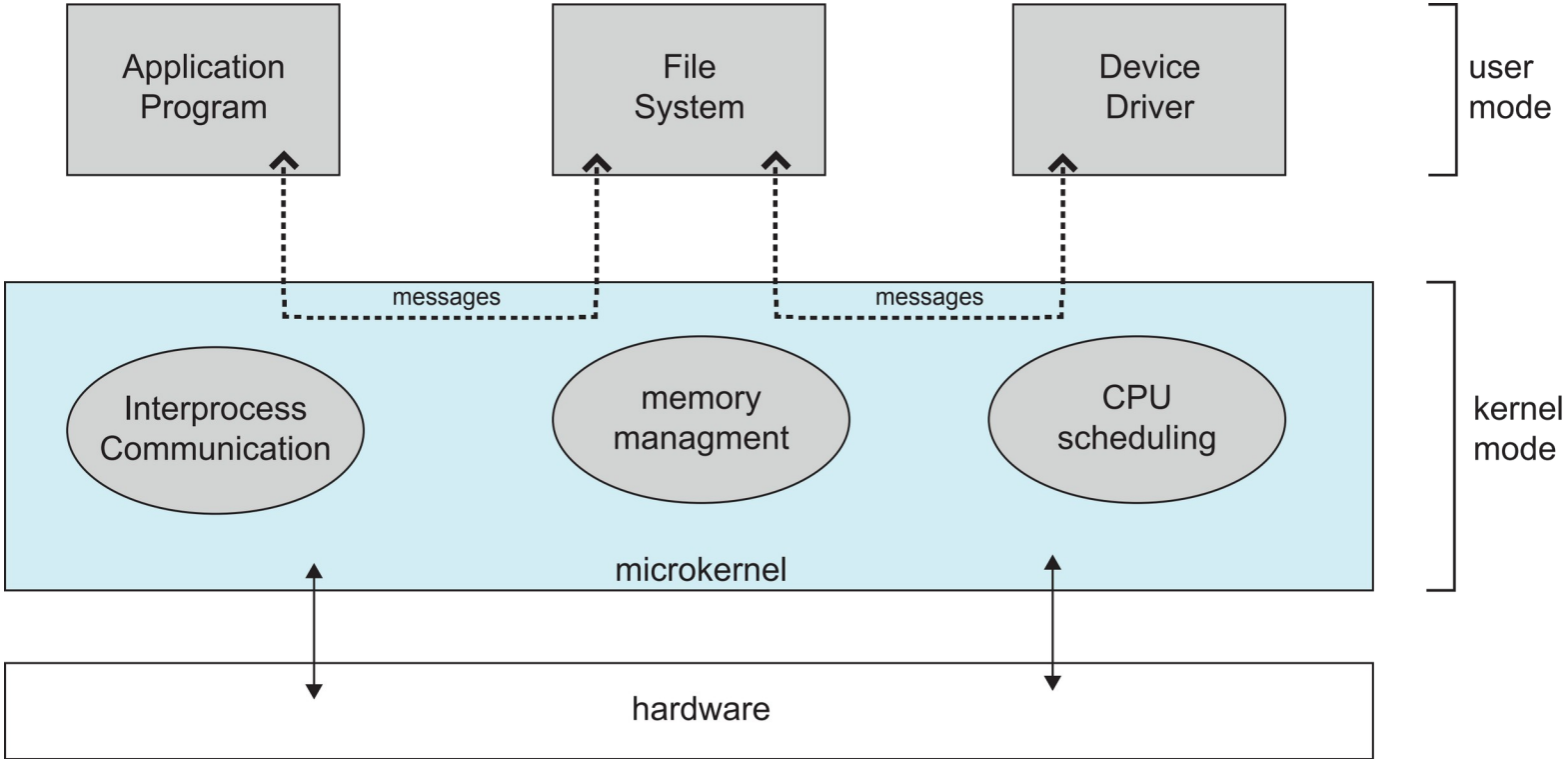


Figura 16: Estrutura em microkernel e a troca de mensagens

Estrutura do SO

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

Estrutura de módulo: Solaris

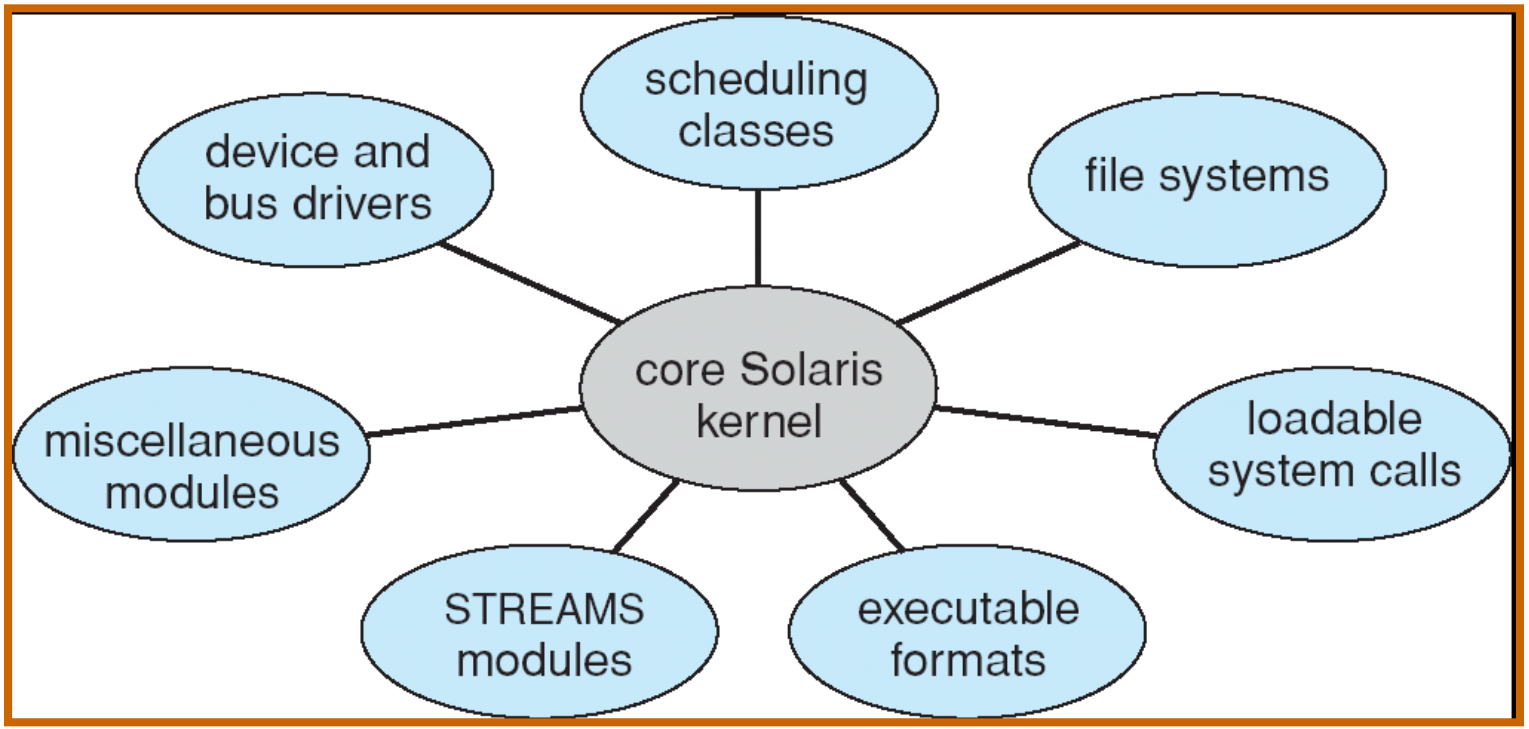


Figura 17: Módulos carregáveis do Solaris

Estrutura de módulo: Linux

- Execute:
- **$ /sbin/lsmod**
  - 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 memoria, comunicação interprocesso e chamada de procedimento remoto.
- BSD: sistema de arquivo, interface de comandos, redes e APIs POSIX;

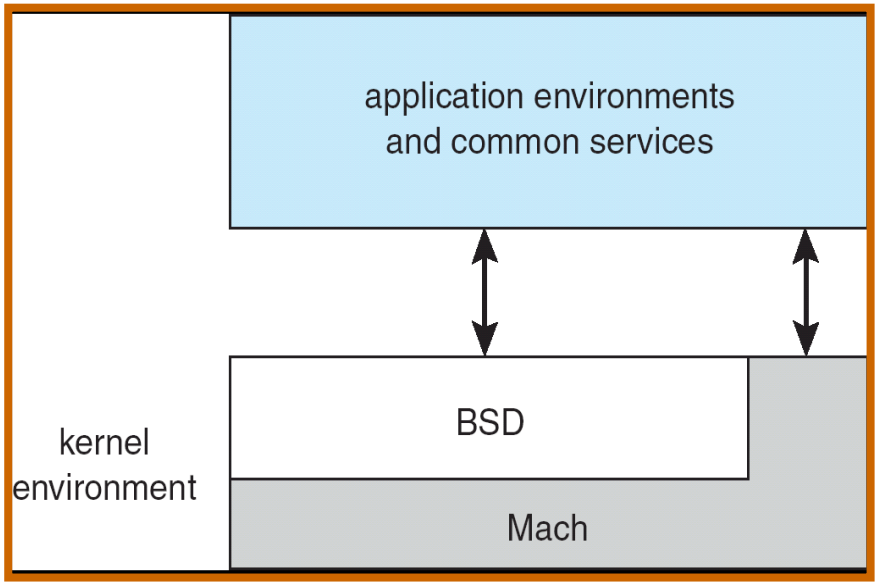


Figura 18: Estrutura do Mac OS X

Monitoramento de um SO

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

Monitoramento de um SO

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

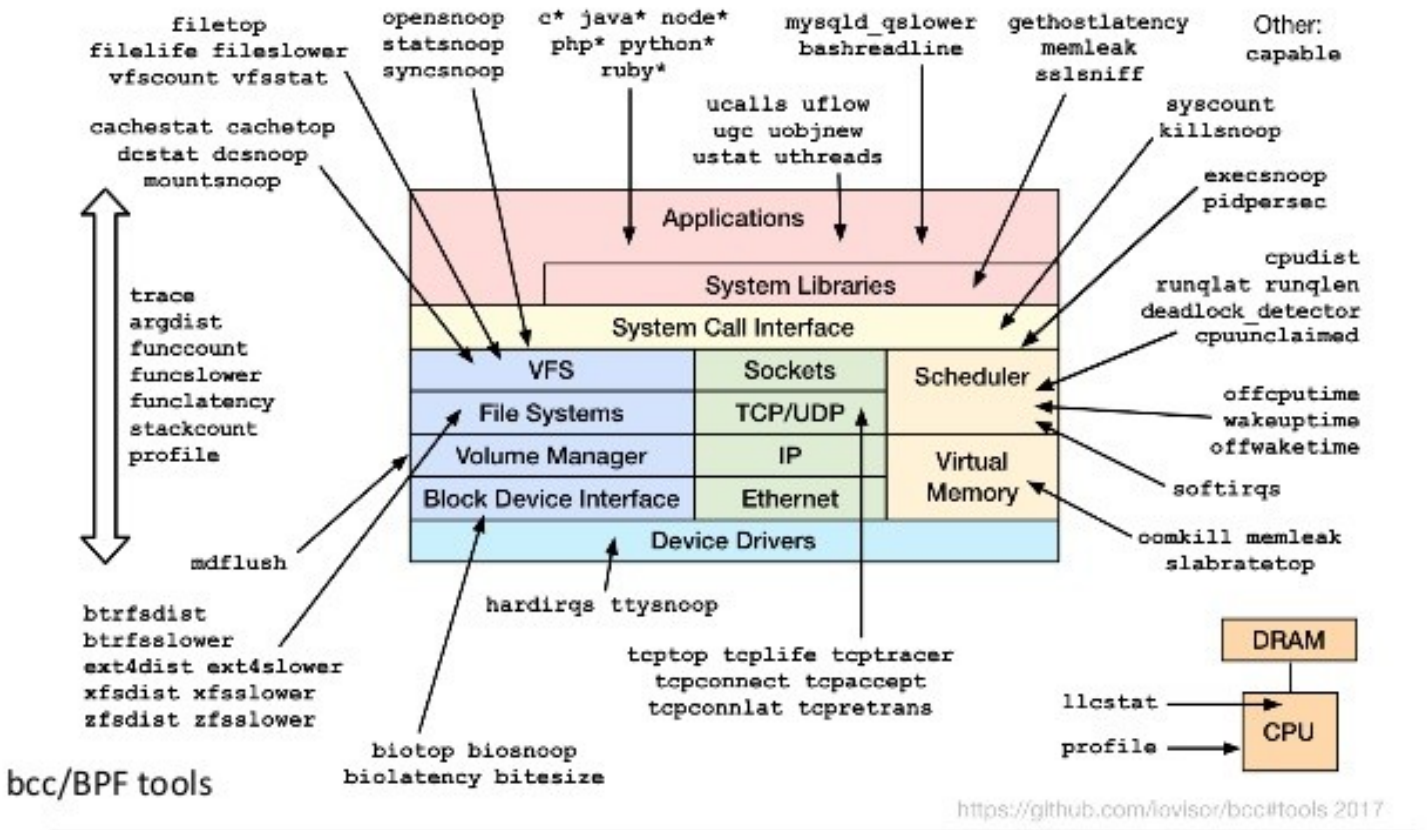


Figura 19: Coleção de compiladores BPF (Berkeley Packet Filter)

Arqutitetura do Windows

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

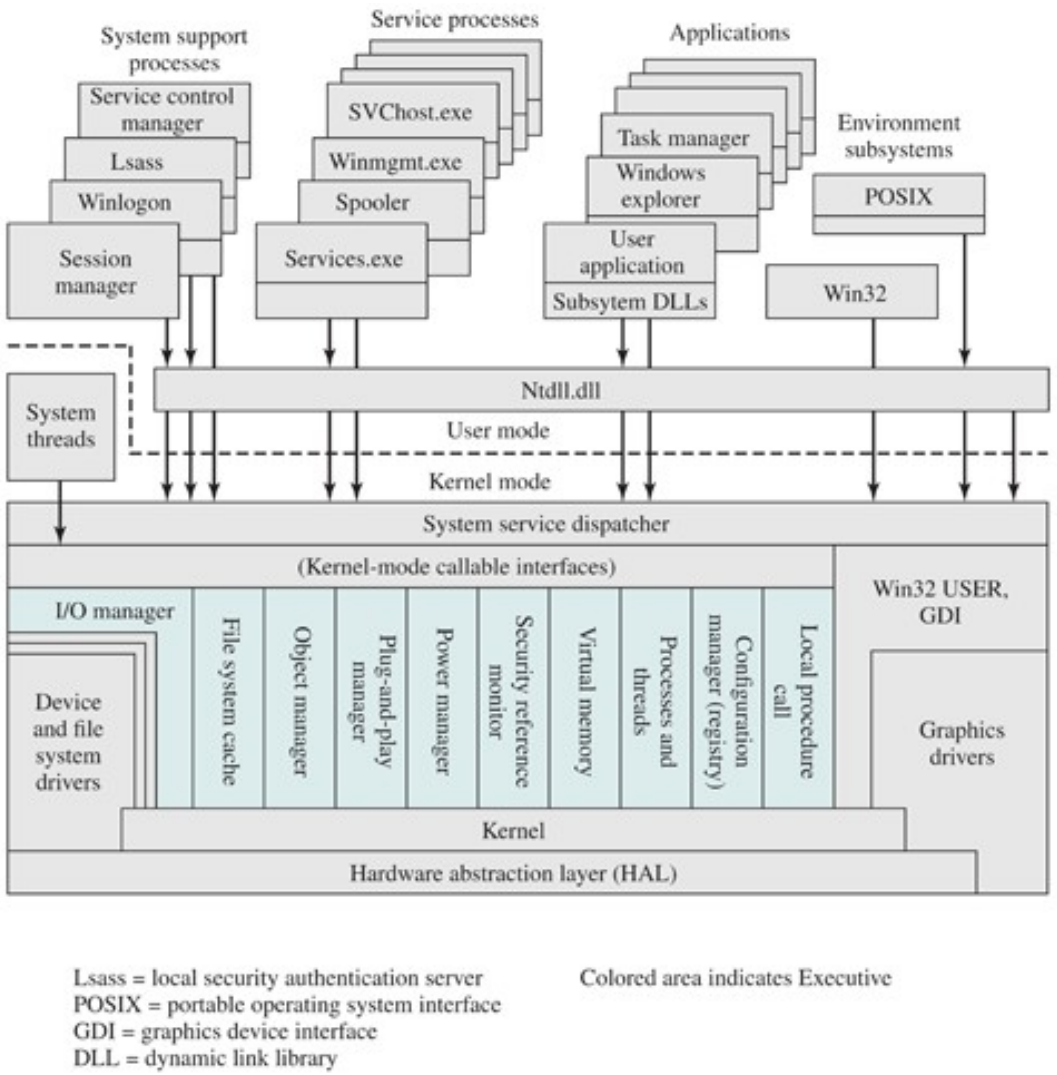


Figura 19: Arquitetura do Windows

Arquitetura do Linux

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

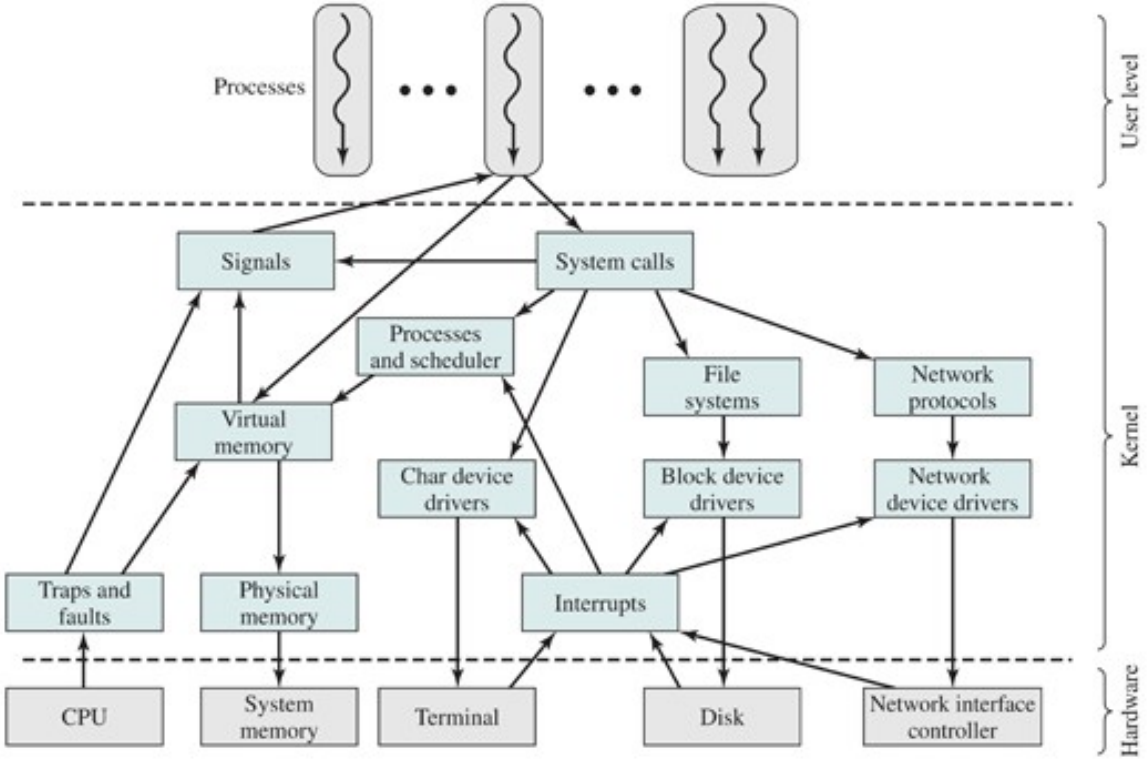
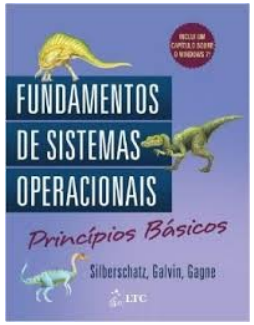


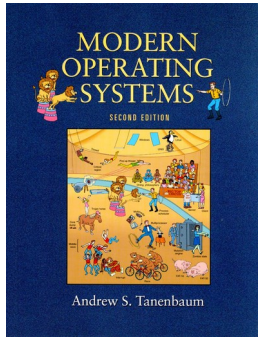
Figura 20: Arquitetura do Linux

Leituras Sugeridas

- SILBERSCHATZ, A. & GALVIN, P. B. & GAGNE, G. Fundamentos de Sistemas Operacionais. 8ª Edição, Ed. LTC – Capítulo 1 e 2.



- Andrew S. Tanembaum. 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