

Sistemas Operacionais

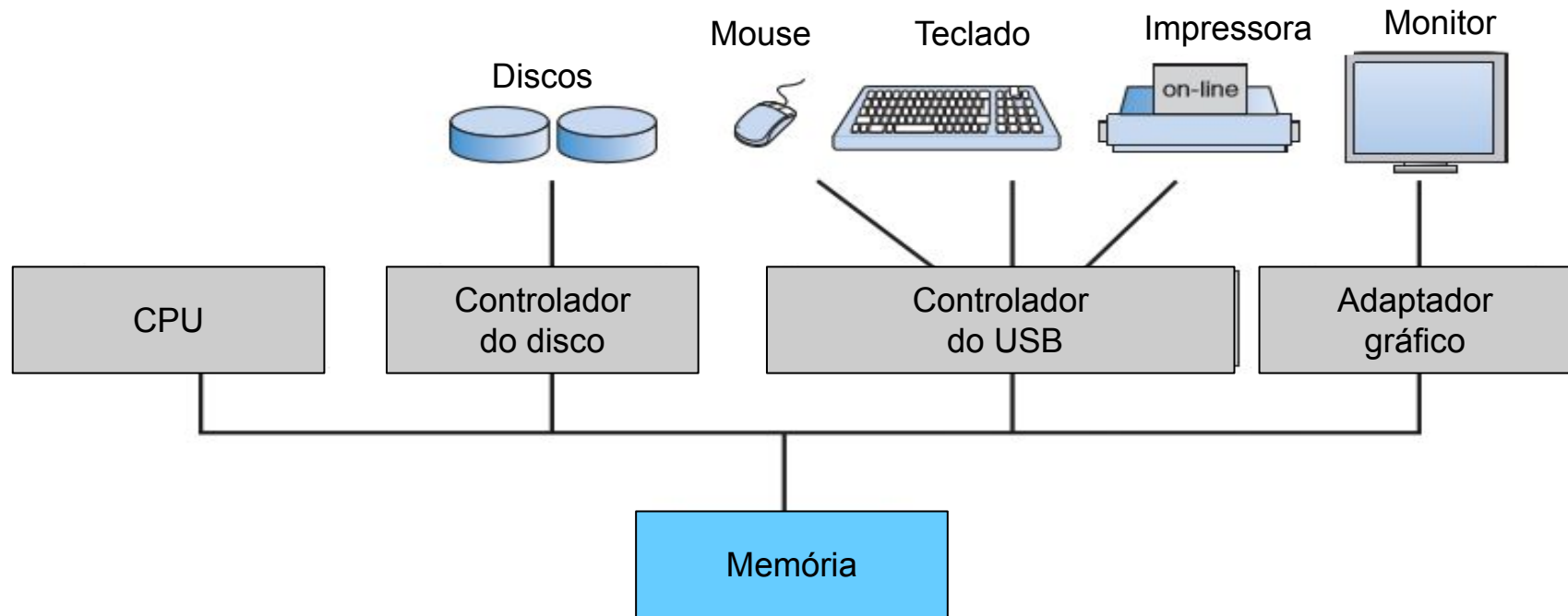
# Contextualização e Funcionamento de Sistemas Operacionais

**Lesandro Ponciano**

# Objetivos da Aula

- Contextualizar os sistemas operacionais (SOs)
- Analisar os componentes de SOs
- Apresentar conceitos básicos associados ao
  - Gerenciamento de processos
  - Gerenciamento da memória
  - Gerenciamento do armazenamento

# Diversos Hardwares Interligados



# Funcionamento Interligado

- Dispositivos entrada e saída (I/O)
  - Executam ao mesmo tempo que o processador (**concorrentemente**)
  - Controlador responsável pela sua operação
- Memória e *Buffers*
  - Cada controlador de dispositivo tem um *buffer* local
  - CPU move dados entre memória e *buffers*
  - I/O ocorre a partir do dispositivo para o *buffer* local
  - Controlador de dispositivo informa a CPU que terminou a sua operação

# Desafio da Programação

- Detalhes do funcionamento de todos os dispositivos e dos seus estados de utilização
- E se, para desenvolver uma aplicação, o programador precisasse entender tudo isso?
  - Imensa complexidade; programar se tornaria uma tarefa quase impossível
  - Grande dificuldade de gerenciar todos os dispositivos de modo a utilizá-los de maneira otimizada

Foi preciso pensar em uma forma de facilitar a tarefa para os programadores de aplicações

# Evolução dos Sistemas Operacionais

## 1ª Geração (1945 - 1955)

- Válvulas e Painéis com Plugs
- Operação e controle programados pelo programador

## 2ª Geração (1955 - 1965)

- Transistores
- Sistemas Batch

## 3ª Geração (1965 - 1980)

- Circuitos Integrados
- Multiprogramação e tempo compartilhado (*timesharing*)

## 4ª Geração (1980 - presente)

- Computadores pessoais com interfaces gráficas
- Sistemas operacionais de rede e os sistemas operacionais distribuídos

# Sistema Operacional

- Programa de computador que opera o hardware e fornece a base para todos os outros programas
- Alocador de recursos
  - Decide entre pedidos conflitantes de modo a tornar o uso dos recursos eficiente e justo
- Programa de controle
  - Controla a execução de programas para prevenir erros e uso impróprio do computador

# Tipos de Sistemas Operacionais

- Sistemas **Monoprogramáveis**
  - 1 usuário x 1 tarefa
- Sistemas **Multiprogramáveis**
  - Lote: tarefas sem interação com o usuário
  - 1 usuário x  $N$  tarefas -> Monousuário
  - $N$  usuários x  $N$  tarefas -> Multiusuário
  - Tempo Compartilhado: ênfase em reduzir o tempo de reação
  - Tempo Real: ênfase em limites rígidos para execução



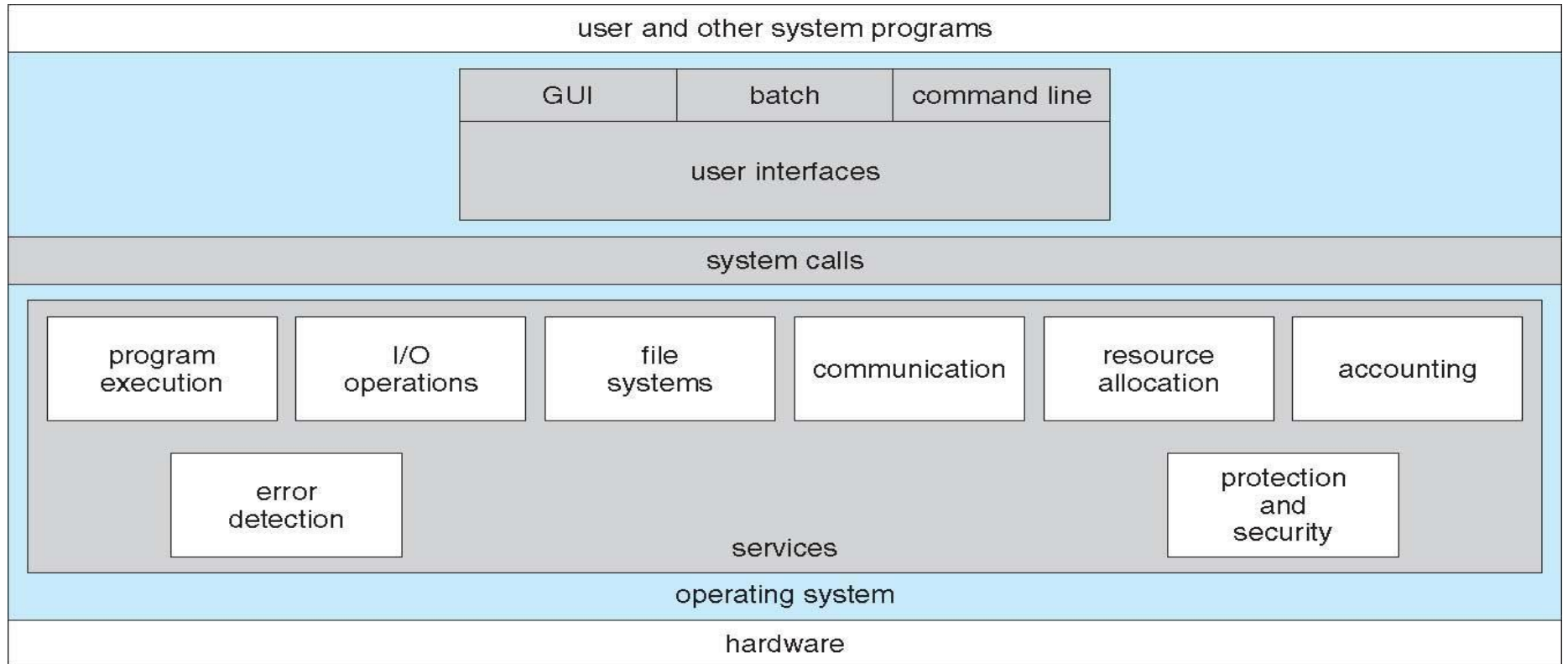
# Tipos de Sistemas Operacionais

- Sistemas Paralelos e Distribuídos
  - Características principais
    - Escalabilidade (acrescentar novos elementos de computação)
    - Disponibilidade (substituir elementos)
    - Balanceamento de carga (distribuir a computação entre os elementos)
  - Desenvolvidos para
    - Multiprocessador
    - Multicomputador

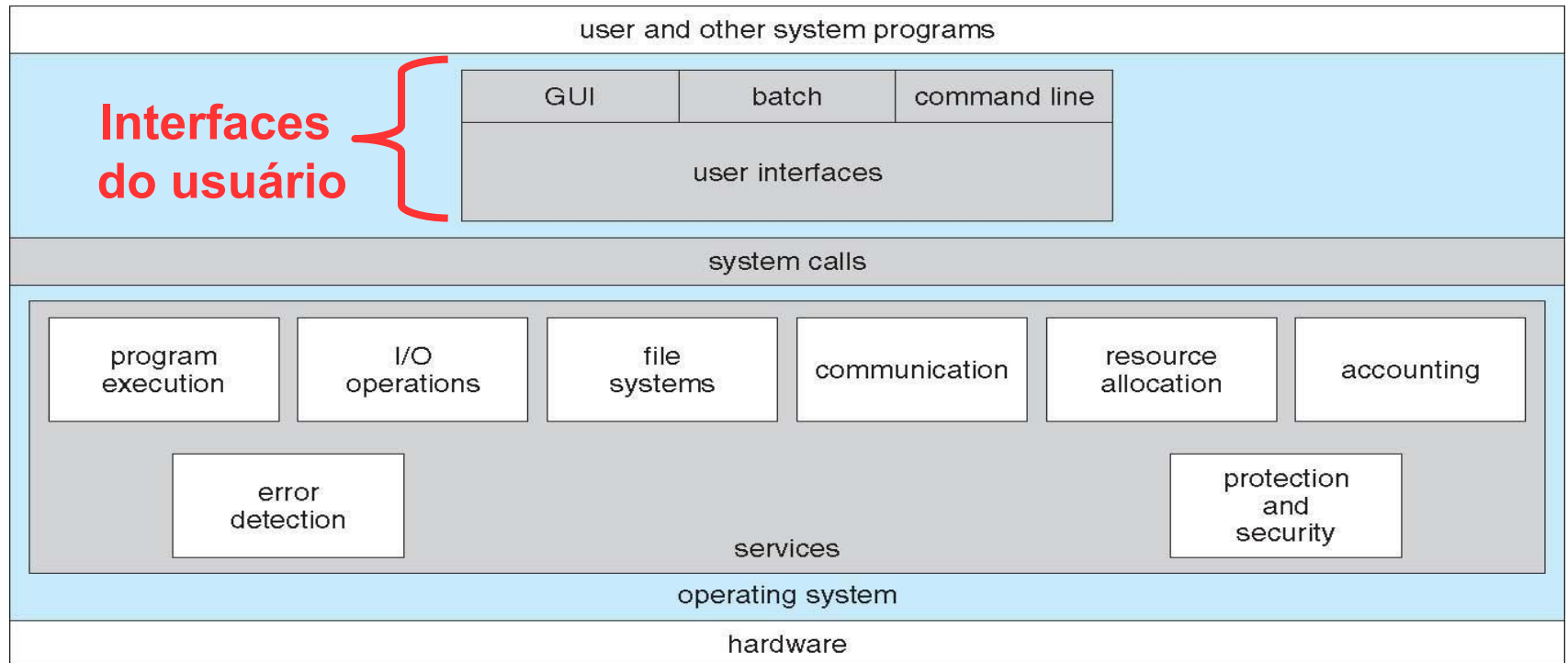
# Inicialização do Sistema Operacional

- Quando a energia do computador é ativada, um programa de *boot* (*bootstrap*) é iniciado
  - Fica armazenado na ROM, EPROM, ou seja *firmware*
  - Carrega o sistema operacional e **inicia** a execução dele
- O sistema operacional é o único programa que permanece executando durante todo o tempo no computador, os demais são
  - Programas de sistema
  - Aplicações dos usuários

# Ambiente do Sistema Operacional



# Interface do Usuário com o SO



# Estilos de Interface do Usuário (UI)

- *Shell*, interpretador (ou *prompt*) de comandos

- ex.: cd, ls, move, ssh

Comando: rm

```
rm arquivo.txt
```

Parâmetro: arquivo a ser removido

- GUI, *graphical user interface*

- ex.: menus, ícones e caixas de diálogo

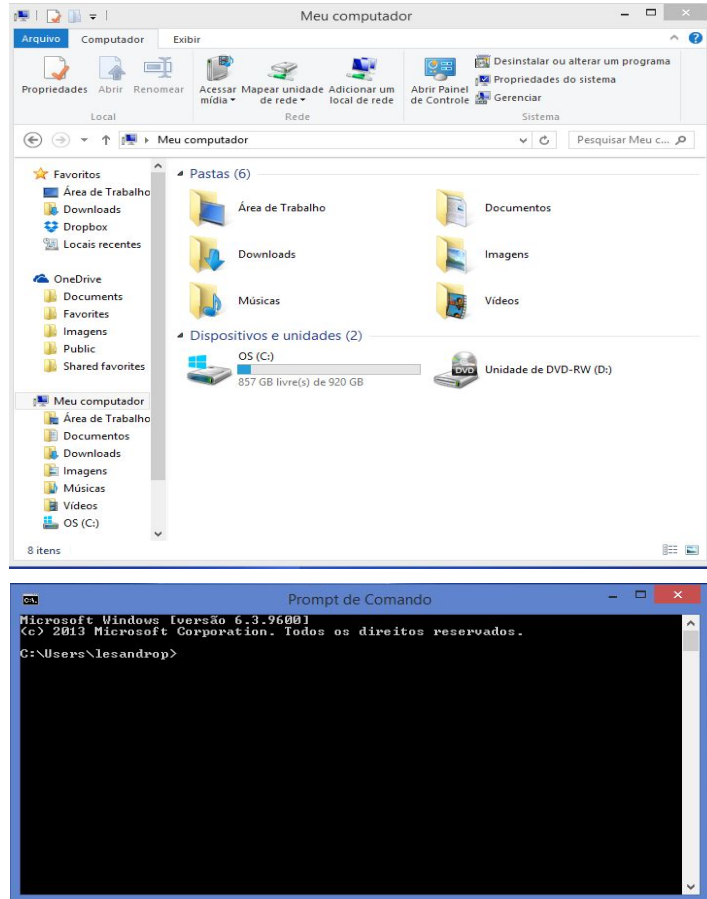
- VUI, *voice user interface*

- comandos em voz, muito comum em *smartphones*

- Interface de programação

- bibliotecas

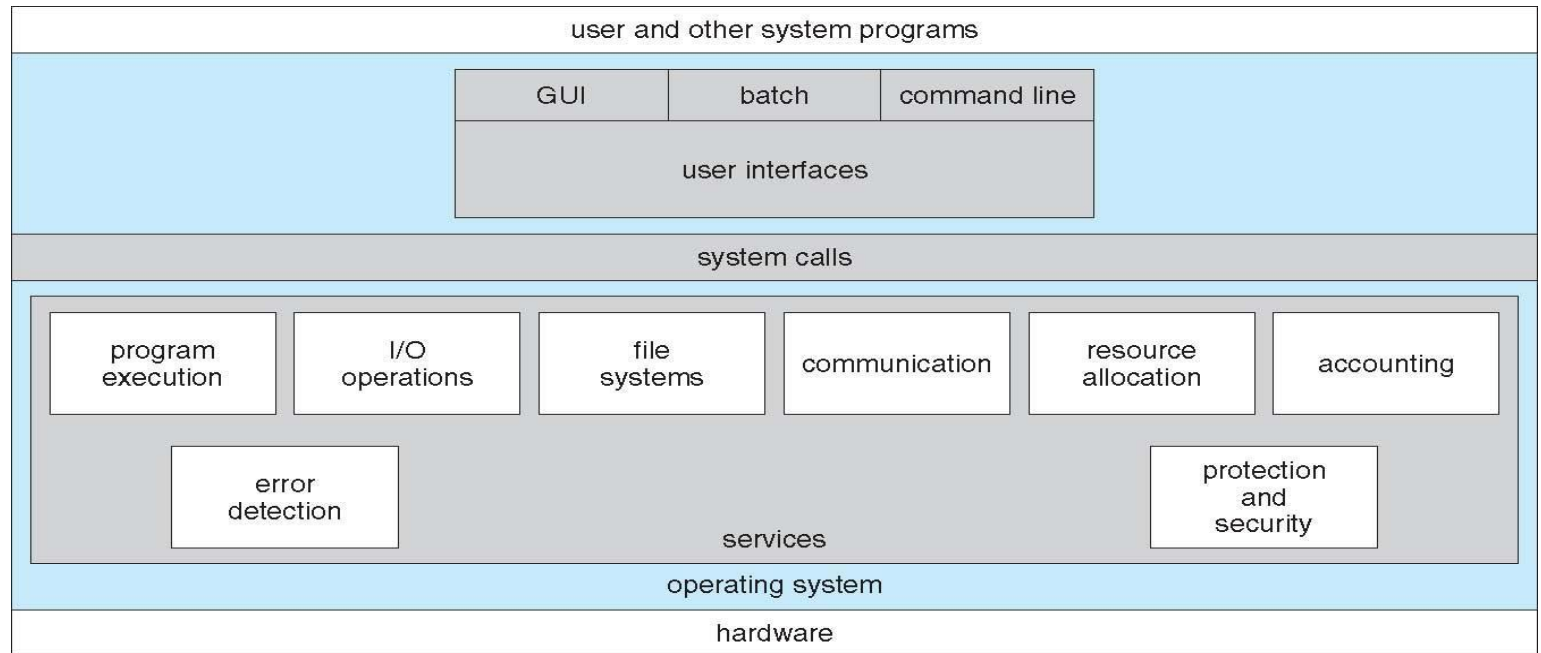
# Exemplos de UIs



- Não fazem parte do núcleo (kernel) do sistema operacional
- Programas que operam em modo usuário, mas eles sabem “realizar chamadas” ao núcleo do sistema operacional (TANENBAUM, 2009)

# Chamadas de Sistema (*system calls*)

Chamadas de Sistema {



# Modos de Operação

## ■ Modo Usuário

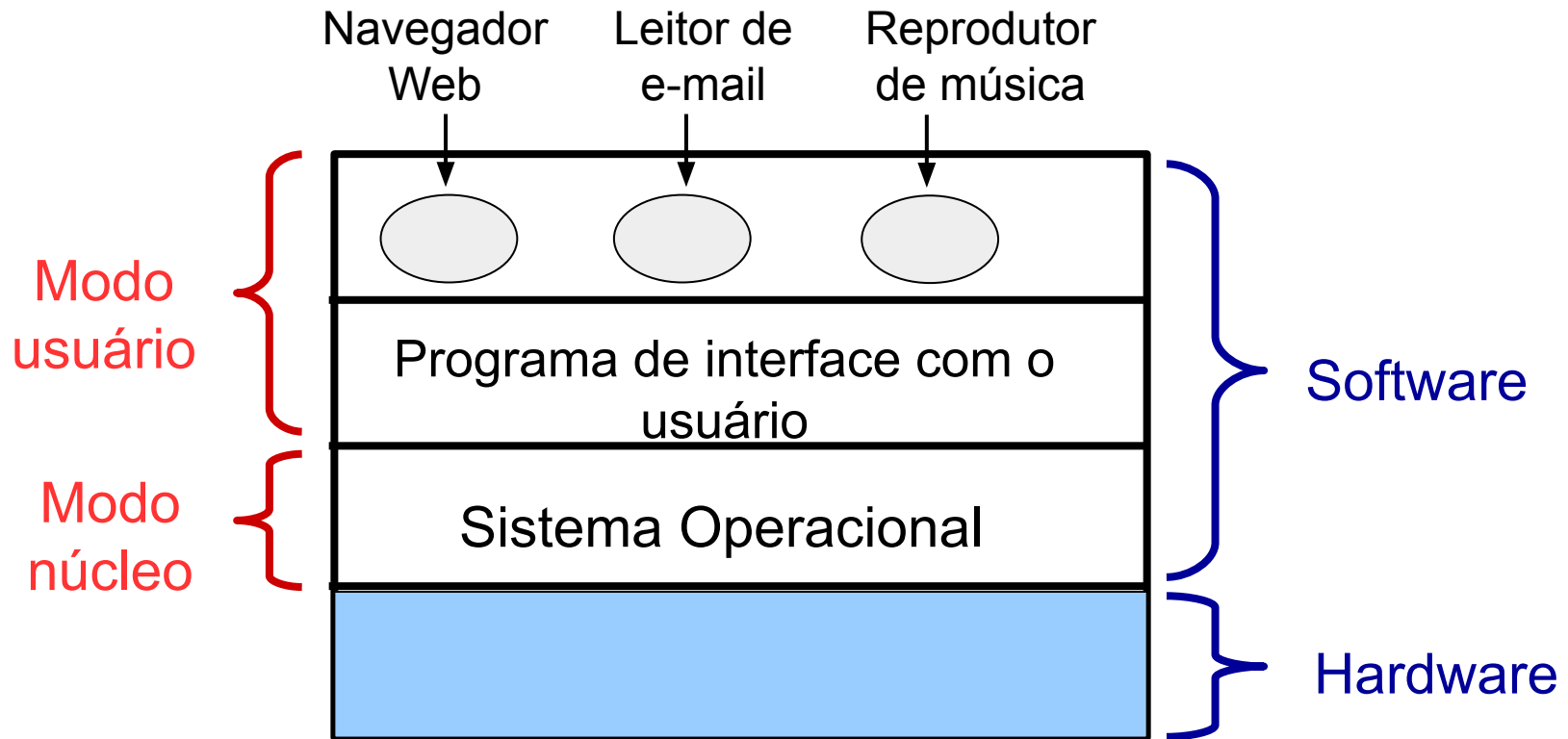
- Apenas um subconjunto de instruções está disponível
- Não estão disponíveis as instruções que afetam o controle do hardware e as que realizam operações de entrada e saída de dados
- Ex.: Não é possível mudar o *clock* do processador

## ■ Modo Núcleo

- Também chamado de modo supervisor ou modo *Kernel*
- Tem acesso a todo o hardware e pode executar qualquer instrução que o hardware é capaz de executar
- Ex.: É possível alterar o *clock* do processador

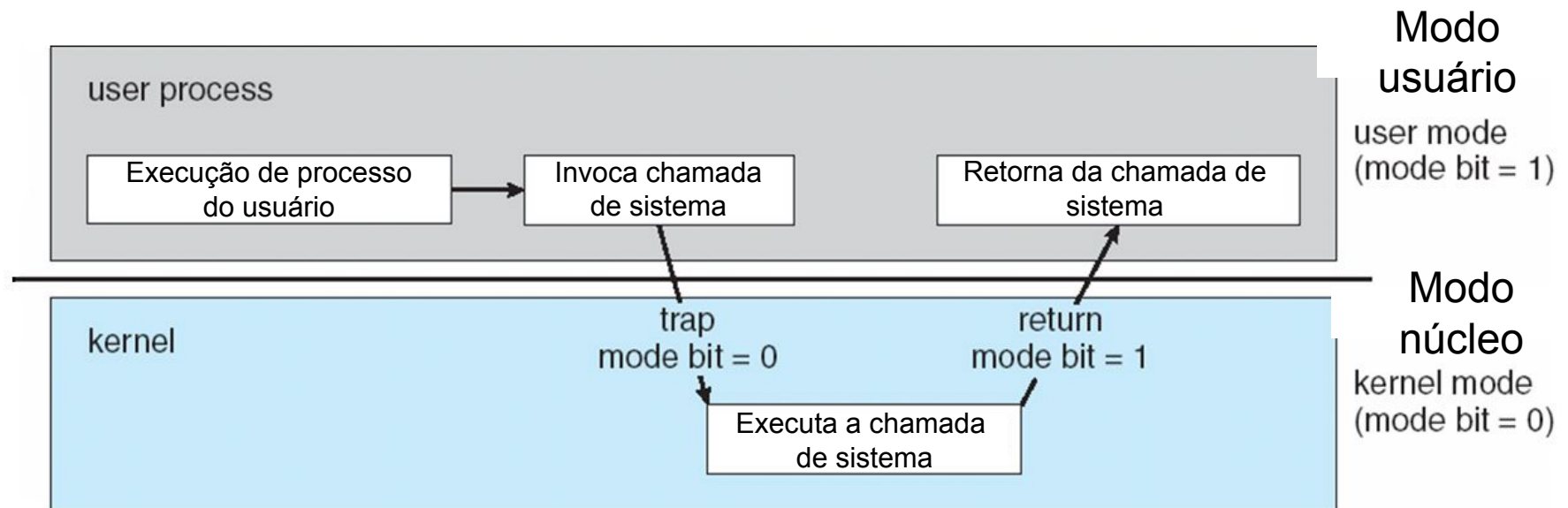


# Do Modo Usuário para o Modo Núcleo



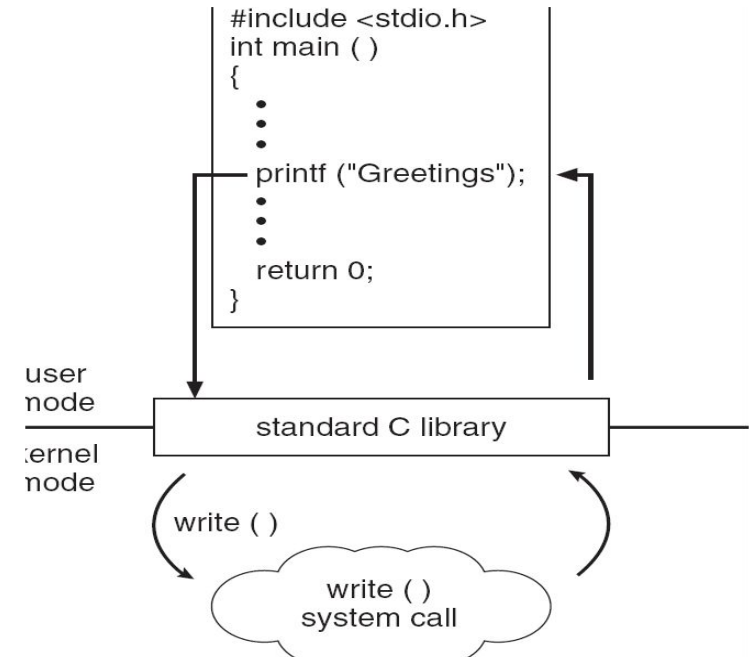
# Invocação e Retorno

- Alternância entre **modo usuário** e **modo núcleo**
  - Software pedindo ao sistema operacional para executar uma instrução



# Linguagens de Programação

- Linguagens de programação fornecem uma interface para chamadas ao SO
- Os detalhes de implementação são escondidos do programador
- O chamador não precisa saber nada sobre como a chamada de sistema é implementada
  - Apenas o nome, parâmetros e retornos



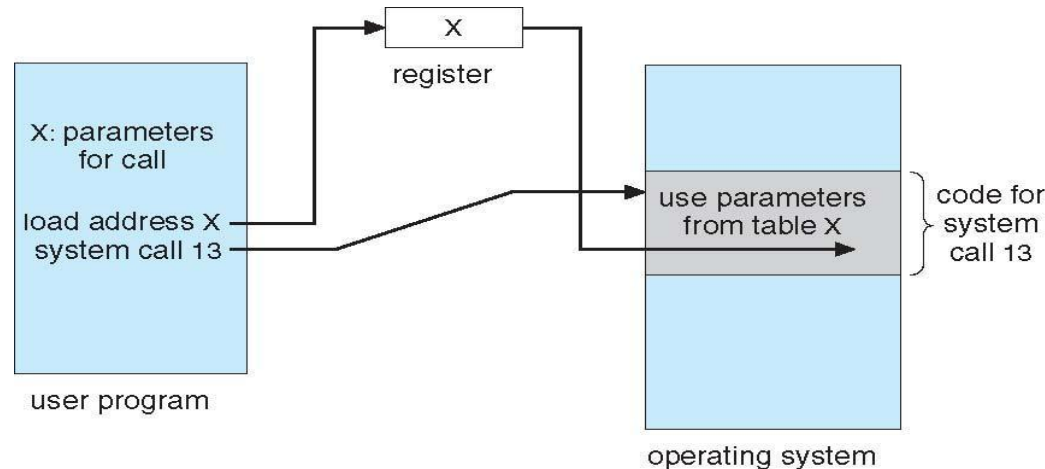
Exemplo de um programa em C, que invoca o procedimento printf() da biblioteca, que por sua vez realiza a chamada de sistema write()

# Processamento da Chamada

- Há um número associado a cada chamada de sistema
  - A interface de chamada de sistema mantém uma tabela indexada de acordo com esses números
  - Permite identificar as chamadas que ainda estão para ser executadas
- A interface de chamada de sistema invoca a chamada e retorna o *status* dela e quaisquer valores de retorno

# Passagem de Parâmetros

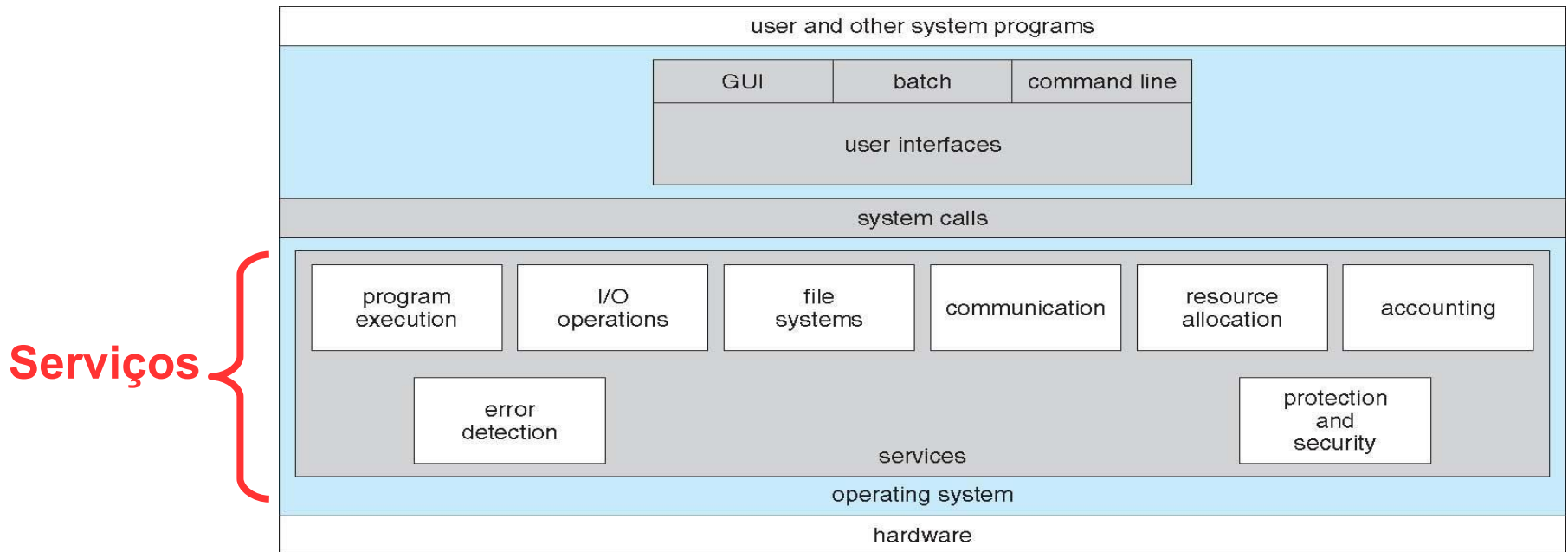
- Na chamada de sistema, programas em execução podem passar parâmetros ao SO por meio de
  - Registradores
  - Blocos (ou tabelas) da memória
  - Pilha (itens adicionados pelo programa e removidos pelo SO)



# Chamadas no Windows e Unix

	Windows	Unix
<b>Controle de processo</b>	<code>CreateProcess()</code> <code>ExitProcess()</code> <code>WaitForSingleObject()</code>	<code>fork()</code> <code>exit()</code> <code>wait()</code>
<b>Gerenciamento de arquivo</b>	<code>CreateFile()</code> <code>ReadFile()</code> <code>WriteFile()</code> <code>CloseHandle()</code>	<code>open()</code> <code>read()</code> <code>write()</code> <code>close()</code>
<b>Gerenciamento de dispositivo</b>	<code>SetConsoleMode()</code> <code>ReadConsole()</code> <code>WriteConsole()</code>	<code>ioctl()</code> <code>read()</code> <code>write()</code>
<b>Manutenção de informações</b>	<code>GetCurrentProcessID()</code> <code>SetTimer()</code> <code>Sleep()</code>	<code>getpid()</code> <code>alarm()</code> <code>sleep()</code>
<b>Comunicação</b>	<code>CreatePipe()</code> <code>CreateFileMapping()</code> <code>MapViewOfFile()</code>	<code>pipe()</code> <code>shmget()</code> <code>mmap()</code>
<b>Proteção</b>	<code>SetFileSecurity()</code> <code>InitializeSecurityDescriptor()</code> <code>SetSecurityDescriptorGroup()</code>	<code>chmod()</code> <code>umask()</code> <code>chown()</code>

# Serviços do SO



# Serviços Úteis aos Usuários

- **Execução** de programas
  - O SO deve carregar o programa na memória e possibilitar sua execução pelo processador
- **Operações** de I/O
  - Programas em execução podem requerer ao SO operações de entrada e saída de dados
- **Manipulação** de sistema de arquivos
  - Programas em execução podem requerer ao SO a leitura, escrita, criação, exclusão e pesquisa em arquivos e diretórios



# Serviços Úteis aos Usuários

## ■ Comunicação

- Programas em execução podem se comunicar uns com os outros
- Cabe ao SO gerenciar essa comunicação

## ■ Detecção de erros

- Erros podem ocorrer no hardware da CPU, memória, dispositivos de I/O, programas do usuário, etc
- O SO deve detectar o erro e tomar a medida correta que assegure o processamento correto e consistente do restante do sistema

# Serviços para Operação do Sistema

- **Alocação** de recursos

- Quando há vários programas em execução, o SO deve implementar meios de prover recursos a eles

- **Contabilização**

- Controle de quais usuário utilizam que quantidade e que tipo de recursos no computador

- **Proteção** e Segurança

- Garantir que um programa em execução não interfira no funcionamento de outros programas e nem no SO
- Garantir a segurança do sistema contra invasores

# Funcionamento dos SOs

- Principais gerências
  - Gerência de Processos
  - Gerenciamento de Memória
  - Gerenciamento de Armazenamento e de Entrada e Saída

# Atividade de Fixação

A delimitação de “modo usuário” e “modo núcleo” no projeto de Sistemas Operacionais é necessária para:

- a) prover ao usuário uma interface gráfica amigável e de fácil uso.
- b) possibilitar a inicialização do sistema operacional quando a energia é ativada.
- c) delimitar instruções que podem ser executadas apenas pelo sistema operacional.
- d) implementar a gerência de memória, processos e armazenamento.

# Material Complementar

- **Tabela.** "Tabela de Chamadas ao sistema no Linux" Disponível em <https://www.ime.usp.br/~kon/MAC211/syscalls.html> Acesso em: 04 Fev. 2024
- **Prática de monitoramento:** "Understanding system calls on Linux with strace" Disponível em: <https://opensource.com/article/19/10/strace> Acesso em: 04 Fev. 2024
- **Documentação:** "strace linux syscall tracer" Disponível em: <https://strace.io/> Acesso em: 04 Fev. 2024

# Referências

SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. Fundamentos de sistemas operacionais: princípios básicos. Rio de Janeiro, RJ: LTC, 2013. xvi, 432 p. (Capítulos 1 e 2)

TANENBAUM, Andrew S. Sistemas operacionais modernos. 3. ed. São Paulo: Pearson Prentice Hall, 2009. xvi, 653 p. (Capítulos 1)

O sistema operacional é um programa de computador que opera o hardware e fornece a base para todos os outros programas, implementando a alocação de recursos e o controle do uso desses recursos.

## Sistemas Operacionais

**Prof. Dr. Lesandro Ponciano**

<https://orcid.org/0000-0002-5724-0094>