



INTRODUÇÃO À ARQUITETURA DE SISTEMAS OPERACIONAIS

Prof. MSc. Marco Antonio Alves Pereira



**Prof. MSc. Marco Antonio Alves Pereira
FATEC – Faculdade de Tecnologia de Ribeirão Preto**

VERSÃO 1.1 – JANEIRO/2026.

SUMÁRIO

1	Capítulo 1: A Essência dos Sistemas Operacionais.....	4
1.1	Por que Estudar Sistemas Operacionais?	4
1.2	O S.O. na Arquitetura do Computador	4
1.3	Conteúdo da Disciplina: Teoria e Prática	6
2	Capítulo 2: Visão Geral dos Sistemas Operacionais	8
2.1	Introdução.....	8
2.2	Funções Principais de um S.O.	8
2.3	Tipos de Sistemas Operacionais	9
2.4	Principais Componentes do Sistema Operacional.....	10
2.5	Introdução a Sistemas de Arquivos	14
3	Capítulo 3: Processos e Gerenciamento de Execução	18
3.1	Introdução: Programas vs. Processos	18
3.2	A Estrutura do Processo: O PCB (Bloco de Controle de Processo)	18
3.3	Ciclo de Vida: Estados dos Processos	19
3.4	Escalonamento e Compartilhamento de Tempo (Time Slice)	21
3.5	Componentes do Escalonamento: Escalonador e Dispatcher.....	22
3.6	Tipos de Processos.....	23
3.7	Threads (Linhas de Execução).....	24
3.8	Processos do Sistema.....	25
3.9	Sinais	26
4	Capítulo 4: Programação Concorrente e Comunicação entre Processos	27
4.1	Introdução à Programação Concorrente	27
4.2	Comunicação entre Processos (IPC)	28
4.3	Exclusão Mútua e Região Crítica.....	29
4.4	Problemas Clássicos de Concorrência.....	29
4.5	Semáforos	31
5	Capítulo 5: Gerência de Memória e Memória Virtual.....	33
5.1	Introdução à Gerência de Memória.....	33
5.2	Primeiras Técnicas de Alocação de Memória	33
5.3	Swapping	34
5.4	Memória Virtual.....	36
5.5	Swapping e Gerenciamento de Ausência de Páginas	37
5.6	Compartilhamento de Memória.....	39
6	Glossário	40
7	CONSIDERAÇÕES FINAIS	42
8	BIBLIOGRAFIA CONSULTADA	43

1 Capítulo 1: A Essência dos Sistemas Operacionais

1.1 Por que Estudar Sistemas Operacionais?

Você interage com a computação todos os dias: seu smartphone, seu notebook, o caixa eletrônico do banco. Por trás de cada clique, toque ou transação, existe uma peça de software trabalhando incansavelmente para que tudo funcione: o Sistema Operacional (S.O.).

Estudar Sistemas Operacionais é mergulhar na **alma da computação**. Esta disciplina não é apenas teórica; é o entendimento de como o software e o hardware conversam, o que torna a multitarefa possível e como seu computador gerencia seus recursos mais valiosos.

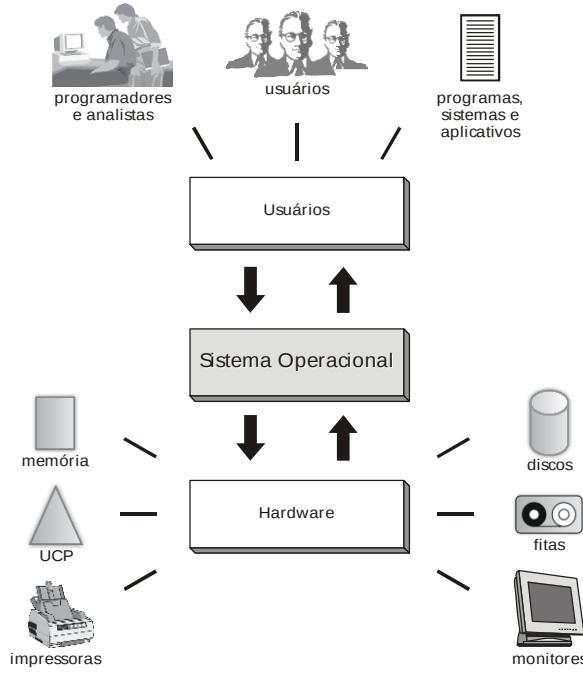
Para a sua Carreira:

- **Fundamento de Redes e Servidores:** Administradores de rede, profissionais de segurança e engenheiros de *cloud* precisam de um conhecimento profundo do S.O. para configurar, otimizar e solucionar problemas em servidores (como Windows Server e Ubuntu Server).
- **Otimização de Software:** Programadores que entendem de processos, *threads* e gerência de memória escrevem códigos mais rápidos, mais eficientes e livres de *bugs* de concorrência.
- **Cibersegurança:** O conhecimento sobre o *kernel*, permissões e sistemas de arquivos é crucial para defender sistemas e entender como atacantes exploram vulnerabilidades.

Em suma, dominar Sistemas Operacionais é o que transforma um usuário básico em um profissional capaz de entender o **"porquê"** e o **"como"** as coisas realmente funcionam dentro da máquina.

1.2 O S.O. na Arquitetura do Computador

O Sistema Operacional ocupa uma posição estratégica e única na arquitetura de um sistema de computador. Ele se encontra exatamente entre o hardware e o usuário (ou seus aplicativos).



Podemos visualizá-lo como um intermediário essencial, dividido em duas grandes partes funcionais:

1.2.1. O Gerente de Recursos (Kernel)

O **Kernel** é o coração do S.O., a parte mais fundamental e que reside permanentemente na memória. Sua principal função é ser o **Gerente de Recursos** do computador, cuidando de:

- 1. Gerenciamento do Processador (CPU):** Decide qual programa (processo) deve rodar, quando e por quanto tempo (Escalonamento).
- 2. Gerenciamento da Memória:** Controla a alocação de memória RAM para cada processo e implementa a Memória Virtual.
- 3. Gerenciamento de Entrada e Saída (E/S):** Controla discos, teclados, placas de rede e impressoras.
- 4. Gerenciamento de Arquivos:** Cria, deleta e organiza arquivos e diretórios em sistemas de arquivos (como NTFS ou EXT4).

O S.O. é quem gerencia o hardware. Se um aplicativo de usuário precisasse acessar diretamente o disco rígido ou a CPU, a bagunça seria generalizada (um programa poderia corromper a memória de outro). O S.O. fornece uma interface

segura (chamada de **Chamadas de Sistema** - *System Calls*) que os aplicativos usam para pedir recursos ao *hardware*.

1.2.2. A Interface com o Usuário (Shell)

O *Shell* é a camada mais externa, responsável pela comunicação com o usuário. Pode ser:

- **Gráfico (GUI):** A interface com ícones, janelas e mouse (ex: Windows Explorer, GNOME, KDE).
- **Linha de Comando (CLI):** Onde o usuário digita comandos em texto (ex: Prompt de Comando, PowerShell, Terminal Linux).

1.3 Conteúdo da Disciplina: Teoria e Prática

Nesta disciplina de Introdução a Sistemas Operacionais, faremos uma jornada que combina a **Teoria Fundamental** que rege todos os S.O. (Windows, Linux, etc) com a **Prática de Servidores** que você encontrará no mercado de trabalho.

A. Fundamentos Teóricos

A teoria é o "motor" da computação. Entenderemos os conceitos de forma profunda para resolver problemas de alto nível.

- **Visão Geral:** Definição, funções e tipos de S.O.
- **Processos e Escalonamento:** O que é um processo, como ele nasce, morre e como o S.O. o gerencia para criar a multitarefa (Time Slice e Escalonador).
- **Programação Concorrente:** O desafio de fazer vários processos ou *threads* rodarem juntos sem travar, explorando técnicas como Semáforos e Exclusão Mútua.
- **Gerência de Memória e Memória Virtual:** Como a RAM é usada de forma eficiente, a ilusão da Memória Virtual e o papel da Paginação e do Swapping.

B. Aplicação Prática e Servidores

A partir da metade do curso, aplicaremos a teoria em dois dos S.O. mais relevantes para o ambiente de servidores e infraestrutura:

- **Microsoft Windows Server:** Abordaremos sua estrutura, gerenciamento de usuários, grupos, sistemas de arquivos (NTFS) e o papel do S.O. em ambientes corporativos.
- **Ubuntu Server (Linux):** Exploraremos a linha de comando (CLI), a gestão de permissões, sistemas de arquivos (EXT4) e a administração básica de serviços em um ambiente Open Source, amplamente utilizado em *cloud computing* e *data centers*.

Este curso é o seu ponto de partida para se tornar um profissional que não apenas utiliza sistemas, mas que os entende profundamente e é capaz de gerenciá-los com maestria. Dedique-se à teoria, pois ela será a chave para desbloquear a prática no futuro.

Resumo do Capítulo 1: A Essência dos Sistemas Operacionais

Neste capítulo, estabelecemos que o Sistema Operacional (S.O.) é o software central que funciona como **Gerente de Recursos** e **Máquina Estendida**, mediando entre o usuário/aplicações e o hardware. Vimos que o S.O. se divide fundamentalmente em duas partes: o **Kernel**, que é o núcleo privilegiado responsável por todas as operações de baixo nível e gerenciamento de recursos, e o **Shell**, que é a interface de comunicação com o usuário (seja ela gráfica - GUI, ou de linha de comando - CLI). O estudo do S.O. é vital para as áreas de **Infraestrutura**, **Otimização de Software** (devido ao entendimento de threads) e **Cibersegurança**, pois o conhecimento do funcionamento do kernel e das permissões é crucial para proteger o sistema.

2 Capítulo 2: Visão Geral dos Sistemas Operacionais

2.1 Introdução

Ao iniciarmos o estudo da computação moderna, deparamo-nos inevitavelmente com a pergunta fundamental: **O que é um Sistema Operacional (S.O.)?**

Em termos técnicos, um Sistema Operacional é uma coleção complexa de programas e rotinas responsáveis por gerenciar o hardware do computador. Ele atua como um intermediário vital entre o usuário (e seus aplicativos) e a parte física da máquina. Suas responsabilidades primárias incluem:

- **Gerenciamento de Hardware:** Controlar processadores, memória, discos e periféricos.
- **Rotinas de Controle:** Fornecer instruções básicas para que dispositivos funcionem corretamente.
- **Gerência de Tarefas:** Decidir qual programa roda, quando e por quanto tempo (escalonamento).
- **Integridade do Sistema:** Garantir que o sistema permaneça estável e seguro, prevenindo erros críticos.

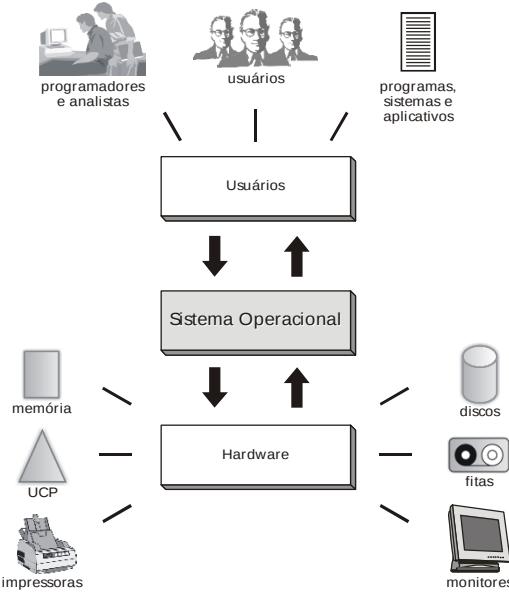
Podemos imaginar o S.O. como o "gerente" de uma grande empresa (o computador), garantindo que todos os departamentos (memória, CPU, dispositivos) trabalhem de forma sincronizada para atender às demandas dos clientes (usuários e softwares).

2.2 Funções Principais de um S.O.

Embora um Sistema Operacional execute milhares de operações por segundo, todas elas convergem para duas funções macroscópicas essenciais:

1. **Abstração (Facilitar o Acesso):** O S.O. esconde a complexidade do hardware. O usuário não precisa saber como mover a agulha de um disco rígido para salvar um arquivo; ele apenas clica em "Salvar". O S.O. traduz esse comando simples em instruções complexas de máquina.
2. **Gerenciamento (Compartilhar Recursos):** O S.O. organiza e protege os recursos. Se dois programas tentarem imprimir um documento ao mesmo tempo, o S.O. decide quem vai primeiro (fila de impressão), evitando

conflitos e garantindo que o processador e a memória sejam compartilhados de forma justa e protegida.



Fonte: Arquitetura de Sistemas Operacionais – Machado e Maia

2.3 Tipos de Sistemas Operacionais

Os sistemas operacionais evoluíram para atender a diferentes necessidades e hardwares. Eles são classificados em cinco grupos principais:

2.3.1. S.O. Monotarefa

Projetados para que um único usuário execute apenas uma tarefa de cada vez. Eram comuns em dispositivos mais antigos ou limitados, onde a execução de múltiplos programas simultâneos não era necessária ou possível devido a restrições de hardware.

- **Exemplo:** Palm OS (usado nos antigos PDAs).

2.3.2. S.O. Monousuário / Multitarefa

É o tipo mais comum em computadores pessoais (desktops e laptops) hoje. Permite que um único usuário execute diversos programas simultaneamente. O sistema gerencia a memória e o processador para que você possa ouvir música, navegar na internet e editar um texto ao mesmo tempo.

- **Exemplos:** Microsoft Windows e Apple macOS.

2.3.3. S.O. Multusuário

Permite que diversos usuários utilizem os recursos de um único computador simultaneamente. O sistema deve ser robusto para balancear as solicitações de todos, garantindo que o processamento pesado de um usuário não trave o sistema para os outros. É muito utilizado em grandes servidores.

- **Exemplos:** Unix, VMS e sistemas de Mainframes (como MVS).

2.3.4. S.O. de Tempo Compartilhado (Time-Sharing)

Esta é a técnica por trás da multitarefa. Na realidade, um processador (com um único núcleo) executa apenas uma instrução por vez. O S.O. de tempo compartilhado fatia o tempo do processador em pedaços minúsculos e alterna rapidamente entre os programas. Essa alternância é tão veloz que cria a **ilusão** de que tudo está rodando simultaneamente.

2.3.5. S.O. de Tempo Real (RTOS)

São sistemas focados em precisão temporal, não necessariamente em velocidade para o usuário final. São usados para controlar máquinas industriais, sistemas de aviação ou instrumentos científicos.

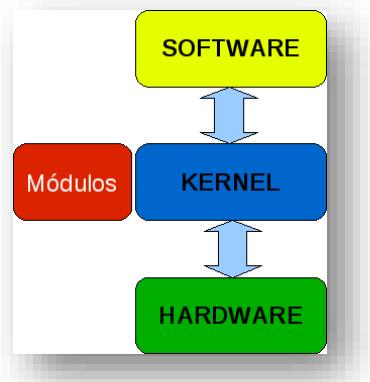
- **Características:** Geralmente são "caixas seladas" (sem interface gráfica complexa para o usuário). A prioridade é garantir que uma tarefa específica seja executada exatamente dentro de um prazo rígido (deadline).

2.4 Principais Componentes do Sistema Operacional

A arquitetura de um S.O. é composta por camadas e módulos que interagem entre si. Os quatro componentes fundamentais são:

2.4.1. Kernel (Núcleo)

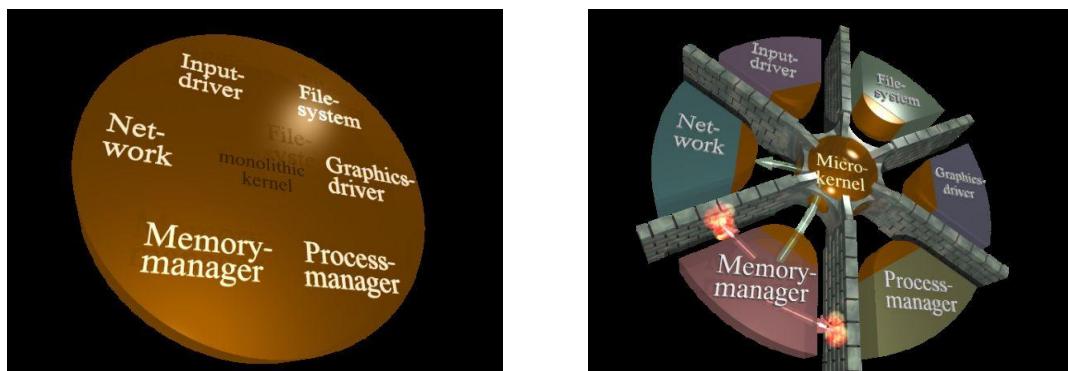
O Kernel é o coração do sistema operacional. É a primeira parte do S.O. a ser carregada na memória quando ligamos o computador e fica lá até desligarmos. Ele é responsável por detectar o hardware (vídeo, processador, etc.) e gerenciar as operações de baixo nível.



<http://www.vivaolinux.com.br/artigo/Novidades-do-Kernel-2.6.35>

Existem duas arquiteturas principais de Kernel:

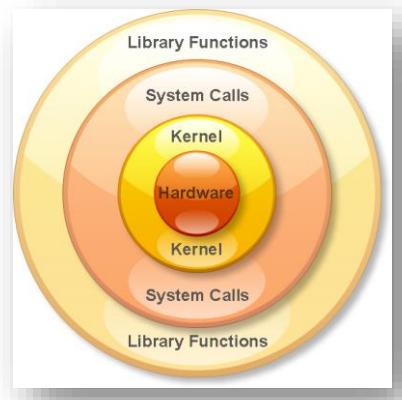
- **Kernel Monolítico:** Todos os subsistemas (gerenciamento de memória, rede, drivers de arquivos) estão agrupados em um único bloco de código executável. É rápido, mas um erro em um driver pode travar todo o sistema.
- **Microkernel:** Mantém apenas o mínimo essencial no núcleo. Outros serviços (como sistemas de arquivos e drivers) rodam como processos separados (módulos), comunicando-se por troca de mensagens. Isso torna o sistema mais modular e estável.



2.4.2. System Call (Chamada de Sistema)

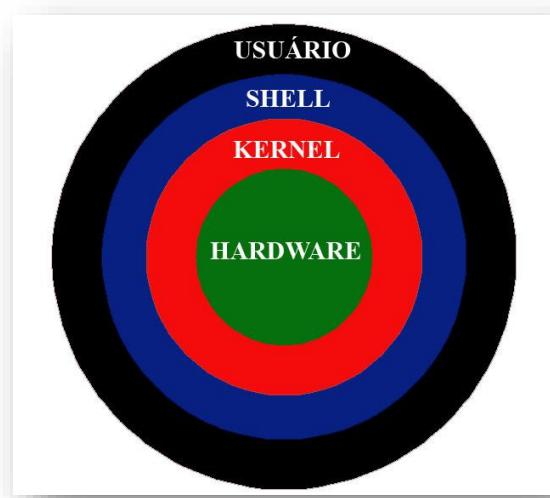
É a porta de comunicação entre os programas (software do usuário) e o Kernel. Por segurança, os programas não têm permissão para acessar o hardware diretamente (para evitar que um software malicioso apague o disco, por exemplo). Quando um programa precisa ler um arquivo ou desenhar na tela, ele faz uma

System Call solicitando esse serviço ao Kernel. O Kernel verifica se o pedido é válido e, se for, executa a ação.

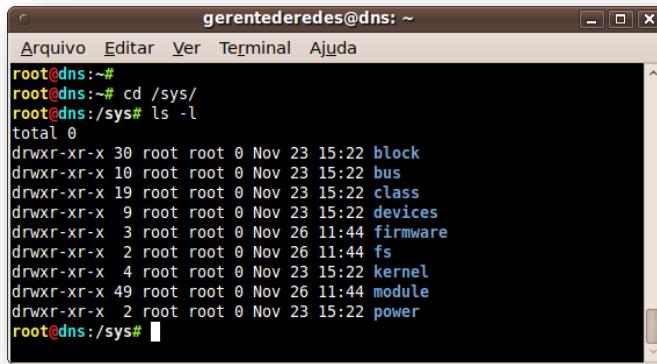


2.4.3. Shell (Interpretador de Comandos)

O Shell é a interface visível para o usuário. Ele recebe os comandos (seja via texto ou cliques de mouse), interpreta o que o usuário deseja e instrui o sistema a fazer.



- **Interface de Linha de Comando (CLI):** Onde o usuário digita comandos (ex: Prompt de Comando, Bash no Linux).



```
gerentederedes@dns: ~
Arquivo Editar Ver Terminal Ajuda
root@dns:~#
root@dns:# cd /sys/
root@dns:/sys# ls -l
total 0
drwxr-xr-x 30 root root 0 Nov 23 15:22 block
drwxr-xr-x 10 root root 0 Nov 23 15:22 bus
drwxr-xr-x 19 root root 0 Nov 23 15:22 class
drwxr-xr-x 9 root root 0 Nov 23 15:22 devices
drwxr-xr-x 3 root root 0 Nov 26 11:44 firmware
drwxr-xr-x 2 root root 0 Nov 26 11:44 fs
drwxr-xr-x 4 root root 0 Nov 23 15:22 kernel
drwxr-xr-x 49 root root 0 Nov 26 11:44 module
drwxr-xr-x 2 root root 0 Nov 23 15:22 power
root@dns:/sys#
```

- **Interface Gráfica (GUI):** Onde o usuário interage visualmente (janelas, ícones, menus). Embora pareça diferente, a interface gráfica é uma capa visual que, no fundo, envia instruções ao sistema.



2.4.4. Loader (Carregador)

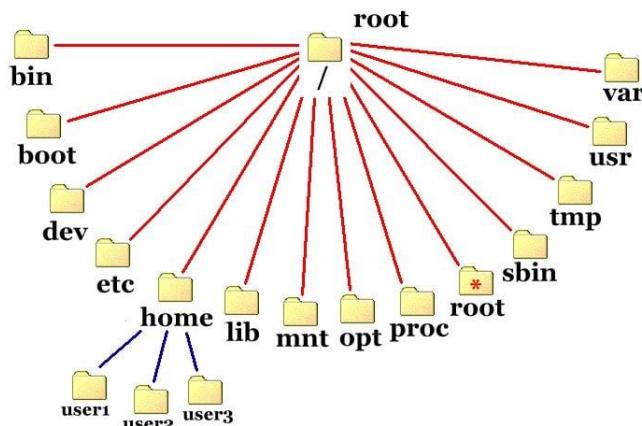
É um utilitário essencial responsável por pegar um programa que está armazenado no disco (HD/SSD) e colocá-lo fisicamente na memória RAM para que possa ser executado. Sem o loader, o processador não conseguiria acessar as instruções dos programas.

2.5 Introdução a Sistemas de Arquivos

Para o computador, tudo são dados. O Sistema de Arquivos é a lógica que organiza como esses dados são armazenados e recuperados em dispositivos não voláteis (Discos Rígidos, SSDs, Pen Drives).

2.5.1. Conceitos Básicos

- **Arquivo:** Um conjunto de bytes armazenado sob um nome.
- **Metadados:** Informações sobre o arquivo, como nome, extensão, permissões de acesso (quem pode ler/escrever), tamanho e localização física no disco.
- **Diretório (Pasta):** Tecnicamente, é um arquivo especial que serve apenas para armazenar referências (metadados) de outros arquivos, criando uma organização lógica.
- **Estrutura em Árvore:** Os diretórios são organizados hierarquicamente, partindo de uma raiz (Root / no Linux ou C:\ no Windows) e ramificando-se em subdiretórios.



2.5.2. Armazenamento Físico e Lógico

- **Formatação Lógica:** Prepara o disco para receber o S.O., organizando os setores físicos em **Clusters** (unidades lógicas de alocação).
- **SSD vs HD:**
 - **HD (Hard Disk):** Mecânico, magnético.

- **SSD (Solid State Drive):** Usa memória Flash NAND. Não possui partes móveis, o que garante menor latência e maior velocidade. Porém, possui um ciclo de vida limitado de escritas (embora a tecnologia atual tenha mitigado muito esse problema).

2.5.3. Unidades de Medida de Dados

Na computação, medir a capacidade de armazenamento e memória não é apenas uma questão de listar siglas. Existe uma distinção fundamental entre como o hardware (fabricantes de discos) e o software (Sistemas Operacionais) enxergam esses números.

As Duas Bases de Cálculo

1. Base Decimal: Utilizada por fabricantes de HDs, SSDs e pen drives. Para eles, 1 KB equivale a 1.000 bytes. É por isso que, ao comprar um "HD de 500 GB", o Windows frequentemente mostra um valor menor; o fabricante usa a base 10, enquanto o sistema usa a base 2.

2. Base Binária: Utilizada pelos Sistemas Operacionais e para medir a memória RAM. Aqui, 1 KB (ou KiB) equivale a 1.024 bytes.

A Escala Exponencial de Armazenamento

À medida que a tecnologia avança, passamos de KB para escalas colossais que o Sistema Operacional precisa ser capaz de gerenciar:

- KB (Kilobyte): 10^3 bytes. Antigamente, o tamanho de um arquivo de texto simples.
- MB (Megabyte): 10^6 bytes. Escala comum para fotos e músicas (MP3).
- GB (Gigabyte): 10^9 bytes. Padrão atual para memória RAM e pequenos arquivos de vídeo. Um processador de 32 bits, por exemplo, consegue endereçar no máximo 4 GB de RAM.
- TB (Terabyte): 10^{12} bytes. Escala comum para HDs e SSDs modernos de uso doméstico.
- PB (Petabyte): 10^{15} bytes. Utilizado em grandes Data Centers e servidores de Cloud Computing.
- EB (Exabyte): 10^{18} bytes. É o limite prático de tecnologias modernas. Por exemplo, o sistema de arquivos NTFS do Windows e o EXT4 do Linux suportam

volumes de até 16 Exabytes. Além disso, processadores de 64 bits têm a capacidade teórica de endereçar até 16 EB de RAM.

- ZB (Zettabyte): 10^{21} bytes. Refere-se à escala de todo o tráfego global da internet. Sistemas de arquivos avançados, como o ZFS, foram projetados para lidar com essa magnitude.
- YB (Yottabyte): 10^{24} bytes. A maior unidade oficialmente reconhecida, representando uma quantidade de dados quase inimaginável para os padrões atuais.

Por que isso importa para o S.O.? O Sistema Operacional deve possuir uma arquitetura (como 64 bits) e sistemas de arquivos (como NTFS ou XFS) que consigam "enxergar" e gerenciar essas quantidades imensas de dados sem perda de desempenho ou erros de endereçamento.

Nome	Símbolo	Múltiplo
byte	B	10^0
kilobyte	kB	10^3
megabyte	MB	10^6
gigabyte	GB	10^9
terabyte	TB	10^{12}
petabyte	PB	10^{15}
exabyte	EB	10^{18}
zettabyte	ZB	10^{21}
yottabyte	YB	10^{24}

2.5.4. Exemplos de Sistemas de Arquivos

No mundo Windows:

- **FAT16 / FAT32:** Antigos, com limitações de tamanho de arquivo e segurança. Muito compatíveis com pendrives.
- **NTFS (New Technology File System):** O padrão atual do Windows. Suporta:
 - Arquivos maiores que 4GB.
 - **Journaling:** Um registro de transações que evita corrupção de dados em caso de queda de energia.
 - Permissões de acesso (segurança) e criptografia nativa.
 - Compactação de dados.

No mundo Linux:

- **EXT4 (Fourth Extended Filesystem):** O padrão mais comum. Robusto, suporta volumes de até 1 Exabyte e arquivos de até 16 Terabytes. Possui journaling.
- **XFS:** Padrão em sistemas corporativos como Red Hat. Excelente para gerenciar arquivos gigantescos e alto desempenho.
- **BTRFS:** Um sistema moderno focado em tolerância a falhas, snapshots (cópias instantâneas do estado do sistema) e fácil gerenciamento de volumes.

Resumo do Capítulo 2: Visão Geral dos Sistemas Operacionais

Continuando, examinamos as funções primárias do S.O., que são a **Abstração** (simplificar o uso do hardware) e a **Gerência de Recursos** (controlar o acesso à CPU, memória e I/O). Para garantir a estabilidade e a segurança, o processador opera em dois modos: o **Modo Kernel (Privilegiado)**, onde apenas o código do S.O. pode executar instruções críticas, e o **Modo Usuário**, onde rodam as aplicações restritas. Qualquer serviço de hardware solicitado por um aplicativo de usuário deve ser feito através de uma **Chamada de Sistema (System Call)**. Também estudamos as estruturas de Kernel: o **Monolítico**, que é rápido, mas menos tolerante a falhas, e o **Microkernel**, que é mais estável e modular, mas incorre em maior overhead de comunicação. Por fim, classificamos os S.O., destacando os sistemas de **Tempo Real**, onde a resposta dentro de um deadline é crítica.

3 Capítulo 3: Processos e Gerenciamento de Execução

3.1 Introdução: Programas vs. Processos

Para entender como um sistema operacional moderno funciona, precisamos fazer uma distinção fundamental entre dois conceitos: **Programa** e **Processo**.

- **O Programa (Estático):** É o código, um conjunto passivo de instruções armazenado no disco.
- **O Processo (Dinâmico):** É a instância de um programa em execução. Ele é a entidade ativa que consome recursos (CPU, memória) e existe apenas enquanto o programa estiver rodando.

O Sistema Operacional utiliza o conceito de Processo para organizar a execução multitarefa.

3.2 A Estrutura do Processo: O PCB (Bloco de Controle de Processo)

Quando um processo é criado, o Sistema Operacional precisa de uma estrutura de dados para gerenciar todas as informações vitais sobre ele. Essa estrutura é o **PCB (Process Control Block)**. O PCB é a "ficha técnica" do processo e reside na memória, sendo o único local que o S.O. precisa consultar para saber tudo sobre um processo.

O PCB armazena informações essenciais, divididas em três grandes grupos:

	Contexto de Software						Espaço de Endereç.			Contexto de Hardware		
	Nome	PID	PPID	UID	Prior.	Dt/Hr	End1	End2	End3...	Reg1	Reg2	Reg3...
PCBa												
PCBb												
PCBc												

3.2.1. Contexto de Software

São informações lógicas e de identificação:

- **PID (Process ID):** Identificação única do processo.
- **PPID (Parent Process ID):** Identificação do processo pai (quem o criou).
- **UID (User ID):** Usuário proprietário, definindo permissões de acesso e segurança.
- **Prioridade:** Nível de importância do processo para o Escalonador.

3.2.2. Contexto de Hardware

Quando um processo está em execução, ele utiliza os registradores da CPU. Quando ele é interrompido, o S.O. salva o estado desses registradores no PCB para que a execução possa ser retomada exatamente de onde parou.

- **Registradores Gerais:** Valores temporários usados nos cálculos.
- **Program Counter (PC):** O endereço da próxima instrução de código a ser executada.

3.2.3. Espaço de Endereçamento

Define o limite de memória RAM (Memória Principal) alocada para aquele processo. Isso garante que um processo não acesse nem corrompa a memória de outro (proteção de memória).

3.3 Ciclo de Vida: Estados dos Processos

Em um ambiente multitarefa, um processo alterna entre o uso da CPU e a espera por recursos.

- Processos **nascem**
 - No momento de sua criação (via chamada de sistema)
- Processos **vivem**
 - Executam na CPU, liberam a CPU (E/S)...
 - Executam:
 - Programas dos usuários
 - Programas do sistema (daemons)

- Processos ***morrem***
 - Ou porque terminaram sua execução
 - Ou porque um outro processo os matou:
 - Erro, acesso não-autorizado, falha

Os processos passam pelos seguintes estados:

Estado	Descrição
Novo (New)	O processo está sendo criado. O S.O. está alocando o PCB e recursos iniciais.
Pronto (Ready)	O processo está na memória, tem todos os recursos necessários e está apto a rodar, mas está aguardando pela CPU .
Execução (Running)	O processo está usando ativamente o processador e executando suas instruções. Em um sistema com um núcleo de CPU, apenas um processo pode estar neste estado por vez.
Espera / Bloqueado (Waiting)	O processo liberou voluntariamente a CPU para aguardar por um evento lento, como uma operação de Entrada/Saída (E/S), leitura de disco ou uma ação do usuário (ex: digitar no teclado).
Terminado (Terminated)	O processo encerrou sua execução. O S.O. libera seus recursos e memória.

3.3.1. Transição de Estados

- **De Pronto para Execução:** O Escalonador (Scheduler) seleciona o processo.
- **De Execução para Pronto:** O tempo de uso do processador (Time Slice) se esgotou (preempção).
- **De Execução para Espera:** O processo solicita uma operação de E/S.
- **De Espera para Pronto:** O evento de E/S solicitado é concluído e o processo está novamente pronto para competir pela CPU.



3.4 Escalonamento e Compartilhamento de Tempo (Time Slice)

O S.O. cria a ilusão de multitarefa em um único processador ao dar a cada processo uma fatia muito curta de tempo na CPU.

- **Time Slice (ou Quantum):** É o tempo máximo que um processo pode usar o processador de forma contínua. Por ser extremamente curto (geralmente entre 10ms e 100ms), a troca constante entre processos é imperceptível para o usuário.
- **Escalonamento (Scheduling):** É o ato de gerenciar e decidir qual processo da fila de **Pronto** será o próximo a ganhar a CPU. Essa é uma das tarefas mais críticas do Sistema Operacional, pois afeta diretamente o desempenho e a responsividade do sistema.

3.4.1. Tipos de Escalonamento

Os algoritmos de escalonamento são geralmente classificados em dois tipos, dependendo de como lidam com o processo que está em execução:

1. **Não-Preemptivo:** O processo, uma vez na CPU, só a libera voluntariamente, seja porque terminou sua execução, seja porque entrou em estado de **Espera** (solicitou E/S). O S.O. não pode forçar a retirada.
2. **Preemptivo:** O S.O. pode interromper o processo em execução a qualquer momento, geralmente ao fim do **Time Slice**, forçando-o a voltar para o estado **Pronto**. A maioria dos sistemas operacionais modernos utiliza escalonamento preemptivo para garantir a justiça e a interação em tempo real.

3.4.2 O que é Preempção?

A preempção é a capacidade soberana do S.O. de suspender a execução de um processo para priorizar outro ou para garantir que todos os processos recebam uma "fatia" do processador. Ela pode ser disparada por dois motivos principais:

1. Preempção por Tempo: Ocorre quando o Time Slice (ou Quantum) do processo chega ao fim. O timer do sistema gera uma interrupção, e o S.O. coloca o processo atual de volta na fila de espera para que o próximo da fila possa rodar.

2. Preempção por Prioridade: Acontece quando um processo com prioridade maior do que o que está executando torna-se Pronto (por exemplo, ao terminar uma leitura de disco). O S.O. interrompe o processo menos importante para dar lugar ao mais urgente.

Benefícios da Preempção:

- **Justiça (Fairness):** Garante que nenhum processo monopolize o processador por tempo indefinido, permitindo que todos tenham oportunidade de execução.
- **Responsividade:** Permite que aplicativos interativos (como o mouse ou o teclado) respondam rapidamente ao usuário, mesmo que o computador esteja realizando cálculos pesados em segundo plano.
- **Estabilidade:** Evita que um erro (como um loop infinito) em um aplicativo trave o sistema inteiro, pois o S.O. ainda conseguirá interrompê-lo via preempção.

3.5 Componentes do Escalonamento: Escalonador e Dispatcher

Para que a troca de processos ocorra de forma eficiente, dois módulos principais do S.O. trabalham em conjunto:

- **Escalonador (Scheduler):** É o módulo que executa o Algoritmo de Escalonamento. Sua função é puramente lógica: Decidir qual processo da fila de "Pronto" será o próximo a ser executado.
- **Dispatcher:** É o módulo que executa a Mudança de Contexto (Context Switch). Sua função é operacional: Retirar o processo que está saindo da CPU, salvar seu contexto (PCB), carregar o contexto (PCB) do processo escolhido pelo Escalonador e, finalmente, entregar a CPU a esse novo processo.

3.5.1 Principais Algoritmos de Escalonamento Atualmente

- Completely Fair Scheduler (Linux)
- Multilevel Feedback Queue Scheduler (Windows).

3.6 Tipos de Processos

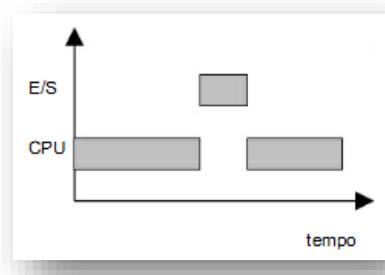
Os processos podem ser categorizados pelo seu nível de interação com o usuário e pela sua demanda por recursos:

3.6.1. Pela Interação (Foreground vs. Background)

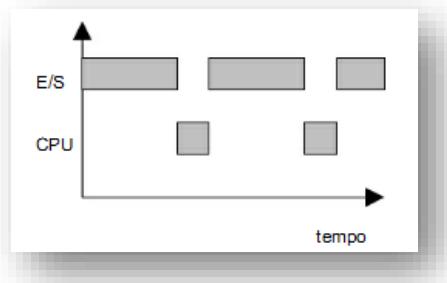
- **Foreground (Primeiro Plano):** Processos que interagem diretamente com o usuário e, portanto, exigem respostas rápidas (ex: Navegador, editor de texto).
- **Background (Segundo Plano):** Processos que rodam "escondidos" e não interagem com o usuário (ex: Servidores de e-mail, Antivírus, serviços de sistema).

3.6.2. Pelo Uso de Recursos (CPU Bound vs. I/O Bound)

- **CPU Bound (Ligado à CPU):** Processos que passam a maior parte do tempo realizando cálculos (ex: Renderização de vídeo, simulações científicas). Eles tendem a usar o **Time Slice** completo.



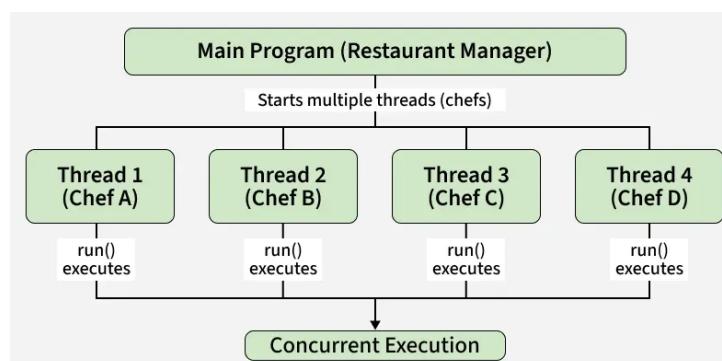
- **I/O Bound (Ligado à E/S):** Processos que passam a maior parte do tempo esperando por operações de Entrada/Saída (ex: Banco de Dados, Editores de texto). Eles tendem a liberar a CPU antes do **Time Slice** terminar.



3.7 Threads (Linhas de Execução)

O processo é a unidade de alocação de recursos. Criar um processo novo é "pesado" para o S.O. Para aumentar a concorrência sem o alto custo de processos completos, utiliza-se o conceito de **Threads** (Linhas de Execução).

- **Definição:** Uma Thread é um "pedaço" de um processo que pode ser executado concorrentemente.
- **Vantagem Principal:** Threads dentro do **mesmo processo** **compartilham a mesma memória** e recursos globais. Isso torna a comunicação entre elas muito mais rápida e eficiente.
- **TCB (Thread Control Block):** A thread possui uma estrutura de controle menor que o PCB, armazenando apenas o estado dos registradores e o contador de programa.



Regra Fundamental:

- O Processo é a unidade de **Alocação de Recursos** (dono da memória).
- A Thread é a unidade de **Escalonamento** (quem o S.O. põe na CPU).

3.8 Processos do Sistema

Embora a maioria dos processos que percebemos sejam aplicações de usuário (como um navegador ou editor de texto), o conceito de processo é fundamental na própria arquitetura interna do Sistema Operacional.

3.8.1. O S.O. como um Conjunto de Processos

Em sistemas modernos, muitos serviços que antigamente eram "embutidos" dentro do núcleo (Kernel) agora são executados como processos independentes. Esta abordagem traz vantagens críticas:

- Estabilidade: Ao retirar código do núcleo e colocá-lo em processos de serviço, o S.O. torna-se mais estável. Se um serviço (como um driver de rede) falhar, ele apenas encerra seu processo, mas o Kernel continua rodando.
- Segurança: Processos de serviço rodam com privilégios limitados em comparação ao núcleo, dificultando que uma falha de software comprometa todo o hardware.
- Arquitetura Microkernel: Esta filosofia é a base do modelo de Microkernel, que mantém no núcleo apenas o essencial (como comunicação e escalonamento), delegando o restante para processos servidores.

3.8.2. Exemplos Práticos: Daemons e Serviços

A nomenclatura e a implementação desses processos variam conforme o sistema:

- No Unix/Linux (Daemons): Os processos do sistema são conhecidos como daemons. Eles são criados automaticamente durante a inicialização e realizam tarefas administrativas em segundo plano (background).

Exemplos: cron (agendamento de tarefas), lpd (gerenciamento de impressoras) e swapper (gerência de memória).

- No Microsoft Windows (Serviços): São chamados de serviços do sistema e são fundamentais para o gerenciamento de sessões, segurança e rede.

Exemplos: LSASS (segurança local), Winlogon (logon de usuários) e o Service Control Manager.

3.8.3. Diferença entre Processos e Serviços

Embora rodem como processos, os Serviços possuem características únicas:

1. Independência de Usuário: Podem ser iniciados antes mesmo de qualquer usuário fazer login e continuam rodando após o logout.
2. Prioridade: Geralmente possuem prioridade elevada para garantir que funções essenciais (como a rede) não sofram atrasos causados por aplicativos comuns.

3.9 Sinais

Sinais são notificações assíncronas enviadas pelo S.O. ou por outro processo para interferir no fluxo de execução de um processo. Quando um processo recebe um sinal, ele para sua execução normal para tratar o evento.

- **Exemplo:** O comando Ctrl+C em um terminal envia um sinal de interrupção (SIGINT) para o processo, instruindo-o a parar imediatamente.

Resumo do Capítulo 3: Processos e Escalonamento

O foco deste capítulo foi o **Processo**, a unidade de alocação de recursos que representa um programa em execução. A chave para a gerência de processos é o **PCB (Process Control Block)**, uma estrutura de dados que armazena todo o estado do processo, incluindo os registradores da CPU e o **Program Counter**. O S.O. utiliza a **Mudança de Contexto (Context Switch)** para alternar rapidamente entre os processos, criando a ilusão de multitarefa; este é um custo administrativo (overhead) inevitável. O **Escalonador** é o módulo que decide qual processo passará para o estado de **Execução** (selecionado da fila de **Pronto**), podendo ser **Preemptivo** (o S.O. força a retirada da CPU) ou **Não-Preemptivo**. Finalmente, introduzimos as **Threads (Linhas de Execução)**, que são unidades de escalonamento mais leves que compartilham o espaço de memória do processo, proporcionando concorrência mais eficiente.

4 Capítulo 4: Programação Concorrente e Comunicação entre Processos

4.1 Introdução à Programação Concorrente

Até agora, estudamos processos como entidades isoladas. No entanto, em sistemas modernos, a eficiência depende da capacidade de realizar múltiplas tarefas simultaneamente. O termo **Programação Concorrente** (*Concurrent Programming*) refere-se à essa execução sobreposta de tarefas.

É importante fazer uma distinção sutil, porém valiosa:

- **Concorrência:** Lidar com muitas coisas ao mesmo tempo. É uma forma de estruturar o programa para que tarefas independentes progridam (ex: o sistema operacional alternando rapidamente entre o navegador e o reproduutor de música).
- **Paralelismo:** Fazer muitas coisas ao mesmo tempo. Requer hardware com múltiplos núcleos (*multicore*), onde instruções são literalmente executadas no mesmo instante físico.

Na prática, dizemos que um programa é concorrente quando, durante sua execução, ele origina diferentes fluxos de execução (threads ou subprocessos) que interagem entre si para alcançar um objetivo comum.

4.1.1. Especificação de Concorrência

Didaticamente, existem primitivas clássicas para especificar que partes de um código devem rodar em paralelo:

1. Fork / Wait (ou Join):

- **Fork:** Um processo principal "bifurca" sua execução, criando um processo filho para realizar uma tarefa específica (ex: Processo A chama fork para lançar o Processo B).
- **Wait:** O processo principal continua executando até chegar a um ponto onde ele *precisa* do resultado do filho. O comando wait faz o pai pausar e esperar o filho terminar.

2. Parbegin / Paren:

- É uma estrutura de bloco. Todos os comandos colocados entre PARBEGIN e PARENDE são executados concorrentemente. O

programa só continua para a próxima linha após o PARENDO quando todos os comandos do bloco tiverem terminado.

```
PROGRAM Expressao
VAR X, Temp1, Temp2, Temp3 : REAL;
BEGIN
PARBEGIN
    Temp1 := SQRT (1024);
    Temp2 := 35.4 * 0.23;
    Temp3 := 302 / 7;
PARENDO;
X := Temp1 + Temp2 - Temp3;
WRITELN ('x = ', X);
END
```

Para exemplificar o funcionamento dos comandos PARBEGIN e PARENDO em uma aplicação concorrente, o programa Expressao realiza o cálculo do valor da expressão a seguir: $X := \text{SQRT}(1024) + (35.4 * 0.23) - (302 / 7)$

Os comandos de atribuição situados entre PARBEGIN e PARENDO são executados concorrentemente. O cálculo final de X só pode ser realizado quando todas as variáveis dentro da estrutura tiverem sido calculadas

4.2 Comunicação entre Processos (IPC)

Processos concorrentes raramente trabalham isolados. Frequentemente, eles precisam trocar dados ou usar recursos comuns (como uma variável na memória, um arquivo de log ou uma impressora).

Quando dois processos compartilham recursos, surge a necessidade vital de **sincronização**. Se a comunicação não for rigidamente estruturada, o resultado da execução pode ser imprevisível e catastrófico.

4.2.1. Condição de Corrida (Race Condition)

A condição de corrida é o "pesadelo" da programação concorrente. Ela ocorre quando dois ou mais processos tentam ler e escrever em um dado compartilhado ao mesmo tempo, e o resultado final depende de qual processo "correu mais rápido" ou teve sorte no escalonamento da CPU.

Exemplo Clássico: A Conta Bancária

Imagine que Zé tem **\$1.000** no banco.

1. Zé tenta sacar **\$100**.
2. O pai de Zé tenta depositar **\$100** simultaneamente.

Se não houver controle, o cenário de erro seria:

- Processo do Zé lê o saldo: **\$1.000**.
- Processo do Pai lê o saldo: **\$1.000** (antes do Zé atualizar).
- Processo do Zé calcula $\$1.000 - \$100 = \$900$ e grava.
- Processo do Pai calcula $\$1.000 + \$100 = \$1.100$ e grava.
- **Resultado:** O saldo final vira **\$1.100**, e o saque de Zé foi "esquecido". O banco perdeu dinheiro.

4.3 Exclusão Mútua e Região Crítica

Para evitar a condição de corrida, utilizamos a técnica de **Exclusão Mútua**. A ideia é simples: se um processo está mexendo em um dado compartilhado, ninguém mais pode mexer até que ele termine.

4.3.1. Região Crítica (Critical Section)

A Região Crítica é o trecho específico do código onde ocorre o acesso ao recurso compartilhado.

- **A Regra de Ouro:** Dois processos nunca podem estar dentro de suas respectivas regiões críticas ao mesmo tempo se elas acessam o mesmo recurso.

O gerenciamento de concorrência se resume a proteger a entrada e a saída dessa Região Crítica.

4.4 Problemas Clássicos de Concorrência

Para entender e testar soluções de sincronização, a ciência da computação utiliza problemas teóricos clássicos:

4.4.1. O Problema do Produtor-Consumidor

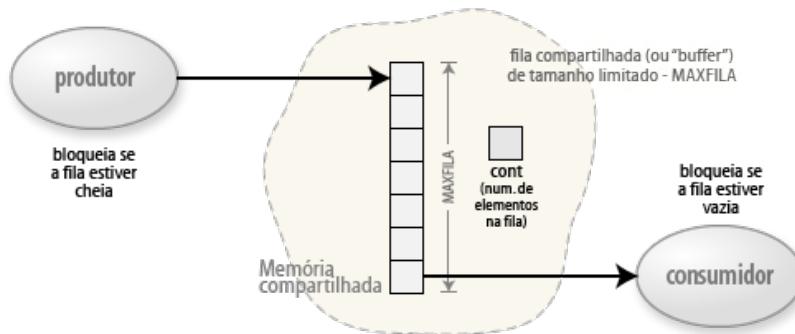
Imagine um buffer (uma fila de dados) com tamanho limitado.

- **Produtor:** Gera dados e os coloca no buffer.
- **Consumidor:** Retira dados do buffer para processar.

Os desafios:

1. **Buffer Cheio:** O Produtor não pode colocar novos dados se não houver espaço. Ele deve "dormir" (bloquear) até surgir uma vaga.
2. **Buffer Vazio:** O Consumidor não pode retirar dados se não houver nada lá. Ele deve "dormir" até que chegue um dado.

Solução: O uso de variáveis de controle (contadores) e sinais de bloqueio/desbloqueio para coordenar quem dorme e quem acorda.



4.4.2. O Jantar dos Filósofos (Dining Philosophers)

Proposto por Edsger Dijkstra, este problema ilustra a disputa por recursos limitados.

- **Cenário:** 5 filósofos sentados em uma mesa redonda. Entre cada par de pratos, há apenas 1 garfo (total de 5 garfos).
- **Regra:** Para comer, um filósofo precisa de **dois garfos** (o da esquerda e o da direita).
- **Ciclo:** O filósofo Pensa -> Sente Fome -> Tenta pegar garfos -> Come -> Devolve garfos -> Pensa.

Problemas Potenciais:

- **Deadlock (Impasse):** Se todos os filósofos sentirem fome ao mesmo tempo e pegarem o garfo da direita simultaneamente, ninguém conseguirá o garfo da esquerda. Todos ficarão segurando um garfo eternamente, esperando pelo outro. O sistema trava.
- **Starvation (Inanição):** Um filósofo pode nunca conseguir os dois garfos simultaneamente se seus vizinhos forem muito rápidos (comem e pensam rápido), fazendo com que ele morra de fome esperando sua vez.



4.5 Semáforos

Para resolver esses problemas de exclusão mútua e sincronização, Dijkstra (1965) propôs o conceito de **Semáforo**.

O Semáforo é uma variável inteira especial, usada para sinalizar status. Ela só pode ser manipulada por duas operações atômicas (que não podem ser interrompidas pela metade):

1. P (Proberen - Testar / Down / Wait):

- Tenta diminuir o valor do semáforo.
- Se o valor for maior que 0, decrementa e o processo entra na Região Crítica.
- Se o valor for 0 (recurso ocupado), o processo é **bloqueado** (dorme) até que o semáforo seja liberado.

2. V (Verhogen - Incrementar / Up / Signal):

- Incrementa o valor do semáforo (indicando que o recurso foi liberado).
- Se houver processos dormindo na fila de espera desse semáforo, um deles é acordado para assumir o recurso.

```
PROGRAM Semaforo_I;
  VARs Semaforo := 1;
  PROCEDURE Processo_A;
  BEGIN
    REPEAT
      DOWN(s);
      Regiao_Critica_A;
      UP(s);
    UNTIL False;
  END;
  PROCEDURE Processo_B;
  BEGIN
    REPEAT
      DOWN(s);
      Regiao_Critica_B;
      UP(s);
    UNTIL False;
  END;
  BEGIN
    PARBEGIN
      Processo_A;
      Processo_B;
    PARENDE;
  END.
```

Resumo do Capítulo 4: Programação Concorrente e Sincronização

Aprofundamos o estudo da execução simultânea, distinguindo **Concorrência** (intercalação de fluxos) de **Paralelismo** (execução real em múltiplos núcleos). O principal desafio da concorrência é a **Condição de Corrida (Race Condition)**, que ocorre quando o resultado depende da ordem imprevisível de acesso a um dado compartilhado, gerando inconsistência. A solução exige a **Exclusão Mútua da Região Crítica**. Para isso, usamos mecanismos de sincronização como **Semáforos** (variáveis inteiras controladas pelas operações atômicas P e V) e **Monitores** (abstrações de alto nível que garantem exclusão mútua automática). Por fim, analisamos o **Deadlock**, uma situação de bloqueio permanente que exige o cumprimento simultâneo das quatro condições: Exclusão Mútua, Posse e Espera, Não-Preempção e Espera Circular.

5 Capítulo 5: Gerência de Memória e Memória Virtual

5.1 Introdução à Gerência de Memória

A memória principal (RAM) é um dos recursos mais críticos e limitados de um sistema de computador. A função do **Gerenciamento de Memória** no Sistema Operacional é dupla:

1. **Alocação:** Decidir qual parte da memória está livre e alocar espaço para os processos que estão sendo carregados e executados.
2. **Proteção:** Garantir que um processo não acesse a área de memória de outro processo ou a área reservada ao próprio Sistema Operacional (evitando travamentos).

Se o S.O. não for eficiente em gerenciar a memória, o desempenho cai drasticamente e a multitarefa se torna impossível.

5.2 Primeiras Técnicas de Alocação de Memória

As primeiras abordagens de gerenciamento de memória eram mais simples, mas apresentavam grandes limitações na era da multiprogramação.

5.2.1. Alocação Contígua Simples

Utilizada nos primeiros sistemas operacionais monoprogramáveis (apenas um processo de usuário por vez).

- **Divisão:** A memória era dividida em duas áreas fixas: uma para o Sistema Operacional e a outra para o programa do usuário.
- **Limitação:** O processo de usuário não podia ser maior que a área disponível. Havia desperdício, pois um programa pequeno ocupava toda a área restante, impedindo outros de rodar.

5.2.2. Alocação Particionada Estática

Essa técnica surgiu para permitir a multiprogramação (vários processos rodando).

- **Partições:** A memória era dividida em pedaços de tamanho fixo (**partições**) já na inicialização do sistema.

- **Problema: Fragmentação Interna.** Se uma partição tiver 4MB e o programa do usuário precisar de apenas 2.5MB, os 1.5MB restantes dentro da partição são desperdiçados e não podem ser usados por outro programa. Para alterar o tamanho das partições, era necessário reiniciar o sistema.

5.2.3. Alocação Particionada Dinâmica

Buscava resolver o desperdício da alocação estática.

- **Partições Variáveis:** As partições são criadas dinamicamente, exatamente do tamanho do processo que está sendo carregado.
- **Problema: Fragmentação Externa.** Com o tempo, os processos (que saem e entram) criam pequenos "buracos" de memória livre espalhados. Embora o espaço *total* livre seja suficiente para um novo programa, esse espaço não é **contíguo** (junto). O S.O. precisava mover os processos (compactação) para juntar os buracos livres, o que era caro computacionalmente.

5.3 Swapping

Nos primórdios da multiprogramação, os processos ativos permaneciam na memória principal (RAM) durante todo o tempo, mesmo quando estavam no estado de bloqueado, aguardando por algum evento lento (como uma entrada do usuário ou leitura de disco). Essa abordagem limitava drasticamente o número de programas que podiam coexistir no sistema, já que a RAM é um recurso finito e caro.

5.3.1. O que é Swapping?

O Swapping é uma técnica de gerenciamento que permite ao Sistema Operacional transferir processos inteiros entre a memória principal e a memória secundária (disco). O objetivo principal é resolver o problema de novos processos que aguardam por espaço livre adequado na RAM para serem executados.

Com o swapping, os processos não precisam mais residir na memória o tempo todo. Isso permite que o sistema operacional execute mais programas do que a capacidade física da memória RAM permitiria isoladamente, usando o disco como uma extensão temporária.

5.3.2. O Ciclo de Funcionamento: Swapped-Out e Swapped-In

O mecanismo funciona através de um ciclo de movimentação de dados gerenciado pelo sistema operacional:

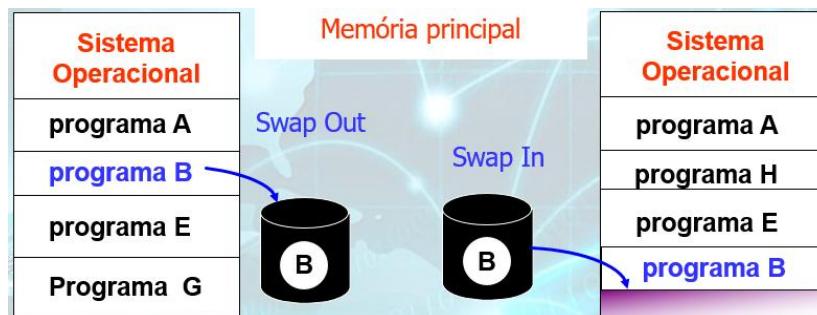
- **Swapped-Out (Saída):** Quando a RAM torna-se insuficiente para novos processos ou para garantir a execução dos mais prioritários, o sistema seleciona um processo residente (geralmente um que esteja inativo ou bloqueado há mais tempo) e o leva para uma área especial no disco chamada arquivo de swap.
- **Swapped-In (Entrada):** Quando o processo que foi enviado ao disco precisa voltar a ser executado, o sistema o traz de volta para a memória principal.

Transparência para o Usuário: Uma característica fundamental dessa técnica é que o processo retorna à memória e retoma sua execução exatamente de onde parou, sem "perceber" que foi temporariamente movido para o disco.

5.3.3. Vantagens e Limitações

A implementação do swapping traz benefícios claros, mas também impõe custos ao sistema:

- **Aumento do Throughput:** Permite que um número maior de processos seja executado simultaneamente, aumentando a produtividade geral do computador.
- **Custo de Desempenho (Overhead):** O acesso ao disco é significativamente mais lento do que o acesso à RAM. Portanto, a movimentação constante de processos cria um "custo administrativo" (overhead) que pode resultar em atrasos perceptíveis se for realizada em excesso.



5.4 Memória Virtual

A **Memória Virtual** é a técnica central dos sistemas operacionais modernos. Seu principal objetivo é permitir que os programas sejam executados mesmo que a soma de seus tamanhos seja maior que a memória RAM física disponível.

Conceito: A memória virtual faz com que o sistema pareça ter uma memória RAM muito maior do que realmente possui. Ela faz isso usando o disco rígido (HD ou SSD) como uma extensão da RAM.

5.4.1. Paginação (Paging)

A paginação é a técnica utilizada para implementar a memória virtual de forma eficiente, resolvendo o problema da fragmentação externa.

- **Páginas (Pages):** O programa do usuário (o código e os dados) é dividido em blocos de tamanho fixo, geralmente 4KB. Esses blocos são chamados de **Páginas**.
- **Frames (Quadros de Página):** A memória RAM física é dividida em blocos de mesmo tamanho das páginas. Esses blocos são chamados de **Frames** (ou Quadros de Página).

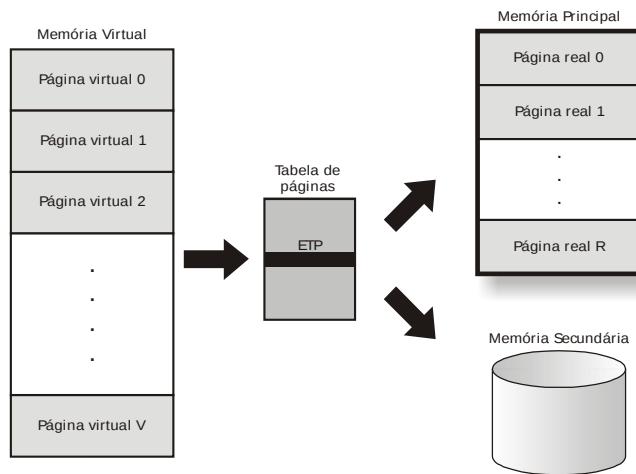
O Funcionamento: O S.O. não precisa carregar o programa inteiro na RAM de uma vez. Ele carrega apenas as Páginas que estão sendo usadas no momento em qualquer Frame livre da RAM. Isso permite que as páginas de um processo fiquem espalhadas por vários Frames não contíguos na RAM, acabando com a fragmentação externa.

5.4.2. Endereçamento e a Tabela de Páginas

Para que a CPU encontre o Frame correto na RAM, o S.O. deve manter um mapa de tradução:

- **Endereço Virtual:** O endereço que o programa (processo) "acha" que está usando. Cada processo tem seu próprio espaço de endereçamento virtual, geralmente começando em zero.
- **Endereço Real (Físico):** O endereço real do Frame na memória RAM.

Essa tradução é feita pela **Tabela de Páginas**. Cada processo tem sua própria Tabela de Páginas, que mapeia cada **Página Virtual** para seu respectivo **Frame Físico**.



5.4.3. TLB (Translation Lookaside Buffer)

A tradução de endereço virtual para real através da Tabela de Páginas consome tempo. Para acelerar esse processo, os processadores modernos usam um cache dedicado chamado **TLB (Translation Lookaside Buffer)**.

- **Função:** O TLB é uma pequena memória de alta velocidade (dentro do cache do processador) que armazena as últimas e mais frequentes traduções de endereços realizadas.
- **Economia de Tempo:** Antes de consultar a Tabela de Páginas na RAM (que é lento), o S.O. consulta o TLB. Se a tradução estiver lá (TLB Hit), ela é usada instantaneamente, economizando tempo e evitando o acesso à RAM.

5.5 Swapping e Gerenciamento de Ausência de Páginas

O cerne da Memória Virtual é o uso do disco como extensão da RAM, processo conhecido como **Swapping**.

5.4.1. Swapping

Swapping em memória virtual é o processo de mover uma ou mais páginas de um processo da memória física (RAM) para uma área de armazenamento secundário (disco), que é chamada de **Espaço de Swap** (ou *Swap File/Partition*).

5.4.2. Page Fault (Falta de Página)

O Page Fault é um evento normal e esperado na memória virtual. Ele ocorre quando a CPU tenta acessar uma instrução ou dado em uma **Página Virtual** que não está carregada no momento na memória RAM.

O Processo de Resolução (Page Fault Handling):

1. A CPU tenta acessar a Página X.
2. O S.O. verifica a Tabela de Páginas e vê que o bit de presença está desligado (Página X está no disco, não na RAM). **Page Fault!**
3. O S.O. encontra um Frame livre na RAM para carregar a Página X.
4. Se não houver Frame livre, o S.O. usa um **Algoritmo de Substituição de Páginas** (como FIFO, LRU, etc.) para escolher uma página "velha" (que não está sendo usada) para ser movida para o disco (swapped out) e liberar o Frame.
5. O S.O. carrega a Página X do disco para o Frame recém-liberado.
6. A Tabela de Páginas é atualizada.
7. A CPU recomeça a execução da instrução que causou a falta de página.

5.4.3. Thrashing (Batelada)

Thrashing é um estado de degradação do desempenho. Ocorre quando há uma **excessiva transferência de páginas** (Swapping) entre a memória principal e o disco.

- **Causa:** O sistema está sobrecarregado, com muito mais processos ativos do que a RAM pode suportar.
- **Sintomas:** O S.O. passa a maior parte do tempo gerenciando faltas de página e trocando dados com o disco, em vez de executar instruções úteis.
- **Solução:** Se o Thrashing for crônico (causado por falta de memória real), a única solução é aumentar a quantidade de memória RAM física do computador.

5.6 Compartilhamento de Memória

A memória virtual não apenas protege a memória, mas também permite o compartilhamento eficiente de recursos entre processos.

5.5.1. Reentrância

Refere-se ao código que pode ser executado por múltiplos processos simultaneamente sem causar problemas de concorrência.

- **Exemplo:** O código de um compilador ou de uma biblioteca comum do sistema (DLLs no Windows, bibliotecas compartilhadas no Linux).
- **Mecanismo:** Vários processos podem ter suas Tabelas de Páginas apontando para os mesmos Frames Físicos na RAM que contêm o código reentrante. Isso economiza RAM, pois o código da biblioteca só precisa ser carregado uma vez.

5.5.2. Compartilhamento de Dados

Dois ou mais processos também podem compartilhar dados (variáveis, estruturas). Para isso, basta que suas respectivas Tabelas de Páginas tenham entradas diferentes, mas que ambas apontem para o **mesmo Frame Físico** na memória.

- **Uso:** Essencial para a Comunicação Interprocessos (IPC) rápida e eficiente.

Resumo do Capítulo 5: Gerência de Memória e Memória Virtual

Iniciamos o estudo da gerência de memória, revisando os problemas das alocações contíguas: **Fragmentação Interna** (desperdício dentro do bloco) e **Fragmentação Externa** (espaço livre disperso). A solução moderna é a **Memória Virtual**, que separa o *Endereço Lógico* (virtual) do *Endereço Físico* (RAM) e utiliza a **Paginação** para carregar **Páginas** virtuais em **Frames** não contíguos na RAM. A tradução do endereço é feita pela **MMU** usando a **Tabela de Páginas (TP)** de cada processo e acelerada pelo **TLB (Translation Lookaside Buffer)**, um cache de traduções ultrarrápido na CPU. Vimos que um acesso a uma página não presente na RAM gera uma **Page Fault**, que dispara o **Swapping** (troca com o disco) e, se a carga de trabalho for excessiva, pode levar ao **Thrashing**, onde o sistema gasta mais tempo trocando páginas do que processando.

6 Glossário

Termo	Definição
Abstração	Função do S.O. de fornecer uma visão simplificada e de alto nível do hardware para o usuário e o programador, escondendo a complexidade física.
Chamada de Sistema (System Call)	A interface programática usada por um aplicativo de usuário para solicitar um serviço ao Kernel (Ex: acesso ao disco, criação de processo).
Concorrência	A execução intercalada de dois ou mais fluxos de execução (processos ou <i>threads</i>), dando a ilusão de simultaneidade (geralmente em uma única CPU).
Condição de Corrida (Race Condition)	Ocorre quando o resultado da execução depende da ordem imprevisível com que processos ou <i>threads</i> acessam e modificam um dado compartilhado, podendo levar à inconsistência.
Deadlock (Impasse)	Situação em que dois ou mais processos estão bloqueados permanentemente, esperando por recursos que estão sendo mantidos um pelo outro.
Escalonador	Módulo do S.O. responsável por decidir qual processo ou <i>thread</i> na fila de Pronto será alocado à CPU para ser executado.
Exclusão Mútua	Mecanismo que garante que, em um dado momento, apenas uma única <i>thread</i> possa estar executando o código dentro da Região Crítica .
Falta de Página (Page Fault)	Interrupção gerada quando um processo tenta acessar uma página de memória virtual que não está carregada na Memória RAM física no momento.
Fragmentação Externa	Ocorre quando o espaço total livre na memória é suficiente para carregar um programa, mas está dividido em pequenos blocos não contíguos (problema resolvido pela Paginação).
Fragmentação Interna	Ocorre quando um bloco de memória alocado é maior do que o tamanho do dado/programa, gerando desperdício de espaço dentro do bloco.
Frame (Quadro)	Uma unidade de tamanho fixo da Memória RAM física, utilizada para armazenar uma Página de memória virtual.
Gerente de Recursos	O papel principal do S.O. de controlar, alocar, proteger e arbitrar o uso de recursos de hardware (CPU, memória, I/O).
Kernel	O núcleo essencial do S.O., um código privilegiado que reside permanentemente na memória e é o responsável direto pela gerência de recursos.
LRU (Least Recently Used)	Algoritmo de substituição de páginas que remove da RAM a página que foi usada há mais tempo, baseado no princípio de localidade.
Memória Virtual	Técnica que separa o espaço de endereçamento lógico (vista pelo programa) do espaço de endereçamento físico (RAM real), permitindo que programas maiores que a RAM sejam executados.
Microkernel	Estrutura de Kernel onde apenas as funções essenciais rodam em Modo Privilegiado; outras funções do S.O. rodam como servidores em Modo Usuário (foco na estabilidade).

MMU (Memory Management Unit)	Hardware dedicado na CPU responsável pela tradução, em tempo real, dos endereços virtuais para endereços físicos.
Modo Kernel (Privilegiado)	O modo de operação da CPU onde o código do S.O. é executado e possui acesso total e irrestrito a todas as instruções e ao hardware.
Modo Usuário	O modo de operação da CPU onde as aplicações rodam, com restrições de acesso a instruções privilegiadas.
Monolítico	Estrutura de Kernel onde todos os serviços do S.O. (drivers, gerência de processos e memória) estão compilados em um único programa em Modo Kernel (foco na velocidade).
Mudança de Contexto (Context Switch)	A operação pela qual o S.O. salva o estado do processo atual (no PCB) e carrega o estado de outro processo (do PCB dele) para a CPU, permitindo a multitarefa.
Paginação	O método primário de Memória Virtual que divide o espaço de endereçamento em Páginas e a RAM em Frames, eliminando a Fragmentação Externa.
Paralelismo	A execução verdadeiramente simultânea de múltiplos fluxos de execução em múltiplas CPUs ou núcleos.
PCB (Process Control Block)	Bloco de Controle de Processo. Estrutura de dados que armazena todas as informações de estado de um processo (registradores, PC, prioridade, etc.).
Processo	Uma instância em execução de um programa. A unidade de alocação de recursos do S.O.
Região Crítica	O segmento de código onde um recurso compartilhado é acessado ou modificado. Deve ser protegido pela Exclusão Mútua.
Semáforo	Variável inteira usada para sincronização, manipulada por operações atômicas P (decrementa/bloqueia) e V (incrementa/acorda).
Swapping	O processo de mover uma página de memória do disco (área de swap) para a RAM, ou vice-versa, geralmente em resposta a uma Page Fault .
Tabela de Páginas (TP)	O mapa por processo usado pela MMU para traduzir o endereço virtual (Página) para o endereço físico (Frame).
Thrashing	Sobrecarga do sistema causada pela taxa excessivamente alta de Page Faults , onde o S.O. gasta a maior parte do tempo realizando Swapping .
Thread (Linha de Execução)	Unidade de execução leve que compartilha o espaço de endereçamento do processo pai. É a unidade de Escalonamento do S.O.
TLB (Translation Lookaside Buffer)	Um cache de alta velocidade que armazena as traduções de endereço mais usadas, evitando a necessidade de consultar a Tabela de Páginas na RAM a cada acesso.

7 CONSIDERAÇÕES FINAIS

Ao longo deste curso de Sistemas Operacionais, exploramos diversas facetas práticas e teóricas que compõem o funcionamento dos sistemas que estão no cerne de toda a computação moderna. Entre os aspectos abordados, a teoria de sistemas operacionais desempenha um papel crucial, fornecendo os fundamentos necessários para compreender não apenas como os sistemas funcionam, mas também por que funcionam dessa maneira.

Estudar a teoria de sistemas operacionais não se limita a memorizar conceitos ou algoritmos. Ela proporciona uma compreensão profunda dos princípios fundamentais que orientam o design, a implementação e a evolução dos sistemas que controlam recursos de hardware e facilitam a execução de programas. A teoria nos permite explorar questões complexas, como o gerenciamento de processos e threads, a alocação de memória, o controle de dispositivos e a segurança do sistema, entre outros aspectos críticos.

Além disso, compreender a teoria de sistemas operacionais é essencial para acompanhar as rápidas mudanças e inovações na tecnologia da informação. À medida que novos dispositivos, plataformas e paradigmas de computação emergem, os princípios teóricos nos fornecem um arcabouço sólido para avaliar e adaptar essas novidades ao ambiente computacional.

Ao dominar a teoria de sistemas operacionais, vocês não apenas se tornam capazes de resolver problemas complexos de forma eficiente, mas também adquirem uma base sólida para explorar novas áreas de pesquisa e desenvolvimento dentro da computação. Mais do que nunca, em um mundo cada vez mais dependente da tecnologia, o conhecimento teórico se revela um diferencial crucial para os profissionais da área.

Portanto, ao concluir este curso, encorajo-os a continuar explorando e aprofundando seus conhecimentos na teoria de sistemas operacionais. Este é um investimento que não apenas fortalecerá suas habilidades técnicas, mas também ampliará suas perspectivas sobre como a tecnologia pode ser aplicada para resolver desafios futuros.

Desejo a todos sucesso em suas jornadas acadêmicas e profissionais!

Professor Marco Antonio Alves Pereira, MSc.

8 BIBLIOGRAFIA CONSULTADA

Deitel, Harvey M.; Deitel, Paul J.. Sistemas Operacionais. 3^a ed. São Paulo: Pearson Prentice Hall, 2011.

Machado, Francis B.; Maia, Luiz Paulo. Arquitetura de Sistemas Operacionais. 5^a ed. São Paulo: LTC, 2013.

Peter Baerwolf. Sistemas Operacionais Modernos. 3^a ed. Porto Alegre: Bookman, 2015.

Silberschatz, Abraham; Galvin, Peter B.; Gagne, Greg. Fundamentos de Sistemas Operacionais. 9^a ed. Rio de Janeiro: Elsevier, 2018.

Stallings, William. Sistemas Operacionais. 9^a ed. São Paulo: Pearson Prentice Hall, 2014.

Tanenbaum, Andrew S.. Sistemas Operacionais Modernos. 5^a ed. São Paulo: Pearson Prentice Hall, 2016.