SISTEMAS OPERACIONAIS I

Prof. Me. Fábio Nascimento

2025

Sumário

1. INTRODUÇÃO AOS SISTEMAS OPERACIONAIS

1.1 O que é um Sistema Operacional

Um **Sistema Operacional (SO)** é um **software fundamental** que atua como intermediário entre o usuário e o hardware do computador. Ele é responsável por gerenciar os recursos do sistema – como o processador, memória, dispositivos de entrada/saída e armazenamento – e oferecer uma interface que permita a execução de programas de forma eficiente e segura.

Sem um sistema operacional, o usuário teria que interagir diretamente com o hardware por meio de instruções de baixo nível – algo complexo e impraticável para a maioria das pessoas.

1.2 Objetivos de um Sistema Operacional

Os principais objetivos de um sistema operacional são:

- **Gerenciar os recursos do hardware** de forma eficiente (CPU, memória, dispositivos, etc.);
- **Fornecer uma interface para o usuário**, seja ela gráfica (GUI) ou por linha de comando (CLI);
- **Executar e controlar programas de aplicação**;
- **Prover segurança, isolamento e proteção** entre processos e usuários;
- **Multiplexar recursos entre múltiplas aplicações e usuários**;
- **Oferecer serviços ao sistema e aos programas**, como sistemas de arquivos, bibliotecas, drivers e protocolos de rede.

1.3 Funcionalidades Essenciais

a) Multiprogramação

Permite que vários programas estejam na memória ao mesmo tempo, aumentando a utilização da CPU.

b) Multitarefa

Habilidade de alternar rapidamente entre tarefas, dando a ilusão de execução simultânea.

c) Tempo Compartilhado (Time Sharing)

Distribui o tempo do processador entre múltiplos usuários de forma justa.

d) Gerenciamento de Segurança

Controla o acesso a arquivos, dados e dispositivos por meio de autenticação, criptografia e permissões.

e) Suporte à Rede

Permite comunicação com outros sistemas via protocolos de rede, essencial em ambientes modernos e conectados.

1.4 Importância do Sistema Operacional

- O sistema operacional é indispensável para o funcionamento de qualquer dispositivo computacional moderno. Ele:
- Garante o funcionamento adequado do hardware;
- Permite a execução de softwares;
- Cria um ambiente estável e seguro para o usuário e para os desenvolvedores;
- Otimiza o uso dos recursos computacionais;
- Suporta tecnologias modernas como virtualização, computação em nuvem e dispositivos móveis.

1.5 Histórico e Evolução

- Anos 1980: MS-DOS
- Anos 1990: Windows 3.x, 95, 98, ME
- Anos 2000: Windows 2000, XP, Vista
- Anos 2010: Windows 7, 8, 10
- Atualidade: Windows 11

1.6 Tipos de Sistemas Operacionais

- **Monolítico:** Todo o sistema operacional funciona como um único programa (ex: Windows NT).
- **Em Camadas:** Dividido em camadas hierárquicas.
- **Microkernel:** Parte mínima no kernel, demais funções em modo usuário (não comum no Windows).
- **Híbrido:** Estrutura do Windows NT: núcleo monolítico com módulos.
- **Distribuído:** Conjunto de computadores que colaboram (ex: Windows Server com cluster).
- **Embarcado:** Windows IoT.

1.7 Componentes de um Sistema Operacional

- **Kernel (NTOSKRNL):** Parte central que gerencia recursos
- **API do Windows:** Interface para programas (Win32)
- **Shell:** Interface gráfica (explorer.exe) ou textual (cmd, PowerShell)
- **Gerenciadores de recurso:** CPU, memória, disco, etc.

2. HISTÓRIA DOS SISTEMAS OPERACIONAIS

A evolução dos sistemas operacionais está diretamente ligada ao avanço do hardware e às necessidades computacionais ao longo das décadas. Desde as primeiras máquinas de processamento até os sistemas distribuídos e móveis atuais, os sistemas operacionais passaram por várias gerações e paradigmas.

2.1 Primeira Geração (década de 1940 a início dos anos 1950): Sem Sistema Operacional

Características:

- Computadores como o **ENIAC** e **EDVAC**.
- Programação feita diretamente no **hardware**, com o uso de **painéis com fios**, **interruptores**, e, posteriormente, **linguagem de máquina**.
- **Sem sistema operacional**: os programadores tinham acesso direto à máquina.

Funcionamento:

- Um único programa era executado por vez (monotarefa).
- Toda operação era manual: carregamento, execução e armazenamento dos dados.

Desafios:

- Baixa produtividade.
- Alto custo operacional.
- Erros frequentes devido à complexidade da operação manual.

2.2 Segunda Geração (meados dos anos 1950 a início dos anos 1960): Surgimento dos Sistemas em Lote (Batch Systems)

Características:

- Uso de **cartões perfurados** para entrada de dados e instruções.
- Introdução de **sistemas operacionais rudimentares** para automatizar o processamento em **lote**.

Conceito-chave:

- **Processamento em lote (batch)**: vários programas eram carregados sequencialmente e processados sem intervenção do operador entre eles.
- Exemplo: **IBM 1401** com o sistema **IBM 7090**.

Avanços:

- Introdução dos **monitores residentes** (primeiros programas que permaneciam na memória para controlar a execução dos lotes).
- Melhor aproveitamento da CPU e redução do tempo ocioso.

2.3 Terceira Geração (década de 1960 a início dos anos 1970): Multiprogramação e Tempo Compartilhado

Características:

- Introdução de **mainframes** e **terminais interativos**.
- **Sistemas multiprogramáveis**: vários programas na memória competindo pela CPU.
- **Tempo compartilhado (Time-sharing)**: usuários interagindo com o sistema simultaneamente, com divisão do tempo da CPU.

Exemplos de sistemas:

- **MULTICS (Multiplexed Information and Computing Service)**: precursor de muitos conceitos modernos.

- **UNIX (1969)**: criado por Ken Thompson e Dennis Ritchie nos laboratórios Bell; altamente influente.

Avanços:

- Suporte a múltiplos usuários.
- Gerenciamento de processos, memória e arquivos mais sofisticado.
- Introdução da **linguagem C**, que permitiu que o UNIX fosse portado entre diferentes máquinas.

2.4 Quarta Geração (anos 1970 e 1980): Sistemas Pessoais e Interfaces Gráficas

Características:

- Surgimento dos **microcomputadores (PCs)**.
- Sistemas operacionais mais acessíveis para o **usuário final**.
- Evolução de **interfaces gráficas (GUI)**, tornando os computadores mais fáceis de usar.

Exemplos de sistemas:

- **CP/M (Control Program for Microcomputers)**: usado em microcomputadores com processadores Intel 8080.
- **MS-DOS (Microsoft Disk Operating System)**: dominante na década de 1980.
- **Apple Lisa (1983)** e **Macintosh (1984)**: pioneiros no uso de GUI.
- **Windows 1.0 (1985)**: início da interface gráfica no ambiente Microsoft.

Avanços:

- Interface gráfica (GUI), janelas, mouse.
- Computação mais pessoal e interativa.
- Sistemas operacionais passaram a ter suporte para discos rígidos, drivers, e software de aplicação.

2.5 Quinta Geração (década de 1990 até meados dos anos 2000): Redes, Internet e Sistemas Distribuídos

Características:

- Consolidação da **Internet** e das **redes de computadores**.
- Necessidade de **sistemas operacionais em rede**.
- Consolidação dos sistemas **multiusuário e multitarefa** em desktops.

Exemplos de sistemas:

- **Windows 95, 98, 2000, XP**: evolução da GUI e integração com redes.
- **Linux (1991)**: criado por Linus Torvalds, baseado em Unix, com código aberto e grande flexibilidade.
- **Mac OS X (2001)**: baseado em UNIX, com forte apelo gráfico.
- **Windows Server, Red Hat, Debian**: sistemas robustos para servidores.

Avanços:

- Sistemas operacionais com **pilhas de rede** embutidas.
- Suporte a **multimídia, USB, plug-and-play**, e protocolos como TCP/IP.
- **Virtualização** começou a ser explorada.
-

2.6 Sexta Geração (anos 2000 até hoje): Mobilidade, Virtualização e Computação em Nuvem

Características:

- Explosão dos **dispositivos móveis** e **sistemas operacionais móveis**.
- Popularização da **virtualização** e da **computação em nuvem**.
- Evolução da **segurança da informação** e **multiplataforma**.

Exemplos de sistemas:

- **Android (2008)**: baseado em Linux, dominante no mercado móvel.
- **iOS (2007)**: sistema móvel da Apple.

- **Windows 10/11**: integração com nuvem, assistentes virtuais, e atualizações contínuas.
- **Distribuições Linux modernas** (Ubuntu, Fedora, Arch Linux): para desktops, servidores e IoT.
- **Hypervisores e sistemas em nuvem**: VMware ESXi, Proxmox, Microsoft Azure, Amazon EC2.

Avanços:

- Computação em nuvem (SaaS, PaaS, IaaS).
- Virtualização de servidores, desktops e redes.
- Contêineres (Docker), orquestração (Kubernetes).
- Suporte a múltiplas arquiteturas (ARM, x86, RISC-V).
- Inteligência artificial **e** aprendizado de máquina integrados aos sistemas operacionais.

2.7 Família de Sistemas Operacionais Windows

A família Windows**,** desenvolvida pela Microsoft**,** é uma das mais populares do mundo, especialmente no ambiente de computadores pessoais (PCs). Ela evoluiu ao longo das décadas desde um ambiente gráfico sobre o MS-DOS até sistemas operacionais modernos, robustos, com suporte a redes, segurança avançada, virtualização e computação em nuvem.

2.7.1 Origem e Evolução Inicial

MS-DOS (1981) – o ponto de partida

Antes do Windows, a Microsoft lançou o MS-DOS (Microsoft Disk Operating System)**,** um sistema operacional baseado **e**m linha de comando (CLI), usado em PCs IBM e compatíveis. Ele não tinha interface gráfica, multitarefa ou suporte a redes.

- Windows 1.0 a 3.11 (1985–1993) – as primeiras GUIs
- Windows 1.0 (1985)**:** ambiente gráfico sobre MS-DOS, com interface baseada em janelas sobrepostas.
- Windows 3.1/3.11**:** popularizou o uso gráfico; incluía o Gerenciador de Programas, suporte a fontes TrueType e melhor desempenho gráfico.

Essas versões ainda dependiam do MS-DOS para funcionar.

2.7.2 Linha Windows 9x: DOS + GUI (1995–2000)

Versão	Ano	Destaques
Windows 95	1995	GUI moderna (menu Iniciar), suporte a plug and play, multitarefa cooperativa.
Windows 98	1998	Integração com a Internet, suporte a dispositivos USB.
Windows ME (Millennium Edition)	2000	Foco em mídia e usuários domésticos, mas com instabilidade frequente.

Essas versões ainda usavam o MS-DOS como base, com multitarefa limitada e pouca estabilidade.

2.7.3 Linha Windows NT (New Technology): Arquitetura Moderna

A família Windows NT surgiu para oferecer uma plataforma mais robusta, segura e escalável. Ao contrário da linha 9x, ela não depende do MS-DOS**.**

Características da arquitetura NT:

- Kernel híbrido (monolítico + microkernel)
- Gerenciamento de processos e memória avançado
- Suporte a múltiplos usuários e multitarefa preemptiva
- Sistema de arquivos NTFS
- Camada de abstração de hardware (HAL)

Versão	Ano	Uso principal
Windows NT 3.1 a 4.0	1993–1996	Estações de trabalho e servidores
Windows 2000	2000	Ambiente corporativo, domínio, Active Directory
Windows XP	2001	Unificou as linhas doméstica e empresarial, GUI aprimorada
Windows Vista	2007	Foco em segurança (UAC), Aero, mas desempenho criticado
Windows 7	2009	Estável, rápido, popular em ambientes corporativos
Windows 8 / 8.1	2012–2013	Interface "Metro", foco em touch, mas rejeitado por muitos usuários
Windows 10	2015	Plataforma unificada para desktops, tablets e IoT
Windows 11	2021	Requisitos mais modernos (TPM 2.0), novo visual, foco em produtividade

2.7.4 Windows Server: Família Corporativa

A linha Windows Server é voltada para ambientes corporativos, com foco em:

- Gerenciamento de rede, domínios e permissões (Active Directory)
- Serviços de arquivos, impressão, DHCP, DNS, VPN
- Virtualização (Hyper-V)
- Alta disponibilidade e clustering

Versão	Ano	Destaques
Windows Server 2003	2003	Suporte a .NET, AD aprimorado
Windows Server 2008	2008	Introdução do Hyper-V
Windows Server 2012	2012	Interface Metro, foco em cloud

Windows Server 2016	2016	Containers, Nano Server
Windows Server 2019	2018	Melhorias em segurança e Kubernetes
Windows Server 2022	2021	Performance, segurança e Azure integration

2.7.5 Principais Características da Família Windows

Interface Gráfica (GUI)

- Ponto forte da família Windows desde suas origens.
- Evoluiu do Gerenciador de Programas para o menu Iniciar**,** Aero Glass e, mais recentemente, o Fluent Design**.**

Compatibilidade com Aplicações

- Um dos maiores trunfos da Microsoft.
- Suporte a bilhões de aplicativos legados e modernos.

Sistema de Arquivos

- FAT16/FAT32**:** usado até o Windows ME.
- NTFS: desde o Windows 2000, com recursos como criptografia, compressão e permissões avançadas.

Segurança

- Autenticação via contas locais, domínios e biometria.
- Windows Defender, BitLocker, UAC, Firewall**.**
- Integração com soluções corporativas como Active Directory, Group Policy.

Multitarefa Preemptiva

- Cada processo tem controle isolado da CPU.
- Gerenciamento sofisticado de memória virtual e threads.
- Atualizações Automáticas
- Windows Update: fornece patches de segurança e melhorias constantes.

2.8 Família de Sistemas Operacionais Linux

O Linux é um sistema operacional baseado em Unix, de código aberto (open source) e software livre, criado inicialmente por Linus Torvalds em 1991. Ao contrário do Windows, não é um produto único de uma empresa, mas sim um núcleo (kernel) que serve de base para centenas de distribuições adaptadas a diferentes necessidades — de servidores corporativos até smartphones e sistemas embarcados.

2.8.1 Origem e Evolução

A gênese (1991)

- Linus Torvalds, estudante na Universidade de Helsinque (Finlândia), desenvolveu um **kernel** inspirado no **MINIX** (um pequeno sistema Unix usado para ensino).
- O código foi disponibilizado gratuitamente na internet, permitindo que programadores do mundo todo contribuíssem.
- Primeira mensagem histórica no grupo *comp.os.minix* em 25 de agosto de 1991:
  "I'm doing a (free) operating system (just a hobby, won't be big and professional like GNU)..."

Parceria com o Projeto GNU

- O projeto GNU, iniciado por Richard Stallman em 1983, já havia criado várias ferramentas de sistema (compiladores, shells, bibliotecas).
- Ao unir o kernel Linux com as ferramentas GNU, nasceu um sistema Unix-like completo e livre: o GNU/Linux.

2.8.2 Arquitetura e Filosofia

Características fundamentais:

- Código aberto sob a licença GPL.
- Kernel monolítico modular: funções centrais no kernel, mas com suporte a módulos carregáveis.
- Multiusuário e multitarefa preemptiva.
- Segurança robusta baseada em permissões e separação de processos.
- Portabilidade: roda em arquiteturas x86, ARM, RISC-V, PowerPC e outras.
- Flexibilidade: adaptável desde supercomputadores até relógios inteligentes.

2.8.3 Linha do Tempo e Marcos Importantes

Ano	Evento
1991	Linus Torvalds lança a primeira versão do kernel Linux (0.01).
1993	Surge o Slackware, uma das primeiras distribuições.
1994	Lançado o Linux 1.0 (suporte TCP/IP básico).
1998	Grandes empresas (IBM, Oracle) começam a investir em Linux.
2004	Surge o Ubuntu, popularizando o Linux no desktop.
2008	Google lança o Android, baseado no kernel Linux.
2010+	Linux domina servidores, supercomputadores e dispositivos móveis.
Hoje	Kernel estável, suporte a milhares de dispositivos e grande comunidade ativa.

2.8.4 Principais Distribuições (Distros)

Como Linux é um kernel, cada distribuição acrescenta pacotes, gerenciadores, interfaces e ferramentas específicas.

Distribuição	Origem	Público-alvo	Características
Debian	1993	Servidores e desktop	Estável, grande repositório de pacotes.
Ubuntu	2004	Desktop, servidores e nuvem	Amigável, suporte da Canonical.
Fedora	2003	Desenvolvedores e entusiastas	Recursos recentes, patrocínio Red Hat.
Red Hat Enterprise Linux (RHEL)	2000	Corporativo	Suporte profissional, estabilidade.
CentOS / AlmaLinux / Rocky Linux	2004/2021	Servidores	Alternativas gratuitas ao RHEL.
Arch Linux	2002	Avançado	Rolling release, totalmente customizável.
Kali Linux	2013	Segurança e pentest	Ferramentas pré-instaladas para auditoria.
Android	2008	Dispositivos móveis	Base Linux adaptada para touch e sensores.

2.8.5 Segurança e Estabilidade

- Permissões de arquivos (usuário, grupo e outros).
- Controle de acesso obrigatório (MAC): SELinux (Red Hat) e AppArmor (Ubuntu).
- Isolamento de processos: cada processo opera no seu próprio espaço de memória.
- Comunidade ativa: vulnerabilidades corrigidas rapidamente.

2.8.6 Linux no Mercado

Servidores

- Mais de 70% dos servidores web usam Linux (Apache, Nginx).
- Popular em data centers, cloud computing (AWS, Azure, Google Cloud).

Supercomputadores

- Segundo o ranking TOP500, 100% dos supercomputadores rodam Linux.

Dispositivos móveis

- Android domina o mercado de smartphones, todos usando kernel Linux.

IoT e sistemas embarcados

- Linux embarcado está em roteadores, Smart TVs, câmeras e automóveis.

2.8.7 Comparação com Windows (Resumo)

Critério	Linux	Windows
Licença	GPL (livre)	Proprietária
Custo	Gratuito (na maioria)	Pago
Código-fonte	Aberto	Fechado
Segurança	Alta	Moderada
Interface	Variável (KDE, GNOME, XFCE...)	Padrão da Microsoft
Desempenho	Excelente em hardware limitado	Pode ser pesado
Mercado	Servidores, supercomputadores, dispositivos móveis	PCs, notebooks e alguns servidores

2.9 Família de Sistemas Operacionais da Apple

A Apple desenvolveu, ao longo de sua história, diversos sistemas operacionais, inicialmente para seus computadores pessoais e, posteriormente, para dispositivos móveis e outros eletrônicos. A família de sistemas operacionais da Apple é reconhecida por design refinado, integração entre hardware e software, segurança e usabilidade.

2.9.1 Primeira Fase – Computadores Apple II e Macintosh Clássico

Apple DOS (1978–1983)

- Primeiro sistema operacional da Apple, criado para a série Apple II.
- Baseado em linha de comando (CLI).
- Suporte a disquetes e gerenciamento básico de arquivos.

ProDOS (1983–1993)

- Sucessor do Apple DOS, mais rápido e com suporte a discos rígidos.
- Estrutura hierárquica de diretórios.

Mac OS Clássico (1984–2001)

- Lançado com o Macintosh original em 1984.
- Pioneiro em Interface Gráfica (GUI) para o usuário doméstico.
- Uso intenso do mouse e janelas sobrepostas.
- Limitações técnicas: ausência de multitarefa preemptiva e segurança de processos.

2.9.2 Segunda Fase – A transição para o Mac OS X (Base UNIX)

Contexto

Nos anos 90, o Mac OS clássico enfrentava limitações técnicas graves. A Apple adquiriu a empresa NeXT (fundada por Steve Jobs) em 1996, trazendo o NeXTSTEP, um sistema baseado em UNIX com interface avançada.

Mac OS X (2001–2012)

- Baseado no Darwin (núcleo híbrido XNU, que combina Mach e BSD).
- Suporte a multitarefa preemptiva, memória protegida e estabilidade de nível UNIX.
- Interface gráfica Aqua com efeitos visuais avançados.
- Nomeado com felinos (Cheetah, Puma, Jaguar, Panther, Tiger, Leopard, Snow Leopard, Lion, Mountain Lion).

2.9.3 Terceira Fase – macOS e Integração com Dispositivos Móveis

macOS (2016–presente)

- Mudança de nome para alinhar com iOS e watchOS.
- Integração profunda com iCloud e dispositivos Apple.
- Suporte a processadores Apple Silicon (M1, M2, M3) a partir de 2020.
- Versões nomeadas com lugares da Califórnia (Mavericks, Yosemite, Catalina, Big Sur, Monterey, Ventura, Sonoma).

2.9.4 Expansão para Dispositivos Móveis e Outros

A partir do sucesso do iPod e do iPhone, a Apple expandiu seu ecossistema de sistemas operacionais:

iOS (2007–presente)

- Baseado no kernel Darwin, como o macOS.
- Criado para o iPhone e depois adaptado ao iPad (iPadOS).
- Interface multitouch, App Store, segurança avançada (sandbox, criptografia).
- Base do ecossistema móvel Apple.

iPadOS (2019–presente)

- Versão do iOS adaptada para iPads, com multitarefa mais avançada, suporte a teclado e mouse.

watchOS (2015–presente)

- Sistema para Apple Watch.
- Foco em saúde, fitness e notificações rápidas.
- Integração com iPhone e ecossistema Apple.

tvOS (2015–presente)

- Sistema da Apple TV.
- Baseado no iOS, adaptado para controle remoto e consumo de mídia.

2.9.5 Linha do Tempo Resumida

Ano	Sistema	Plataforma
1978	Apple DOS	Apple II
1983	ProDOS	Apple II
1984	Mac OS Clássico	Macintosh
2001	Mac OS X	Macintosh
2007	iOS	iPhone
2015	watchOS	Apple Watch
2015	tvOS	Apple TV
2016	macOS	Mac
2019	iPadOS	iPad

2.10 Família de Sistemas Operacionais Android

O Android é um sistema operacional baseado em Linux desenvolvido inicialmente pela Android Inc. e adquirido pelo Google em 2005. É hoje o SO móvel mais usado do mundo, presente em bilhões de smartphones, tablets, TVs, carros e dispositivos vestíveis.
Sua popularidade vem da flexibilidade, código aberto, ampla base de fabricantes e enorme ecossistema de aplicativos.

2.10.1 Origem e Primeiros Passos

Criação (2003–2005)

- Desenvolvido pela Android Inc., fundada por Andy Rubin, Rich Miner, Nick Sears e Chris White.
- Inicialmente pensado para câmeras digitais inteligentes, mas rapidamente adaptado para telefones móveis.
- Baseado no kernel Linux para fornecer estabilidade e compatibilidade com hardware.

Aquisição pelo Google (2005)

- Google compra a Android Inc., garantindo recursos e equipe para o desenvolvimento.
- Visão: criar um sistema aberto e gratuito para fabricantes, competindo com Symbian, Windows Mobile e, futuramente, iOS.

2.10.2 Primeiras Versões (2008–2010)

Android 1.0 (2008)

- Lançado com o HTC Dream (T-Mobile G1).
- Recursos básicos: navegador, Gmail, Maps, YouTube, Android Market (primeira loja de apps).
- Interface simples, focada em funcionalidades.

Android 1.5 Cupcake / 1.6 Donut / 2.0 Eclair

- Introdução do teclado virtual.
- Suporte a múltiplos tamanhos de tela.
- Melhorias no GPS e integração com redes sociais.

2.10.3 Evolução e Popularização (2010–2014)

Android 2.2 Froyo / 2.3 Gingerbread

- Otimização de desempenho.
- Melhorias de bateria.
- Suporte a NFC (Gingerbread).

Android 3.x Honeycomb (2011)

- Exclusivo para tablets.
- Interface otimizada para telas grandes.

Android 4.x Ice Cream Sandwich / Jelly Bean / KitKat

- Interface Holo (design mais moderno).
- Google Now e recursos de voz.
- Integração profunda com serviços Google.

2.10.4 Maturidade e Consolidação (2014–2019)

Android 5.x Lollipop

- Introdução do Material Design.
- Notificações na tela de bloqueio.
- Suporte a múltiplos usuários.

Android 6.x Marshmallow

- Permissões de apps mais controladas.
- Suporte a leitor de impressões digitais.

Android 7.x Nougat

- Modo de tela dividida.
- Melhorias de desempenho e segurança.

Android 8.x Oreo

- Notificações agrupadas.
- Picture-in-Picture.

Android 9 Pie

- Navegação por gestos.
- Recursos de inteligência artificial para economia de energia.

2.10.5 Era Moderna (2019–presente)

Android 10

- Tema escuro.
- Fim dos nomes de doces (passa a usar apenas números).
- Melhorias de privacidade.

Android 11

- Conversas em bolhas.
- Melhor controle de permissões temporárias.

Android 12

- Interface Material You (personalização de cores e temas).

- Desempenho e segurança otimizados.

Android 13

- Expansão do Material You.
- Recursos específicos para tablets e dispositivos dobráveis.

Android 14 (2023)

- Melhorias em acessibilidade e compatibilidade.
- Otimizações para dispositivos com múltiplas telas.

2.10.6 Arquitetura do Android

O Android é composto por camadas:

1. Kernel Linux
    - Gerencia hardware, processos, memória e segurança.
    - Adaptado para dispositivos móveis.
2. Bibliotecas Nativas
    - Gráficos (OpenGL ES), áudio, banco de dados SQLite.
3. Android Runtime (ART)
    - Substituiu a Dalvik VM.
    - Executa apps de forma mais eficiente.
4. Framework de Aplicações
    - APIs para desenvolvedores (GPS, câmera, notificações).
5. Aplicativos de Sistema
    - Apps básicos (Telefone, Mensagens, Configurações).

3. PROCESSOS

3.1 Conceito

Um processo é um programa em execução. Enquanto um programa é apenas um conjunto de instruções armazenadas em disco, o processo é a instância ativa dessas instruções carregadas na memória, juntamente com todos os recursos que ele precisa para ser executado.

3.2 Estrutura de um Processo

Cada processo é formado por diferentes partes na memória:

- **Seção de Código (Text)** → Instruções do programa.
- **Seção de Dados** → Variáveis globais e estáticas.
- **Heap** → Área para alocação dinâmica de memória (malloc/new).
- **Pilha (Stack)** → Armazena variáveis locais e endereços de retorno.
- **Contexto de Execução** → Estado atual do processador (registradores, contadores de programa, flags, etc.).
- **Descritor de Processo (PCB – Process Control Block)** → Estrutura de dados que o SO usa para gerenciar o processo (identificador, prioridade, estado, etc.).

3.3 Estados de Processos

Durante sua execução, um processo pode estar em diferentes **estados**:

1. **Novo (New)** → Criado, mas não pronto para executar.
2. **Pronto (Ready)** → Aguardando ser escalonado para a CPU.
3. **Executando (Running)** → Usando o processador no momento.
4. **Bloqueado (Waiting)** → Esperando por um recurso ou evento (ex.: leitura de disco).
5. **Finalizado (Terminated)** → Execução concluída.

Troca de Contexto (Context Switch) → Quando a CPU alterna entre processos, o sistema salva o estado do processo atual e carrega o do próximo.

3.4 Tipos de Processos

- **Processos de Usuário** → Criados por aplicativos (navegador, editor de texto, etc.).
- **Processos do Sistema** → Criados pelo sistema operacional (gerenciadores de memória, drivers).
- **Processos em Primeiro Plano (Foreground)** → Interagem diretamente com o usuário.
- **Processos em Segundo Plano (Background)** → Executam sem interação direta (serviços, daemons).

3.5 Controle de Processos – PCB

O **Process Control Block (PCB)** é uma estrutura de dados que o sistema operacional usa para armazenar todas as informações sobre um processo:

- Identificador do processo (PID)
- Estado atual
- Registradores da CPU
- Informações de memória
- Arquivos abertos
- Estatísticas de uso de CPU/E/S

Quando o sistema troca a CPU de um processo para outro (**context switch**), ele salva o estado no PCB do processo que foi interrompido e carrega o PCB do processo que vai assumir a execução.

4. THREADS

4.1 Conceito

Uma *thread* (ou **linha de execução**) é a menor unidade de processamento que um sistema operacional pode agendar para execução. Enquanto um **processo** é como um “programa em execução” com seu próprio espaço de memória, uma *thread* é como um “fluxo de instruções” dentro desse processo.

4.2 Diferença entre Processos e Threads

Característica	Processo	Thread
Espaço de memória	Isolado	Compartilhado com outras threads do mesmo processo
Criação	Mais “pesada” e lenta	Mais leve e rápida
Comunicação	Necessita mecanismos como *pipes* ou *sockets*	Pode compartilhar variáveis na memória
Independência	Se um falha, os outros continuam	Falha de uma pode afetar todo o processo

4.3 Modelo de Processos Multithread

O **modelo multithread** permite que um único processo tenha várias threads executando em paralelo ou concorrência, aumentando a **eficiência** e a **responsividade**.

Principais vantagens

- **Melhor desempenho** em sistemas com múltiplos núcleos.
- **Menor consumo de recursos** em comparação à criação de vários processos.
- **Maior responsividade** — útil para interfaces gráficas e aplicações interativas.
- **Compartilhamento de dados simplificado**.

4.4 Tipos de Threads

O SO pode gerenciar *threads* de duas formas principais:

Threads de Usuário (User-Level Threads - ULT)

- Gerenciadas por bibliotecas na aplicação (não pelo núcleo do SO).
- Não exigem chamadas frequentes ao kernel.
- Mais rápidas para criar e trocar.
- Limitação: se uma thread bloquear, todas podem parar.

Exemplo: Bibliotecas POSIX Threads (Pthreads).

Threads de Kernel (Kernel-Level Threads - KLT)

- Gerenciadas diretamente pelo núcleo do sistema operacional.
- Melhor aproveitamento de múltiplos núcleos.
- Mudança de contexto mais “pesada” que ULT.
- Cada thread é conhecida pelo SO individualmente.

Exemplo: Implementações do Linux, Windows e macOS.

4.5 Modelos de Mapeamento de Threads

O SO pode combinar threads de usuário e de kernel de diferentes formas:

Many-to-One

- Muitas threads de usuário → Uma thread de kernel.
- Simples, mas não aproveita múltiplos núcleos.

One-to-One

- Uma thread de usuário → Uma thread de kernel.
- Melhor desempenho em paralelo, mas maior custo.

Many-to-Many

- Muitas threads de usuário → Muitas threads de kernel.
- Flexível e eficiente.

4.6 Aplicações Práticas de Multithreading

- **Navegadores Web**: cada aba pode ser uma thread separada.
- **Jogos**: uma thread para gráficos, outra para física, outra para som.
- **Servidores**: uma thread por cliente conectado.
- **Aplicativos de edição de vídeo**: renderização e interface rodando em paralelo.

5. ESCALONAMENTO DE PROCESSOS

5.1 Conceito

O **escalonamento de processos** é o mecanismo pelo qual o sistema operacional decide **qual processo será executado** e **por quanto tempo**, quando há múltiplos processos prontos para utilizar o processador (CPU). Ele é fundamental para **otimizar o desempenho do sistema**, **garantir justiça no uso dos recursos** e **atender requisitos de tempo real**, quando necessário.

Um **processo** pode estar em diferentes estados:

1. **Novo**: acabou de ser criado.
2. **Pronto**: aguarda para usar a CPU.
3. **Em execução**: está usando a CPU.
4. **Bloqueado**: espera por I/O ou outro evento.
5. **Encerrado**: finalizou.

O escalonador atua **principalmente** sobre processos no estado **pronto**, decidindo qual será o próximo a executar.

5.2 Tipos de Escalonadores

No sistema operacional, existem três níveis principais de escalonamento:

1. Escalonador de Longo Prazo

- Decide quais processos entram no sistema para execução.
- Atua no controle da carga do sistema (quantidade de processos na memória).
- Mais comum em sistemas batch (lote).
- Ex.: Em um servidor de impressão, decidir quais trabalhos entram na fila.

2. Escalonador de Médio Prazo

- Pode suspender processos para liberar memória (swap out) e depois reativá-los (swap in).
- Usado para balancear a carga de CPU e I/O.

3. Escalonador de Curto Prazo (ou dispatcher)

- Atua constantemente, decidindo qual processo pronto será o próximo a usar a CPU.
- É o mais crítico em termos de desempenho, pois a decisão precisa ser rápida.

5.3 Objetivos do Escalonamento

O escalonamento busca **equilibrar múltiplos objetivos**, que podem variar conforme o tipo de sistema:

Em sistemas de tempo compartilhado (desktop, servidores):

- Maximizar o uso da CPU.
- Garantir resposta rápida ao usuário.
- Evitar que um processo monopolize a CPU.

Em sistemas batch (lote):

- Aumentar a vazão (quantidade de processos finalizados por unidade de tempo).
- Reduzir o tempo médio de espera.

Em sistemas de tempo real:

- Cumprir prazos (deadlines) estritamente.
- Minimizar a latência.

5.4 Algoritmos de Escalonamento de CPU

Existem vários algoritmos, cada um com vantagens e desvantagens.

5.4.1 First-Come, First-Served (FCFS)

- **Funcionamento:** Processos são atendidos na ordem de chegada.
- **Vantagem:** Simples de implementar.
- **Desvantagem:** Pode gerar o problema de **convoy effect** (um processo longo atrasa os demais).
- **Aplicação típica:** Sistemas batch simples.

5.4.2 Shortest Job Next (SJN) / Shortest Job First (SJF)

- **Funcionamento:** O processo com menor tempo estimado de CPU é escolhido primeiro.
- **Vantagem:** Minimiza o tempo médio de espera.
- **Desvantagem:** Difícil prever o tempo de execução; pode causar **starvation** (processos longos podem nunca executar).
- **Aplicação típica:** Sistemas batch onde o tempo de execução é conhecido.

5.4.3. Round Robin (RR)

- **Funcionamento:** Cada processo recebe um **quantum** (fatia de tempo) fixo para executar; depois é colocado no fim da fila de prontos.
- **Vantagem:** Bom para sistemas interativos; evita monopolização da CPU.
- **Desvantagem:** Quantum muito pequeno → overhead alto; muito grande → perde responsividade.
- **Aplicação típica:** Sistemas de tempo compartilhado.

5.4.4. Priority Scheduling

- **Funcionamento:** Cada processo tem uma prioridade; o de maior prioridade executa primeiro.
- **Vantagem:** Permite atender processos críticos rapidamente.
- **Desvantagem: Starvation** para processos de baixa prioridade (solução: *aging*, aumentando a prioridade com o tempo).
- **Aplicação típica:** Sistemas de tempo real e servidores.

5.4.5 Multilevel Queue Scheduling

- **Funcionamento:** Processos são divididos em filas separadas (interativos, batch, sistema, etc.), cada uma com seu próprio algoritmo de escalonamento.
- **Vantagem:** Permite tratar diferentes tipos de processos de forma especializada.
- **Desvantagem:** Rigidez; um processo não muda de fila.

5.4.6. Multilevel Feedback Queue

- **Funcionamento:** Similar ao anterior, mas processos podem mudar de fila conforme comportamento (I/O-bound ou CPU-bound).
- **Vantagem:** Alta flexibilidade; ajusta-se dinamicamente.
- **Aplicação típica:** Sistemas modernos como Windows e Linux.

5.4.7. Escalonamento em Tempo Real

- **Rate Monotonic Scheduling (RMS):** Prioridade fixa baseada no período da tarefa.
- **Earliest Deadline First (EDF):** Executa o processo com o prazo mais próximo.
- Usados em sistemas embarcados, robótica, e aplicações críticas.

6. COMUNICAÇÃO ENTRE PROCESSOS (IPC)

6.1 Conceito

A Comunicação entre Processos é o conjunto de mecanismos e técnicas que permitem que processos distintos troquem informações e coordenem suas ações.

Em sistemas operacionais modernos, processos normalmente não compartilham o mesmo espaço de endereçamento de memória, por questões de segurança e isolamento.
Portanto, para que possam interagir, o SO precisa fornecer métodos controlados de comunicação.

Esses métodos podem ser:

- Dentro de um mesmo computador (IPC local).
- Entre diferentes computadores (IPC distribuído).

Objetivos principais do IPC:

1. **Troca de dados** – Enviar e receber informações de forma confiável.
2. **Sincronização** – Coordenar a execução de processos para evitar conflitos (ex.: acesso simultâneo a um arquivo).
3. **Gerenciamento de recursos** – Compartilhar dispositivos ou memória de forma eficiente.

6.2 Modelos de Comunicação

Existem dois modelos principais:

6.2.1 Comunicação por Troca de Mensagens

- Os processos enviam e recebem pacotes de dados (*mensagens*).
- Geralmente implementada via **chamadas de sistema**.
- Não há compartilhamento direto de memória; o SO é responsável por transportar a informação.

Vantagens:

- Simples de implementar.
- Mais seguro (não há acesso direto à memória de outro processo).

Desvantagens:

- Pode ser mais lento devido à cópia de dados e intervenção do kernel.

6.2.2 Comunicação por Memória Compartilhada

- Dois ou mais processos compartilham uma região da memória física.
- Permite leitura e escrita direta, sem passar pelo kernel (após configuração inicial).
- Necessita mecanismos de **sincronização** para evitar condições de corrida.

Vantagens:

- Alto desempenho (acesso direto à memória).

Desvantagens:

- Mais complexa, exige controle rigoroso de concorrência.

6.3 Mecanismos de IPC

O Sistema Operacional fornece várias ferramentas para implementar IPC.

6.3.1 Pipes (Tubos)

- Canal unidirecional para transmissão de dados entre processos.
- Funciona como um “arquivo” temporário em memória.
- Usado em Unix/Linux com o operador `|` no shell (ex.: `ls | grep .txt`).

Tipos:

- **Pipes anônimos** – Apenas entre processos relacionados (pai-filho).
- **Pipes nomeados (FIFOs)** – Permitem comunicação entre processos não relacionados.

6.3.2 Sockets

- Interface de comunicação que permite IPC **local ou remoto** (via rede).
- Suporta **TCP** (orientado a conexão) e **UDP** (sem conexão).
- Muito usado para comunicação cliente-servidor.

6.3.3 Memória Compartilhada

- Regiões de memória mapeadas para múltiplos processos.
- Exige **semáforos ou mutex** para evitar escrita simultânea.
- Extremamente rápida, usada em aplicações de alto desempenho.

6.3.4 Semáforos

- Estruturas de controle usadas para **sincronizar acesso a recursos**.
- Dois tipos principais:
  - **Semáforo binário** (mutex) – Controle exclusivo.
  - **Semáforo contador** – Permite múltiplos acessos limitados.

6.3.5 Filas de Mensagens

- Armazenam mensagens estruturadas que processos podem enviar/receber de forma assíncrona.
- Mais flexíveis que pipes, pois permitem mensagens identificadas e ordenadas.

6.3.6 Sinais

1. Notificações assíncronas enviadas a um processo para indicar que um evento ocorreu.
2. Ex.: Em Unix, `SIGINT` (Ctrl+C) envia um sinal para encerrar um processo.

6.4 Problemas e Cuidados no IPC

- **Condição de corrida (Race Condition)** – Quando dois processos acessam e modificam dados simultaneamente de forma não controlada.
- **Deadlocks** – Dois ou mais processos ficam esperando indefinidamente por recursos que nunca serão liberados.
- **Starvation (Inanição)** – Um processo fica indefinidamente sem acesso ao recurso.
- **Segurança** – IPC pode ser usado como vetor de ataque se não houver controle de acesso.

7 DEADLOCK

Deadlock (ou impasse) é uma situação em que dois ou mais processos ficam **permanentemente bloqueados**, cada um esperando por um recurso que está sendo mantido por outro processo do conjunto. Nenhum deles pode continuar — o sistema ficou “travado” nessa parte.

7.1 Condições necessárias (Coffman — 4 condições)

Para que um deadlock ocorra, as quatro condições abaixo precisam ocorrer simultaneamente:

1. **Mutual exclusion (Exclusão mútua)**
   Pelo menos um recurso é não-compartilhável (somente um processo por vez).

2. **Hold and wait (Segurar e esperar)**
   Um processo mantém pelo menos um recurso e solicita outros recursos adicionais.

3. **No preemption (Sem preempção)**
   Recursos não podem ser retirados do processo que os possui até que ele os libere voluntariamente.

4.

5. **Circular wait (Espera circular)**

Existe uma cadeia circular de processos P1 espera por recurso de P2, P2 espera por recurso de P3, …, Pn espera por recurso de P1.

Quebrar qualquer uma dessas condições evita deadlock.

7.2 Detecção de deadlocks

1) Caso: recursos com uma única instância por tipo.

Basta detectar ciclos no **Resource Allocation Graph**. Detectar ciclo em um grafo direcionado é O(V + E) (DFS).

2) Caso: recursos com múltiplas instâncias.

Usa-se um algoritmo baseado em matrizes — semelhante ao algoritmo do banqueiro/safety check:

Defina:

- `Available` — vetor de recursos livres.
- `Allocation[i]` — vetor do que o processo `i` já tem.
- `Request[i]` (ou `Need[i]`) — vetor do que `i` ainda precisa.
-

Algoritmo de detecção (resumido):

- Work = Available
- Finish[i] = false para todos i
- Procure um i tal que Finish[i] == false e Request[i] <= Work. Se encontrado: Work = Work + Allocation[i]; Finish[i] = true; repetir passo 3.
- Ao fim, qualquer processo com Finish[i] == false está em deadlock.

Exemplo (deadlock verdadeiro)

Totais: A=1, B=1, C=1

Alocações:
P1: A=1,B=0,C=0
P2: A=0,B=1,C=0
P3:A=0,B=0,C=1
Requests:
P1 quer B (0,1,0)
P2 quer C (0,0,1)
P3 quer A (1,0,0)

Available = (0,0,0). Não existe processo cujo Request <= Work, então todos ficam com Finish=false → deadlock entre P1,P2,P3.

7.3 Evitação de deadlocks (Banker's algorithm — visão geral)

A evitação exige conhecimento prévio do **máximo** que cada processo pode requisitar.

- **Banker's algorithm**: antes de conceder uma requisição, o SO simula a concessão e verifica se o sistema permanece **em estado seguro** (existe uma sequência de finalização para todos os processos). Se ficar inseguro, recusa ou faz o processo esperar.

Prós: evita deadlock.

Contras: requer declaração de necessidades máximas, overhead e pouco prático em ambientes dinâmicos.

7.4 Prevenção de deadlocks (quebrar condições de Coffman)

Estratégias comuns:

1. **Proibir hold-and-wait**
   Requer que processos peçam *todos* os recursos de uma vez (reduz concorrência e pode diminuir paralelismo).
   Outra forma: liberar recursos antes de requisitar novos.
2. **Permitir preempção**
   Permitir retirar recursos de processos (complexo: precisa rollback/checkpoint).
3. **Prevenir circular wait**
   Cada recurso tem uma ordem global; processos devem requisitar recursos em ordem crescente. Isso elimina esperas circulares.
4. **Eliminar mutual exclusion**
   Só se recurso puder ser compartilhado (nem sempre possível: impressora, por exemplo, é excludente).

Trade-offs: prevenção costuma sacrificar desempenho, paralelismo ou simplicidade.

7.5 Recuperação de deadlocks

Quando você detecta deadlock, opções comuns:

- **Abortar processos**: terminar um ou mais processos do ciclo (escolher com base em custo — tempo de execução, recursos, prioridade).
- **Preempção de recursos**: retirar recursos de processos (é preciso checkpoint/rollback para reverter trabalhos).
- **Timeouts**: se um processo aguarda por muito tempo, matar/rollback — simples mas pode abortar sem necessidade.

Cada método tem custo: perda de trabalho, complexidade de restauração, impacto no usuário.

7.6 Recapitulando em poucas linhas

- **Deadlock** = processos esperando uns pelos outros indefinidamente.
- Ocorre quando as 4 condições de Coffman estão presentes.
- **Prevenção** quebra uma das condições (p. ex. ordem de locks).
- **Evitação** (Banker) garante que o sistema permaneça em estado seguro.
- **Detecção** encontra ciclos (ou usa algoritmo de Work/Finish) e **recuperação** mata processos/rollback.
- Em sistemas reais, combina-se: evitar via design, usar timeouts, detectar/recuperar quando necessário.

8 SISTEMAS DE ARQUIVOS

Um **Sistema de Arquivos (File System)** é o componente de um sistema operacional responsável por **organizar, armazenar, recuperar e gerenciar** dados em dispositivos de armazenamento, como HDs, SSDs, pendrives ou discos ópticos. Ele define **como os dados são gravados** e **como as informações sobre esses dados (metadados)** são mantidas.

Sem um sistema de arquivos, os dados seriam apenas um fluxo bruto de bits, sem estrutura, tornando impossível identificar onde começa ou termina um arquivo.

8.1 Funções Principais

- **Organização** – Estruturar dados em arquivos e diretórios.
- **Endereçamento** – Mapear a posição física dos dados no disco.
- **Gerenciamento de Espaço** – Controlar áreas livres e ocupadas.
- **Segurança** – Controlar permissões de acesso.
- **Recuperação** – Possibilitar restauração em caso de falhas.
- **Consistência** – Garantir integridade dos dados mesmo após quedas de energia ou falhas.

8.3 Conceitos Fundamentais

a) Arquivo

- Unidade lógica de armazenamento que contém dados.
- Pode armazenar texto, imagens, vídeos, executáveis etc.
- Possui **metadados**: nome, tamanho, data de criação, permissões.

b) Diretório (ou Pasta)

- Estrutura hierárquica para organizar arquivos.
- Pode conter outros diretórios (subdiretórios).
-

c) Metadados

- Informações sobre o arquivo, como:
- Nome
- Tipo
- Tamanho
- Datas (criação, modificação, acesso)
- Permissões de acesso
- Localização no disco

d) Blocos

- O disco é dividido em **blocos de tamanho fixo** (ex.: 4 KB).
- Arquivos ocupam um ou mais blocos.
- Mesmo que um arquivo tenha apenas 1 byte, ele usará um bloco inteiro (fragmentação interna).

8.4 Estrutura Interna

Um sistema de arquivos geralmente possui:

- **Boot Block** – Área com informações para inicializar o sistema.
- **Superbloco** – Contém dados gerais sobre o sistema de arquivos (tamanho, número de blocos, área livre).
- **Tabela de Alocação** – Mapa que indica quais blocos estão livres ou ocupados.
- **Área de Dados** – Onde os arquivos efetivamente são armazenados.

8.5 Tipos de Organização

a) Alocação Contígua

- Arquivos ocupam blocos consecutivos no disco.
- **Vantagem:** Acesso rápido.
- **Desvantagem:** Difícil expansão, fragmentação externa.

b) Alocação Encadeada

- Cada bloco contém um ponteiro para o próximo bloco do arquivo.
- **Vantagem:** Fácil expansão.
- **Desvantagem:** Acesso mais lento.

c) Alocação Indexada

- Um bloco especial (bloco de índice) contém todos os endereços dos blocos que formam o arquivo.
- **Vantagem:** Acesso rápido e direto.
- **Desvantagem:** Bloco de índice ocupa espaço extra.

8.6 Tipos de Sistemas de Arquivos

- FAT16 / FAT32 – Usados no MS-DOS e Windows antigos; simples, mas limitados.
- NTFS – Usado no Windows moderno; suporta permissões avançadas, compressão e criptografia.
- EXT2 / EXT3 / EXT4 – Muito comuns em sistemas Linux; EXT4 suporta grandes volumes e journaling.
- HFS / HFS+ / APFS – Usados nos sistemas Apple; APFS otimizado para SSDs.
- exFAT – Desenvolvido para pendrives e cartões de memória, compatível com Windows e macOS.
- ISO 9660 – Padrão para CDs e DVDs.

9 GERENCIAMENTO DE MEMÓRIA

O **gerenciamento de memória** é a função do sistema operacional (SO) responsável por **controlar, alocar, liberar e otimizar** o uso da memória principal (RAM) e, em alguns casos, também a memória secundária (disco) quando há necessidade de troca (*swap*).

O objetivo principal é:

- Garantir que **cada processo** tenha o espaço necessário para execução.
- **Evitar conflitos** de acesso entre processos.
- **Maximizar o uso da memória** de forma eficiente.

9.1 Tipos de Memória no Contexto do SO

Memória Principal

- **RAM (Random Access Memory)** – rápida, volátil e utilizada diretamente pela CPU.
- O sistema operacional gerencia **endereços lógicos** (usados por programas) e **endereços físicos** (posições reais na RAM).

Memória Secundária

- Disco rígido ou SSD.
- Usada para armazenar dados de forma permanente ou para ampliar virtualmente a RAM via **memória virtual**.

9.2 Principais Tarefas do Gerenciamento de Memória

1. Rastreamento de uso
    - O SO mantém tabelas (como a tabela de páginas) para saber quais blocos de memória estão livres ou ocupados.
1. Alocação
    - Reservar espaço para processos em execução.
    - Escolher onde colocar cada processo na memória física.
2. Desalocação
    - Liberar a memória quando um processo termina ou quando parte dela não é mais necessária.
3. Proteção e isolamento
    - Impedir que um processo acesse áreas de memória de outro processo (violação de acesso).

9.3 Estratégias de Alocação de Memória

1 Alocação Contígua

- O processo ocupa um bloco contínuo de memória física.
- **Vantagens:** simples e rápido.
- **Desvantagens:** fragmentação externa.
- **Técnicas de escolha:**
    - **First-fit:** pega o primeiro bloco livre suficiente.
    - **Best-fit:** pega o menor bloco livre que caiba.
    - **Worst-fit:** pega o maior bloco livre.

2 Paginação

- Divide a memória física em **quadros** (frames) e os processos em **páginas**.
- O mapeamento entre páginas e quadros é feito pela **tabela de páginas**.
- **Vantagem:** evita fragmentação externa.
- **Desvantagem:** pode ter sobrecarga de gerenciamento.

3 Segmentação

- A memória é dividida logicamente em **segmentos** (código, dados, pilha, etc.).
- Cada segmento pode ter tamanho variável.
- Boa para facilitar **proteção e compartilhamento**.

4 Memória Virtual

- Permite que o processo use **mais memória do que a RAM disponível**.
- Páginas menos usadas são movidas para o disco (swap) e trazidas de volta quando necessário (**page fault**).

9.4 Gerenciamento de Memória Virtual

Troca de Páginas (*Page Replacement*)

- Quando a RAM está cheia e uma nova página precisa ser carregada.
- Algoritmos comuns:
    - **FIFO (First In, First Out)** – remove a página mais antiga.
    - **LRU (Least Recently Used)** – remove a página não usada há mais tempo.
    - **Optimal** – remove a página que não será usada por mais tempo (teórico).

Thrashing

- Ocorre quando há excesso de troca de páginas entre RAM e disco, causando queda no desempenho.

9.5 Algoritmos de Gerenciamento de Concorrência e Alocação de Memória

Quando um processo precisa de memória, o sistema operacional precisa decidir **onde** e **como** alocar esse espaço na **memória principal**. Os métodos **First Fit**, **Next Fit** e **Best Fit** são estratégias clássicas para resolver esse problema.

9.5.1 First Fit (Primeiro Ajuste)

Definição:

- O sistema percorre a **lista de blocos livres** de memória a partir do início.
- Aloca o **primeiro espaço livre** suficientemente grande para o processo.

Como funciona:

1. O SO mantém uma lista dos blocos livres (ordenados pelo endereço).
2. Quando um processo solicita memória, começa-se a busca do **início da lista**.
3. O primeiro bloco que for **igual ou maior** que o tamanho solicitado é selecionado.
4. Se sobrar espaço no bloco, ele é dividido; o restante volta para a lista de blocos livres.

Exemplo:

- Blocos livres: [10KB, 50KB, 20KB, 35KB]
- Processo pede 15KB:
  - Pega o primeiro bloco grande o bastante → 50KB.
  - Aloca 15KB e sobra 35KB.

Vantagens:

- Simples e rápido (para listas pequenas).
- Menos sobrecarga de busca.

Desvantagens:

- Pode gerar **fragmentação externa** rapidamente (pequenos espaços livres espalhados).
- Pode favorecer alocação no início da memória, deixando final subutilizado.

9.5.2 Next Fit (Próximo Ajuste)

Definição:

- Parecido com o First Fit, mas a busca **não recomeça do início** a cada alocação.
- Continua a partir do **ponto onde parou** na última busca.

Como funciona:

1. O SO guarda um **ponteiro** para o local da última alocação.
2. Quando um processo pede memória, a busca começa **a partir desse ponto**.
3. Ao chegar no fim da lista, volta ao início e continua até encontrar um espaço adequado.

Exemplo:

- Blocos livres: [10KB, 50KB, 20KB, 35KB]
- Última alocação foi no segundo bloco (50KB).
- Novo processo pede 18KB:
  - Busca a partir do bloco seguinte (20KB), encontra espaço e aloca.

Vantagens:

- Evita concentração de alocações no início da memória.
- Distribui melhor a fragmentação.

Desvantagens:

- Pode ser mais lento que First Fit, já que pode dar uma volta completa para encontrar espaço.

9.5.3 Best Fit (Melhor Ajuste)

Definição:

- Procura o **menor bloco livre** que seja suficientemente grande para o processo.
- Objetivo: minimizar o espaço desperdiçado.

Como funciona:

1. Percorre toda a lista de blocos livres.
2. Escolhe o bloco mais justo (menor sobra possível após alocação).
3. Se sobrar espaço, o bloco é dividido.

Exemplo:

- Blocos livres: [10KB, 50KB, 20KB, 35KB]
- Processo pede 18KB:
  - O bloco mais justo é 20KB (sobra apenas 2KB).

Vantagens:

- Reduz desperdício imediato de memória.
- Pode diminuir fragmentação no curto prazo.

Desvantagens:

- Mais lento (precisa percorrer todos os blocos para achar o “melhor”).
- Pode aumentar **fragmentação externa** de blocos muito pequenos inutilizáveis.

9.5.4 Comparação Resumida

Algoritmo	Velocidade de busca	Fragmentação	Comentário
First Fit	Rápido	Alta	Simples e rápido, mas concentra fragmentos no início
Next Fit	Moderado	Média	Distribui melhor a fragmentação, mas pode ser mais lento
Best Fit	Lento	Variável	Menor sobra imediata, mas pode criar muitos blocos pequenos

10 CONCEITOS DE MEMÓRIA VIRTUAL

A **memória virtual** é uma técnica usada pelos sistemas operacionais para dar a impressão de que o computador tem mais memória RAM do que realmente possui. Ela faz isso **usando parte do disco rígido ou SSD como uma extensão da RAM**, permitindo que mais programas sejam executados simultaneamente, mesmo que a RAM física seja limitada.

10.1 Objetivos principais

1. Ampliar o espaço de endereçamento:
- Permite que programas usem endereços maiores do que a memória física real.
2. Isolamento e proteção:
- Cada processo “enxerga” seu próprio espaço de memória, sem interferir nos outros.
3. Execução de programas grandes:
- Um programa pode ser maior do que a RAM disponível, pois partes dele ficam no disco.
4. Gerenciamento eficiente:
- O sistema operacional decide quais partes ficam na RAM e quais vão temporariamente para o disco.

10.2 Como funciona

A memória virtual usa o conceito de **paginação** (mais comum) ou **segmentação** (menos usada hoje).

- **Paginação**:
  A memória é dividida em blocos de tamanho fixo chamados **páginas** (geralmente 4 KB).O disco contém uma **área de troca** (*swap*), onde páginas que não estão em uso imediato são armazenadas.
- **Mapeamento de endereços**:
  - **Endereço lógico**: o que o programa “acha” que está usando.
  - **Endereço físico**: onde realmente os dados estão (RAM ou disco).
  - O **MMU** (*Memory Management Unit*) faz a tradução entre eles.

Exemplo:

Imagine que um computador tenha **4 GB de RAM**, mas esteja rodando programas que juntos precisem de **6 GB**.

O SO mantém **4 GB** ativos na RAM e **2 GB** ficam no arquivo de *swap* no disco. Quando um programa precisa de uma parte que está no *swap*, ocorre um **page fault** (falha de página) e o sistema troca (swap) o conteúdo.

10.3 Vantagens vs Desvantagens

Vantagens

- Permite executar programas maiores que a RAM física.
- Aumenta a multitarefa, mantendo mais programas ativos.
- Isola processos para segurança.

Desvantagens

- **Desempenho menor**: acesso ao disco é muito mais lento que à RAM.
- Pode causar **thrashing**: quando o sistema passa mais tempo trocando páginas entre RAM e disco do que executando programas.

10.4 Memória Virtual e o Sistema Operacional

A **memória virtual** é uma técnica que permite que um sistema operacional execute programas como se houvesse muito mais memória RAM disponível do que realmente existe fisicamente. Isso é feito usando **espaço em disco** (normalmente o *swap* ou *page file*) para armazenar temporariamente partes da memória que não estão sendo usadas no momento.

O **Sistema Operacional (SO)** é o responsável por gerenciar essa memória virtual, fazendo a **tradução de endereços lógicos para endereços físicos** e movendo dados entre a RAM e o disco conforme necessário.

10.5 O papel da Tabela de Páginas

Cada processo tem seu **espaço de endereçamento virtual**, dividido em **páginas** (blocos de tamanho fixo, como 4 KB).

O SO mantém para cada processo **uma tabela de páginas**, que faz o mapeamento:

- **Endereço lógico** (usado pelo programa) → **Endereço físico** (posição real na RAM).
- Se a página não estiver na RAM no momento, a tabela indica que ela está no disco, gerando uma **falha de página (page fault)**.

Estrutura típica de uma entrada na tabela de páginas:

- **Número da moldura (frame)** na RAM (se presente).
- **Bit de presença** (indica se a página está na RAM ou no disco).
- **Bit de modificação** (dirty bit — se a página foi alterada).
- **Bits de proteção** (se é leitura, escrita, execução).

Quando a RAM está cheia e ocorre uma **falha de página**, o SO precisa decidir **qual página existente será removida** para abrir espaço para a nova. É aí que entram **os algoritmos de substituição de páginas**.

10.6 Algoritmos de substituição de páginas

10.6.1 FIFO (First In, First Out)

- **Lógica:** Remove a página que está na memória há mais tempo.
- **Implementação:** Pode ser feito com uma fila simples.
- **Prós:** Simples de implementar.
- **Contras:** Pode remover páginas que ainda estão sendo muito usadas, causando o fenômeno de **anomalia de Belady** (mais memória → mais *page faults*).

Exemplo:

Se as páginas chegaram na ordem A, B, C e chega D, a primeira a sair será A.

10.6.2 LRU (Least Recently Used)

- Lógica: Remove a página que não foi usada há mais tempo.
- Implementação: Usa estruturas como pilhas ou contadores de tempo para rastrear acessos.
- Prós: Geralmente tem desempenho melhor que o FIFO.
- Contras: Mais complexo, exige controle de histórico de acessos.

Exemplo:

Se as páginas A, B, C estão na RAM e a mais antiga sem uso é B, ela será substituída.

10.6.3 Clock (ou Segunda Chance)

- **Lógica:** É uma implementação otimizada do LRU.
- **Funcionamento:** As páginas são organizadas como um **anel circular** (clock).
    - Cada página tem um **bit de referência**.
    - O "ponteiro do relógio" varre as páginas:
        - Se o bit de referência for 0, a página é substituída.
        - Se for 1, o bit é zerado e o ponteiro avança, dando uma **segunda chance**.
    - **Prós:** Mais eficiente que o LRU puro.
    - **Contras:** Ainda não é perfeito para todos os padrões de acesso.

10.7 Resumo visual

Algoritmo	Critério de remoção	Complexidade	Eficiência
FIFO	Mais antigo na memória	Baixa	Baixa a média
LRU	Menos recentemente usado	Média a alta	Alta
Clock	Baseado em referência e segunda chance	Média	Alta (boa relação custo/benefício)