CURSO SUPERIOR DE ADS

Gerenciamento de Memória e Processos

Prof. Fernando Marlon Soares Figueiredo

Disciplina: Introdução à Computação e suas aplicações

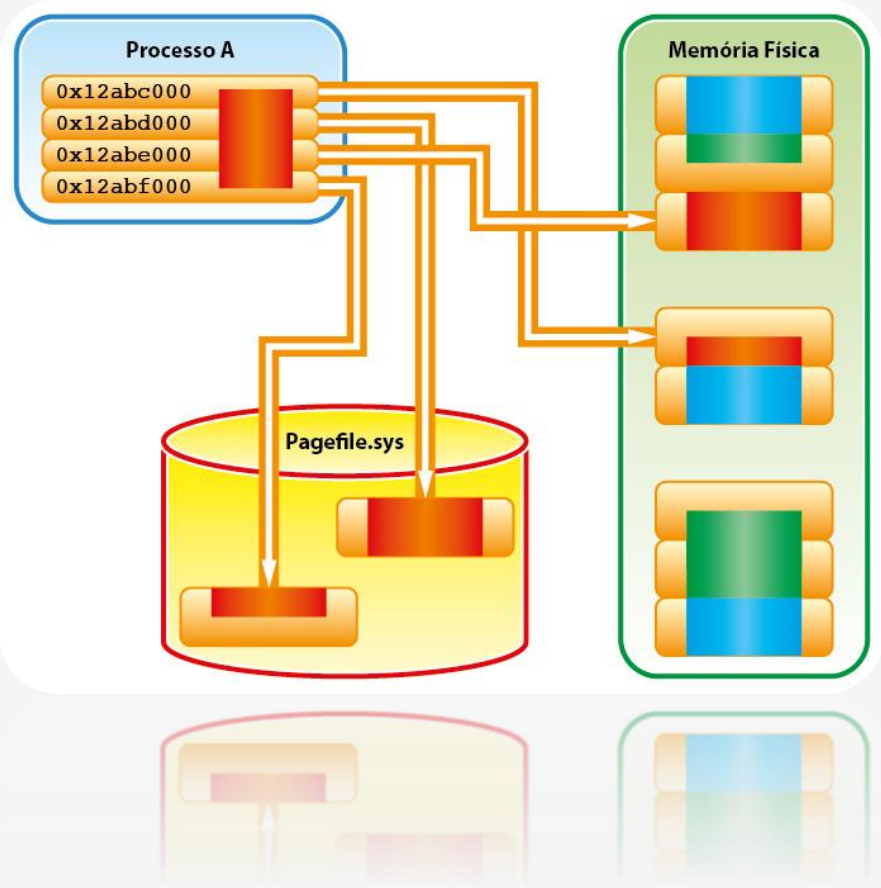
SECTRAS

ADS

Introdução ao Gerenciamento de Memória e Processos

O que veremos hoje:

- **Gerenciamento de Memória:** Como o sistema operacional (SO) organiza a memória para que diferentes programas possam rodar ao mesmo tempo sem interferir entre si.
- **Gerenciamento de Processos:** Como o SO lida com múltiplos programas (processos) sendo executados simultaneamente.



O Papel do Sistema Operacional (SO)

Funções principais do SO:

- **Gerenciamento de Processos:** Responsável por criar, executar e terminar os processos. O SO decide qual processo vai ser executado pela CPU, quando e por quanto tempo.
- **Gerenciamento de Memória:** O SO gerencia o uso da memória (RAM) para garantir que cada processo tenha a quantidade necessária de recursos e evitar interferência entre eles.
- **Gerenciamento de Entrada/Saída (E/S):** Organiza o fluxo de dados entre os processos e os dispositivos externos, como discos, teclados e monitores.
- **Gerenciamento de Arquivos:** Controla a leitura, escrita e organização de arquivos no sistema de arquivos do computador.

Função geral: O SO organiza e coordena todos esses recursos para garantir que o sistema funcione de forma eficiente e sem falhas.

O Conceito de Processo

Definição:

Um **processo** é uma instância de execução de um programa. Quando um programa é executado, o SO cria um processo que contém todos os recursos necessários para a execução do código.

Estrutura de um Processo:

- **Código:** O conjunto de instruções do programa que está sendo executado.
- **Dados:** As variáveis e informações que o processo usa durante sua execução.
- **Estado:** O status do processo em determinado momento (executando, esperando, etc.).

Importância do Conceito:

Processos são essenciais para o funcionamento de qualquer sistema operacional, pois é através deles que os programas são executados e interagem com o hardware.

Programa Vs Processo

A compreensão do conceito de processo exige, necessariamente, a distinção entre programa e execução.

Um programa consiste em um **conjunto de instruções armazenadas**, **caracterizando-se como uma entidade estática**, que não possui comportamento dinâmico por si só. Ele representa apenas o potencial de execução.

Por outro lado, o processo corresponde à **execução ativa desse programa**, incorporando não apenas as instruções, mas também o estado atual da execução, os dados utilizados e os recursos alocados pelo sistema operacional.

Dessa forma, um mesmo programa pode originar múltiplos processos simultaneamente, cada um com seu próprio contexto e ciclo de vida.

Essa distinção é fundamental para compreender como o sistema operacional organiza e gerencia a execução de diferentes aplicações de forma concorrente.

Estrutura de Memória do Processo

Para que um processo possa ser executado de forma organizada, ele não é armazenado na memória de maneira aleatória.

Sua execução depende de uma **estrutura de memória bem definida**, que permite ao sistema operacional controlar e gerenciar seu funcionamento.

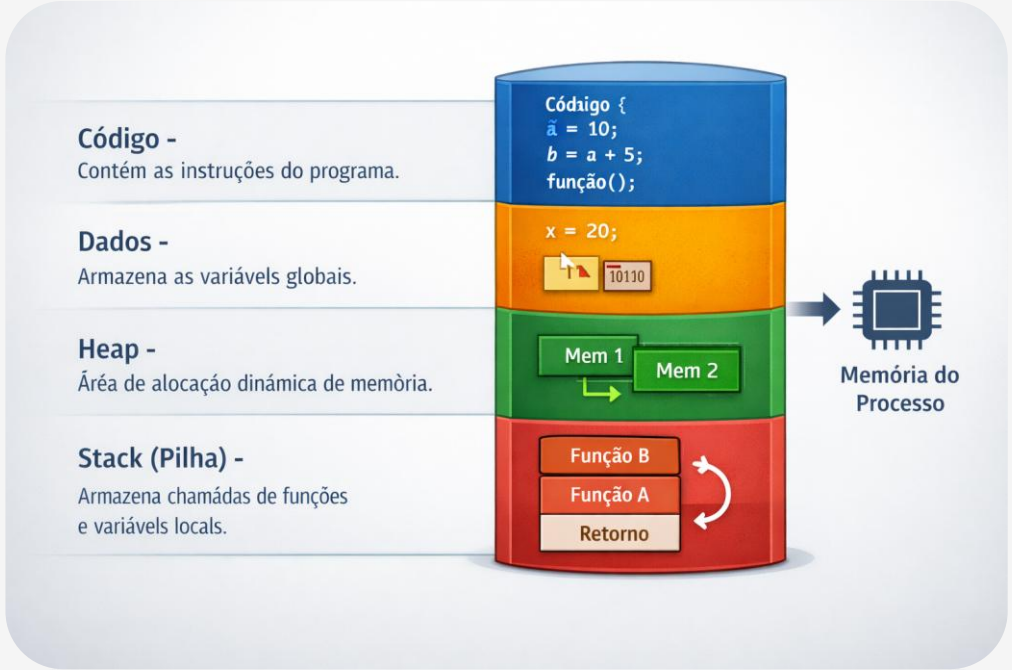
De forma geral, um processo é dividido em diferentes regiões, cada uma com uma função específica.

A região de **código** armazena as instruções que serão executadas, representando a lógica do programa. A área de dados contém as variáveis globais utilizadas durante a execução.

A **pilha (stack)** é responsável por controlar as chamadas de funções, parâmetros e variáveis locais, sendo fundamental para o fluxo de execução do programa.

Já a área conhecida como **heap** é utilizada para **alocação dinâmica de memória**, permitindo que o processo utilize recursos adicionais conforme necessário.

Essa organização garante que o processo seja executado de forma **estruturada**, **previsível** e **controlada**, evitando conflitos e facilitando a atuação do sistema operacional.



Controle do Processo (PCB)

A execução simultânea de múltiplos processos exige que o sistema operacional mantenha um **controle rigoroso** sobre cada um deles.

Para isso, o sistema utiliza uma estrutura fundamental chamada **Bloco de Controle de Processo (Process Control Block — PCB)**.

O PCB funciona como um **registro administrativo do processo**, armazenando todas as informações necessárias para que o sistema operacional possa gerenciar sua execução de forma precisa.

Entre as principais informações contidas no PCB, destacam-se o **identificador do processo (PID)**, o **estado atual**, os **registradores da CPU**, o **contador de programa** e as informações relacionadas ao uso da memória.

É por meio dessa estrutura que o sistema operacional consegue **interromper**, **salvar e retomar a execução de um processo**, garantindo que ele continue exatamente do ponto onde foi pausado.

Sem o PCB, o gerenciamento de múltiplos processos seria inviável, pois não haveria controle sobre o estado e o progresso de cada execução.

Estados de um Processo

Existem 5 estados principais de um processo:

1. **Novo:** O processo está sendo criado. O SO está alocando os recursos necessários para que ele comece a execução.

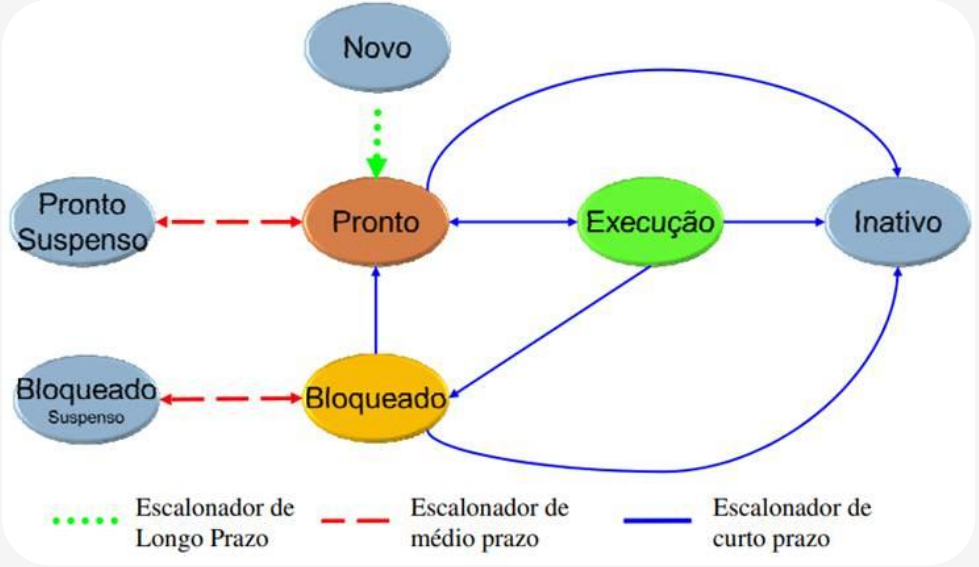
2. **Pronto:** O processo está pronto para ser executado, mas está aguardando na fila para que a CPU o processe.

3. **Executando:** O processo está sendo executado pela CPU. Neste momento, ele está realizando suas operações e interagindo com a memória.

4. **Esperando (ou Bloqueado):** O processo não pode continuar sua execução no momento. Pode estar aguardando a conclusão de uma operação de entrada/saída (E/S), como ler ou escrever dados de um disco.

5. **Finalizado:** O processo terminou sua execução e o SO libera todos os recursos que ele estava utilizando.

Esse ciclo é contínuo e dinâmico. Quando um processo está executando, ele pode ser interrompido (por exemplo, por um processo de maior prioridade ou por um erro) e passar para o estado de esperando ou finalizado.



Transições de Estado do Processo

Embora os estados de um processo representem diferentes momentos da execução, é fundamental compreender que esses estados **não são fixos**, mas sim parte de um **fluxo dinâmico e contínuo**.

A mudança de um estado para outro ocorre em função de eventos específicos controlados pelo sistema operacional.

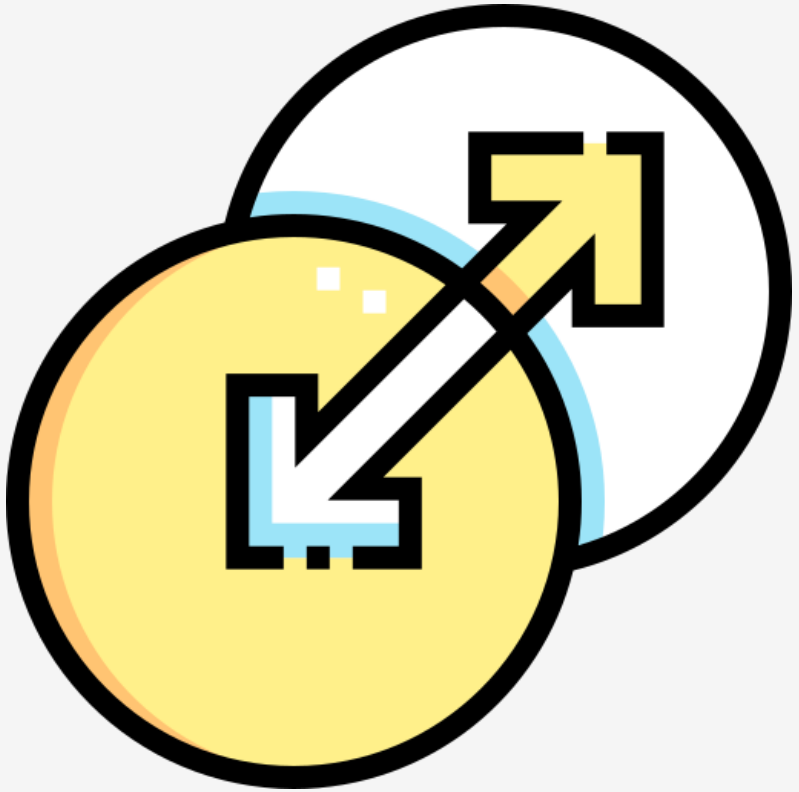
Quando um processo está em estado de **pronto**, ele pode ser direcionado para execução quando a **CPU é disponibilizada pelo escalonador**.

Durante a execução, o processo pode ser **interrompido**, retornando ao estado de pronto, geralmente devido à necessidade de compartilhamento da CPU com outros processos.

Além disso, um processo pode entrar em estado de **espera (bloqueado)** ao depender de operações externas, como acesso a disco, entrada de dados ou comunicação com outros dispositivos.

Uma vez que essa operação é concluída, o processo retorna ao estado de **pronto**, aguardando novamente sua vez de execução.

Esse ciclo de transições demonstra que a execução de um processo não é linear, mas sim **intercalada**, **controlada e dependente de múltiplos fatores internos e externos ao sistema**.



Exemplo Real do Ciclo de Execução

Para compreender de forma concreta o funcionamento dos estados de um processo, é importante analisar uma situação real de execução dentro de um sistema computacional.

Considere a ação de abrir um navegador e acessar uma página na internet.

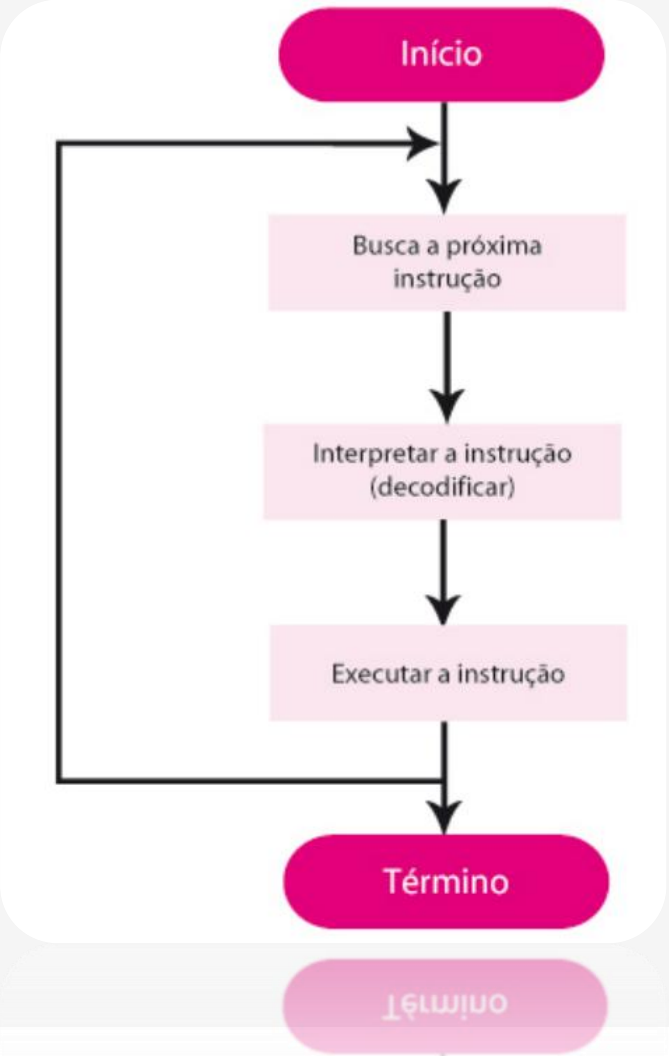
Inicialmente, o processo é criado e entra no estado de **pronto**, aguardando a disponibilidade da CPU.

Ao ser selecionado pelo sistema operacional, ele passa para o estado de **executando**, iniciando a carga dos dados necessários para exibir a página.

Durante esse processo, ao solicitar informações externas, como dados vindos da rede, o processo entra em estado de **espera**, pois depende de uma operação de entrada/saída.

Assim que os dados são recebidos, o processo retorna ao estado de **pronto** e, posteriormente, volta a ser **executado**, continuando sua tarefa até a conclusão.

Esse exemplo evidencia que a execução de um processo ocorre de forma **interrompida, intercalada e dependente de eventos externos**, e não como uma sequência contínua e ininterrupta.



Escalonamento de Processos

Definição:

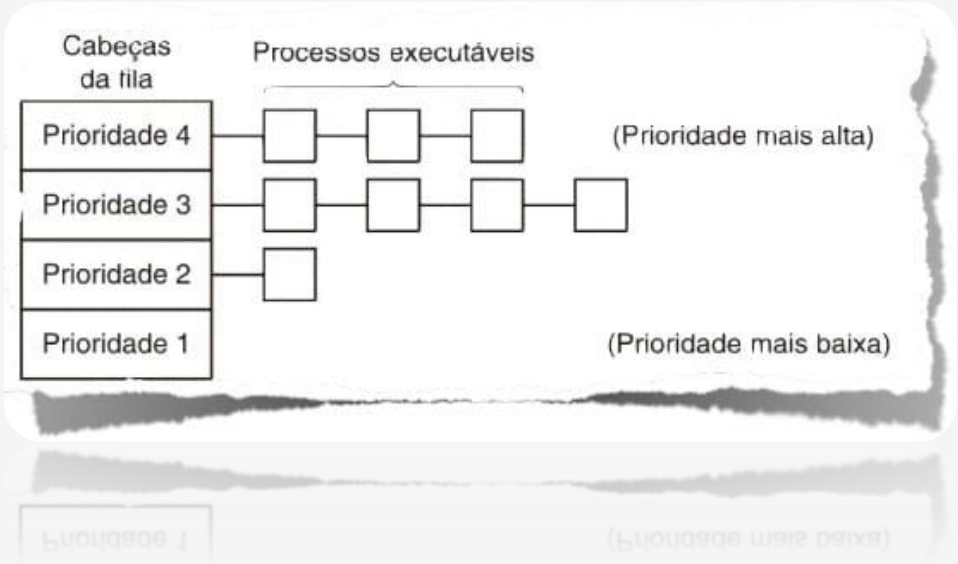
O escalonamento de processos é o processo usado pelo sistema operacional para decidir qual processo será executado pela CPU em um determinado momento.

Objetivos do Escalonamento:

- **Eficiência:** Garantir que a CPU seja usada da melhor maneira possível, evitando períodos de inatividade.
- **Justiça:** Garantir que todos os processos tenham a chance de ser executados de forma justa, sem que nenhum processo fique esperando eternamente (a não ser que seja necessário por algum motivo técnico).

Tipos de Escalonamento:

- **Preemptivo:** O escalonador pode interromper um processo em execução para iniciar outro, geralmente com base em prioridades ou quantidade de tempo de CPU utilizada.
- **Não Preemptivo:** O processo em execução só é interrompido quando termina sua execução ou passa para um estado que não pode continuar, como esperando ou finalizado.



Critérios De Escalonamento

O escalonamento de processos não se resume apenas à escolha de qual processo será executado, mas envolve a definição de **critérios que orientam essa decisão**.

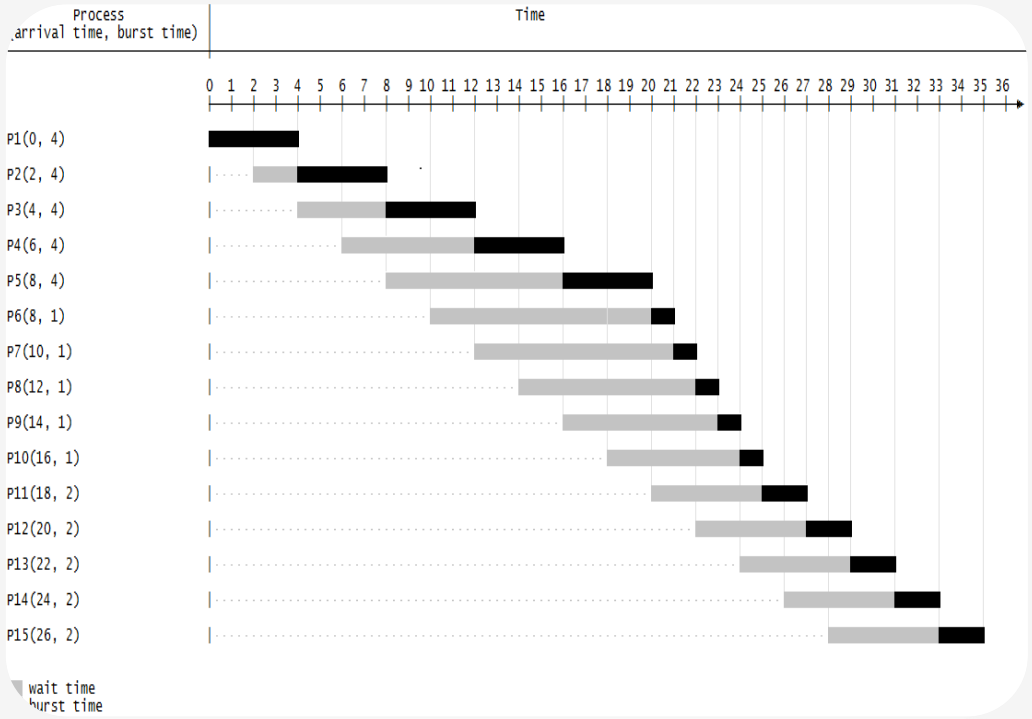
Diferentes algoritmos de escalonamento podem produzir resultados distintos, e, por isso, é necessário avaliar o desempenho do sistema com base em métricas específicas.

Entre os principais critérios utilizados, destaca-se o **tempo de espera**, que representa o período em que um processo permanece na fila aguardando execução.

Outro critério relevante é o **tempo de resposta**, que indica o intervalo entre a solicitação de execução e o início efetivo do processamento, sendo especialmente importante em sistemas interativos.

Além disso, o **tempo de turnaround** corresponde ao tempo total de vida do processo, desde sua criação até sua finalização, refletindo a eficiência global do sistema.

A análise desses critérios permite compreender que não existe um único algoritmo ideal, mas sim diferentes abordagens que priorizam aspectos distintos, como **desempenho, justiça ou tempo de resposta**.



Exemplo Prático de Escalonamento

Para compreender de forma concreta o impacto dos algoritmos de escalonamento, é necessário analisar um cenário prático envolvendo múltiplos processos.

Considere um conjunto de processos com diferentes tempos de execução:

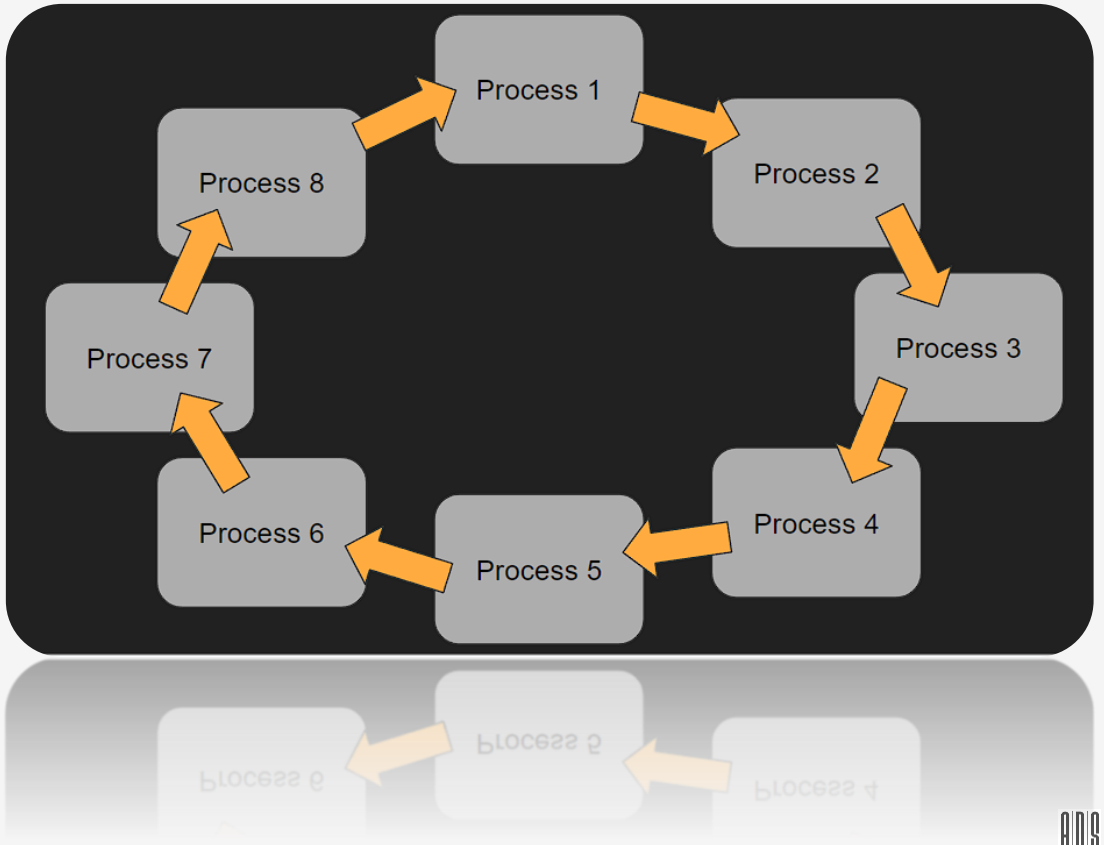
- Processo **P1** com tempo de CPU igual a **5 unidades**
- Processo **P2** com tempo de CPU igual a **3 unidades**
- Processo **P3** com tempo de CPU igual a **8 unidades**

Ao aplicar um algoritmo como o **FIFO (First In, First Out)**, os processos são executados na ordem de chegada, sem considerar o tempo de execução de cada um.

Nesse caso, processos com menor tempo podem acabar aguardando por longos períodos, evidenciando uma possível **ineficiência no tempo de espera**.

Por outro lado, ao utilizar o **Round Robin**, cada processo recebe uma fração de tempo da CPU (quantum), permitindo uma execução mais equilibrada e justa entre os processos.

Essa comparação demonstra que diferentes algoritmos produzem comportamentos distintos, impactando diretamente métricas como tempo de espera, **tempo de resposta e eficiência do sistema**.



Algoritmos de Escalonamento

1. **FIFO (First In, First Out):**

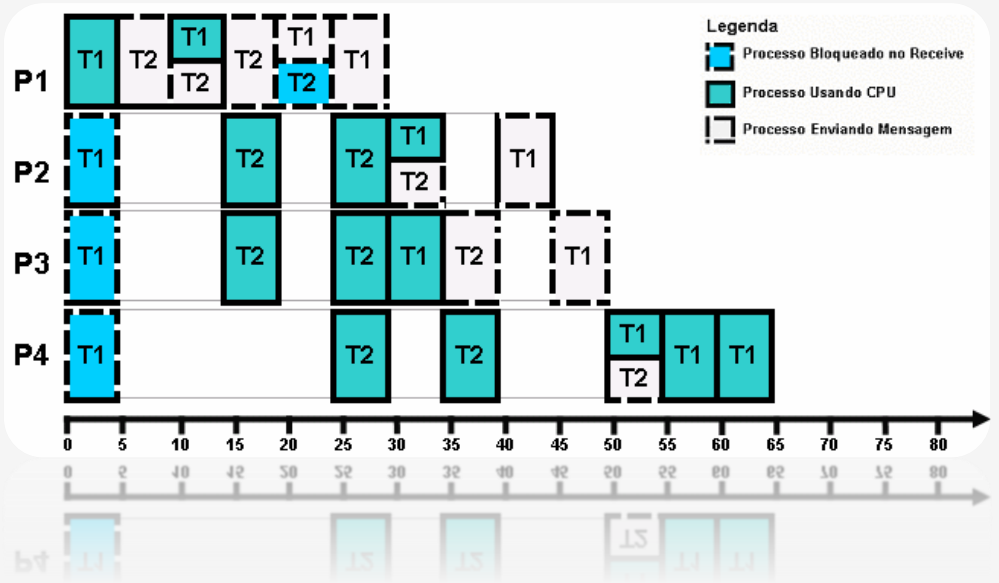
- ❖ O primeiro processo a entrar é o primeiro a sair.
- ❖ Simples de implementar, mas pode resultar em processos pequenos ficando presos atrás de processos grandes (convoy effect).

2. **Round Robin:**

- ❖ Cada processo recebe um pedaço de tempo (quantum) para ser executado. Quando o tempo se esgota, o processo é interrompido e colocado no final da fila de prontos.
- ❖ Ideal para sistemas interativos, como servidores ou interfaces gráficas.

3. **Prioridade:**

- ❖ Cada processo recebe uma prioridade. O processo com maior prioridade é executado primeiro.
- ❖ Pode ser preemptivo (interrompe processos com prioridade menor) ou não preemptivo.



Comparação Entre Algoritmos de Escalonamento

A análise dos algoritmos de escalonamento evidencia que cada abordagem apresenta **vantagens e limitações**, dependendo do contexto em que é aplicada.

O algoritmo **FIFO (First In, First Out)** se caracteriza pela sua simplicidade, executando os processos na ordem de chegada. No entanto, essa abordagem pode gerar situações de **injustiça**, especialmente quando processos curtos ficam aguardando a finalização de processos mais longos.

O **Round Robin**, por sua vez, introduz o conceito de compartilhamento da CPU por meio de um **quantum de tempo**, promovendo maior **equilíbrio entre os processos**.

Apesar disso, o aumento na quantidade de trocas de contexto pode gerar **sobrecarga no sistema**, impactando o desempenho.

Já o escalonamento por **prioridade** permite que processos mais importantes sejam executados primeiro, o que pode ser essencial em determinados cenários.

Entretanto, essa abordagem pode levar ao problema de **starvation**, no qual processos de baixa prioridade permanecem indefinidamente sem execução.

Dessa forma, a escolha do algoritmo de escalonamento envolve um equilíbrio entre **desempenho, justiça e complexidade**, não existindo uma solução única ideal para todos os sistemas.

Atividade em Sala – Escalonamento de Processos

1. Dividir a turma em 3 grupos.
2. Cada grupo pesquisa rapidamente (pode ser em celular/notebook, se tiver), para discute em sala sobre um **tipo de escalonamento:**
   - ❖ Grupo 1: FIFO (First In, First Out)
   - ❖ Grupo 2: Round Robin
   - ❖ Grupo 3: Escalonamento por Prioridade
3. Cada grupo deve anotar:
   - ❖ Definição do algoritmo.
   - ❖ Principais características (vantagens e desvantagens).
   - ❖ Exemplo de sistema operacional que utiliza (ou utilizou) esse algoritmo.

Troca de Contexto (Context Switch)

Definição:

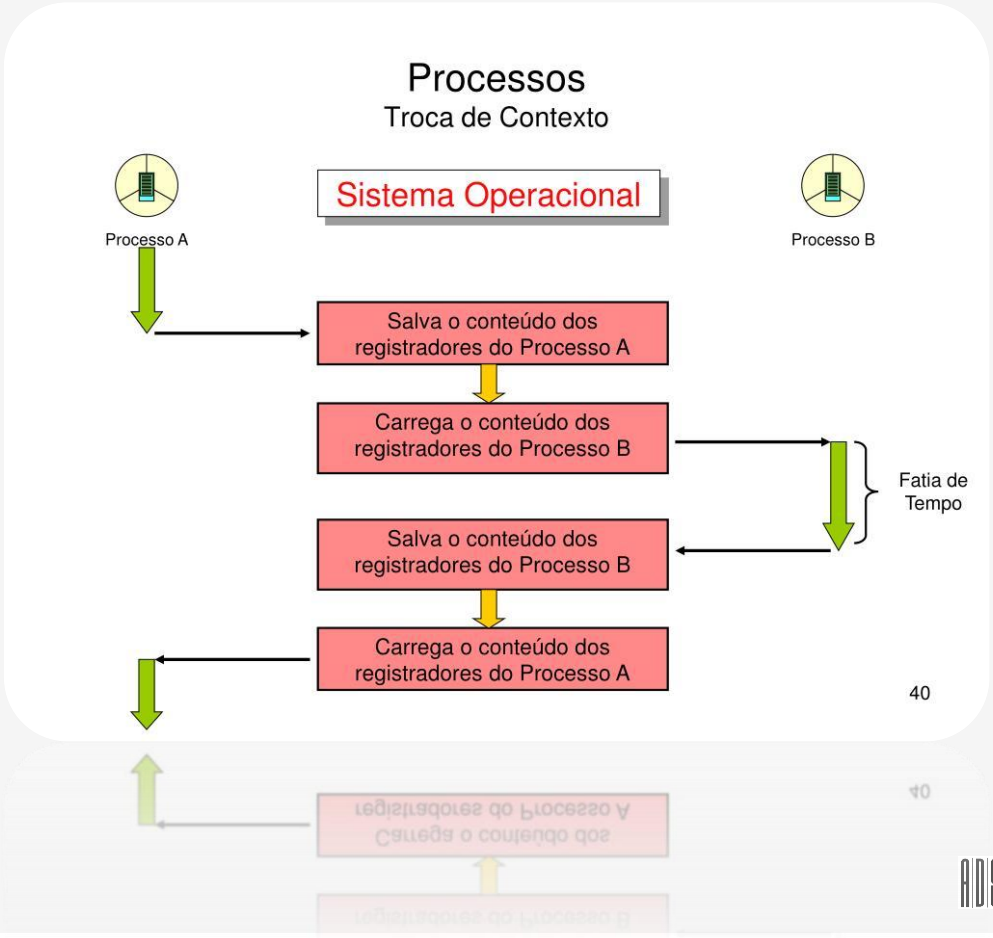
A troca de contexto ocorre quando o SO interrompe a execução de um processo e salva seu estado atual (contexto), para retomar a execução de outro processo. Isso é essencial para garantir que os processos sejam executados de maneira eficiente e sem falhas.

Processo de Troca de Contexto:

1. O SO salva o estado do processo que está sendo interrompido (seu contexto).
2. O SO carrega o estado do próximo processo que será executado.
3. O novo processo começa a execução.

Impacto:

Trocas de contexto são caras em termos de desempenho, pois envolvem salvar e carregar registros, o que consome tempo de CPU. Por isso, o SO tenta minimizar a quantidade de trocas de contexto.



O Que é Salvo na Troca de Contexto

A troca de contexto envolve a interrupção de um processo em execução e a posterior retomada de outro. Para que isso ocorra de forma correta, o sistema operacional precisa preservar todas as informações necessárias para que o processo interrompido possa continuar sua execução posteriormente.

Esse conjunto de informações corresponde ao **contexto do processo**, que é armazenado no momento da interrupção.

Entre os principais elementos que compõem esse contexto, destacam-se os **registradores da CPU**, que armazenam valores temporários utilizados durante a execução das instruções.

Além disso, o **contador de programa (Program Counter - PC)** indica qual será a próxima instrução a ser executada, sendo essencial para garantir a continuidade correta do processo.

Também são preservadas informações relacionadas ao **estado do processo** e aos **dados de memória**, permitindo que a execução seja retomada exatamente do ponto onde foi interrompida.

Esse mecanismo evidencia que a execução de processos não ocorre de forma contínua, mas sim por meio de **interrupções controladas e retomadas precisas**, coordenadas pelo sistema operacional.

Impacto da Troca de Contexto no Desempenho

Embora a troca de contexto seja essencial para permitir a execução concorrente de múltiplos processos, esse mecanismo não ocorre sem custo.

Cada troca de contexto exige que o sistema operacional **interrompa o processo atual**, salve seu estado e carregue o contexto de outro processo, o que demanda tempo e utilização da CPU.

Esse custo é conhecido como **overhead**, representando um tempo em que a CPU não está executando diretamente instruções úteis dos processos, mas sim realizando operações de gerenciamento.

Em sistemas com muitos processos ou com frequentes trocas de contexto, esse overhead pode se tornar significativo, impactando negativamente o desempenho geral.

Além disso, quanto menor for o tempo de execução de cada processo antes da troca (como em quantuns muito pequenos no Round Robin), maior será a quantidade de trocas de contexto, aumentando ainda mais esse custo.

Dessa forma, o sistema operacional precisa encontrar um equilíbrio entre **responsividade** e **eficiência**, evitando tanto a monopolização da CPU quanto o excesso de trocas de contexto.

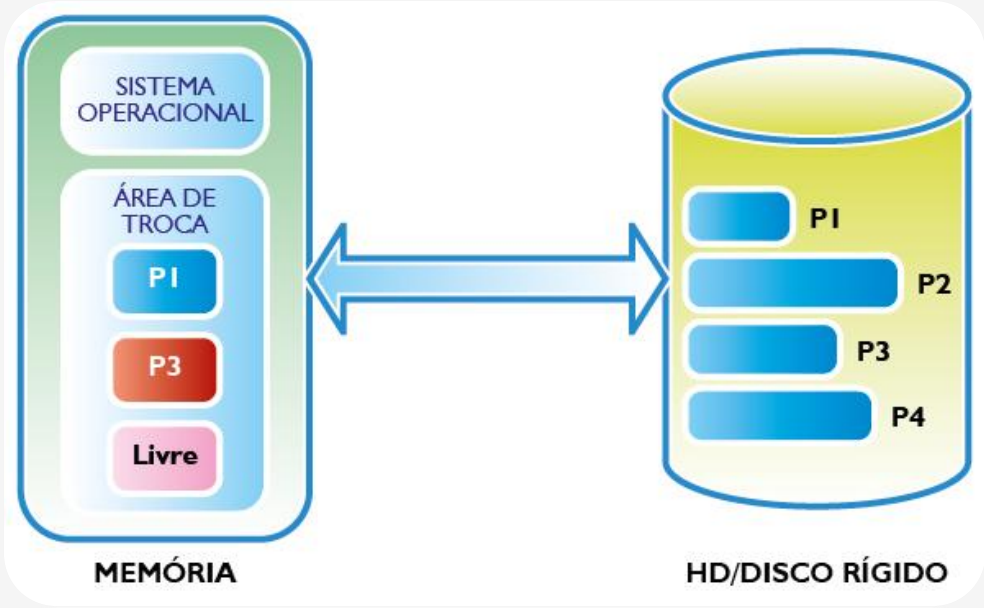
Gerenciamento de Memória

Definição:

O **gerenciamento de memória** refere-se ao processo de alocar e liberar espaço na memória (RAM) para os processos em execução. O SO deve garantir que a memória seja utilizada de forma eficiente e que cada processo tenha acesso à memória de forma isolada.

Objetivos do Gerenciamento de Memória:

- **Otimização:** Garantir que a memória seja usada de forma eficiente, evitando desperdícios.
- **Isolamento:** Evitar que um processo sobrescreva dados de outro processo. Isso é crucial para a estabilidade e segurança do sistema.



Tipos de Memória no Sistema

Para compreender o gerenciamento de memória, é necessário reconhecer que o sistema computacional não utiliza apenas um único tipo de memória, mas sim uma **hierarquia de memórias**, cada uma com características específicas de desempenho, capacidade e custo.

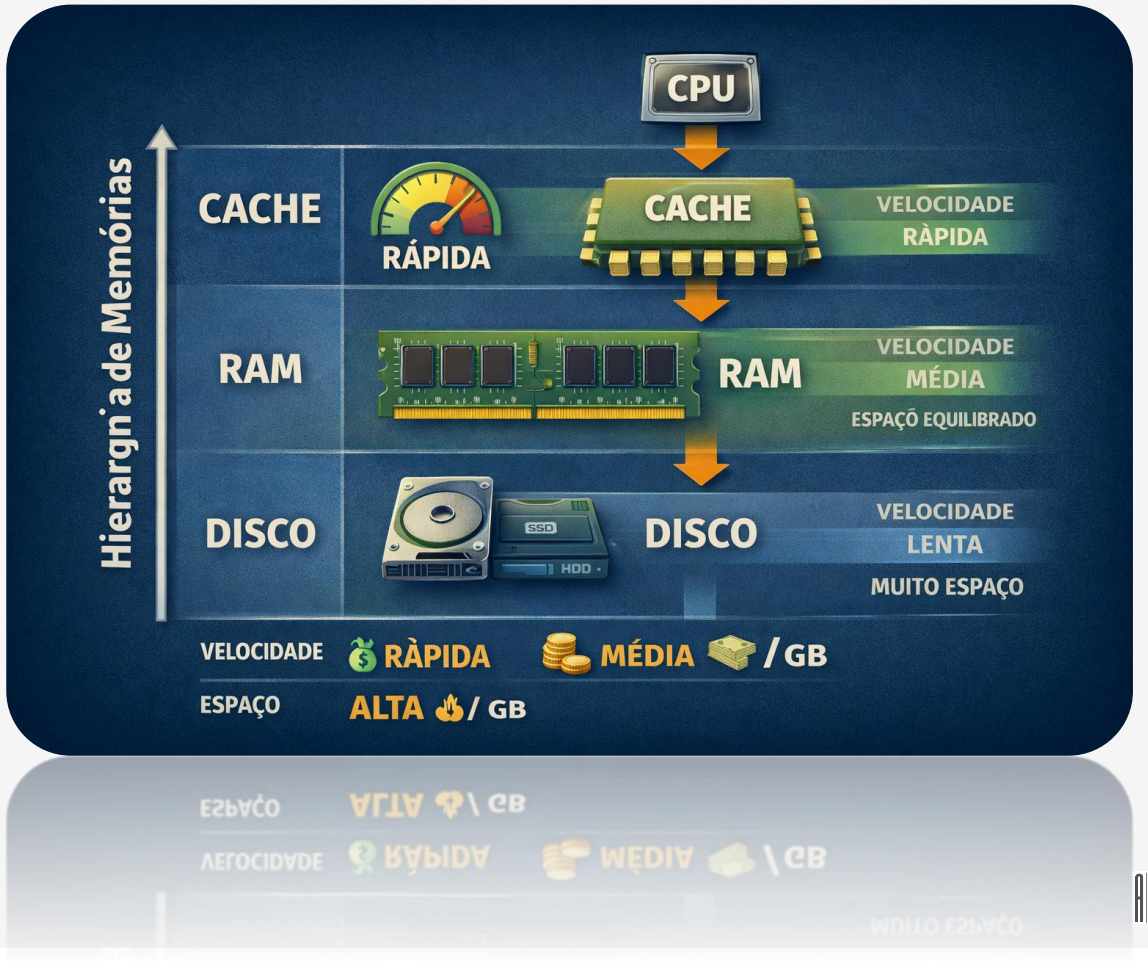
Entre os principais tipos, destaca-se a **memória cache**, localizada mais próxima da CPU e responsável por armazenar dados frequentemente acessados, permitindo um acesso extremamente rápido.

A **memória principal (RAM)** é utilizada para armazenar os processos em execução, sendo o principal espaço de trabalho do sistema operacional.

Já a **memória secundária**, como o disco rígido ou SSD, apresenta maior capacidade de armazenamento, porém com tempos de acesso significativamente mais elevados.

Essa organização hierárquica permite que o sistema combine **velocidade e capacidade**, utilizando memórias rápidas para operações imediatas e memórias maiores para armazenamento de longo prazo.

Dessa forma, o gerenciamento de memória não envolve apenas a alocação de espaço, mas também a **coordenação entre diferentes níveis de memória**, visando otimizar o desempenho do sistema.



Técnicas de Gerenciamento de Memória

1. Particionamento Fixo:

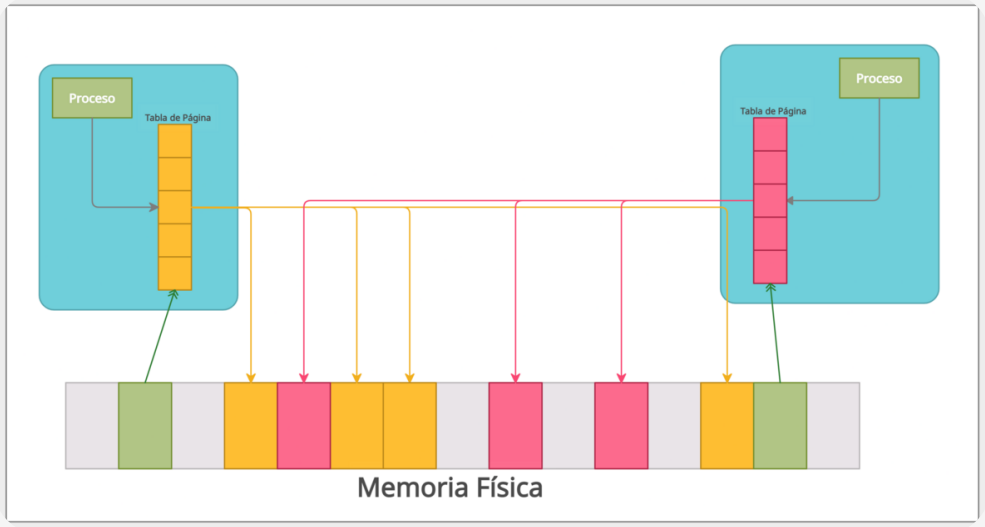
- A memória é dividida em blocos de tamanho fixo. Cada processo é alocado em um desses blocos.
- Problema: Fragmentação interna, onde processos menores ocupam um bloco maior do que o necessário.

2. Particionamento Dinâmico:

- A memória é dividida dinamicamente, conforme a necessidade de cada processo. Isso permite uma alocação mais eficiente, mas pode gerar fragmentação externa (espaços vazios na memória).

3. Memória Virtual:

- Técnica que utiliza parte do disco rígido como memória, permitindo que os processos utilizem mais memória do que a fisicamente disponível na RAM.
- Importante para permitir que programas grandes ou múltiplos programas sejam executados ao mesmo tempo, sem esgotar a memória RAM.



Fragmentação de Memória

O gerenciamento de memória, embora essencial para o funcionamento do sistema, pode gerar problemas relacionados ao uso ineficiente do espaço disponível.

Um desses problemas é conhecido como **fragmentação de memória**, que ocorre quando a memória não é utilizada de forma contínua, resultando em desperdício de espaço.

A **fragmentação interna** acontece quando um processo ocupa um bloco de memória maior do que o necessário, deixando parte desse espaço inutilizado dentro da própria alocação.

Já a **fragmentação externa** ocorre quando existem pequenos espaços livres distribuídos na memória, mas que não podem ser utilizados para alocar novos processos maiores, mesmo que a soma total de memória livre seja suficiente.

Esse problema evidencia que não basta haver memória disponível, é necessário que ela esteja **organizada de forma adequada** para permitir sua utilização eficiente.

Dessa forma, o sistema operacional precisa adotar estratégias que minimizem a fragmentação, garantindo melhor aproveitamento dos recursos de memória.

Paginação (Memória Virtual)

Diante dos problemas causados pela fragmentação de memória, surge a necessidade de técnicas que permitam uma utilização mais eficiente do espaço disponível.

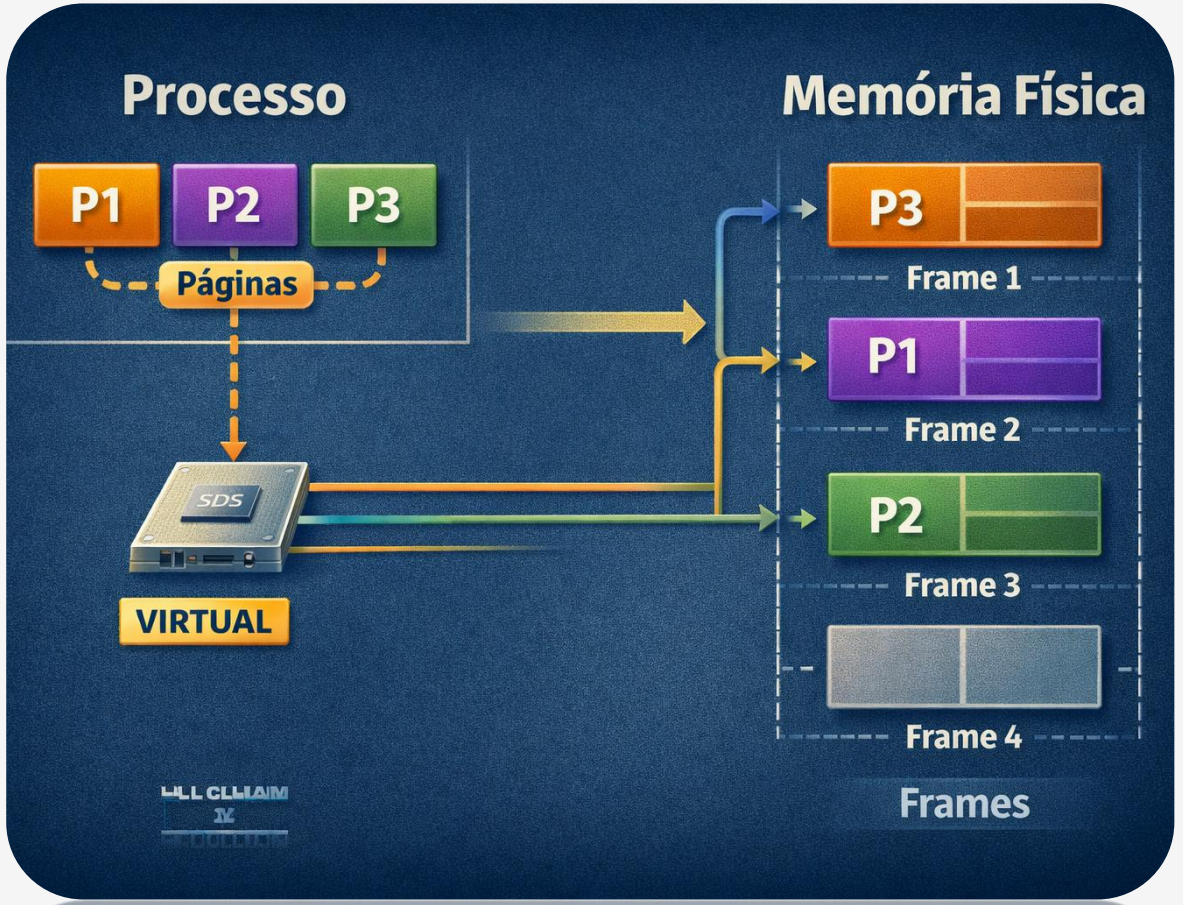
Uma das principais soluções adotadas pelos sistemas operacionais modernos é a **paginação**, que consiste na divisão da memória em unidades de tamanho fixo.

Nesse modelo, a memória física é dividida em **frames (quadros)**, enquanto os processos são divididos em **páginas**, ambas com o mesmo tamanho.

Essa abordagem permite que as páginas de um processo sejam armazenadas em diferentes regiões da memória, eliminando a necessidade de alocação contígua.

Dessa forma, a paginação reduz significativamente o problema da **fragmentação externa**, permitindo um melhor aproveitamento da memória disponível.

Além disso, essa técnica possibilita a implementação da **memória virtual**, na qual parte dos dados pode ser armazenada temporariamente fora da memória principal, ampliando a capacidade de execução do sistema.



Page Fault (Falha De Página)

Embora a paginação permita uma utilização mais eficiente da memória, nem todas as páginas de um processo permanecem carregadas na memória principal o tempo todo.

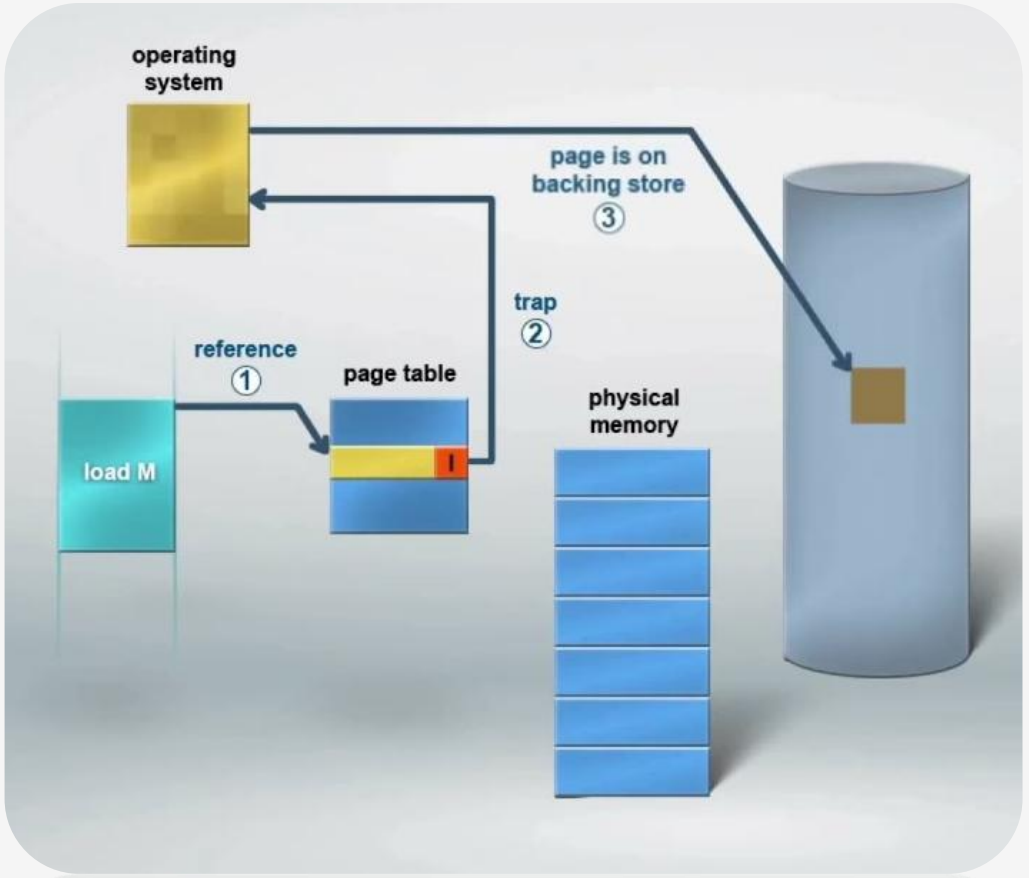
Quando um processo tenta acessar uma página que não está presente na memória RAM, ocorre um evento conhecido como **page fault (falha de página)**.

Nesse momento, o sistema operacional precisa **interromper a execução do processo** e localizar essa página na memória secundária, como o disco.

Após encontrar a página, o sistema realiza sua transferência para a memória principal, atualizando as estruturas de controle e permitindo que o processo retome sua execução.

Esse processo, embora essencial, é significativamente mais lento, pois envolve acesso a dispositivos de armazenamento, que possuem tempos de resposta muito superiores à memória RAM.

Dessa forma, a ocorrência frequente de page faults pode impactar diretamente o desempenho do sistema, tornando a execução mais lenta.



Relação entre Processos e Memória

Interdependência CPU + RAM:

A CPU processa os dados dos processos, mas precisa que esses dados estejam carregados na memória RAM. Sem memória suficiente ou eficiente, o desempenho da CPU pode ser drasticamente afetado.

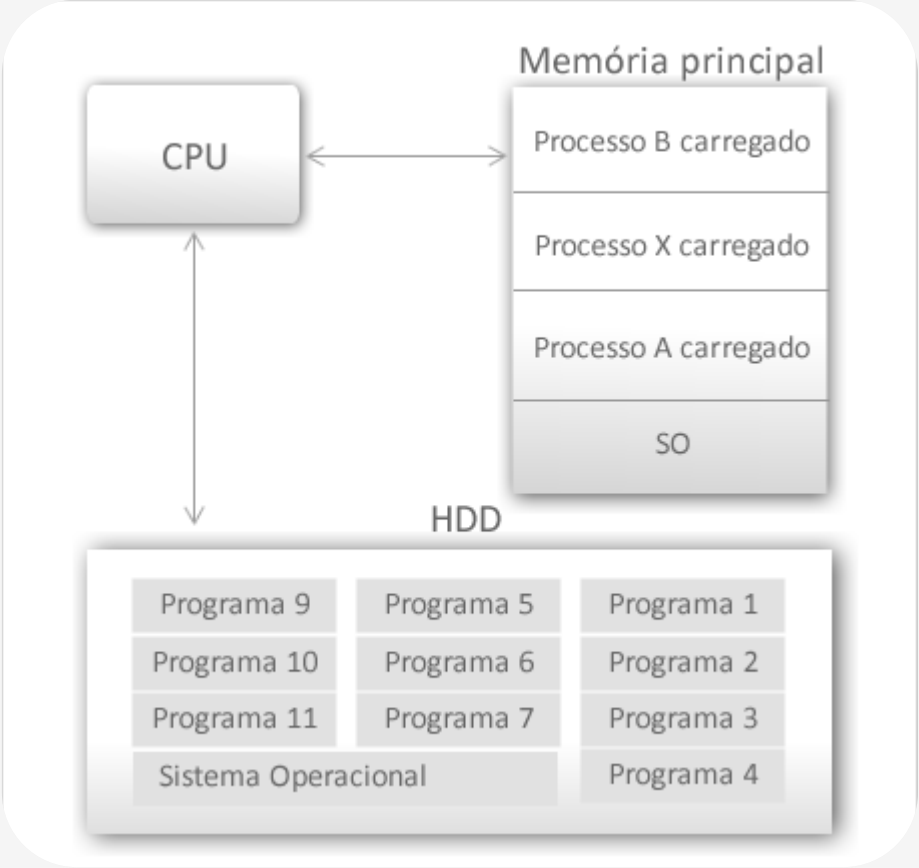
Problemas comuns:

1. Deadlock:

- Quando dois ou mais processos ficam esperando indefinidamente por recursos uns dos outros, criando um ciclo sem fim.

2. Thrashing:

- Quando o sistema começa a trocar processos entre a RAM e o disco de forma excessiva (geralmente devido à falta de memória), tornando o sistema extremamente lento.



Deadlock (Interbloqueio)

O **deadlock**, também conhecido como **interbloqueio**, é uma situação em que dois ou mais processos ficam **indefinidamente bloqueados**, aguardando recursos que nunca serão liberados.

Esse problema ocorre quando há uma dependência circular entre processos, impedindo que qualquer um deles avance em sua execução.

Para que o deadlock aconteça, é necessário que quatro condições ocorram simultaneamente.

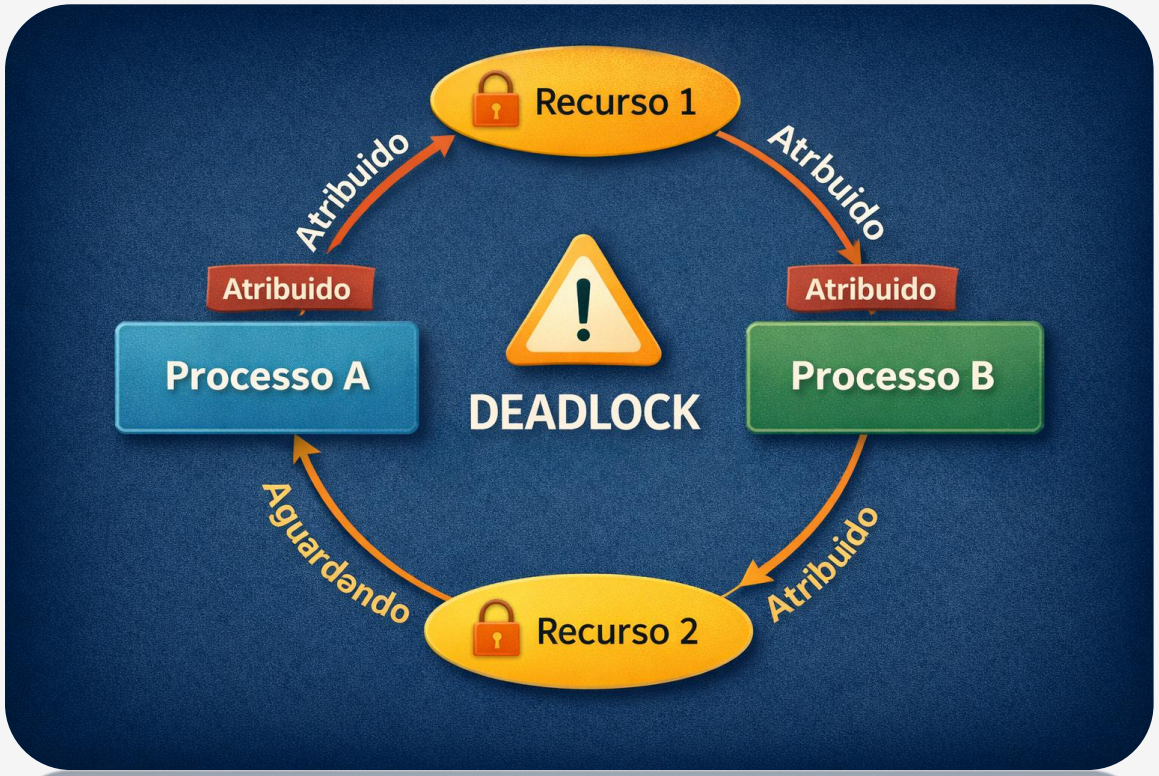
A primeira é a **exclusão mútua**, na qual um recurso só pode ser utilizado por um processo por vez.

A segunda é a **posse e espera (hold and wait)**, em que um processo mantém recursos já adquiridos enquanto aguarda por novos.

A terceira condição é a **não preempção**, que impede que o sistema operacional retire à força um recurso de um processo.

Por fim, a **espera circular** ocorre quando existe uma cadeia de processos, na qual cada um aguarda por um recurso que está sendo utilizado pelo próximo.

Quando essas condições se manifestam simultaneamente, o sistema entra em um estado no qual nenhum dos processos consegue prosseguir, comprometendo a execução.



Thrashing (Colapso de Memória)

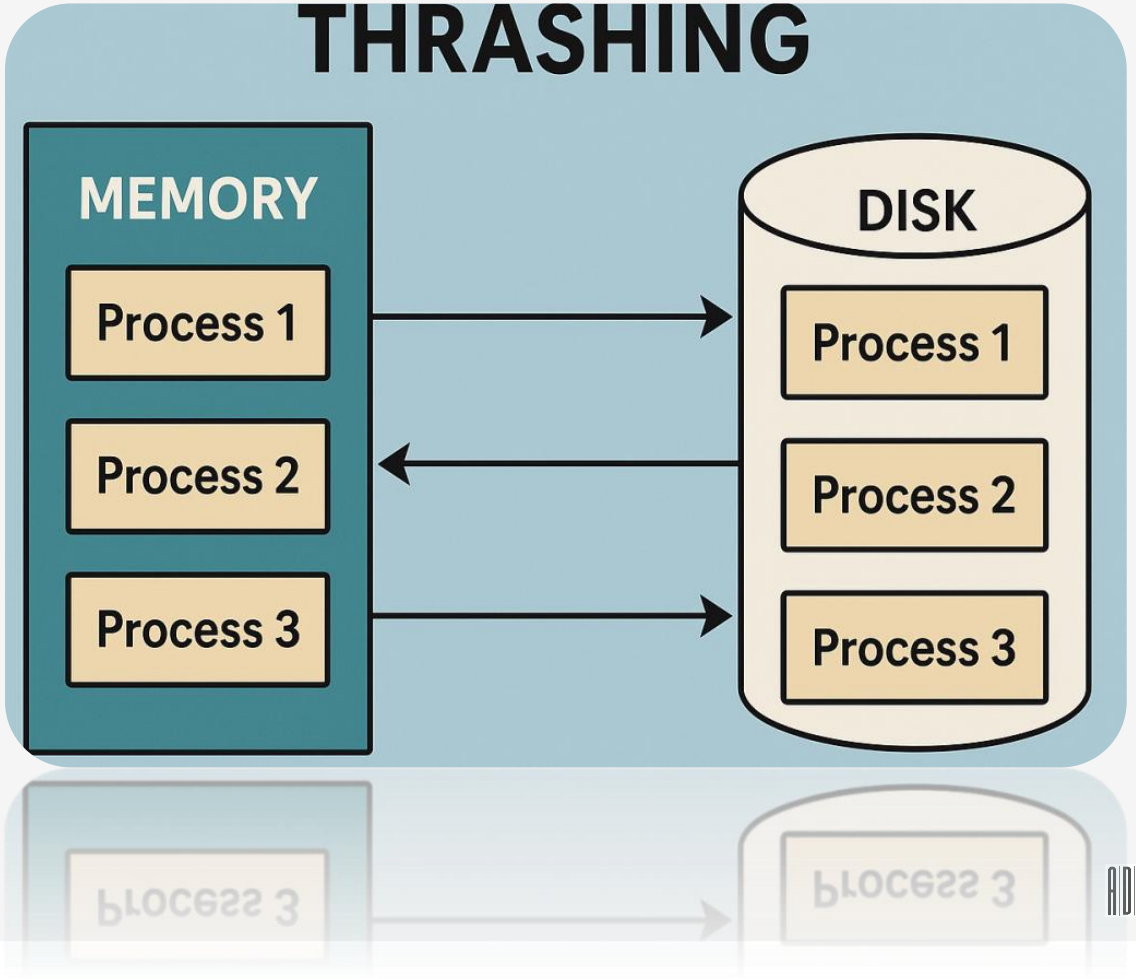
O **thrashing** é uma situação crítica em que o sistema operacional passa a utilizar a maior parte do tempo realizando **trocas de páginas entre a memória RAM e o disco**, em vez de executar efetivamente os processos.

Esse problema ocorre quando há uma quantidade excessiva de processos ativos ou quando a memória disponível é insuficiente para manter as páginas necessárias em execução.

Como consequência, o sistema entra em um ciclo contínuo de **page faults**, no qual páginas são constantemente removidas e recarregadas, sem que o processamento avance de forma significativa.

Nesse cenário, a CPU permanece ocupada com operações de gerenciamento de memória, enquanto o desempenho geral do sistema é drasticamente reduzido.

O thrashing evidencia que o uso excessivo de memória virtual, sem controle adequado, pode comprometer completamente a eficiência do sistema.



Integração: Processo, CPU e Memória

O funcionamento de um sistema computacional não pode ser compreendido de forma isolada, pois envolve a **integração entre processos, CPU e memória**.

Os **processos** representam as tarefas em execução, sendo responsáveis por realizar operações e manipular dados dentro do sistema.

A **CPU**, por sua vez, atua como o elemento responsável pelo processamento dessas tarefas, executando instruções de forma sequencial e controlada.

No entanto, para que esse processamento ocorra de forma eficiente, é necessário que os dados estejam disponíveis na **memória principal (RAM)**, evidenciando a dependência entre esses componentes.

O sistema operacional atua como o elemento central dessa integração, sendo responsável por **gerenciar a execução dos processos, controlar o uso da CPU e organizar a alocação de memória**.

Problemas como **deadlock**, **thrashing** e **overhead de troca de contexto** demonstram que o equilíbrio entre esses elementos é essencial para o bom desempenho do sistema.

Dessa forma, compreender a interação entre esses componentes permite uma visão mais completa do funcionamento dos sistemas operacionais, indo além da análise isolada de cada conceito.