Sistemas Operacionais I

Criado em	@7 de outubro de 2024 08:38
Tags	

Conteúdos

- [x] ~~Mutex/Semáforos~~
- [x] ~~Monitores~~
- [x] ~~Deadlock~~
- [x] ~~Gerenciamento de Memória~~
- [x] ~~Permuta entre Processos (Swapping)~~
- [x] ~~Alocação de Memória Contígua~~
- [x] ~~Paginação~~
- [x] ~~Memória Virtual~~
- [x] ~~Escalonamento de CPU~~
- [x] ~~Escalonamento com Múltiplas CPUs~~

Permuta entre Processos (Swapping)

- A permuta de processos (swapping) é uma técnica de gerenciamento de memória utilizada por sistemas operacionais para otimizar o uso da memória física, evitando a sobrecarga;
- Esse processo envolve mover **temporariamente** processos **inativos ou que estão em espera** para o armazenamento secundário e, posteriormente, trazê-los de volta à memória principal quando estiverem prontos para retomar a execução;

- A permuta ajuda a liberar espaço na RAM para que outros processos possam ser executados, maximizando a utilização de recursos e mantendo a capacidade de multiprogramação.

Estratégias para Gerenciar Memória

- **Swapping (Permuta-Padrão)**: procedimento que traz o processo inteiro para a memória, executa-o por um período, e depois o move de volta para o disco;
- **Memória Virtual**: procedimento que permite que processos sejam executados parcialmente na memória principal, sem a necessidade de carregá-los integralmente.

Permuta-Padrão

A permuta-padrão envolve a troca **completa** de processos entre a memória principal e a memória de retaguarda:

- Quando o escalonador decide executar um processo que não está na memória principal, o despachante verifica se há uma região livre de memória. Se não houver, o sistema permuta um processo da memória com o processo desejado;
- A memória secundária precisa ser grande o suficiente para armazenar cópias de todos os processos e garantir acesso rápido para que esses dados sejam carregados novamente;
- A maior parte do **tempo de permuta** é gasta na **transferência** dos processos entre a memória principal e o disco, o que pode reduzir significativamente o desempenho;
- O sistema pode otimizar a permuta se souber exatamente a quantidade de memória que o processo está utilizando. **Chamadas de sistema**, como `request_memory()` e `release_memory()`, podem informar o sistema sobre as mudanças de necessidades de memória, minimizando o tempo de permuta.
- A permuta deve **ocorrer apenas** quando o processo está **completamente ocioso**, pois operações de I/O pendentes podem interferir.

Alocação de Memória Contígua

A memória principal deve acomodar tanto o sistema operacional quanto os diversos processos de usuário. Por isso, é preciso alocar a memória principal da maneira mais eficiente possível.

A memória é normalmente dividida em duas partições:

- Uma parte para o sistema operacional;
- E a outra parte para os processos de usuário.

Pode-se alocar o sistema operacional tanto na **memória baixa** quanto na **memória alta**.

O **principal fator** que afeta essa decisão é a localização do **vetor de interrupções** (uma estrutura de dados usada por sistemas operacionais para gerenciar e processar interrupções no computador). Como este vetor costuma estar na memória baixa, os programadores também costumam alocar o sistema operacional para a memória baixa.

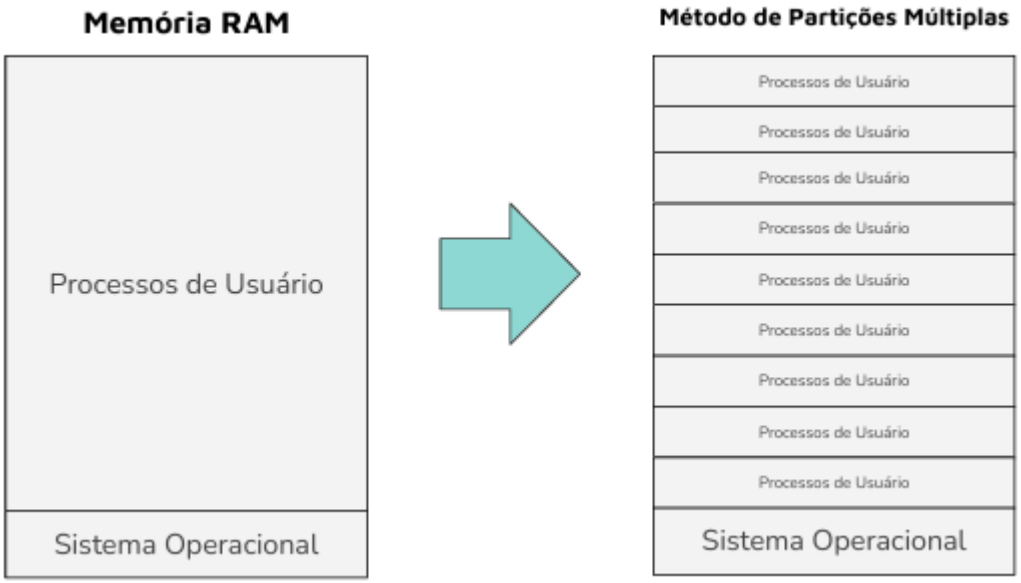
- A memória baixa refere-se à parte da memória RAM do computador que está alocada para o sistema operacional, geralmente destinada ao gerenciamento do próprio sistema e ao carregamento de programas essenciais.

Alocação de Memória

Um dos métodos mais simples para alocação de memória é separar a memória em várias partições de mesmo tamanho. Portanto, cada partição pode conter exatamente um processo, pois o grau de multiprogramação é limitado pelo número de partições.

Nesse método de partições múltiplas, quando uma delas está livre, um processo é selecionado na fila de entrada e carregado na partição que está disponível naquele momento.

Esse método é chamado de MFT, porém não está mais em uso.



- No esquema de partições variáveis, o sistema operacional mantém uma tabela indicando quais partes da memória estão disponíveis e quais estão ocupadas.;
- Inicialmente, toda a memória está disponível para processos de usuário e é considerada um grande bloco de memória disponível;
- Conforme os processos entram no sistema, eles são colocados em uma fila de entrada;

- O sistema operacional leva em consideração os requisitos de memória de cada processo e o montante de espaço disponível na memória para determinar que processos devem ter memória alocada;
- Quando um processo recebe espaço, ele é carregado na memória e pode, então, competir por tempo da CPU;
- Quando um processo termina, libera sua memória que o sistema operacional pode então preencher com outro processo da fila de entrada.

Brechas

Os blocos de memória disponíveis compõem um **conjunto de brechas** de vários tamanhos espalhadas pela memória. Quando um processo chega e precisa de memória, o sistema procura no conjunto por uma **brecha** que seja suficientemente grande para esse processo.

Esse procedimento é uma instância específica do problema de alocação de memória dinâmica geral que diz respeito a como satisfazer uma solicitação de tamanho n a partir de uma lista de brechas livres.

Há muitas soluções para esse problema, como as estratégias do Primeiro-apto (first-fit), Mais-apto (best-fit) e Menos-apto (worst-fit). Essas são as soluções mais usadas para selecionar uma brecha livre no conjunto de brechas disponíveis.

- **Primeiro-Apto (First-Fit)**
    - O sistema busca o **primeiro bloco** de memória livre que seja grande o suficiente para atender à necessidade do processo;
    - É uma **solução rápida**, pois o sistema não precisa explorar toda a memória;
    - Porém, pode gerar fragmentação externa, já que o processo de alocação é feito no primeiro lugar disponível, mesmo que existam blocos melhores para a tarefa.
- **Mais-Apto (Best-Fit)**

    - O sistema busca o **menor bloco** de memória livre que seja **suficientemente grande para o processo**;
    - **Reduz a quantidade de memória não utilizada** dentro dos blocos de memória alocados, maximizando a utilização da memória;
    - No entanto, pode resultar em **fragmentação mais severa** e processos mais lentos devido à necessidade de verificar toda a memória disponível para encontrar o bloco ideal.
- **Menos-Apto (Worst-Fit)**
    - O sistema escolhe o **maior bloco** de memória livre, na esperança de que, ao dividir o bloco para alocar o processo, o restante seja grande o suficiente para outros processos;
    - Pode **reduzir o número de pequenos fragmentos** ao final da alocação;
    - Porém, pode **gerar blocos de memória muito grandes e subutilizados**, além de **aumentar a fragmentação externa** ao longo do tempo.

OBS. Simulações têm mostrado que tanto o primeiro-apto quanto o mais-apto são melhores do que o menos-apto em termos de redução de tempo e utilização de memória.

Nem o primeiro-apto, nem o mais-apto é claramente melhor do que o outro em termos de utilização de memória, mas o primeiro-apto geralmente é mais rápido

Fragmentação

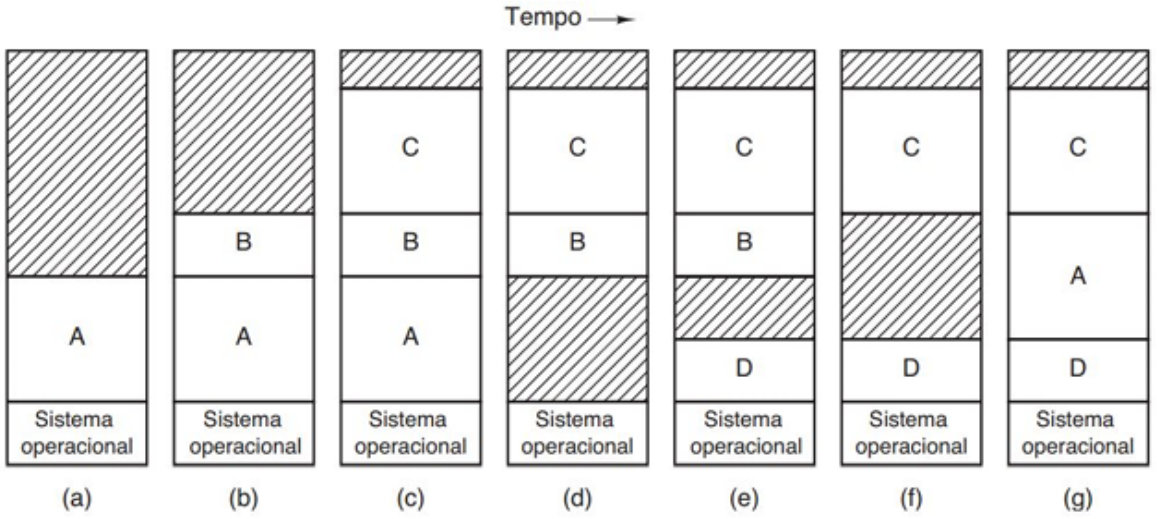
A fragmentação é um problema que ocorre quando **a memória disponível não pode ser alocada de forma eficiente**, resultando em espaços não utilizados ou subutilizados na memória.

Existem dois tipos principais de fragmentação:

- **Fragmentação Interna**

- Ocorre quando a memória alocada para um **processo é maior do que a memória realmente necessária**, resultando em espaço não utilizado dentro do bloco de memória alocado;
- Esse tipo de fragmentação é mais comum em sistemas de alocação fixa, onde os blocos têm tamanhos fixos.

- **Fragmentação Externa**:
  - Ocorre quando **há muitos blocos de memória livre, pequenos e dispersos**, mas **nenhum suficientemente grande** para alocar um novo processo;
  - Isso pode acontecer, por exemplo, em alocação **first-fit**, onde os processos são alocados nos primeiros blocos disponíveis, mas com o tempo, esses blocos se tornam pequenos demais para processos maiores.



Desfragmentação

Outra maneira de tratar a fragmentação externa consiste em desfragmentar a memória periodicamente.

Como áreas que não podem ser acessadas durante a desfragmentação, é importante que esse procedimento seja executado rapidamente e com pouca frequência.

Segmentação

A segmentação é um técnica de gerenciamento de memória que **divide o espaço de memória em segmentos de diferentes tamanhos**.

Cada segmento pode corresponder a d**iferentes partes de um programa, como código, dados e pilha**.

Ela **não utiliza blocos de tamanhos fixos**, diferentemente da paginação.

Características Principais

- Cada segmento tem um propósito específico (ele segue uma divisão lógica, separando-se em código, dados ou pilha);
- O tamanho de cada segmento pode variar de acordo com as necessidades do processo;
- Como os segmentos podem variar de tamanho dinamicamente, a segmentação é mais flexível em relação ao uso da memória;
- A segmentação pode levar a fragmentação externa, onde a memória livre é dividida em pequenas partes, o que pode dificultar a alocação de segmentos grandes.

Paginação

A paginação é uma técnica em que a memória é dividida em páginas de tamanho fixo, enquanto a memória física é dividida em quadros (frames) de igual tamanho.

Quando um processo precisa de memória, o sistema operacional aloca páginas em quadros de memórias disponíveis.

Diferentemente da segmentação, a paginação evita a fragmentação externa e a necessidade de compactação.

Ela também resolve o considerável problema de acomodar trechos de memória de vários tamanhos na memória de retaguarda (secundária).

Características Principais

- Tanto as páginas quanto os quadros de memória têm **tamanho fixo**, o que simplifica o gerenciamento e reduz a fragmentação interna;
- A **fragmentação interna** pode ocorrer dentro de uma página, caso a quantidade de dados do processo não ocupe totalmente o tamanho da página;
- A **fragmentação externa** é minimizada, pois a memória é gerenciada em blocos fixos.

Características Específicas do Slide

- A paginação pode usar dispositivos de armazenamento secundário como extensão da memória RAM;
- Partes ociosas da memória podem ser transferidas para um disco, liberando a memória RAM para outros usos;
- Caso algum processo tente acessar esse conteúdo posteriormente, ele deverá ser trazido de volta à memória RAM;
- A transferência de dados entre memória e disco é feita pelo sistema operacional, de forma transparente para os processos.

A ideia central da paginação em disco consiste em transmitir páginas ociosas da memória RAM para outra área em disco, liberando memória para outras páginas.

Modelo Básico

O modelo básico de paginação envolve a fragmentação da memória em **páginas**e **quadros de memória**.

Quando um processo é carregado na memória, suas páginas são distribuídas entre os quadros disponíveis.

Como funciona

1. Na memória lógica, o processo é fragmentado em páginas de tamanho fixo, e cada página tem um número que a identifica.
2. A memória física também é fragmentada em quadros de tamanho fixo. Quando o processo é carregado, as páginas são mapeadas para quadros.

OBS. O sistema usa uma **tabela de páginas** para mapear a relação entre as páginas do processo e os quadros de memória físicos. Cada entrada da tabela de páginas contém o número do quadro físico onde a página está armazenada.

Memória lógica = fragmentada em páginas do mesmo tamanho;

Memória física = fragmentada em quadros do mesmo tamanho.

Quando um processo está para ser executado, suas páginas são carregadas em quaisquer quadros de memória disponíveis a partir de sua origem (um sistema de arquivos ou a memória de secundária).

Suporte de Hardware à Paginação

- Cada endereço gerado pela CPU é dividido em duas partes: um número de página (p) e um deslocamento de página (d);
- O número de páginas é usado como índice em uma tabela de páginas;
- A tabela de páginas contém o endereço base de cada página na memória física;
- Esse endereço base é combinado com o deslocamento de página para definir o endereço de memória física que é enviado à unidade de memória.

Critérios de Seleção

- Idade da página: Há quanto tempo a página está na memória; páginas muito antigas talvez sejam pouco usadas (Página mais antiga = FIFO);
- Frequência de acessos à página: páginas mais acessadas em um passado recente possivelmente ainda o serão em um futuro próximo;
- Data do último acesso: páginas há mais tempo sem acessar possivelmente serão pouco acessadas em um futuro próximo (sobretudo se os processos respeitarem o princípio da localidade de referências);
- Prioridade do processo proprietário: processos de alta prioridade, ou de tempo real, podem precisar de suas páginas de memória rapidamente; se elas estiverem no disco, seu desempenho ou tempo de resposta poderão ser prejudicados;
- Conteúdo da página: páginas cujo conteúdo seja código executável exigem menos esforço do mecanismo de paginação, porque seu conteúdo já está mapeado no disco (dentro do arquivo executável correspondente ao processo). Por outro lado, páginas de dados que tenham sido alteradas precisam ser salvas na área de troca;
- Páginas especiais: páginas contendo buffers de operações de entrada/saída podem ocasionar dificuldades ao núcleo caso não estejam na memória no momento em que ocorrer a transferência de dados entre o processo e o dispositivo físico. O processo também pode solicitar que certas páginas contendo informações sensíveis (como senhas ou chaves criptográficas) não sejam copiadas na área de troca, por segurança.

Memória Virtual

A memória virtual é uma técnica que permite a execução de processos que não estão completamente na memória, possibilitando que programas maiores que a memória física sejam executados.

A memória virtual abstrai a memória principal em um array de armazenamento uniforme extremamente grande, separando a memória lógica da memória física.

> → Essa técnica deixa os programadores livres de preocupações com as limitações de armazenamento de memória, tornando a tarefa de programar muito mais fácil.

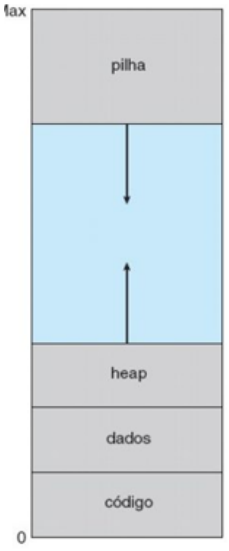
A memória virtual também permite que os processos compartilhem arquivos facilmente e implementem a memória compartilhada. Além disso, ela fornece um mecanismo eficiente para a criação de processos.

OBS. No entanto, a memória virtual não é fácil de implementar e pode piorar o desempenho se não for usada cautelosamente.

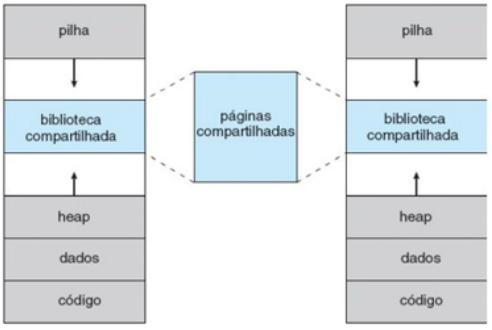
Características Importantes

- **Espaço de Endereçamento Virtual**
  - O espaço de endereçamento virtual de um processo representa a “visão lógica” que o sistema tem da memória;
  - O processo enxerga a memória de forma contígua e não se preocupa e não se preocupa com a disposição física dos dados da RAM;
  - A separação entre a memória lógica (endereço virtual) e a memória física (endereço real) permite que programas utilizem uma quantidade de memória além dos limites físicos do sistema.

- **Estrutura do Espaçamento de Endereço**
  - Normalmente, o espaço de endereçamento é estruturado para que o heap e a pilha possam crescer em direções opostas, permitindo a expansão dinâmica desses segmentos conforme necessário;
  - A lacuna entre o heap e a pilha forma uma área de endereçamento "esparsa", preenchida com páginas conforme a necessidade de alocação de memória dinâmica ou chamadas de função;
  - Esse design maximiza o uso da memória virtual, especialmente para processos que carregam bibliotecas dinamicamente.



- **Compartilhamento de Memória e Arquivos**
  - Uma das vantagens é a possibilidade de compartilhamento entre processos;
  - Bibliotecas de sistemas, por exemplo, podem ser mapeadas para espaços de endereçamento virtual de diferentes processos;
  - Essa capacidade reduz o consumo de memória ao permitir que vários processos compartilhem a mesma área de memória para dados comuns, como bibliotecas de código;



    - Essa estrutura também facilita a comunicação entre processos que compartilham memória.

Paginação por Demanda

A paginação é o método mais comum para implementar memória virtual. Ela fragmenta a memória em blocos de tamanho fixo, chamados de páginas (no espaço lógico) e quadros (na memória física).

Cada página lógica de um processo é mapeada para um quadro físico de memória, permitindo que apenas as páginas necessárias sejam carregadas.

Características

- **Paginação por Demanda**
    - Nessa técnica, apenas as páginas que um processo está utilizando no momento são carregadas na memória. Assim, quando um processo começa a execução, somente as páginas imediatamente necessárias são trazidas do disco para a memória física;
    - As demais páginas permanecem no disco até que sejam requisitadas, momento em que ocorre um erro de página;
    - Quando um processo está para ser inserido na memória, o paginador avalia as páginas que serão usadas antes que o processo seja removido novamente;
    - No lugar de inserir um processo inteiro, o paginador traz apenas essas páginas para a memória;
    - É preciso um suporte de hardware para realizar este método.
- **Erro de Página**
    - Quando um processo tenta acessar uma página não carregada, ocorre um erro de página. Esse evento sinaliza que a página precisa ser trazida para a memória. O sistema operacional interrompe o processo, localiza a página no disco, a carrega em um quadro livre e atualiza a tabela de páginas;

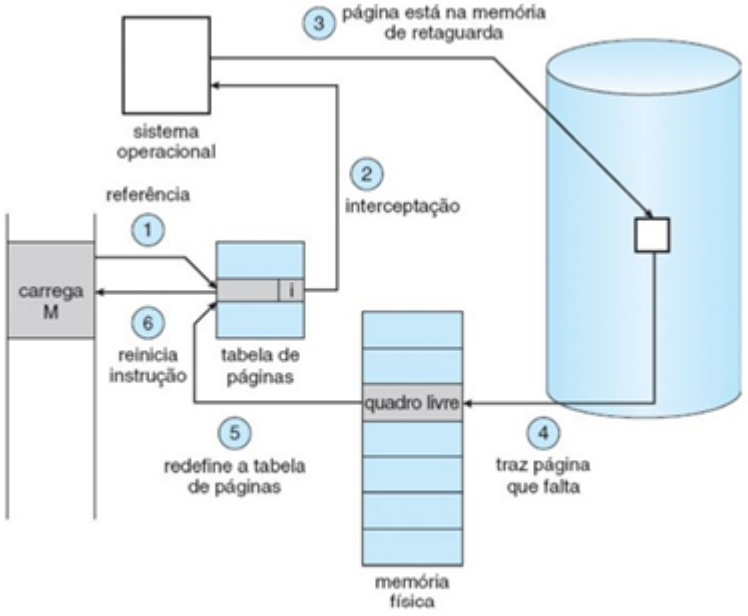
    - Uma vez completado o processo, o sistema retoma a execução do processo exatamente no ponto de interrupção;
    - Um requisito crucial da paginação por demanda é a capacidade de reiniciar qualquer instrução após um erro de página.
- **Tabela de Páginas**
    - A tabela de páginas armazena informações de mapeamento entre páginas lógicas e quadros físicos. Cada entrada da tabela indica se a página está na memória ou no disco;
    - Um bit de validade ajuda a identificar se a página está disponível na memória principal, auxiliando o sistema operacional a decidir as operações de leitura e escrita entre memória e disco.

Substituição de Páginas

A substituição de páginas é fundamental para o gerenciamento de memória virtual.

Quando ocorre um erro de página, o sistema operacional executa os seguintes passos:

1. **Localiza a Página no Disco**: o sistema encontra a página requisitada no disco para trazê-la para a memória;
2. **Busca um Quadro Livre**: se houver um quadro livre, ele é usado. Caso contrário, o sistema usa o algoritmo de substituição de páginas para selecionar uma página "vítima" para ser substituída, aplicando um dos critérios de seleção;
3. **Atualiza Tabelas e Transferências**: o “quadro vítima” é gravado no disco (caso tenha sido modificado) e a tabela de páginas é atualizada para refletir a nova localização da página requisitada;
4. **Retoma o Processo**: após a atualização, o processo retoma a execução a partir do ponto onde foi interrompido pelo erro de página.



Passos para a manipulação de um erro de página.

Escalonamento de CPU

O escalonamento de CPU gerencia a ordem de execução dos processos na CPU, essencial em sistemas multiprogramados onde vários processos competem pela CPU.

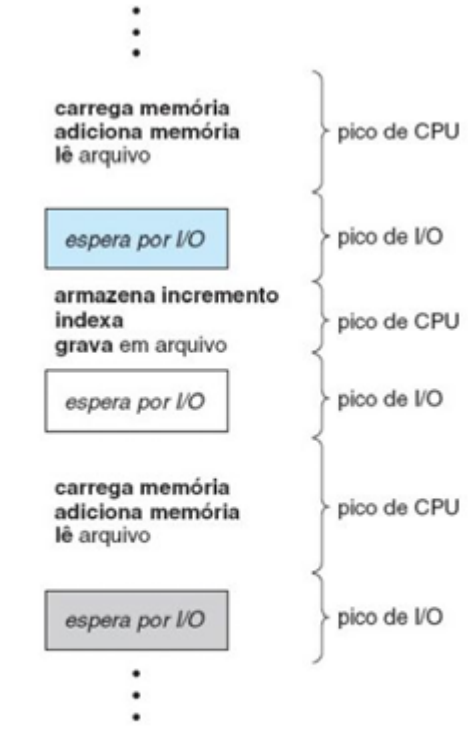
A parte do sistema operacional que gerencia essas decisões é o escalonador (scheduling), e o algoritmo utilizado para definir a ordem de execução dos processos é chamado de algoritmo de escalonamento.

> → Quando mais de um processo está no estado pronto e existe apenas uma CPU disponível, o sistema operacional deve decidir qual deles vai executar primeiro.

Ciclo de CPU

A execução de processo geralmente consiste em um ciclo entre execução na CPU e espera por I/O.

A alternância entre esses estados ajuda a identificar quais processos estão prontos para execução e quais aguardam operações de I/O, facilitando a escolha do algoritmo de escalonamento ideal.



Sequência alternada de picos de CPU e de I/O.

Critérios de Escalonamento

Os algoritmos de escalonamento da CPU são avaliados com base em vários critérios, sendo eles:

- **Utilização da CPU**, isto é, mantê-la ocupada, variando entre 40% a 90%;
- **Throughput**, que seria o número de processos completados por unidade de tempo;
- O **tempo de "Turnaround"**, ou seja, o tempo total desde a submissão até a conclusão de um processo;
- O **tempo de espera** que um processo passa na fila de processos prontos;
- O **tempo de resposta** entre a submissão de uma solicitação e a primeira resposta.

Quando o escalonamento é necessário?

O escalonamento pode ocorrer em momentos específicos, como quando:

→ um processo termina;

→ é bloqueado em uma operação I/O;

→ ou é interrompido por um semáforo.

Em cada um desses casos, o processo que estava em execução se torna não apto a continuar, de modo que outro processo deva ser escolhido para executar em seguida.

Existem três outras ocasiões em que o escalonamento é normalmente, embora não seja absolutamente necessário nesses momentos:

1. Quando um novo processo é criado;
    a. No caso de um novo processo, faz sentido reavaliar as prioridades nesse momento. Em alguns casos, o processo pai pode solicitar uma prioridade diferente para o filho.
2. Quando ocorre uma interrupção de I/O;
    a. No caso de uma interrupção de E/S, isso normalmente significa que um dispositivo de E/S
    acabou de concluir seu trabalho. Assim, algum processo que estava bloqueado, esperando pela I/O, poderá agora estar pronto (ou apto) para executar.
3. Quando ocorre uma interrupção de relógio.
    a. No caso de uma interrupção de relógio, essa é uma oportunidade para decidir se o processo
    que está correntemente em execução já está executando por um tempo demasiado.

Escalonamento com Múltiplas CPUs

O escalonamento com múltiplas CPUs é uma extensão do escalonamento tradicional de CPU, aplicado a sistemas multiprocessadores. Quando o sistema

possui mais de uma unidade de processamento, o compartilhamento de carga de trabalho entre os processadores se torna essencial para maximizar a utilização de recursos e garantir um desempenho eficiente.

No entanto, essa tarefa é mais complexa por envolver várias decisões de alocação e sincronização.

Objetivos do Escalonamento com Múltiplas CPUs

- Maximizar a utilização das CPUs disponíveis;
- Balancear a carga de trabalho entre os processadores para evitar que algumas CPUs fiquem ociosas enquanto outras estão sobrecarregadas;
- Minimizar o tempo de espera e de resposta para os processos,
- Garantir que o sistema funcione de maneira eficiente e responsiva.

Modelos de Multiprocessamento

Multiprocessamento Assimétrico (AMP)

No modelo AMP, **um único processador**, chamado de processador **mestre**, é responsável por:

→ Realizar todas as decisões de escalonamento;

→ Gerenciar as operações de I/O;

→ Controlar as estruturas de dados do sistema e outras atividades.

- Os demais **processadores (escravos) executam apenas código de usuário** e não acessam diretamente as estruturas do sistema operacional;
- Esse modelo é simples de implementar por **eliminar a necessidade de sincronizaçã**o entre processadores para acessar as estruturas de dados compartilhadas;

- Uma desvantagem deste modelo é que o processador mestre pode se tornar um gargalo, limitando o desempenho em sistemas altamente carregados.

Multiprocessamento Simétrico (SMP)

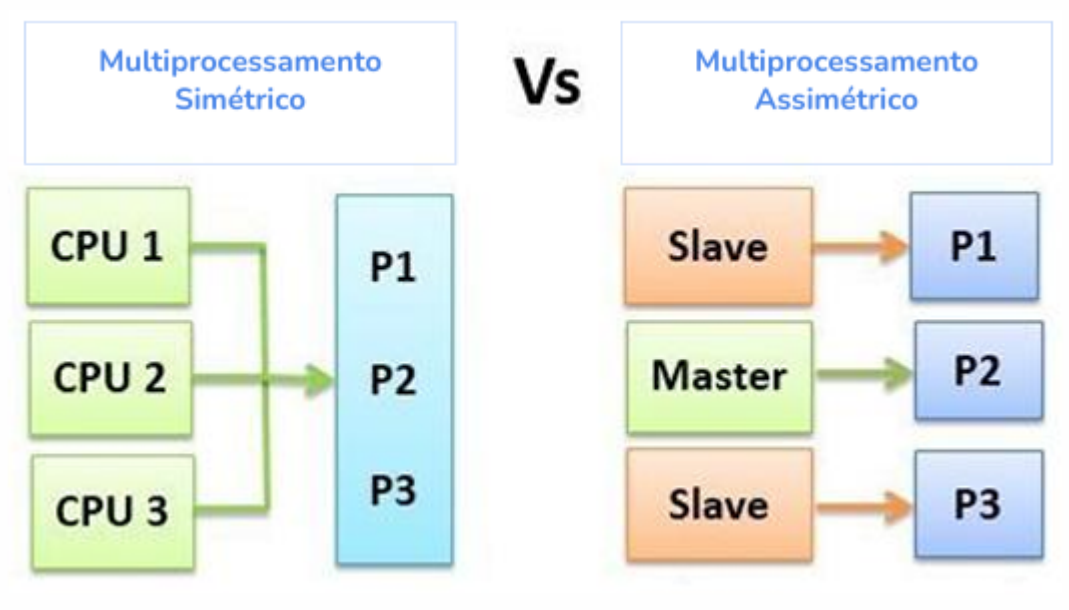
No modelo SMP, **todos os processadores compartilham igualmente as tarefas** do sistema operacional, como escalonamento e I/O. Cada **processador possui o próprio escalonador** e pode executar processos de forma independente.

Existem duas formas de gerenciar as filas de processos, sendo elas:

→ **Fila comum**, onde todos os processadores **compartilham uma única fila de processos** prontos, simplificando o balanceamento de carga, pois qualquer processador pode acessar os processos na fila;

→ **Filas privadas**, onde cada processador **possui a sua própria fila de processos** prontos, reduzindo a contenção de dados, mas é exigido mecanismos de balanceamento de carga para evitar ociosidade nos processadores.

- As vantagens deste modelo são melhor distribuição de carga de trabalho e maior escalabilidade em sistemas com muitos processadores;
- E as desvantagens são maior complexidade para evitar conflitos e a necessidade de sincronização cautelosa ao acessar estruturas compartilhadas.



Afinidade com o Processador

A afinidade com o processador refere-se à **tentativa de manter um processo sempre em execução no mesmo processador**, e isso se dá porque:

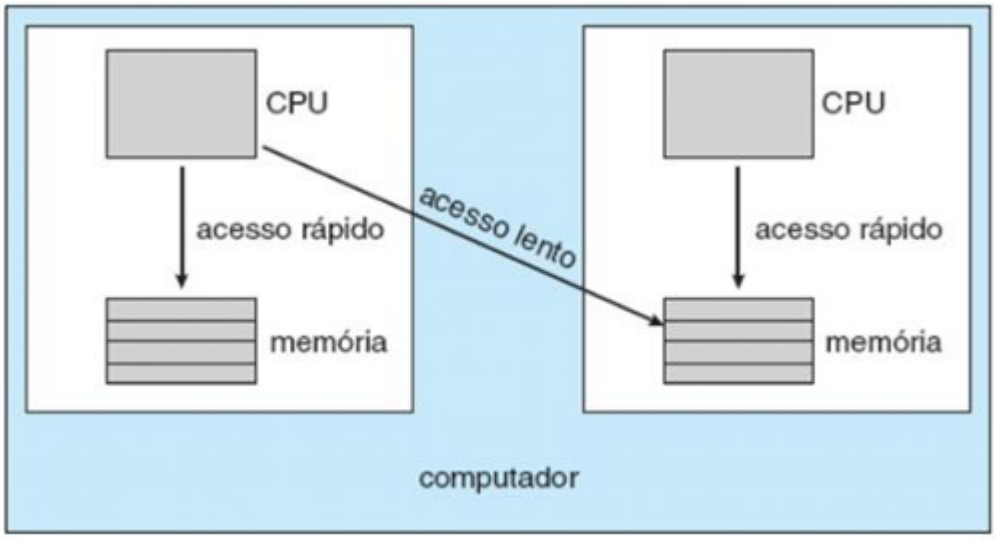
→ O cache do processador armazena dados acessados recentemente pelo processo;

→ Migrar um processo para outro processador invalida o cache do processador anterior e exige repovoamento no novo processador, aumentando o custo de cada execução.

Os tipos de afinidade com o processador são:

- **Afinidade leve**, onde o sistema **tenta manter o processo no mesmo processador**, mas pode migrá-lo se necessário;
- **Afinidade rígida**, onde o **processo especifica um subconjunto de processadores** onde pode ser executado, **restringindo suas migrações**.

OBS. A **arquitetura da memória principal** de um sistema pode **afetar** questões relacionadas com a afinidade com o processador.



Balanceamento de Carga

O balanceamento de carga tem o objetivo de **distribuir o trabalho de forma uniforme entre os processadores**. Caso contrário, um ou mais processadores podem ficar ociosos enquanto outros terão cargas altíssimas de trabalho.

Em sistemas SMP, é importante **manter a carga de trabalho balanceada** entre todos os processadores para que os benefícios do uso de mais de um processador sejam obtidos plenamente.

> → É importante observar que normalmente **o balanceamento de carga é necessário somente em sistemas que cada processador tem a sua própria fila privada de processos** prontos para serem executados!

Existem duas abordagens importantes para o balanceamento de carga, sendo elas:

- **Migração por impulsão**, onde uma tarefa específica verifica periodicamente a carga dos processadores e — quando encontra um desequilíbrio — redistribui os processos uniformemente entre eles;
- **Migração por extração**, onde processos ociosos “extarem” processos de filhas de processadores que estão ocupados.

No entanto, o balanceamento de carga pode entrar em conflito com a afinidade, pois mover um processo de um processador para outro invalida o cache e reduz a eficiência.