AULA 1 – ARQUITETURA DO SISTEMA OPERACIONAL

OBJETIVO DA AULA

Conhecer os componentes do sistema operacional e seu relacionamento.

APRESENTAÇÃO

O sistema operacional é um conjunto de programas que gerencia o *hardware* e o *software* do computador, além de promover uma interface que facilita muito a vida do usuário.

Vamos ver nesta aula como o sistema operacional é organizado, suas principais funções e tarefas e sua participação no desempenho do computador.

Hora de começar!

1. INTRODUÇÃO

Como já vimos em aulas anteriores, o computador é um complexo formado por *hardware* e *software*, que se relacionam, se completam e devem funcionar em harmonia.

O *software* mais importante de todos é o sistema operacional, que tem duas funções principais: gerenciar os recursos de *hardware* e *software* e promover a interface entre o computador e o usuário.

Mas o que ele tem que gerenciar e o que é interface? É o que vamos ver a partir de agora.

Começaremos pela classificação dos sistemas operacionais. E nesta classificação usaremos alguns parâmetros.

O primeiro parâmetro que vamos considerar é quanto à capacidade de execução de tarefas. Temos então:

- Sistemas monotarefa: são aqueles que são capazes de manter apenas um programa de cada vez na memória, desfrutando de todos os recursos da máquina. Embora sejam de implementação simples, têm a desvantagem de desperdiçar recursos como espaço de memória e o próprio processador;
- Sistemas multitarefa: são capazes de suportar mais de um programa na memória. Permitem um uso melhor dos recursos do computador, mas têm que gerenciar a disputa por memória, tempo de processador e acesso aos dispositivos de E/S.

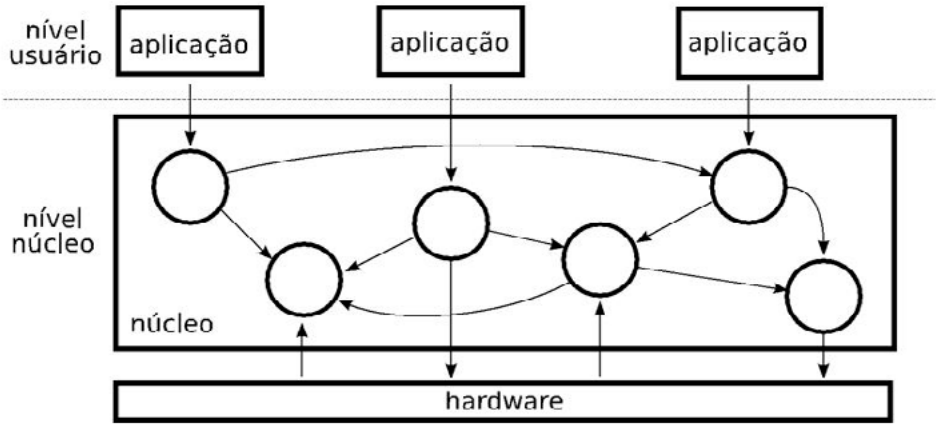
O segundo parâmetro de classificação é quanto à quantidade de usuários que o sistema consegue suportar.

- Sistemas monousuários: são aqueles que suportam um único usuário;
- Sistemas multiusuários: permite que diversos usuários utilizem os recursos do computador, garantindo que as tarefas, preferências e arquivos dos usuários estejam separados e protegidos.

Finalmente, o terceiro parâmetro é quanto à arquitetura do sistema operacional. Temos as seguintes classificações:

- Monolítico: esse tipo de sistema operacional é formado por uma coleção de procedimentos que podem interagir livremente entre si, conforme Figura 1.

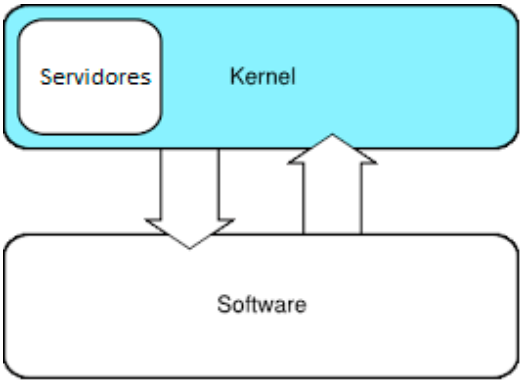
FIGURA 1 | **Estrutura de um sistema monolítico**



Fonte: https://slideplayer.com.br/slide/287939/)

- Microkernel: é uma arquitetura cujas funcionalidades são quase todas executadas fora do núcleo, já que este fornece recursos mínimos necessários ao funcionamento do sistema. Outras funcionalidades são oferecidas através de programas *servidores*, que se localizam no "espaço de usuário". Os processos se comunicam com esse núcleo, usando o mínimo possível o "espaço do sistema", conforme Figura 2.

FIGURA 2 | **Estrutura de um sistema microkernel**



Fonte: https://www.ic.unicamp.br/~islene/2s2007-mo806/slides/Microkernel.pdf

2. ESTRUTURA DOS SISTEMAS OPERACIONAIS

Vamos agora olhar com mais atenção a estrutura do sistema operacional. Ele é formado por um conjunto de rotinas chamadas sempre que é necessária a intervenção do sistema operacional em razão da solução de conflitos ou atendimento a solicitação dos usuários. Como vimos, essas rotinas compõem o que chamamos de *núcleo* ou *kernel* do sistema operacional.

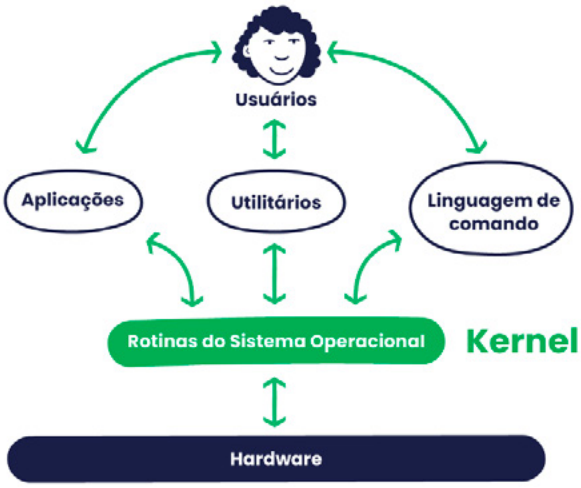
DESTAQUE

Usaremos no texto os termos núcleo ou kernel alternadamente. Mas você precisa saber que os termos significam a mesma coisa.

Mas um sistema operacional não se resume a seu kernel. Há outras partes, como uma linguagem de comandos e alguns utilitários de apoio que veremos a partir de agora.

A maioria dos sistemas operacionais é organizada em camadas e para o usuário isso é transparente, uma vez que ele se utiliza dos utilitários e da linguagem de comandos para se comunicar com o computador.

FIGURA 3 | **Estrutura do sistema operacional**



Fonte: https://slideplayer.com.br/slide/6856346/.

Na Figura 3 vemos que o usuário tem acesso às aplicações, utilitários e linguagem de comando, sem precisar conhecer as rotinas do Kernel. Também podemos ver que essas rotinas controlam o *hardware* em diversos aspectos.

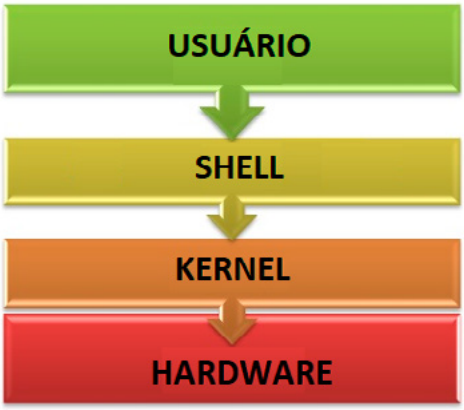
As rotinas do Kernel (núcleo) são críticas em relação à segurança do computador e, como veremos adiante, só podem ser executadas sob supervisão do próprio sistema operacional para não comprometerem a segurança e confiabilidade do sistema.

Entre as rotinas do Kernel estão:

- tratamento de interrupções e exceções (que estudamos na unidade anterior);
- criação e eliminação de processos;
- sincronização e comunicação entre processos;
- gerência de memória;
- gerência do sistema de arquivos;
- gerência do uso da CPU.

Quanto à linguagem de comandos, conhecida como *shell* do sistema operacional, ela é uma interface que oferece ao usuário uma forma de se comunicar com o sistema operacional e usar seus serviços através de uma linha de comandos ou uma interface gráfica.

FIGURA 4 | **Shell como interface entre o usuário e o sistema operacional**



Fonte: https://windowsclub.com.br/windows-core-os-e-andromeda-explicacao-completa-do-novo-windows-10/.

Na Figura 4 vemos que o shell é uma camada que fica entre o usuário e o Kernel. Assim, é através do shell que o usuário pode acionar algumas das rotinas do Kernel diretamente.

3. MODOS DE ACESSO

Como vimos, as rotinas do Kernel podem afetar diretamente a segurança e confiabilidade do computador. Assim, elas só podem ser executadas com um cuidado especial.

Para isso, o sistema operacional oferece, como mecanismo de proteção, dois modos de acesso ao processador: o modo usuário e o modo supervisor (ou modo kernel).

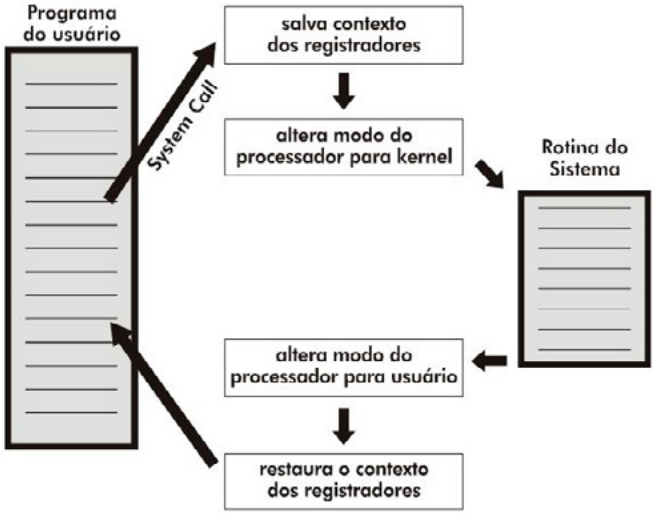
No **modo usuário**, o processador não tem acesso às rotinas do núcleo e somente a algumas instruções que não afetam a integridade do computador.

Já no **modo kernel**, o processador tem permissão para acessar e executar todas as instruções do núcleo, chamadas de *instruções privilegiadas*.

O modo de acesso é definido em um conjunto de bits armazenados no registrador de status.

A cada solicitação de execução de uma instrução privilegiada, o *hardware* verifica o conteúdo desse registrador. Se ele estiver indicando modo kernel, as instruções são executadas normalmente. Caso contrário, as instruções não serão executadas.

FIGURA 5 | **Alternando os modos de acesso do processador**



Fonte: https://slideplayer.com.br/slide/378300/.

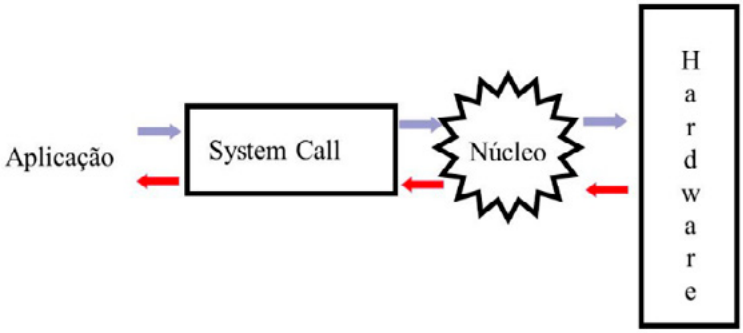
A Figura 5 mostra o funcionamento do sistema de modos de acesso do processador.

Repare que somente enquanto está executando uma rotina do sistema é que o processador fica em modo kernel. Durante a execução dos programas em geral o modo de acesso é o usuário.

4. CHAMADAS DE SISTEMA

Como vimos, as rotinas dos sistemas operacionais compõem o kernel e são instruções privilegiadas e que só podem ser executadas em modo supervisor e o mecanismo que controla isso, através dos modos de acesso, é conhecido como *system calls* (chamadas de sistema).

FIGURA 6 | **Chamada de sistema**



Fonte: https://slideplayer.com.br/slide/42543/.

Os sistemas operacionais definem como chamadas de sistema todas as operações envolvendo acesso a periféricos, acesso a arquivos e operações como criação e eliminação de tarefas, conforme Figura 6. Tais operações, como vimos, se executadas sem supervisão podem comprometer a integridade e a segurança do sistema.

CONSIDERAÇÕES FINAIS

Chegamos ao fim desta aula em que vimos a estrutura dos sistemas operacionais. Tudo o que estudamos aqui vale para a quase totalidade dos sistemas operacionais modernos.

Vimos que o sistema operacional é um gerente dos recursos de *hardware* e *software*, além de promover uma interface entre o usuário e o computador.

O núcleo do sistema operacional é composto por rotinas que devem ser executadas sob determinados cuidados para não comprometer a integridade do computador.

Além disso, os sistemas operacionais são classificados de acordo com diversos parâmetros.

Nossa próxima etapa é estudar as funções de gerenciamento de recursos por parte do sistema operacional, começando pela gerência do processador e em seguida pela gerência da memória.

Até a próxima aula!

MATERIAIS COMPLEMENTARES

Assista a esse breve histórico dos sistemas operacionais. Disponível em: https://www.youtube.com/watch?v=9rC9GilX1Io.

Neste vídeo você aprenderá alguns dos conceitos básicos dos sistemas operacionais, o que será um complemento muito interessante para nosso estudo. Disponível em: https://www.youtube.com/watch?v=T7lCM3l7vAQ.

REFERÊNCIAS

STALLINGS, William. *Arquitetura e Organização de Computadores: projeto para o desempenho*. 8ª edição. Editora Pearson. Livro. (642 p.). ISBN 9788576055648. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576055648>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Organização estruturada de computadores*. 6ª edição. Editora Pearson. Livro. (628 p.). ISBN 9788581435398. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788581435398>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Sistemas Operacionais Modernos*. 3ª edição. Editora Pearson. Livro. (674 p.). ISBN 9788576052371. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576052371>. Acesso em: 16 out. 2022.

AULA 2 – PROCESSOS, THREADS E GERÊNCIA DO PROCESSADOR

OBJETIVO DA AULA

Definir processo e escalonamento de processos no processador.

Apresentação

Olá!

Sabemos que os computadores atualmente trabalham atendendo a várias demandas do usuário simultaneamente. Por exemplo, você pode estar digitando um texto enquanto ouve música pela internet ou vinda de um arquivo no disco rígido e, ao mesmo tempo, estar realizando um download.

Todas essas atividades ativam programas na memória do computador e esses programas em execução são chamados de processos.

Vamos ver nesta aula exatamente o que é um processo e como o sistema operacional permite que vários processos compartilhem o tempo do processador.

1. INTRODUÇÃO

Imagine que você está usando seu computador para digitar um texto e, enquanto trabalha, coloca um vídeo de uma música no YouTube. Neste momento você ativou dois programas: o editor de texto e o browser.

Mas, além desses programas, há outros rodando no computador e na maioria das vezes você sequer sabe o que eles estão fazendo. Porém, eles usam o mesmo processador e esse uso precisa ser dividido eficientemente por todos esses programas.

Quem exerce esse controle e de que forma isso acontece? É o que vamos começar a ver agora.

2. PROCESSO

O primeiro conceito que precisamos conhecer é o de *processo*. Como vimos anteriormente, uma definição simples para processo é *programa em execução*. Mas processo é algo muito maior do que isso.

Podemos definir processo como uma estrutura criada pelo sistema operacional para que um programa possa ser executado. Assim, toda vez que você aciona um programa como um editor de textos ou um jogo, o sistema operacional entra em ação e cria um processo.

Um processo possui as seguintes informações:

- Espaço de endereçamento: é uma faixa de endereços de memória reservada para que o programa possa usar;
- Contexto de *hardware*: é composto pelo conteúdo do banco de registradores a cada momento. Esse contexto é uma espécie de status do programa e deve ser salvo a cada interrupção. Ele é dinâmico, pois seu conteúdo é alterado a cada instrução executada;
- Contexto de *software*: é composto de quatro informações:

1) PID (*Process IDentification*) – é um número de identificação do processo;

2) *Owner* – é a identificação de quem solicitou a criação do processo, que pode ser um usuário ou outro processo;

3) *Quotas* – definem as quantidades de cada recurso que o processo pode utilizar. Por exemplo, a quantidade de arquivos que ele pode abrir simultaneamente ou a quantidade de subprocessos que ele pode criar.

4) Privilégios – definem as ações que o processo pode executar e as que não pode. Em suma, refletem as permissões dadas ao processo.

Quanto aos estados que um processo pode assumir, eles são os seguintes:

- Executando – é quando o processo está com a posse do processador e, portanto, tem suas instruções sendo executadas;
- Pronto – é quando o processo reúne todas as condições para entrar em execução e apenas aguarda sua vez de pegar o processador;
- Espera ou bloqueado – é quando o processo possui alguma pendência – por exemplo, uma variável compartilhada que está sendo usada por outro processo – e, por isso, não pode entrar em execução ainda que o processador esteja ocioso.

As mudanças de estado acontecem mediante a ocorrência de alguns eventos. Vejamos quais são esses eventos:

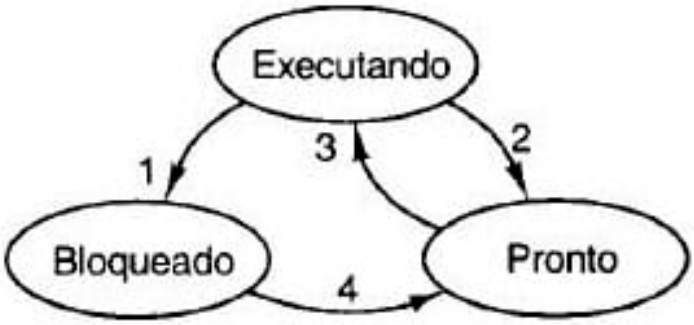
1) De **executando** para **bloqueado**: ocorre quando o processo detecta a necessidade de um recurso (variável ou *hardware*) que está bloqueado por outro processo e sem ele não pode prosseguir;

2) De **executando** para **pronto**: ocorre quando o tempo que o processo tem para usar o processador (*time slice* ou *quantum*) termina ou quando um processo de maior prioridade é criado;

3) De **pronto** para **executando**: ocorre quando chega a vez do processo, que aguarda numa fila (fila de prontos), entrar em execução;

4) De **bloqueado** para **pronto**: ocorre quando a pendência que impedia o processo de entrar em execução é solucionada.

FIGURA 1 | **Mudanças de estado de processo**



Fonte: https://www.oficinadanet.com.br/post/12786-sistemas-operacionais-o-que-e-deadlock.

3. THREADS

As threads, também conhecidas como *processos leves*, são unidades, ou blocos básicos, contendo fluxo de instruções independentes em um mesmo processo que podem ser executados em paralelo, uma vez que não apresentam dependências entre si.

Um processo pode conter várias threads que compartilham todos os seus recursos (quotas e privilégios, além do espaço de endereçamento).

4. ESCALONAMENTO DE PROCESSOS

Essa função executada pelo sistema operacional consiste em gerenciar a fila de processos que disputam o tempo do processador.

Para isso há algumas políticas ou algoritmos de escalonamento.

Antes de estudarmos esses algoritmos, precisamos saber que eles estão divididos em dois tipos: algoritmos preemptivos e algoritmos não preemptivos. No primeiro caso, quando um processo "pega" o processador, ele pode sofrer uma *preempção*, que é um tipo de interrupção causada exclusivamente para dar a vez a outro processo. No segundo caso, a preempção não ocorre, o que significa que o processo que está com o processador fica com ele até terminar todas as suas instruções.

Vale ressaltar que em ambos os casos as interrupções causadas por dispositivos de E/S ocorrem normalmente.

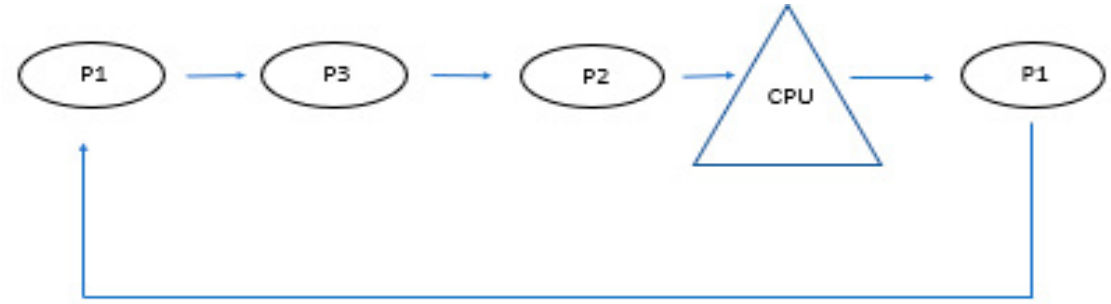
Vamos agora ver os principais algoritmos de escalonamento:

1) FIFO (*First In First Out*) – neste algoritmo os processos são encaminhados ao processador pela ordem de chegada. Como ele é não preemptivo, os processos só "devolvem" o processador quando terminam sua execução.

2) SJF (*Shortest Job First*) – neste algoritmo, também não preemptivo, os processos são enfileirados de acordo com seu tempo de execução, do mais curto. O problema deste algoritmo é que processos com tempo de execução muito grande podem ficar por tempo indeterminado na fila, uma vez que podem ser criados processos com menor tempo de duração. Outro problema, que torna a implementação desse algoritmo difícil, é que nem sempre é possível estimar o tempo que o processo gastará, já que as circunstâncias podem torná-lo mais longo ou mais curto.

3) *Round Robin* (circular) – algoritmo preemptivo, o *Round Robin* atende os processos pela ordem de chegada, dando a cada um uma fatia de tempo (*time slice* ou *quantum*) ao final da qual o processo sofre uma preempção dando lugar ao próximo e retornando ao final da fila onde aguardará a próxima vez. Esse algoritmo é muito usado, pois, além evitar que algum processo não seja atendido, permite um tempo de resposta médio mais interessante quando há vários processos disputando o processador.

FIGURA 2 | **Escalonamento Round Robin**



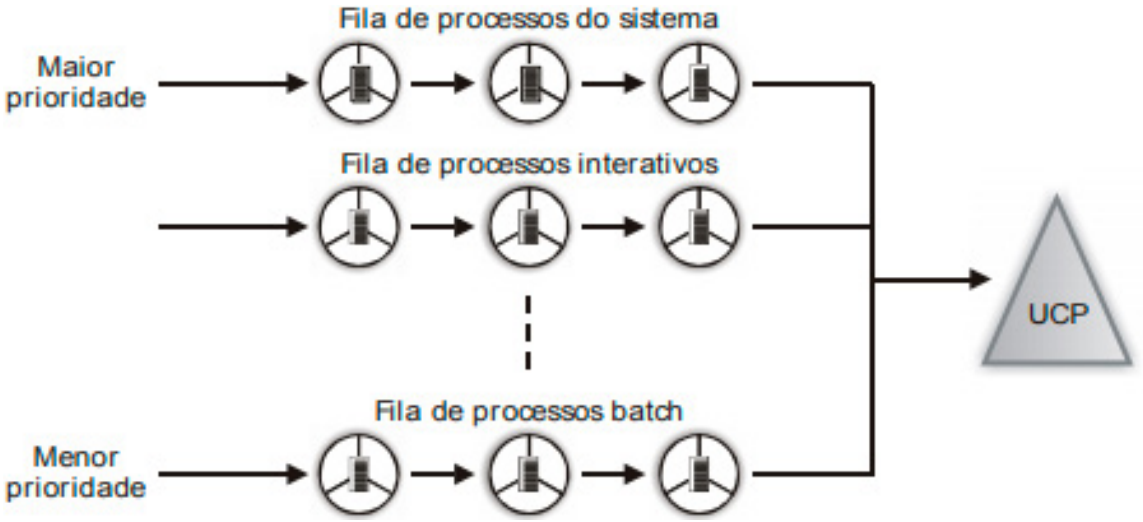
Elaborado pelo autor.

Quanto ao tamanho da fatia de tempo, ela é definida com base na importância do processo e não necessariamente é igual para todos os processos.

4) Escalonamento por prioridades – algoritmo preemptivo em que a cada processo é associada uma prioridade e o atendimento aos processos é feito de acordo com essa prioridade, de forma que os mais importantes são atendidos primeiro. Nesse caso, se um processo de maior prioridade chega na fila, ele "toma" o processador daquele que está executando e só o libera se chegar outro processo com maior prioridade ou quando termina todas as suas tarefas.

5) Escalonamento por múltiplas filas – nesse algoritmo, que também é preemptivo, os processos são organizados em filas por nível de prioridade de forma que uma fila de menor prioridade só é atendida se todas as filas com maior prioridade estiverem vazias. Para evitar que processos das filas de mais baixa prioridade fiquem muito tempo sem serem atendidos, há mecanismos de incremento de prioridade para esses processos para que eles mudem de fila eventualmente.

FIGURA 3 | **Escalonamento por múltiplas filas**



Fonte: https://www.passeidireto.com/arquivo/69218857/escalonamento-por-multiplas-filas.

Na Figura 3 vemos três filas de processos. Na fila de mais alta prioridade estão os processos do próprio sistema operacional, que, como vimos, gerenciam os recursos de *hardware* e *software*.

Na segunda fila estão os processos interativos, ou seja, processos do usuário, que precisam de um tempo de resposta razoável.

Finalmente, na fila de mais baixa prioridade estão os processos cujo resultado não precisa ser tão rápido e por isso são executados somente quando as filas superiores estiverem vazias.

Nesse tipo de escalonamento cada fila pode funcionar com um algoritmo diferente. Assim, por exemplo, na fila de menor prioridade pode ser usado o algoritmo FIFO, na segunda fila o Round Robin e na primeira fila o escalonamento por prioridades.

Cada algoritmo desses que estudamos atende em maior ou menor nível aos seguintes parâmetros:

- Justiça – todo processo deve ser atendido em algum momento;
- Taxa de utilização da CPU – o algoritmo deve evitar ao máximo deixar a CPU ociosa, minimizando a quantidade de trocas de contexto, que ocorrem cada vez que um processo dá lugar a outro no uso da CPU;
- Tempo de resposta – cada processo deverá responder às solicitações do usuário no menor tempo possível;
- *Turnaround time* – é o tempo total em que o processo existe, desde o momento em que foi criado até o momento em que foi encerrado. Isso é um somatório dos tempos em que passou em cada um dos três estados que vimos (executando, pronto e bloqueado) e deve ser minimizado;

- *Throughput* – é a quantidade de processos que são concluídos em uma quantidade específica de tempo. Este parâmetro deve ser maximizado.

O atendimento melhor ou pior a esses parâmetros varia entre os algoritmos estudados. Não é possível atender bem a todos, já que muitas vezes eles são conflitantes entre si.

CONSIDERAÇÕES FINAIS

Chegamos ao fim de mais uma aula.

Nela conhecemos um dos conceitos mais importantes para o estudo de sistemas operacionais: processo. Um processo é o ambiente ou infraestrutura que o sistema operacional cria para que um programa possa ser executado.

Os processos podem estar em três estados: executando, pronto e bloqueado e suas informações são divididas em contexto de *hardware*, contexto de *software* e espaço de endereçamento.

Vimos também que a função de escalonamento de processos consiste em organizar a fila de processos sob algum critério de forma que o uso do processador seja o melhor possível atendendo aos parâmetros de justiça, tempo de resposta, taxa de utilização da CPU, *turnaround time* e *throughput*.

Agora é hora de vermos como o sistema operacional gerencia a memória do computador.

MATERIAIS COMPLEMENTARES

Neste vídeo você entenderá um pouco mais sobre threads: https://www.youtube.com/watch?v=xNBMNKjpJzM.

REFERÊNCIAS

STALLINGS, William. *Arquitetura e Organização de Computadores: projeto para o desempenho*. 8ª edição. Editora Pearson. Livro. (642 p.). ISBN 9788576055648. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576055648>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Sistemas Operacionais Modernos*. 3ª edição. Editora Pearson. Livro. (674 p.). ISBN 9788576052371. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576052371>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Organização estruturada de computadores*. 6ª edição. Editora Pearson. Livro. (628 p.). ISBN 9788581435398. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788581435398>. Acesso em: 16 out. 2022.

AULA 3 – GERENCIAMENTO DE MEMÓRIA

OBJETIVO DA AULA

Entender os mecanismos de controle e proteção de memória.

APRESENTAÇÃO

Olá!

Nesta aula estudaremos a segunda grande tarefa do sistema operacional enquanto gerente dos recursos do processador.

A memória é um recurso muito requisitado pelos processos, que precisam do máximo de espaço possível para trabalharem eficientemente. Porém, como qualquer recurso, tem seus limites físicos.

Assim, o sistema operacional precisa gerenciar essa demanda para permitir que todos os processos possam ser executados satisfatoriamente.

Então, vamos ver como isso é feito.

Mãos à obra!

1. INTRODUÇÃO

A função do gerenciamento de memória é fundamental porque qualquer programa precisa de alguma quantidade dela para ser executado.

Porém, a limitação de espaço e as permutas entre a memória principal e a secundária tornam essa tarefa crucial para o bom desempenho do computador.

A gerência de memória lida com quatro aspectos principais:

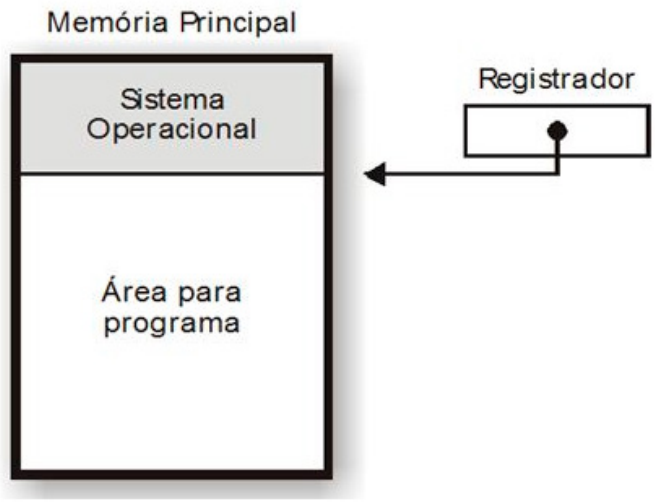
- Controle de ocupação – consiste em registrar os endereços das posições de memória livres e ocupadas;
- Alocação de memória – significa reservar uma faixa de endereços de memória (espaço de endereçamento) para o processo que acabou de ser criado;
- Proteção de memória – como há vários processos ocupando seus espaços de endereçamento, o sistema operacional precisa garantir que um processo não invadirá o espaço do outro;
- *Swapping* – é a permuta entre conteúdos da memória principal e da memória secundária. Esta função será estudada em detalhes quando abordarmos *memória virtual*.

É importante considerar que o sistema operacional deve buscar manter o maior número de processos possível na memória principal, bem como garantir que programas maiores do que o espaço de endereçamento disponível possam ser executados.

2. ALOCAÇÃO DE MEMÓRIA

Na *alocação contígua simples*, usada nos sistemas monoprogramáveis (ou monotarefa), a memória principal tem uma parte destinada ao próprio sistema operacional e o restante é reservado ao programa de usuário e, por haver somente um programa em execução, não há necessidade de proteger esse espaço de endereçamento. Há apenas um registrador que delimita os espaços reservados ao sistema operacional e ao programa de usuário.

FIGURA 1 | **Alocação contígua simples**



Fonte: https://slideplayer.com.br/slide/1235123/)

O principal problema deste modelo de alocação é a subutilização da memória, uma vez que, se o programa de usuário não ocupar todo o espaço disponível, o espaço não ocupado é desperdiçado.

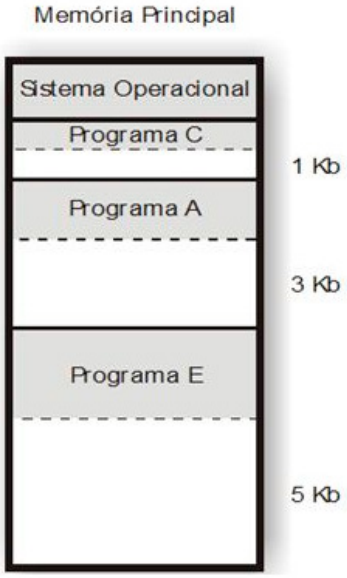
Além disso, se o programa for maior do que esse espaço, ele não caberá ali.

Para resolver isso, é utilizada a técnica de *overlay*, na qual os programas são divididos em módulos que possam ser carregados e executados de forma independente na memória.

A *alocação particionada,* para atender os sistemas multiprogramáveis (multitarefa), pode usar um esquema estático ou dinâmico.

Na alocação particionada estática o tamanho das partições é fixo e isso pode também causar algum desperdício de memória quando as partes em que os programas são divididos não ocupam as respectivas partições totalmente, causando a *fragmentação* da memória, conforme podemos ver na Figura 2 a seguir.

FIGURA 2 | **Fragmentação da memória**



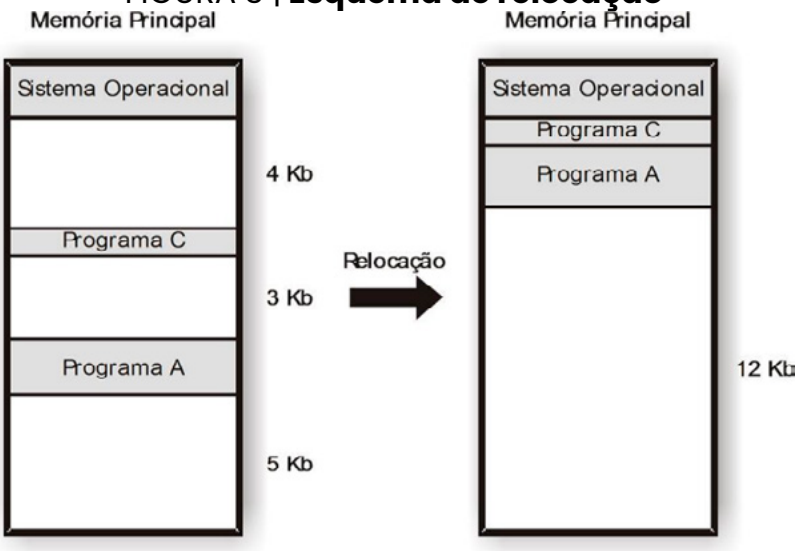
Fonte: https://slidleplayer.com.br/slide/1235123/.

Na Figura 2 as partes em branco são *sobras* ou fragmentos de memória não utilizados.

A *alocação particionada dinâmica* permite que os programas utilizem apenas o espaço necessário para sua execução. Porém, na medida em que vão terminando, os espaços de memória que vão liberando podem trazer também o problema da fragmentação.

A solução para esse problema é a utilização da alocação particionada com *relocação*, em que periodicamente os programas são movimentados na memória para liberar espaços livres.

FIGURA 3 | **Esquema de relocação**



Fonte: https://docplayer.com.br/73250663-Sistemas-operacionais-gerencia-de-memoria.html.

Como podemos observar na Figura 3, após a relocação todo o espaço vazio da memória foi reunido em um único espaço, resolvendo o problema da fragmentação.

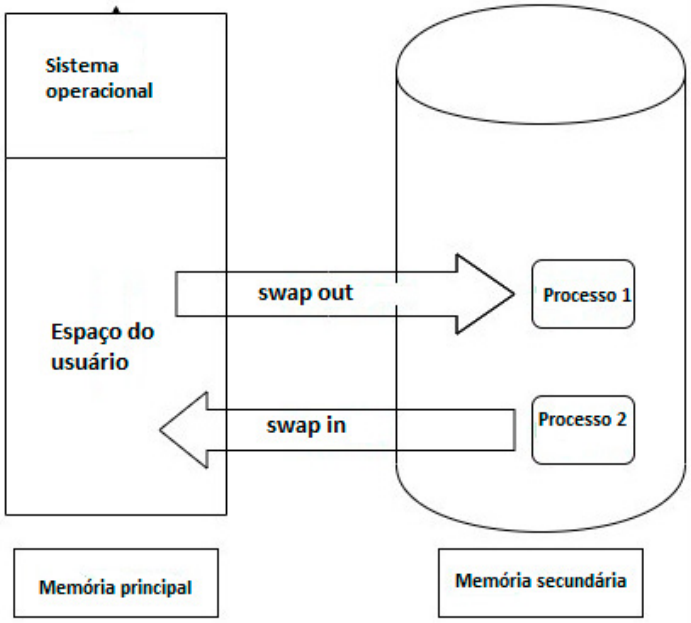
3. SWAPPING

Como sabemos, em geral, os programas são maiores do que o espaço de endereçamento reservado a eles. Assim, é impossível termos alguns programas carregados integralmente na memória principal enquanto estão sendo executados.

O sistema operacional precisa ter alguma forma de garantir que todos os programas – sejam eles do próprio sistema operacional ou do usuário – estejam de alguma forma disponíveis na memória virtual.

A técnica de *swapping* (troca, permuta) permite que o sistema operacional possa realizar as permutas entre a memória principal e a memória secundária, garantindo que todos os programas serão atendidos e que isso ocorrerá no momento em que for necessário.

FIGURA 4 | **Swapping**



Fonte: https://acervolima.com/diferenca-entre-paging-e-swapping-no-so/.

Na Figura 4 vemos que o movimento chamado *swap out* é o de ida da memória principal para a secundária. E o movimento *swap in* é o da memória secundária para a memória principal.

4. MEMÓRIA VIRTUAL

Como já vimos, os programas que demandam a memória são, na maioria das vezes, maiores do que o espaço oferecido a eles.

A memória virtual é uma técnica que utiliza as memórias principal e secundária combinadas de forma que, para o usuário, há a sensação de que a memória disponível é muito maior do que a memória principal.

Essa técnica tem a vantagem de permitir a execução de programas sem qualquer preocupação com seu tamanho, além de tornar possível uma quantidade maior de programas sendo atendidos.

Com a memória virtual os programas não fazem referência a endereços físicos, mas, sim, a endereços *virtuais*. No momento em que é necessário acessar um desses endereços, o sistema operacional se encarrega de traduzir o endereço virtual em um endereço físico e essa operação é denominada *mapeamento*.

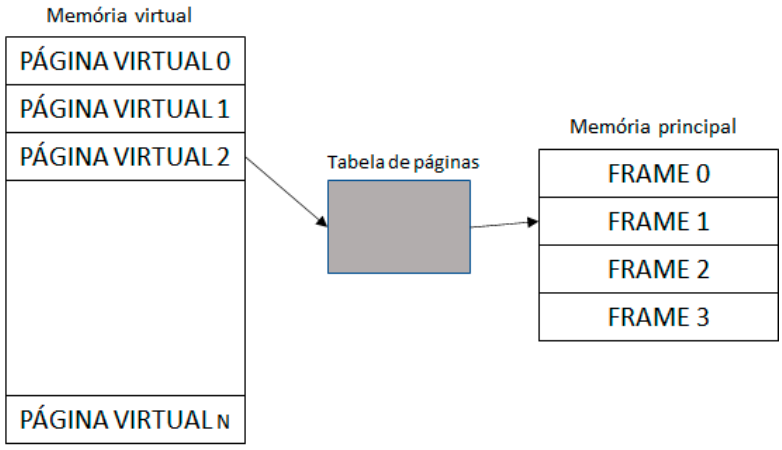
A implementação dessa técnica se dá de duas formas: a *paginação* e a *segmentação*.

Na técnica de paginação, os espaços de endereçamento virtual e real são divididos em pedaços do mesmo tamanho, chamados *páginas*.

Na paginação o sistema operacional utiliza uma estrutura chamada *tabela de páginas*, na qual armazena todas as informações sobre cada página do respectivo processo. Cada processo tem sua própria tabela de páginas.

Quando um programa é executado, suas páginas são transferidas por demanda para *frames* (molduras) na memória principal. Nesta técnica um endereço virtual é formado pelo número da página a que ele pertence mais um deslocamento e o endereço físico é calculado a partir do endereço do frame onde a página se encontra mais o deslocamento. O endereço do frame é encontrado na tabela de páginas.

FIGURA 5 | **Sistema de paginação**



Elaborado pelo autor.

Observe a Figura 5. Pelo que indica a tabela de páginas, a página virtual 2 (PV2) foi carregada no frame 1 (F1). Queremos saber qual o endereço físico correspondente ao endereço virtual PV2 + 4. Suponha que o início de F1 esteja no endereço 1048. Como PV2 está carregada nesse frame, o endereço físico correspondente a PV2 + 4 será 1052 (1048 + 4).

Para o sistema de paginação funcionar corretamente, é necessário que o sistema operacional lide com algumas situações.

A primeira delas é o fato de que há mais páginas nos programas do que frames na memória. Assim, o sistema operacional precisará considerar quais páginas estarão carregadas na memória principal e, quando todos os frames estiverem ocupados, será necessário escolher uma das páginas da memória principal por outra que está chegando da memória secundária.

Há três políticas de substituição de páginas:

1) FIFO (*First In First Out*) – substitui a página que está carregada há mais tempo;

2) LFU (*Least Frequently Used*) – substitui a página que sofreu a menor quantidade de acessos;

3) LRU (*Least Recently Used*) – substitui a página que está há mais tempo sem ser acessada.

Outro aspecto que precisa ser considerado pelo sistema operacional é o controle de quais páginas estão carregadas na memória principal e quais não estão carregadas. Para isso ele usa, na tabela de páginas, um *bit de validade* para cada página, indicando se ela está (1) ou não está (0) carregada na memória.

Finalmente, o sistema operacional também precisa saber se uma determinada página sofreu ou não alteração em seu conteúdo na memória principal. Para isso ele usa o *dirty bit,* que é atualizado para 1 se a página sofreu alterações e permanece 0 se ela não tiver sofrido alterações.

Embora o sistema de paginação seja simples de implementar, como ele não considera a estrutura lógica dos programas ao dividi-los em páginas, isso pode não ser eficiente quando uma estrutura está em parte numa página e em parte em outra.

Na técnica de segmentação, o espaço de endereçamento virtual é dividido em blocos de tamanhos diferentes, de acordo com a estrutura dos programas e há a possibilidade de que os segmentos variem de tamanho para suportar estruturas de dados dinâmicas.

Quanto ao mapeamento de endereços virtuais em endereços físicos, os procedimentos são semelhantes aos da paginação, com os endereços sendo formados pelo número do segmento somado a um deslocamento, no espaço virtual e ocorrendo o mesmo no espaço real, considerando o endereço inicial onde o respectivo segmento foi carregado.

CONSIDERAÇÕES FINAIS

Chegamos ao final dessa aula, na qual vimos como é feito o gerenciamento da memória pelo sistema operacional.

Vimos que são quatro as funções de gerência de memória: controle de ocupação, alocação, proteção e swapping.

Também vimos como o sistema operacional aloca memória, abordando os sistemas de alocação.

Finalmente vimos a técnica de memória virtual, fundamental para os sistemas atuais, uma vez que permite que programas maiores do que o espaço de endereçamento disponível possam ser executados sem problemas.

A memória virtual pode ser implementada basicamente por duas técnicas: a paginação e segmentação.

MATERIAIS COMPLEMENTARES

Neste vídeo você poderá recordar e aprofundar o que vimos aqui a respeito de paginação e segmentação. Disponível: https://www.youtube.com/watch?v=TB19DZBVWRw

REFERÊNCIAS

STALLINGS, William. *Arquitetura e Organização de Computadores: projeto para o desempenho*. 8ª edição. Editora Pearson. Livro. (642 p.). ISBN 9788576055648. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576055648>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Sistemas Operacionais Modernos*. 3ª edição. Editora Pearson. Livro. (674 p.). ISBN 9788576052371. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576052371>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Organização estruturada de computadores*. 6ª edição. Editora Pearson. Livro. (628 p.). ISBN 9788581435398. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788581435398>. Acesso em: 16 out. 2022.

AULA 4 – SINCRONIZAÇÃO ENTRE PROCESSOS

OBJETIVO DA AULA

Abordar questões de sincronização, *deadlocks*, *starvation* e condições de corrida.

APRESENTAÇÃO

Os sistemas operacionais atualmente lidam com diversos processos na memória (multitarefa) e esses processos disputam os recursos de *hardware* e *software* do computador, o que muitas vezes pode ocasionar alguns conflitos que precisam ser detectados e solucionados.

Além disso, os processos podem trocar dados e informações e isso precisa ser feito sincronizadamente para evitar erros e inconsistências.

Nesta aula veremos como o sistema operacional atua no sentido de evitar e resolver conflitos e promover a sincronização entre processos, garantindo o correto funcionamento do computador.

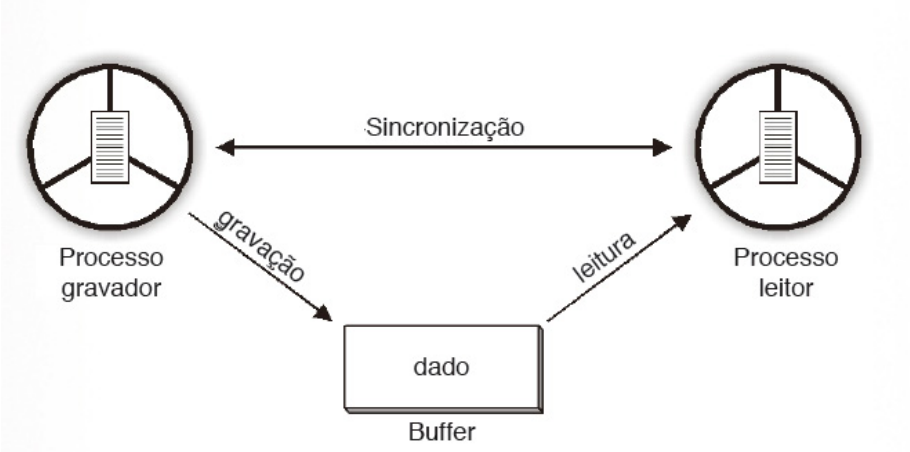
Mãos à obra!

1. INTRODUÇÃO

Com a chegada dos sistemas operacionais multiprogramáveis (multitarefa), surgiu a possibilidade de diferentes programas poderem compartilhar recursos e se comunicar, muitas vezes um processo produzindo informações que funcionam como dados para outro processo.

A comunicação entre processos pode ser realizada através de variáveis compartilhadas na memória principal ou diretamente através do envio de mensagens.

FIGURA 1 | **Sincronização/comunicação entre processos**



Fonte: https://sites.google.com/site/profferreirabrito/aulas/2-sincronizacao-e-comunicacao-de-processos.

Na Figura 1, dois processos utilizam um *buffer* para se comunicar. O processo gravador produz uma informação lida pelo processo leitor. As operações de gravação e leitura devem ocorrer rigorosamente nessa ordem, para evitar a passagem de informações erradas. Esse processo é chamado *sincronização* de processos.

Vamos examinar uma situação que mostra os problemas que podem ser causados por falha na sincronização de processos.

Suponha que dois processos, P1 e P2, manipulam uma variável compartilhada, A. P1 soma um ao valor de A e P2 soma três ao valor de A. Se o valor inicial de A é zero, espera se que seu resultado final, após as execuções de P1 e P2, seja quatro.

A execução de uma instrução de soma é desmembrada em instruções de máquina, que podem ser representadas de forma simples abaixo.

P1	P2
Carregue A, R1	*Carregue A, R2*
Some R1, 1	*Some R2, 3*
Armazene R1, A	*Armazene R2, A*

Se P1 for interrompido após a primeira instrução, teremos o registrador R1 com o valor inicial de A (zero). Se P2 executar suas três instruções antes que P1 retome o processador, ele pegará também o valor inicial de A e somará três a esse valor, e A terminará com valor igual a três.

Em seguida, quando P1 retomar sua execução, ele trabalhará com o valor guardado em R1 (zero), somará um a esse valor e A terminará com valor igual a um e como ele deveria valer quatro, teremos a ocorrência de um erro.

Esse problema é conhecido como *condição de corrida*, e deve ser evitado pelo sistema operacional.

2. EXCLUSÃO MÚTUA

Como vimos, as condições de corrida são nocivas à integridade do sistema, uma vez que podem levar a resultados inconsistentes.

A solução mais simples para evitar que dois ou mais processos acessem um recurso compartilhado e provoquem o erro que vimos é garantir a *exclusão mútua* na utilização desses recursos. De forma simples, a exclusão mútua significa que processos que compartilham recursos só podem usar tais recursos, um de cada vez.

O trecho de código que manipula recursos compartilhado é chamado de *região crítica* e o sistema operacional precisa fornecer protocolos para que as regiões críticas sejam executadas. Esses protocolos são ações que devem ser executadas antes e depois de a execução

de uma região crítica para garantir que os recursos sejam protegidos de erros e não fiquem indefinidamente presos a um processo.

Uma das soluções de *hardware* para isso é a desabilitação das interrupções. Sempre que um processo começar a usar um recurso compartilhado, o sistema operacional impede que ocorram interrupções até que todas as instruções da região crítica tenham sido executadas. Assim, no exemplo que vimos, quando P1 começou a usar a variável A, as interrupções seriam desabilitadas e todas as instruções teriam sido executadas normalmente até o fim, sem o risco de outro processo manipular a mesma variável.

A desabilitação de interrupções é conhecida como um tipo de solução de *hardware*.

Mas a garantia da exclusão mútua também tem soluções de *software* e a mais conhecida delas é a utilização de *semáforos*. Essa solução foi proposta por E. W. Dijkstra em 1965.

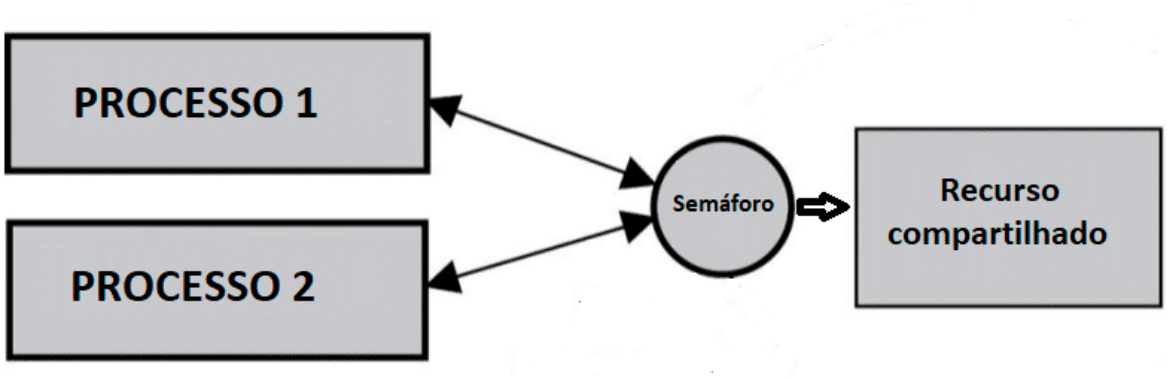
Um semáforo é uma variável inteira e não negativa, sobre a qual só podem ser executadas duas instruções: UP e DOWN. A primeira delas incrementa em uma unidade o semáforo e a segunda o decrementa em uma unidade.

O semáforo é associado a um recurso compartilhado de forma que, quando seu valor for igual a 1, o recurso não está sendo utilizado por nenhum processo e quando for igual a 0, o recurso está em uso.

Seu funcionamento é simples. Quando um processo precisa usar um recurso compartilhado, ele executa a instrução DOWN (protocolo de entrada) no semáforo. Se este estiver valendo 1, ele é decrementado para 0 e o processo pode entrar em sua região crítica. Se ele estiver valendo 0, o processo que executou o DOWN fica impedido de utilizar o recurso e entre em estado de espera, aguardando sua liberação.

Ao terminar de utilizar um recurso, o processo executa a instrução UP (protocolo de saída) e o sistema operacional seleciona um dos processos que está com um DOWN pendente na fila e este pode utilizar o recurso.

FIGURA 2 | **Semáforo para controle de recursos compartilhados**



Elaborado pelo autor.

A importância dos protocolos de entrada e saída das regiões críticas é que, se eles não forem executados corretamente, um ou mais processos poderão ficar aguardando por tempo indefinido a liberação de um recurso e entrar num estado chamado *starvation*.

Por exemplo, suponha que um processo executou a instrução DOWN, utilizou o recurso e não executou a instrução UP ao terminar ou que o sistema operacional desabilitou as interrupções para um processo acessar sua região crítica e não as habilitou novamente ao final. Todos os processos que estiverem aguardando em estado de espera para utilizar tais recursos não conseguirão entrar em execução até que o problema seja resolvido.

3. TROCA DE MENSAGENS

Outra forma de comunicação e sincronização entre processos é a troca de mensagens. Este é um mecanismo que dispensa a existência de variáveis compartilhadas e seu funcionamento se baseia na existência de um canal de comunicação entre os processos que se comunicam e na execução dos comandos SEND e RECEIVE.

É importante que os processos envolvidos na comunicação tenham suas ações sincronizadas, já que não é possível receber uma mensagem que ainda não foi enviada, por exemplo.

A troca de mensagens pode se realizar por *comunicação direta* ou *comunicação indireta*.

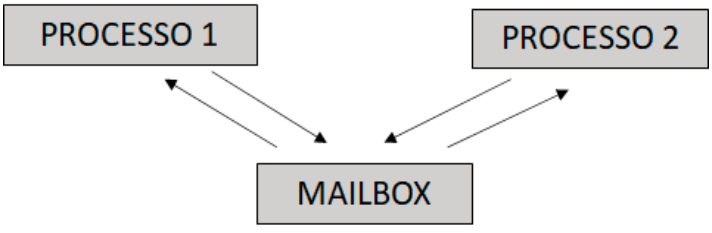
No primeiro caso, os processos envolvidos enderecem de forma explícita o emissor e o receptor e este tipo de comunicação só é possível entre dois processos conforme mostra a Figura 3.

FIGURA 3 | **Comunicação direta**

Elaborado pelo autor.

No caso da comunicação indireta, os processos utilizam uma área compartilhada, chamada *mailbox* ou *port*, cujas características, como, por exemplo, o tamanho, são definidas no momento de sua criação. Neste caso, os parâmetros de endereçamento não se referem mais a processos e, sim, a *mailboxes*. A Figura 4 ilustra a comunicação indireta entre processos.

FIGURA 4 | **Comunicação indireta**



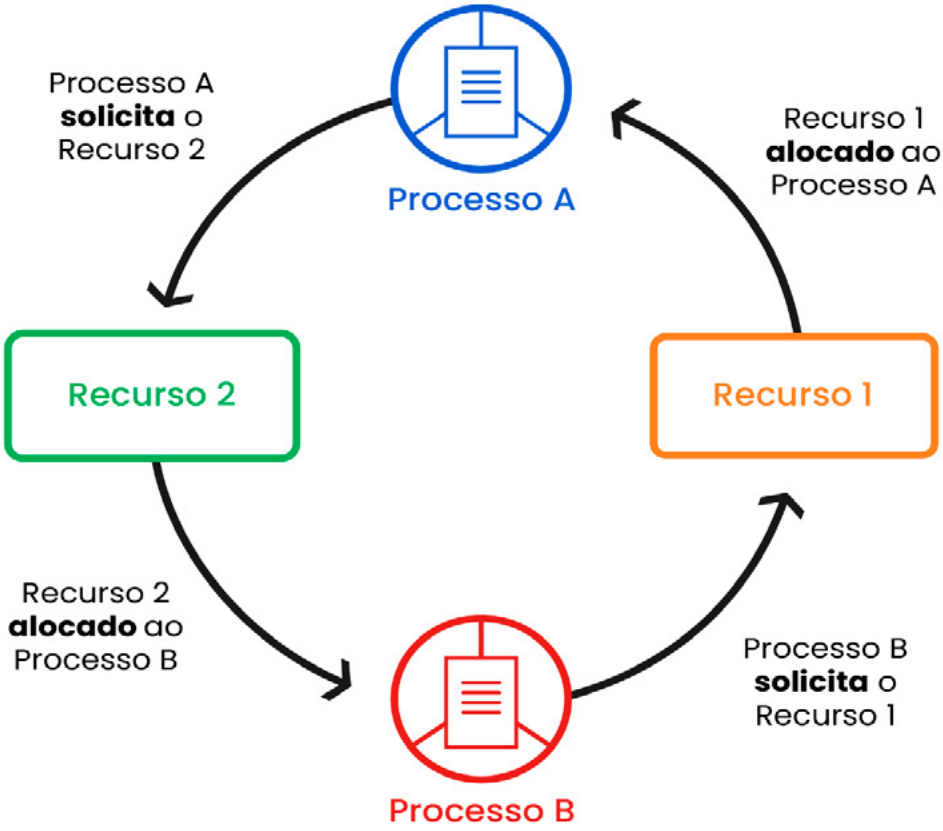
Elaborado pelo autor.

4. DEADLOCK

Deadlock é uma situação em que os processos aguardam por um recurso que não está disponível ou por um evento que nunca ocorrerá.

A tradução da palavra deadlock é *impasse* e isso nos ajuda a começar a entender o que é isso e suas consequências.

FIGURA 5 | **Deadlock**



Fonte: https://saulo.arisa.com.br/wiki/index.php/Comunica%C3%A7%C3%A3º_entre_Processos)

A Figura 5 mostra um deadlock, numa situação conhecida como *espera circular*. Como podemos ver, o processo A está com o recurso 1, mas precisa também do recurso 2 para entrar em execução. Da mesma forma, o processo B está com o recurso 2 e também precisa do recurso 1 para entrar em execução. Dessa forma, nenhum dos dois consegue prosseguir, o que caracteriza um deadlock que, se persistir, poderá envolver outros processos e, num caso mais drástico, causar o travamento do computador.

Há quatro condições que podem levar aos deadlocks:

1) Exclusão mútua: um recurso só pode estar alocado a um processo em um dado momento;

2) Espera por um recurso: um processo pode estar com a posse de alguns recursos e precisar de outros para poder entrar em execução;

3) Não preempção: não liberar um dos recursos de um processo porque outros processos precisam dele;

4) Espera circular: como mostrado na Figura 5, um processo pode estar esperando um recurso alocado a outro processo e vice-versa.

Como gerente de recursos, o sistema operacional precisa atuar da seguinte forma quanto aos deadlocks:

- Prevenção de deadlocks: o sistema operacional deverá garantir que as condições para a ocorrência de deadlocks não sejam satisfeitas;
- Detecção de deadlocks: o sistema operacional deverá ser capaz de perceber a ocorrência de um deadlock o mais rápido possível;
- Diagnóstico de deadlocks: o sistema operacional deverá descobrir a razão da ocorrência do deadlock;
- Solução do deadlock: o sistema operacional deverá tomar alguma providência para desfazer o deadlock. Isso implica em executar algoritmos de *escolha de vítima*, que consistem em sacrificar um dos processos envolvidos no deadlock para que os outros possam prosseguir.

É importante observar que a eficiência do sistema operacional na atuação quanto aos deadlocks podem tornar esse problema, na maioria das vezes, imperceptível ao usuário.

CONSIDERAÇÕES FINAIS

Chegamos ao final de mais uma aula em que vimos a importância das funções de sincronização e comunicação entre processos executada pelo sistema operacional.

Muitos processos compartilham recursos e trocam informações e isso precisa ser rigorosamente controlado para evitar inconsistências na execução dos programas.

Eventos como condições de corrida e deadlocks devem ser evitados pelo sistema operacional porque afetam resultados e eventualmente travam o computador e, portanto, é necessário que o sistema operacional aja preventivamente.

MATERIAIS COMPLEMENTARES

Neste vídeo você poderá aprender um pouco mais sobre deadlocks e seu tratamento. Disponível: https://www.youtube.com/watch?v=SuqbMSnNSEk.

Assista a esse vídeo bastante ilustrativo quanto às condições de corrida. Disponível em: https://www.youtube.com/watch?v=D1S6MlH1l0U.

REFERÊNCIAS

STALLINGS, William. *Arquitetura e Organização de Computadores: projeto para o desempenho*. 8ª edição. Editora Pearson. Livro. (642 p.). ISBN 9788576055648. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576055648>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Sistemas Operacionais Modernos*. 3ª edição. Editora Pearson. Livro. (674 p.). ISBN 9788576052371. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576052371>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Organização estruturada de computadores*. 6ª edição. Editora Pearson. Livro. (628 p.). ISBN 9788581435398. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788581435398>. Acesso em: 16 out. 2022.

AULA 5 – GERÊNCIA DE DISPOSITIVOS DE ENTRADA E SAÍDA

OBJETIVO DA AULA

Entender a participação do sistema operacional nas operações de entrada e saída.

APRESENTAÇÃO

Olá,

O subsistema de entrada e saída é responsável pela comunicação do computador com o mundo externo a ele e, como os outros subsistemas (processamento e armazenamento), é gerenciado pelo sistema operacional.

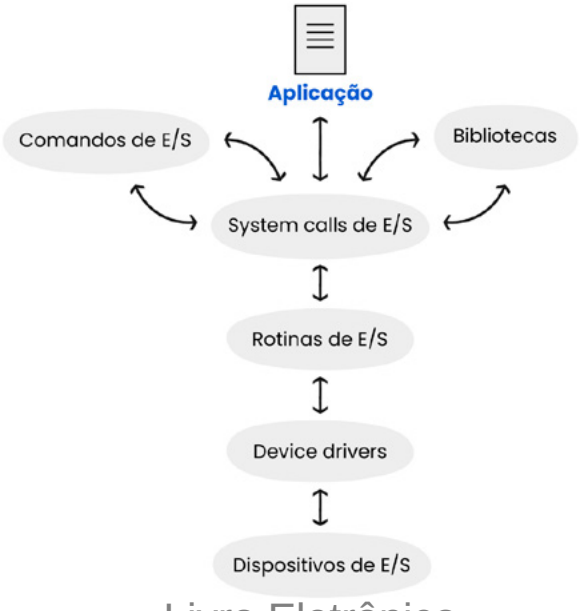
Nesta aula veremos como o sistema operacional gerencia as operações de entrada e saída e como os sistemas de arquivos são organizados.

Então, vamos começar!

1. INTRODUÇÃO

A gerência de dispositivos de entrada e saída é realizada em um sistema de camadas no qual os níveis mais baixos se aproximam do *hardware* e os níveis mais altos se aproximam da interface com o usuário, transformando essas complexas operações em algo mais transparente ao usuário.

FIGURA 1 | **Arquitetura em camadas da gerência de dispositivos**



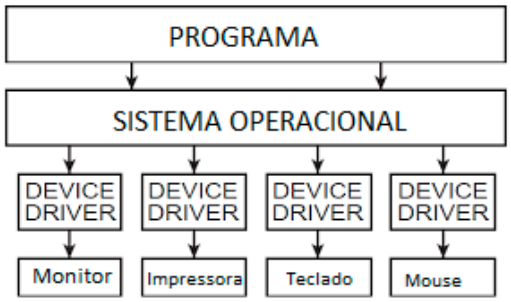
Fonte: http://wiki.foz.ifpr.edu.br/wiki/images/a/a6/Gerencia-dispositivos.pdf.

A Figura 1 ilustra o sistema em camadas da gerência de dispositivos. Com essa estrutura a comunicação com qualquer tipo de dispositivo de E/S é facilitada.

Uma das vantagens desse tipo de estrutura é que torna comunicação com os dispositivos independente de suas características específicas, dando mais flexibilidade ao sistema.

Os *device drivers*, ou simplesmente, *drivers,* funcionam como interfaces entre o subsistema de E/S e cada um dos dispositivos através de suas controladoras. Seu funcionamento é muito simples: eles recebem comandos de camadas superiores e os transformam em comandos específicos do dispositivo controlado. Cada tipo de dispositivo tem suas próprias controladoras e drivers, conforme podemos ver na Figura 2.

FIGURA 2 | **Device drivers**



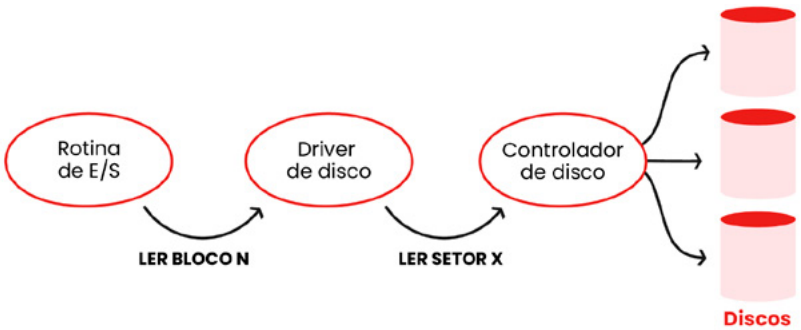
Fonte: https://www.testandtrack.io/usa/index.php/studenttest/givetest/328.

Os drivers são programas que estão diretamente integrados à controladora de um dispositivo ou grupo de dispositivos e, desta forma, trabalham com comandos e instruções específicos desses dispositivos.

Isso é transparente para o processador, uma vez que este não precisa enviar comandos específicos para uma impressora ou um disco rígido, por exemplo, o que além de desperdiçar tempo de processamento das instruções dos programas de usuário, deixaria o sistema bastante limitado em relação aos dispositivos que poderiam ser reconhecidos e usados.

Para o usuário isso também é muito vantajoso, já que este pode escolher os dispositivos que quiser usar conforme a sua conveniência de custo e benefício.

FIGURA 3 | **Drivers de disco**



Fonte: https://slideplayer.com.br/slide/1264625/

Como podemos observar na Figura 3, as rotinas de E/S enviam comandos de alto nível para os drivers que os “traduzem” para que a controladora possa realizar as operações.

As controladoras são componentes de *hardware* cuja função é manipular os dispositivos de E/S.

Sendo dotadas de registradores e memória próprios, elas agem como processadores, executando instruções de baixo nível e utilizando também técnicas de cache para agilizar as operações.

Além disso, como já vimos em aulas anteriores, as controladoras participam do mecanismo de interrupções para entregar ou solicitar serviços e dados do processador.

2. SISTEMAS DE ARQUIVOS

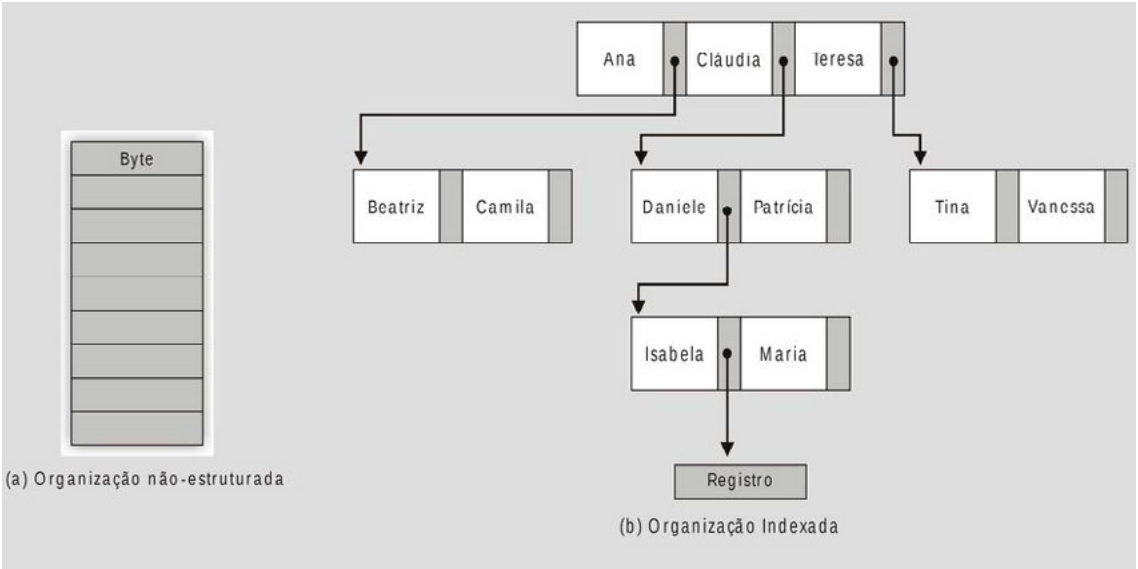
Um arquivo é a reunião de informações relacionadas, identificadas por um nome e alguns atributos, tais como tipo, tamanho, data e hora da criação, data e hora da última atualização e localização.

Os arquivos criados e mantidos no sistema de armazenamento do computador precisam estar organizados de forma que sejam encontrados e acessados sempre que necessário de forma rápida e segura.

Os sistemas de arquivos definem como eles são armazenados e como podem ser salvos ou recuperados.

Uma das formas de se organizar os arquivos é como uma sequência não estruturada de bytes. Outra forma é a *indexada*, como mostrado na Figura 4 abaixo.

FIGURA 4 | **Organização de arquivos**



Fonte: https://docplayer.com.br/15062450-Sistema-de-arquivos-sistemas-de-arquivos.html.

Outro aspecto muito importante em relação aos sistemas de arquivos é seu método de acesso, que define de que forma os registros que compõem os arquivos podem ser recuperados (buscados) ou alterados.

Os métodos de acesso podem ser de dois tipos: sequencial o direto.

No método sequencial o acesso a um registro só ocorre após passar por todos os registros anteriores. Este é o método utilizado em fitas magnéticas.

Já no método direto (ou randômico) a leitura ou gravação de um registro ocorre diretamente no endereço e não há qualquer restrição quanto à ordem em que os registros foram gravados.

Quanto à organização dos arquivos em um disco, a estrutura de diretórios é amplamente usada. Esta estrutura consiste em organizar os arquivos de uma forma hierárquica de pastas e subpastas

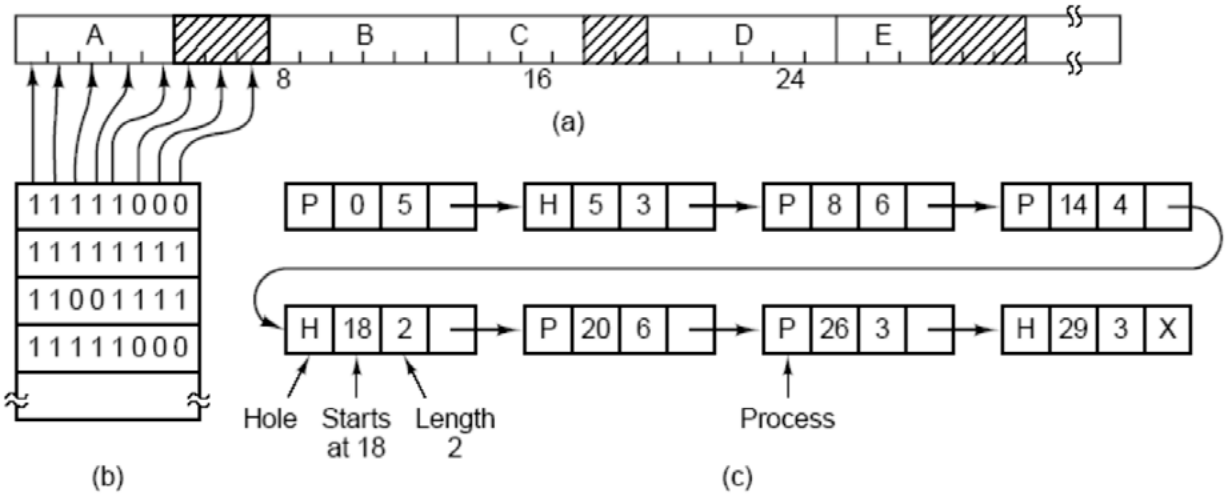
3. GERÊNCIA DE OCUPAÇÃO NO DISCO

Assim como vimos em relação à gerência de memória, o controle de ocupação do disco e demais dispositivos de memória secundária também é uma tarefa do sistema operacional.

A primeira forma de realizar essa tarefa é a utilização de uma tabela chamada *mapa de bits* (bitmap). A cada bit dessa tabela está associado um bloco da memória de forma que, se o bit vale 0, o respectivo bloco está vazio e disponível. Caso contrário (bit = 1), o espaço está ocupado.

Outra forma de organização é a *lista encadeada*, na qual cada nó indica se o espaço está ocupado (P), vazio (H), seu endereço inicial e tamanho. Observe a Figura 5.

FIGURA 5 | **Controle de espaço no disco**



Fonte: http://professor.pucgoias.edu.br/SiteDocente/admin/arquivosUpload/17785/material/AULA%2013%20-%20Gerncia%20de%20Memria.pdf.

Na Figura 5a temos a representação do espaço em disco. A Figura 5b mostra a representação através do mapa de bits, indicando 0 para as posições vazias e 1 para as posições ocupadas.

A Figura 5c mostra a representação do mesmo espaço utilizando a lista encadeada. Observe que:

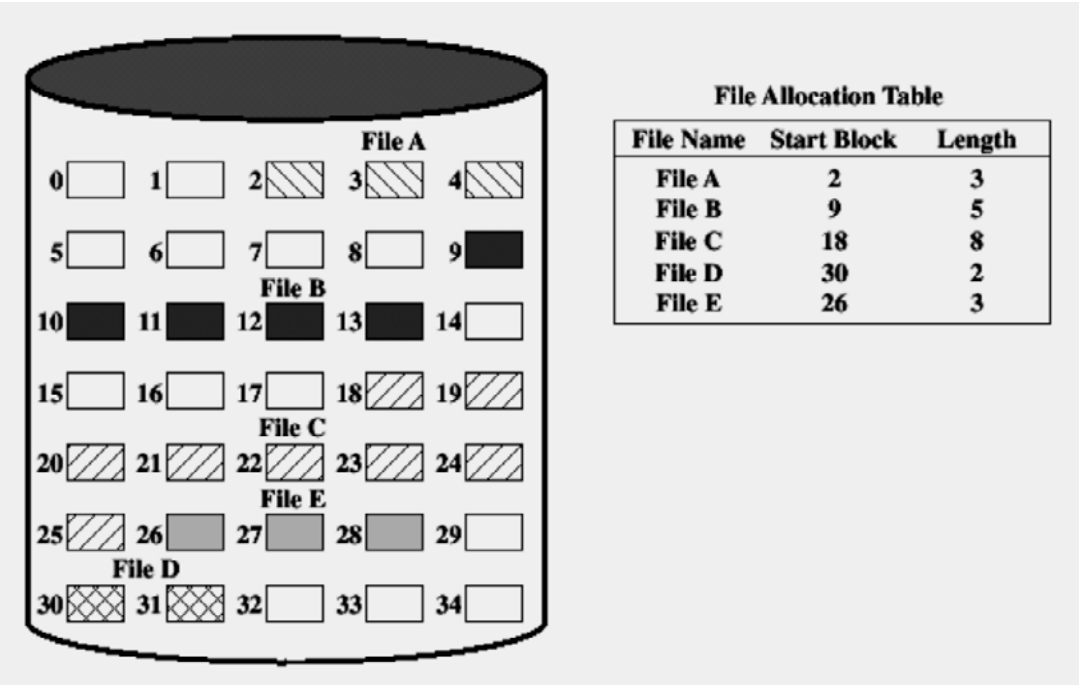
- Cada bloco (vazio ou ocupado) aponta para o próximo bloco;
- A representação de cada bloco possui H ou P para informar se o bloco está vazio ou ocupado; o endereço onde o bloco começa (*starts*) e o tamanho do bloco (*lengh*).

4. ALOCAÇÃO DO ESPAÇO EM DISCO

A alocação do espaço em disco pode se dar de três formas:

1) Alocação contígua – os arquivos são salvos em blocos vizinhos no disco e o endereço é o do primeiro bloco usado (start *block)* e a tabela de alocação também registra a quantidade de blocos usada (lenght), conforme mostra a Figura 6.

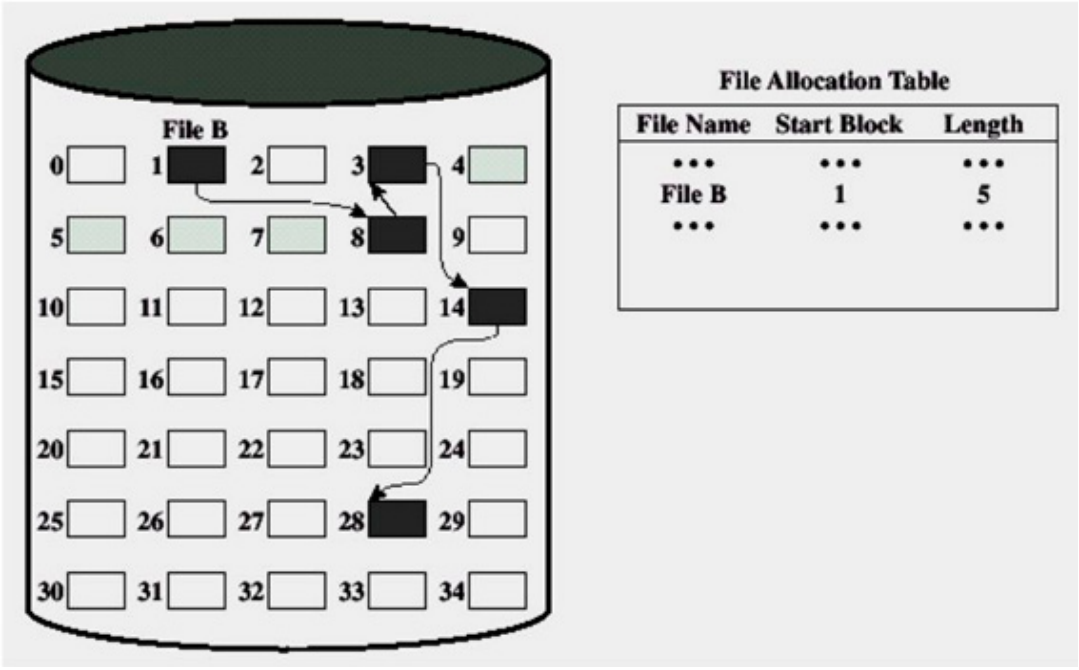
FIGURA 6 | **Alocação contígua**



Fonte: http://www.inf.ufrgs.br/~asc/livro/transparencias/cap8.pdf.

2) Alocação encadeada – nesta forma de alocação os "pedaços" dos arquivos são organizados como um conjunto de blocos ligados logicamente, conforme mostrado na Figura 7.

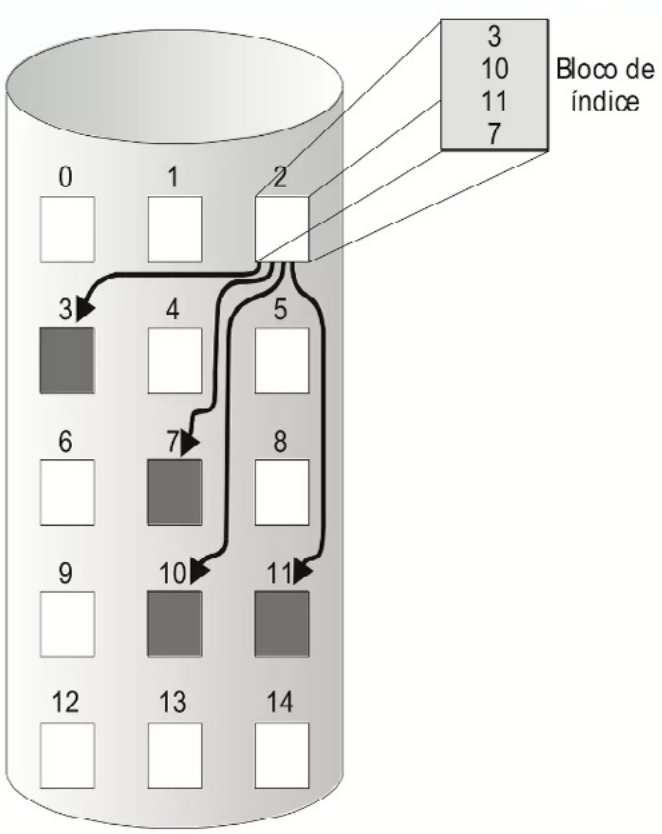
FIGURA 7 | **Alocação encadeada**



Fonte: *https://slideplayer.com.br/slide/3647985/.*

2) Alocação indexada – nessa forma de alocação os ponteiros para todos os blocos dos arquivos ficam em uma tabela chamada *bloco de* índice, como mostrado na Figura 8.

FIGURA 8 | **Alocação indexada**



Fonte: *https://pt.slideshare.net/PauloFonseca1/apostila-8-sistema-de-arquivos-35747696.*

CONSIDERAÇÕES FINAIS

Nesta aula vimos que a gerência de dispositivos de E/S utiliza meios intermediários para tornar essa tarefa mais eficiente e tornar o computador mais flexível, além de exigir menos participação do processador nessas operações.

As controladoras e os drivers funcionam como interface entre os comandos de alto nível e as instruções em baixo nível que manipulam os dispositivos.

Vimos também como o sistema operacional age para organizar e acessar os arquivos e gerenciar o controle de ocupação dos discos.

Como o entendimento dessas funções finalizamos o estudo dos sistemas operacionais como gestores do funcionamento dos computadores.

MATERIAIS COMPLEMENTARES

Neste vídeo você poderá ampliar seus conhecimentos sobre sistemas de arquivos. Disponível: https://www.youtube.com/watch?v=Ol7sjPn8B4º.

REFERÊNCIAS

STALLINGS, William. *Arquitetura e Organização de Computadores: projeto para o desempenho*. 8ª edição. Editora Pearson. Livro. (642 p.). ISBN 9788576055648. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576055648>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Sistemas Operacionais Modernos*. 3ª edição. Editora Pearson. Livro. (674 p.). ISBN 9788576052371. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576052371>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Organização estruturada de computadores*. 6ª edição. Editora Pearson. Livro. (628 p.). ISBN 9788581435398. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788581435398>. Acesso em: 16 out. 2022.

CONSIDERAÇÕES FINAIS DA UNIDADE

Chegamos ao final desta unidade, que foi dedicada aos conceitos a respeito de sistemas operacionais.

Vimos que o sistema operacional tem basicamente duas funções: interface com o usuário e gerência de recursos. Estudamos como o sistema operacional lida com o processador, a memória e finalmente aos sistemas de entrada e saída.

A gerência do processador consiste em controlar o tempo que cada processo vai gastar usando o processador e em organizar a fila dos processos na memória.

A gerência de memória lida com aspectos de ocupação, alocação, proteção e *swapping*, sendo a memória virtual um recurso que permite a execução de programas maiores do que o espaço disponível.

Vimos que, para implementar a memória virtual são usados os esquemas de paginação e segmentação.

As operações de entrada e saída também são gerenciadas pelo sistema operacional, que, com a ajuda dos drivers e controladoras, torna essas operações mais eficientes, além de flexibilizar o uso de quaisquer tipos de dispositivos.

Finalmente, estudamos os sistemas de arquivo, que tratam de como as informações são salvas e recuperadas nos discos rígidos.