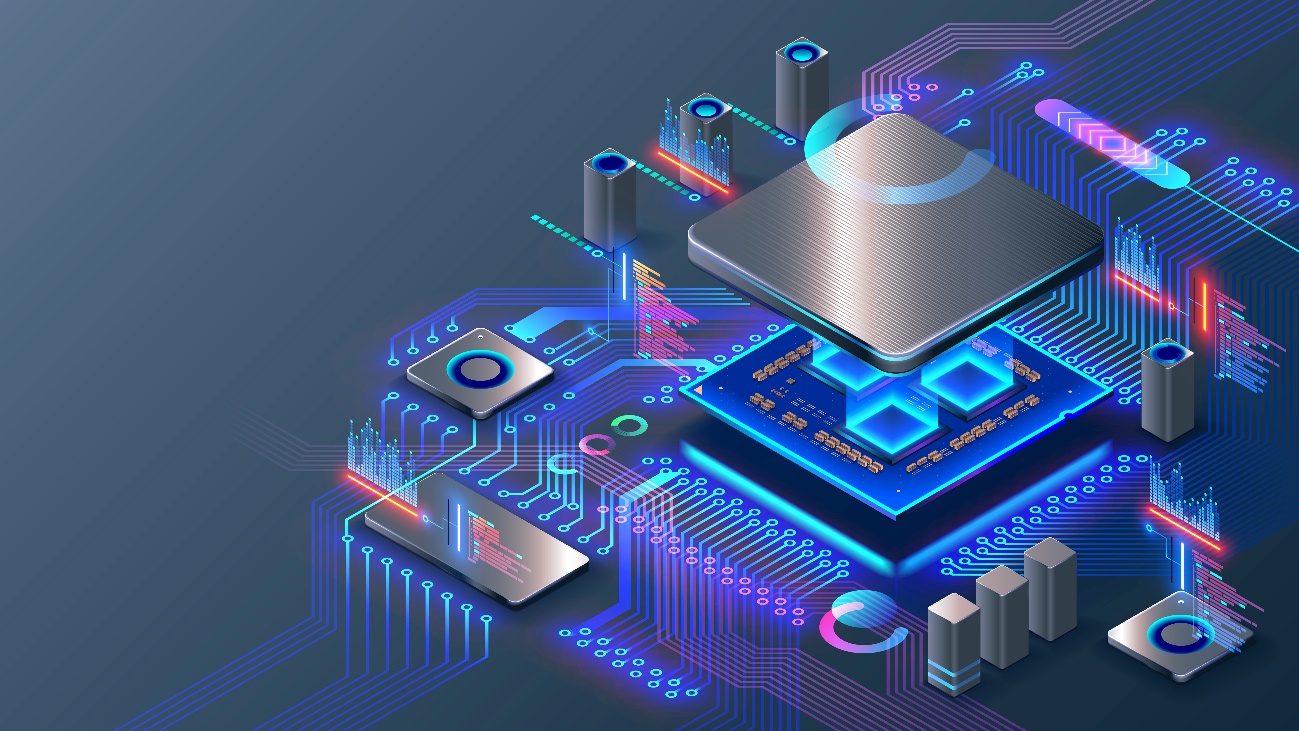
**Introdução da aula**



**Qual é o foco da aula?**

Nessa aula, vamos apresentar os conceitos do *swapping*, os tipos e suas características, como se dá a multiprogramação com partições variáveis e o gerenciamento de memória com mapa de bits, além de entender o funcionamento dos algoritmos de troca de processos.

**Objetivos gerais de aprendizagem**

Ao longo desta aula, você irá:

* definir o método para o gerenciamento de memória: a troca de processos (*swapping*);
* explicar os métodos de gerenciamento de memória com alocação dinâmica: com mapa de bits e com listas encadeadas;
* analisar as estratégias ou algoritmos para definir a área livre dos processos que serão executados por meio da lista encadeada.

**Situação-problema**

O *swapping*(troca de processos) é realizado quando não existe memória principal suficiente para executar todos os programas do computador ao mesmo tempo. No *swapping* um programa é totalmente carregado em memória e executado por um tempo definido, enquanto os demais programas aguardam, em disco, sua vez de executar.

Para exemplificarmos o conceito de *swapping*, imagine que estamos usando o navegador para pesquisar sobre um determinado assunto e, logo que encontramos a informação que buscamos, abrimos o editor de texto para inserir essa informação. Em alguns casos, quando temos vários programas abertos e realizamos essa troca – no caso do navegador e do editor de texto –, percebemos um pouco de lentidão no computador.

Na verdade, o que ocorre é a troca de processos (*swapping*), pois o navegador – que estava sendo executado até então – é tirado da memória e colocado em disco, enquanto o editor de texto, que estava no disco, por falta de utilização, é trazido para a memória e reativado.

Relembrando nosso contexto, você trabalha em uma empresa de infraestrutura como gerente de T.I. e irá representá-la em uma importante feira de tecnologia. Durante a feira, um representante de T.I. de uma empresa de digitalização de documentos visita o seu *stand*.

A empresa digitaliza e gerencia diversos documentos de pessoas físicas e de outras empresas, preservando e prevenindo a perda de documentos importantes. O representante informou que processa um grande volume de documentos diariamente e liberou a memória física dos computadores, usando a partição do HD como uma área de troca (*swap*) em que o sistema pode armazenar áreas de memória que não estão sendo usadas.

Porém quanto menos memória física os computadores usavam e o *swap* era utilizado, ficavam cada vez mais lentos. Assim você é questionado:

* por que ocorre este problema?
* quais as sugestões você dará para resolvê-lo?
* aumentar a memória é sempre a melhor solução para evitar o *swapping* e a lentidão do computador?
* qual o melhor algoritmo para definir em qual área livre o processo deverá ser executado na memória?

Para que você consiga responder esses e outros questionamentos sobre *swapping* e os algoritmos de troca de processos, vamos conhecer mais sobre eles. Lembre-se que ao fim dessa unidade, você deverá entregar um relatório para o seu gestor, com os anteprojetos elaborados durante a feira.

Bons estudos!

**Swapping e multiprogramação com partições variáveis**



Segundo Tanenbaum (2003), em sistemas de compartilhamento de tempo ou computadores pessoais com memória RAM insuficiente para conter todos os processos ativos do sistema, é necessário que eles permaneçam em disco e sejam trazidos dinamicamente para a memória para serem executados somente quando necessário. Neste caso, podem ser utilizados dois métodos para o gerenciamento de memória: a troca de processos (*swapping*) e a memória principal.

***Swapping*: troca de processos**

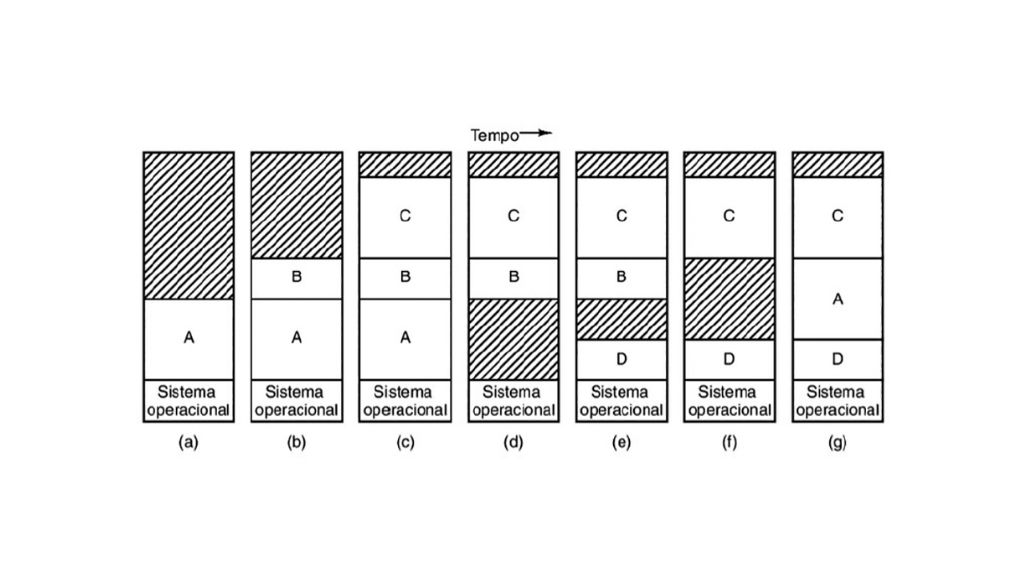
Segundo Tanenbaum (2003), a troca de processos (*swapping*) traz totalmente cada processo para a memória, o executa por algum tempo e o retorna para o disco.

A figura “Troca de processos na memória principal” apresenta o funcionamento da troca de processos, sendo que, inicialmente, somente o processo **A** está na memória (figura (a)).

O processo **B** é trazido do disco para a memória (figura (b)) e em seguida o processo **C** (figura (c)).

O processo **A** é retirado da memória e retorna ao disco (figura (d)) e em seguida o processo **D**é trazido para a memória (figura (e)) e o processo **B** retorna para o disco (figura (f)).

O processo **A** é trazido novamente na memória (figura (g)).

Troca de processos na memória principal. Fonte: Tanenbaum (2003, p. 145).

Segundo Machado e Maia (2007), o *swapping* permite maior compartilhamento da memória principal e utilização dos recursos do sistema computacional. Uma desvantagem é que aumenta o custo operacional de entrada/saída. Quando existe pouca memória RAM disponível, o sistema pode ficar dedicado à execução do *swapping*, deixando de realizar tarefas mais críticas, se tornando ineficiente.

O *swapping* não pode ser utilizado como solução para a questão da falta de memória RAM pelo fato de ser um método lento. Outra questão é que nem sempre adicionando mais memória no computador será resolvido o problema do *swapping* e performance, pois se o HD e a CPU estiverem ultrapassados, influenciarão o desempenho da máquina.

Nos sistemas operacionais atuais, as memórias tendem a ser maiores e raramente acontece a troca entre memória e disco.

\_\_\_\_\_\_

**💭 Reflita**

Qual a vantagem de utilizar o *swapping* uma vez que os computadores atuais possuem memórias com elevada capacidade de processar várias informações ao mesmo tempo? E caso não seja suficiente, o usuário pode adquirir mais memória RAM para ser inserida no computador? Um computador pode ser usado sem *swapping*?

\_\_\_\_\_\_

Além disso, é necessário manter o antivírus do computador sempre atualizado, uma vez que os vírus e programas maliciosos tendem a consumir muita memória, dificultando que programas com prioridade alta executem, forçando a utilização do *swapping*.

Um computador sem a devida utilização do antivírus, também pode ocasionar na demora para carregar os aplicativos e até mesmo travá-lo.

\_\_\_\_\_\_

**📝 Exemplificando**

Para exemplificarmos o conceito de *swapping*, imagine que computador tenha uma memória de 512 MB e você tem quatro processos para serem executados com os tamanhos 481 MB, 508 MB, 380 MB e 369MB, respectivamente. O gerenciador de memória seleciona um processo inteiro para ser executado em memória e os demais processos aguardam em disco a sua vez de executar. Se o primeiro processo (com o tamanho de 481 MB) for selecionado para ser executado, ele é carregado para memória e executado por um tempo determinado.

Assim que o tempo finalizar, o processo retorna para o disco e outro é selecionado para executar. Para a seleção de qual processo será executado são utilizados algoritmos que utilizam de critérios para realizar a escolha.

\_\_\_\_\_\_

Segundo Simioni (2018), no Windows o *swap* é um arquivo que está dentro do diretório raiz (C:/) chamado PAGEFILE.SYS. Este arquivo é criado quando o Windows é instalado. O tamanho do arquivo de *swap*varia de acordo com a quantidade de memória RAM do computador.

Por exemplo, se o seu computador possui 2 GB de memória RAM, o tamanho do arquivo será de 2048MB. Além disso, o tamanho do arquivo pode ser alterado e/ou desativado. No Linux, o *swap* é uma partição no diretório raiz (/) e esta partição pode ser utilizada por vários sistemas operacionais.

\_\_\_\_\_\_

**➕ Pesquise mais**

O grande problema com o uso de *swap* é que o HD é muito mais lento que a memória RAM. Enquanto uma sequência de quatro leituras em um módulo de memória DDR2-800 demora cerca de 35 bilionésimos de segundo, um acesso a um setor qualquer do HD demora pelo menos 10 milésimos.

Para saber mais sobre esse assunto, leia o tutorial hardware para iniciantes, parte 3 [**Swap e cache de disco**](https://www.hardware.com.br/tutoriais/hardware-iniciantes-3/pagina3.html)**.**

\_\_\_\_\_\_

**Multiprogramação com Partições Variáveis**

Segundo Machado e Maia (2007), o esquema conhecido como alocação particionada dinâmica ou variável, consiste em ajustar dinamicamente o tamanho das partições de memória quando os processos chegam para serem executados. Além disso, cada processo utiliza um espaço necessário para executar, não acontecendo a fragmentação interna.

Porém, em alguns casos, pode acontecer a fragmentação externa quando os processos, ao retornarem para o disco, deixarem pequenos espaços na memória, não permitindo que novos processos sejam executados, como por exemplo, o espaço entre os processos B e D (figura “Troca de processos na memória principal” (e)).

A vantagem das partições variáveis é a flexibilidade por não estar preso a um número fixo de partições, melhorando a utilização da memória, porém impactando o gerenciamento das trocas de processos e na alocação e liberação da memória.

Segundo Tanenbaum (2003), quando as trocas deixam muitos espaços vazios é necessário agrupar esses espaços e combiná- los nos endereços mais baixos. Esta técnica é conhecida como compactação de memória, porém não é utilizada devido ao alto tempo de processamento.

**Métodos de gerenciamento de memória com alocação dinâmica**



Quando os processos precisam consumir mais memória durante o processamento, é necessário alocar memória dinamicamente. Existem dois métodos de gerenciamento de memória com alocação dinâmica: com **Mapa de Bits** e com **Listas Encadeadas**.

**Gerenciamento de Memória com Mapa de Bits**

Segundo Tanenbaum (2003), neste método a memória é dividida em unidades de alocação, a qual é associado um bit no mapa de bits. Se o valor do bit for 1, indica que a unidade está ocupada e o se o bit for 0 ela está livre.

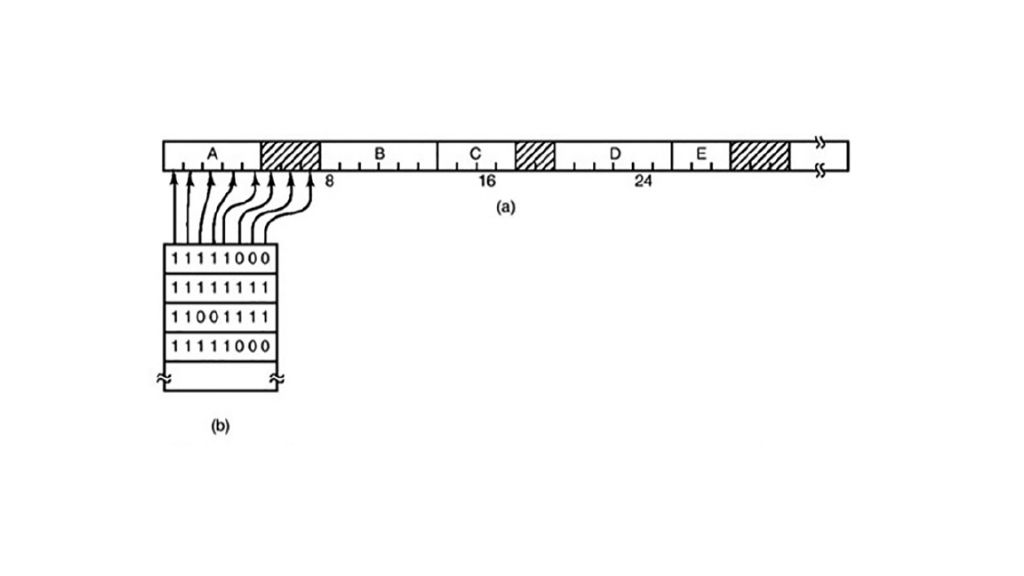
A figura “Mapa de bits e uma parte da memória” apresenta o mapa de bits e a memória dividida em unidades de alocação A figura (a) mostra uma parte da memória com cinco segmentos alocados a processos (**A, B, C, D e E**) e três segmentos de memória livre (espaços vazios). A figura (b) apresenta o mapa de bits correspondente a memória. Segmento é uma área de memória alocada a um processo ou uma área livre de memória entre dois processos.

Por exemplo, na primeira partição da memória representada por 8 bits, foi alocado o processo **A** com cinco unidades de alocação (bits igual a 1) e três unidades de memória livre (bits igual a 0).

Na partição seguinte, o processo **B** alocou seis unidades e o processo **C** somente 2, uma vez que as partições foram divididas em 8 bits cada e não comportava todo o processo.

O processo C utilizou duas unidades de alocação da próxima partição, duas unidades de memória livre e o processo **D** utilizou 4 unidades.

Na última partição do exemplo, o processo **D** utilizou duas unidades de alocação, o processo **E** três unidades e três unidades de memória livre. Quanto menor for a unidade de alocação, maior será o mapa bits.

Mapa de bits e uma parte da memória. Fonte: Tanenbaum (2003, p. 146).

A vantagem do mapa de bits é a simplicidade do gerenciamento de alocação de memória, uma vez que o tamanho do mapa depende somente do tamanho da memória e da unidade de alocação.

A desvantagem surge quando é necessário carregar um processo com um determinado número de unidades consecutivas, necessitando que o gerenciador de memória procure essa sequência no mapa de bits, cujo processo é lento.

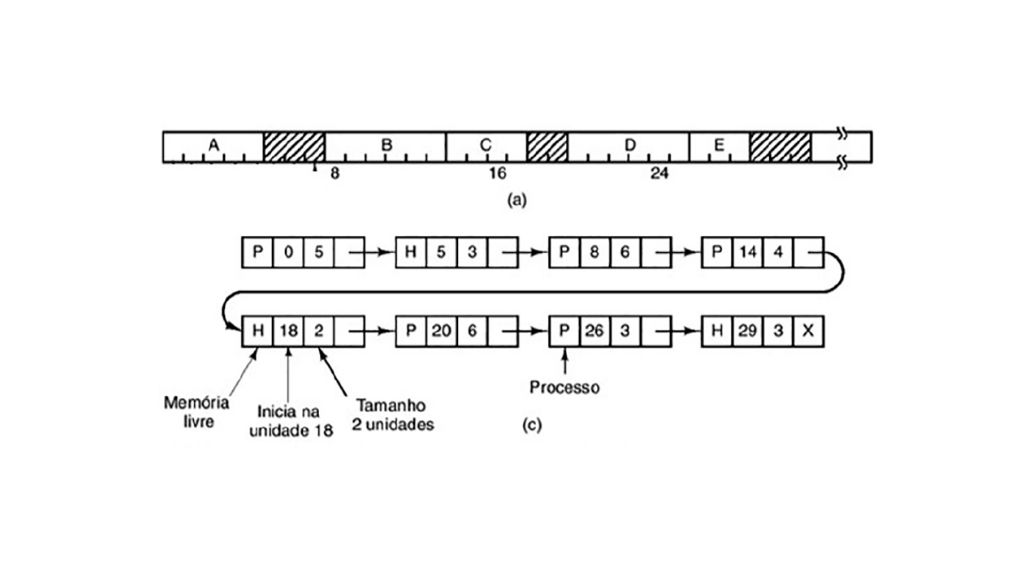
**Gerenciamento de Memória com Listas Encadeadas**

Segundo Tanenbaum (2003), este método consiste em manter uma lista encadeada de segmentos alocados e livres na memória. A figura “Lista encadeada e uma parte da memória (a)” mostra uma parte da memória com cinco segmentos alocados a processos (**A, B, C, D e E**) e três segmentos de memória livre (espaços vazios).

A figura (c) apresenta a lista encadeada e a memória dividida em unidades de alocação. Cada elemento dessa lista especifica um segmento de memória disponível (H) ou de memória alocada ao processo (P), o endereço onde se inicia o segmento e um ponteiro para o próximo elemento da lista.

Por exemplo, o processo **A** possui memória alocada (P), inicia na posição 0 (zero), tem o tamanho de cinco unidades de alocação e possui um ponteiro para o próximo elemento. O próximo elemento é um segmento de memória disponível (H), inicia na posição 5, possui o tamanho de três unidades e aponta para o próximo segmento.

A vantagem deste método é que a lista está ordenada por endereço facilitando a rápida atualização após o término do processo.

Lista encadeada e uma parte da memória. Fonte: Tanenbaum (2003, p. 146).

**Algoritmos de Troca de Processos**



Segundo Machado e Maia (2007), existem estratégias ou algoritmos para definir em qual área livre os processos serão executados por meio da lista encadeada. Estes algoritmos reduzem ou evitam a fragmentação externa.

Confira a seguir os algoritmos de alocação de memória (TANENBAUM, 2003).

* ***First Fit*** (primeiro que couber): é o algoritmo mais simples e que consome menos recurso do sistema. O gerenciador de memória procura ao longo da lista por um segmento livre que seja suficientemente grande para esse processo.
* ***Next Fit*** (próximo que couber): este algoritmo é uma variação do First Fit. A posição em que encontra o segmento de memória disponível é memorizada, não precisando percorrer toda lista quando se quer alocar.
* ***Best Fit*** (melhor que couber): percorre toda lista e escolhe o menor segmento de memória livre suficiente ao processo. Este algoritmo é mais lento uma vez que procura em toda a lista.
* ***Worst Fit*** (pior que couber): sempre é escolhido o maior segmento de memória disponível de maneira que, quando dividido, o segmento disponível restante fosse suficientemente grande para ser útil depois. Simulações mostram que este algoritmo não é uma boa ideia a ser usado.
* ***Quick Fit*** (mais rápido que couber): É um algoritmo rápido e mantém listas separadas por tamanhos de segmentos de memória mais solicitados disponível.

\_\_\_\_\_\_

**🔁 Assimile**

Vimos nesta aula como funcionam os algoritmos de troca de processos. Eles são usados para escolher em qual segmento de memória livre será alocado o processo que será executado. A estratégia a ser usada pelo gerenciador de memória para a escolha de qual algoritmo utilizar irá depender de fatores, como o tamanho dos processos executados no ambiente e das áreas livres disponíveis.

\_\_\_\_\_\_

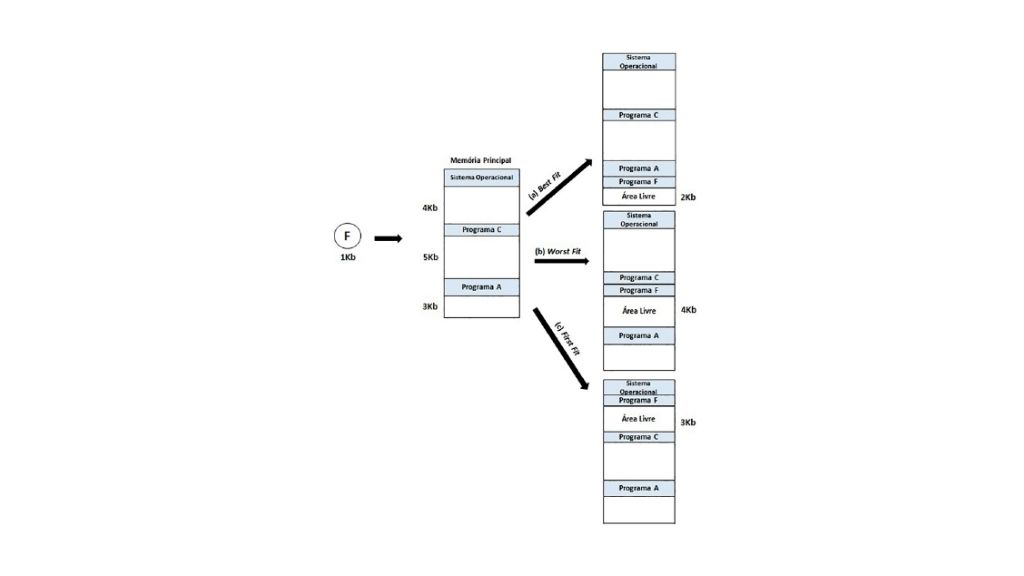
A figura “Algoritmos de alocação de memória” exemplifica o funcionamento dos algoritmos *Best Fit, Worst Fit e First Fit*.

O processo **F**(com tamanho de 1 kb) está aguardando para ser executado na memória principal. Na memória principal está em execução os processos **C** e **A** e contém três segmentos de memória livres com os tamanhos de 4 kb, 5 kb e 3 kb, respectivamente.

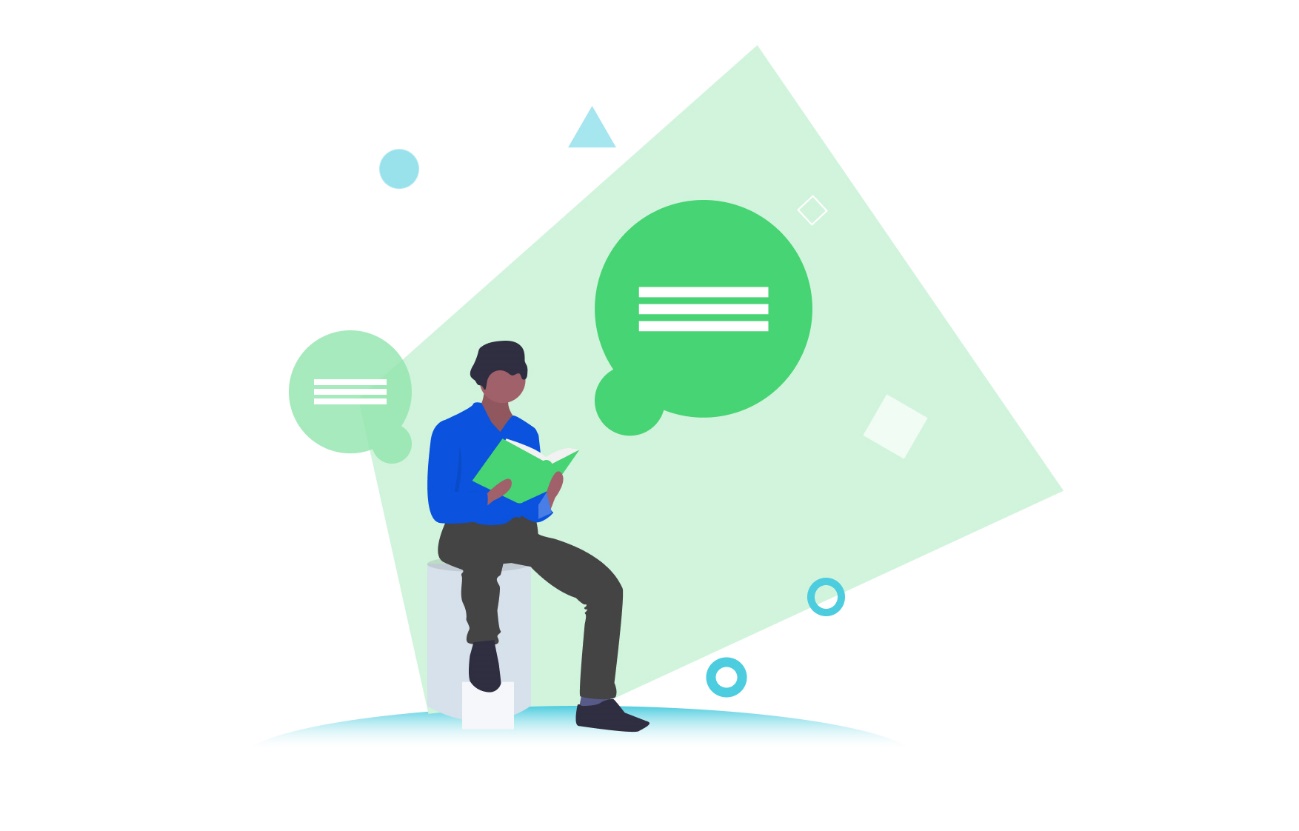
Antes do processo **F** iniciar a execução na memória principal, o algoritmo *Best Fit* seleciona na lista o menor segmento de memória livre. Assim, a partição com o tamanho 3 kb foi selecionada para executar o processo, deixando 2 kb de espaço vazio.

Utilizando o algoritmo *Worst Fit* que escolhe o maior segmento de memória disponível, a partição com o tamanho 5 kb foi selecionada para executar o processo, deixando 4 kb de espaço vazio. Se for utilizado o algoritmo *First Fit* que procura na lista pelo primeiro segmento livre que seja suficientemente grande para esse processo, será selecionada a partição com o tamanho 4 kb para executar o mesmo, deixando 3 kb de espaço vazio.

Nos algoritmos *First Fit* e *Best Fit* ocorre a fragmentação externa.

Algoritmos de alocação de memória. Fonte: Machado e Maia (2007, p. 155).

**Conclusão**



Agora que você já conheceu o funcionamento do *swapping* e dos algoritmos de troca de processos, vamos voltar ao nosso contexto. Você é um gerente de T.I. de uma empresa de infraestrutura e está representando-a numa feira de tecnologia. Um representante de T.I. de uma empresa de digitalização de documentos visita o seu *stand* e relata que digitaliza e gerencia diversos documentos de pessoas físicas e de outras empresas, preservando e prevenindo a perda de documentos importantes.

Ele ainda relatou que os computadores ficavam cada vez mais lentos, apesar de diminuírem o uso de memória física.

* Por que ocorre este problema?
* Aumentar a memória RAM é sempre a melhor solução para evitar o *swapping* e a lentidão do computador?
* Quais as sugestões você dará para resolvê-lo?
* Qual o melhor algoritmo para definir em qual área livre o processo deverá ser executado na memória?

Os sistemas operacionais quando possuem pouca memória RAM utilizam o disco ou um arquivo para processar os programas, carregando-os na memória principal, executando-os e, logo após a execução, o retorna para o disco. Esse método evita que os programas travem, porém impacta diretamente no desempenho do sistema operacional.

O *swapping* não pode ser utilizado como solução para a questão da falta de memória RAM. No caso do estudante, ele reservou uma partição do HD para ser utilizada como uma área de *swapping*. A utilização do *swapping* deixa a máquina mais lenta, porque o processador precisa procurar no disco as informações necessárias.

A memória RAM auxilia o processador, gravando as informações que são usadas frequentemente pelos processos e auxiliando na transferência dos dados entre memória e disco (DAQUINO, 2018). Nem sempre adicionando mais memória será resolvido o problema do *swapping* e performance, pois outros fatores como, HD e a CPU antigos, influenciaram o desempenho da máquina. No caso do representante de T.I. da empesa de digitalização de documentos, é necessário que ele adquira mais memória RAM para o seu computador de forma que um maior número de processos consiga executar diretamente nela, reduzindo o uso do *swapping*.

Além disso, ele deve reduzir o tamanho da área do *swapping* para permitir que a maioria dos processos executem diretamente na memória principal, e somente quando necessário seja realizada a troca de processos. A estratégia utilizada pelo gerenciador de memória para a escolha de qual algoritmo utilizar irá depender de fatores, por exemplo, o tamanho dos processos executados.

O algoritmo *First Fit* (primeiro que couber) é o que consome menos recurso do sistema por escolher a primeira partição livre que comporte a execução do processo, porém existe um grande desperdício de espaço livre.

O algoritmo *Best Fit* (melhor que couber) verifica em toda a memória a melhor partição que comporte a execução do processo, o que o torna mais lento.

Nos algoritmos *First Fit* e *Best Fit* ocorre a fragmentação externa.

Já o algoritmo *Worst Fit* (pior que couber) escolhe a maior partição livre para a execução do processo, porém não é recomendado o seu uso.

Em casos que seja necessário criar mais espaço na memória é recomendado o uso do algoritmo *Worst Fit*. Já para agilidade do processo é recomendado que seja utilizado o algoritmo *Best Fit*.