MEMÓRIAS E DISPOSITIVOS DE ENTRADA E SAÍDA

INTRODUÇÃO

Seguindo a arquitetura de Von Neumann, que define o computador como uma máquina que possui, ao menos, três elementos básicos, que são unidades de processamento, unidades de memória e dispositivos de entrada e saída, estudaremos, nessa unidade, os seguintes assuntos:

01. O que são memórias, como elas participam efetivamente do processamento de informações e como um Sistema Operacional faz o gerenciamento das unidades de memória disponíveis;

02. O processo de utilização de memória virtual, através da qual é possível ampliar a capacidade da memória primária de um sistema computacional;

03. Os dispositivos de interface, também conhecidos como dispositivos de entrada e saída. É através dessas interfaces que o computador é capaz de se comunicar com os usuários e vice-versa.

Utilizando-se desses conceitos, pretende-se que sejam desenvolvidas as seguintes habilidades:

01. Entender a hierarquia de memória e sua utilização no processamento de dados;

02. Compreender como o Sistema Operacional gerencia o uso de memória;

03. Analisar a importância da memória virtual no processamento de dados;

04. Entender os dispositivos de Entrada e Saída como interface com o usuário;

05. Comparar os diversos funcionamentos das interfaces e sua relação com o desempenho computacional.

Dessa forma, a unidade que veremos a seguir traz os conceitos para a compreensão do funcionamento, tanto da memória primária como secundária, bem como a diferenciação entre elas e o uso da memória secundária como parte da memória primária em um processo conhecido como memória virtual. Além disso, a unidade também fala sobre os conceitos de gerenciamento dos dispositivos de entrada e saída, não como meros periféricos, mas sim como interfaces básicas de comunicação do computador com o mundo exterior a ele.

1. MEMÓRIAS E SEU GERENCIAMENTO

Que as memórias são dispositivos capazes de realizar o armazenamento de informações todos sabemos, porém, é preciso fazer um estudo mais detalhado do seu funcionamento, visando o entendimento da gestão desse armazenamento pelos sistemas operacionais e, para iniciar esse entendimento, faremos uma definição da composição da própria informação em Ambientes Computacionais.

A unidade básica de memória, bem como a unidade básica de qualquer outro sistema computacional é o dígito binário, que chamamos comumente de bit. Um bit é a unidade mais básica possível e tem relação direta com o equipamento eletrônico, representando a presença (1) ou não (0) de energia em determinado componente (TANENBAUM, 2013, p. 57).

Sabendo que a informação é composta de bits, pode-se imaginar que um conjunto de bits é capaz de representar uma determinada informação. Assim, um número inteiro é composto por um conjunto de bits, um caractere é composto por um conjunto de bits e assim por diante. Para facilitar o entendimento das grandezas que serão definidas a partir daqui, é preciso introduzir o conceito de bytes (B), que nada mais é que um conjunto de 8 bits. Esse conjunto de 8 bits recebeu um nome especial porque, a partir de 8 bits, é possível representar, ao menos em inglês, todas as letras, dígitos de 0 a 9 e caracteres especiais como vírgula, ponto, travessão e outros que possibilitam a comunicação.

Além disso, existem alguns prefixos para representar valores maiores que 1024 bits. Esses prefixos sãos mostrados na Tabela 1 e representam agrupamentos da unidade básica bit, assim como o quilo, a tonelada etc. representam agrupamentos de uma unidade básica que seria o grama.

Tabela 01. Prefixos representação binária

PREFIXO	REPRESENTAÇÃO	VALOR	ORDEM
K (quilo)	2^{10}	1024	≈ 1 mil
M (mega)	2^{20}	1.048.576	≈ 1 milhão
G (giga)	2^{30}	1.073.741.824	≈ 1 bilhão
T (tera)	2^{40}	1.099.511.627.776	≈ 1 trilhão

Fonte: elaborada pelo autor.

Na teoria de memórias, a esse conjunto de bits que representam uma determinada informação e que podem ser armazenados daremos o nome de palavra, ou dado. Então, uma palavra nada mais é que um conjunto de bits organizado de forma que se possa armazená-lo para utilização posterior.

Cada palavra de memória (que um tamanho definido pelo próprio hardware do computador) é associado um endereço. Assim, conforme mostra a Tabela 2, é possível armazenar inúmeras palavras em uma memória, cada uma delas com o mesmo tamanho, de forma que, tanto o tamanho da palavra como o número de palavras que podem ser armazenadas são definidos pela arquitetura do computador.

Tabela 02. Representação de armazenamento em memória

ENDEREÇO	PALAVRA (em bits)
00	10001101
01	01010101
02	10101010
03	00001111
04	11110000
05	11001100
...	...

Fonte: elaborada pelo autor.

Dessa forma, a arquitetura define o número de bits que comporá cada um dos endereços e o número de bits que comporá cada uma das palavras (dados) armazenados nessa memória.

Suponha que o número de bits utilizado na representação de cada endereço seja 16 (dezesseis). A partir desse número é possível calcular quantos endereços diferentes existirão, sendo calculado utilizando-se 2^{16} = 65.536 (sessenta e cinco mil, quinhentos e trinta e seis) endereços diferentes.

Sabendo que a memória do exemplo possui 65.536 (sessenta e cinco mil, quinhentos e trinta e seis) endereços diferentes, ao se definir o número de bits de cada uma das palavras da memória, consegue definir qual a capacidade dessa memória. No exemplo, suponha que o número de bits de cada palavra armazenada em cada um desses endereços seja de 8 bits, teremos que a capacidade da memória é dada pela Equação 1.

Figura 01. Equação 1 - Capacidade memória (em bits)

$$Capacidadae = 2^{NumBitsEndereço}.NumBitsPalavra$$

Fonte: elaborada pelo autor.

Ou seja, a capacidade da memória do exemplo é $2^{16}.8 = 524.288 bits$, ou 512Kbits, ou ainda 64KBytes.

O número de bits utilizado na representação de cada endereço e o número de bits utilizado na representação de cada dados, conforme visto, pode ser diferente, sempre limitados ao número de bits da arquitetura que está sendo utilizada. Então, em computadores de 32 bits, o número de bits de cada endereço e o número de bits de cada palavra está limitado a esses 32 bits. Além disso, o menor volume de armazenamento de um endereço é uma palavra. Assim, não teremos o endereço de um bit de memória, mas sim, sempre, o endereço de uma palavra de memória e, para se obter um bit específico, é necessário ler a palavra toda e, por software, fazer a separação do bit desejado.

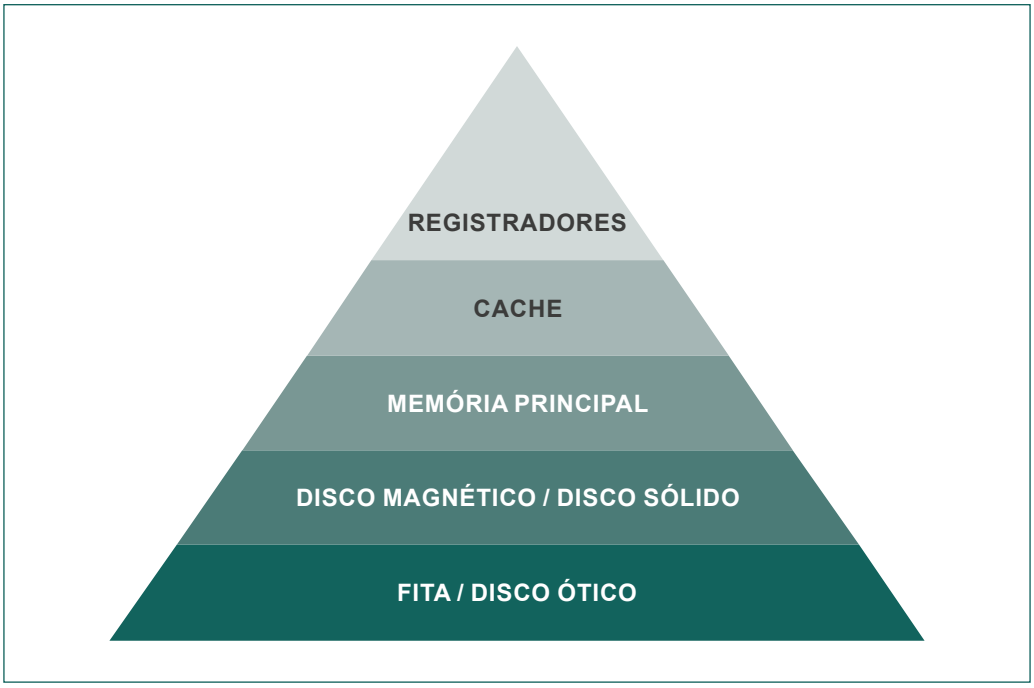
Agora que já se sabe como os dados estão armazenados na memória, pode-se separar as memórias quanto à sua funcionalidade. Existem algumas informações que estão armazenadas na memória e que participam, efetivamente, do processamento de uma determinada tarefa e, por outro lado, existem informações armazenadas na memória que servem apenas como armazenamento, para uso posterior.

Sabendo-se disso, teremos a memória classificada em duas: a memória primária é aquela em cujos dadas armazenados participam efetivamente do processamento (são utilizados pelos processos) e a memória secundária (TANENBAUM, 2013, p. 67) é aquela cujos dados estão armazenados para uso posterior. Embora essa analogia não seja sempre verdadeira, e veremos isso adiante, é possível pensar que a memória primária é a memória RAM (*Random Access Memory*) e a memória secundária o disco, no qual os aplicativos e arquivos estão armazenados.

Sendo assim, para que um aplicativo que está armazenado na memória secundária seja executado, é preciso que seja feito o carregamento desse aplicativo e de seus dados, na memória primária. Somente a partir daí, o processador poderá fazer acesso a esses dados e processá-los.

Isso acontece por dois motivos: a memória primária é construída a partir de uma tecnologia que permite que o acesso à informação a partir dela seja bem mais rápido que a memória secundária, porém, por esse mesmo motivo, ela é mais cara e seria inviável um computador com toda sua capacidade de armazenamento desenvolvida nessa tecnologia; existe uma ligação direta do processador com a memória primária, enquanto que a ligação da memória secundária é feita com a memória primária e não com o processador. *É possível perceber que existe uma hierarquia de memória (TANENBAUM, 2013, p*. 67).

Figura 01. Hierarquia de memória



Fonte: elaborada pelo autor, baseado em Tanenbaum (2013, p. 67).

Observando a Figura 2, quando mais na base da pirâmide uma determinada memória se encontra, maior é sua capacidade de armazenamento e mais barato será o armazenamento, porém, menor será sua velocidade. *Já, subindo para o topo da pirâmide, encontramos memórias com maior velocidade, porém, mais caras e com menor capacidade de armazenamento.*

A memória mais nobre que existe são os registradores, pequenas porções de memória que estão encapsulados juntamente com o processador e são responsáveis diretamente pelo processamento de tarefas. É nos registradores que são armazenador operadores e resultados de operações aritméticas e lógicas, por exemplo.

A memória cache é um tipo especial de memória primária, utilizada apenas para execução de operações que são as que mais se repetem. Com isso, ganha-se na velocidade do processamento e a memória principal é a própria memória RAM do computador, onde todos os dados que serão utilizados pelos processos durante a execução de uma determinada tarefa, devem estar armazenados.

Abaixo da memória principal estão as memórias secundárias, basicamente discos sólidos, como é o caso dos dispositivos SSD (*Solid State Drive*), discos magnéticos como HD (*Hard Disk*), discos óticos como é o caso dos CDs, DVDs e *Blu-rays* e, por fim, as fitas magnéticas, utilizadas basicamente para fazer *backups*.

1.1 GERENCIAMENTO DE MEMÓRIA

Com a multiprogramação, surge a necessidade de se mantar vários processos em execução simultaneamente, o que se torna um desafio no que tange o gerenciamento de memória e de como essa memória é acessada por cada um dos processos. O gerenciamento de memória passa a ser o resultado de duas técnicas distintas:

- Como a memória é vista e como pode ser utilizada pelos processos que estão em execução simultaneamente no sistema;
- Como é o tratamento dado aos processos no que se refere ao uso de memória por eles.

São várias as estratégias utilizadas no gerenciamento de memória visando otimizar o uso desse recurso que, tamanha sua importância no processamento, afeta diretamente o desempenho de todo sistema:

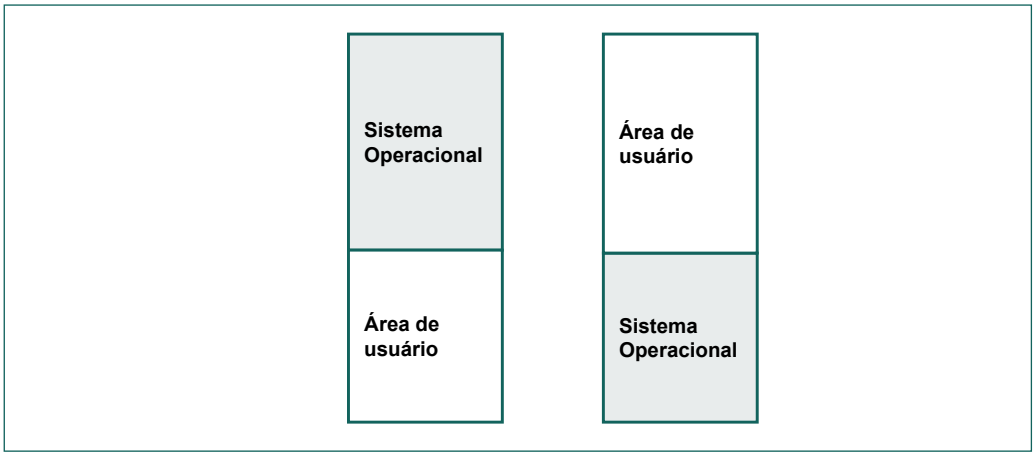
- As estratégias de busca preocupam-se com a busca de quais as próximas instruções precisam estar disponíveis na memória primária para serem as próximas a serem executadas;
- As estratégias de posicionamento que definem em que local da memória primária cada processo será alocado e onde tais processos devem buscar os dados necessários para a execução;
- Estratégias de reposição ou substituição que determinam quais blocos serão enviados à memória secundária para liberar espaço na memória primária para os demais processos em execução.

Visando essas três estratégias, existem modos básicos de organização da memória:

1) Monoprogramado

Nesse tipo de organização, não é permitida a execução de múltiplos processos simultâneos, assim, trata-se de um sistema monotarefa. A memória, conforme mostra o Quadro 1, é dividida em duas partes, uma reservada ao sistema operacional e outra destinada ao processo do usuário.

Quadro 01. Organização de memória em modelo monoprogramado



Fonte: elaborado pelo autor.

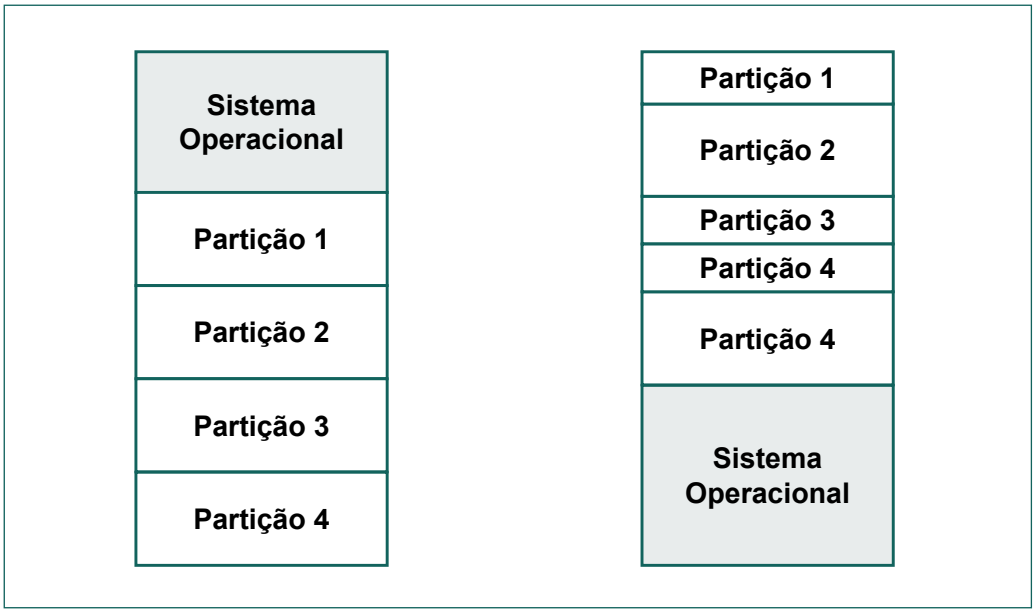
O sistema operacional pode ser alocado na primeira porção de memória ou na última e, ao usuário, só é permitida a execução de um único processo. Dado que o espaço de endereçamento é exatamente àquele correspondente à memória primária, o processo em execução pode utilizar toda memória disponível, exceto àquela utilizada pelo Sistema Operacional.

A vantagem desse tipo de organização é sua simplicidade e facilidade de implementação, porém, como já dito, impossibilita a multiprogramação, o que o torna inviável para sistemas atuais multitarefa.

2) Multiprogramado com particionamento fixo

Como a necessidade de implementação dos sistemas multiprogramados, surge a necessidade de se organizar a memória de maneira diferente. Uma dessas formas é o sistema de particionamento fixo, mostrado no Quadro 2, em que uma das partições continua sendo ocupada pelo Sistema Operacional, porém, a área restante *é dividida em partes* fixas que podem ser iguais ou diferentes.

Quadro 02. Organização de memória em modelo multiprogramado com particionamento fixo



Fonte: elaborado pelo autor.

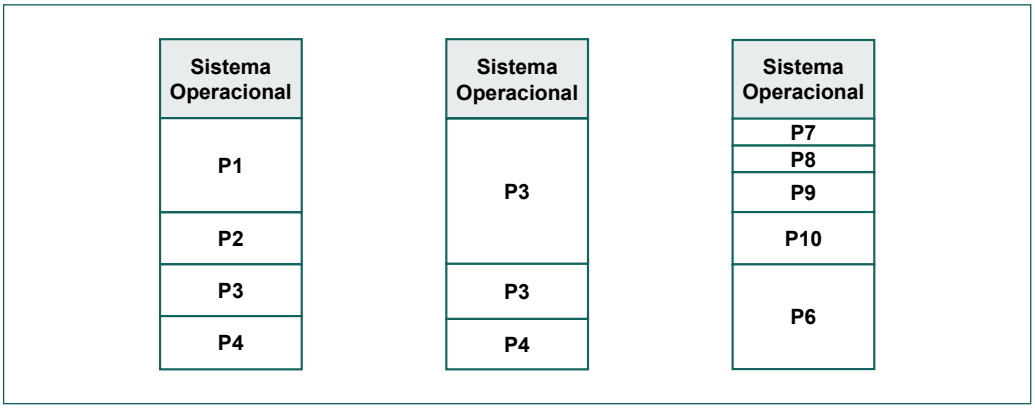
O tamanho de cada partição vai depender da quantidade de memória e do tamanho dos processos que serão executados a partir desse sistema. Como essa maneira de organização permite que sejam alocados mais de um processo, é preciso que seja definida uma estratégia de alocação. São três alocações possíveis (TANENBAUM, 2016, p. 128):

- ***First Fit*:** aloca-se o processo na primeira partição livre que comporte o tamanho do processo, o que minimiza o trabalho com a pesquisa da partição;
- ***Best Fit*:** aloca-se o processo na partição na qual ele melhor se encaixe, minimizando o espaço livre;
- ***Worst Fit*:** aloca-se o processo na partição em que sobre mais espaço. Nesse caso existe um desperdício maior de memória, porém, essa técnica será importante para tipos de organização com partições variáveis.

3) Multiprogramado com particionamento variável

Essa forma de organização é bastante parecida com a forma de particionamento fixo, exceto pelo fato que as partições agora podem mudar de tamanho com o passar o tempo, como mostrado no Quadro 3, o que permite que os processos sejam mais bem alocados, evitando desperdício de memória.

Quadro 03. Organização de memória em modelo multiprogramado com particionamento variável

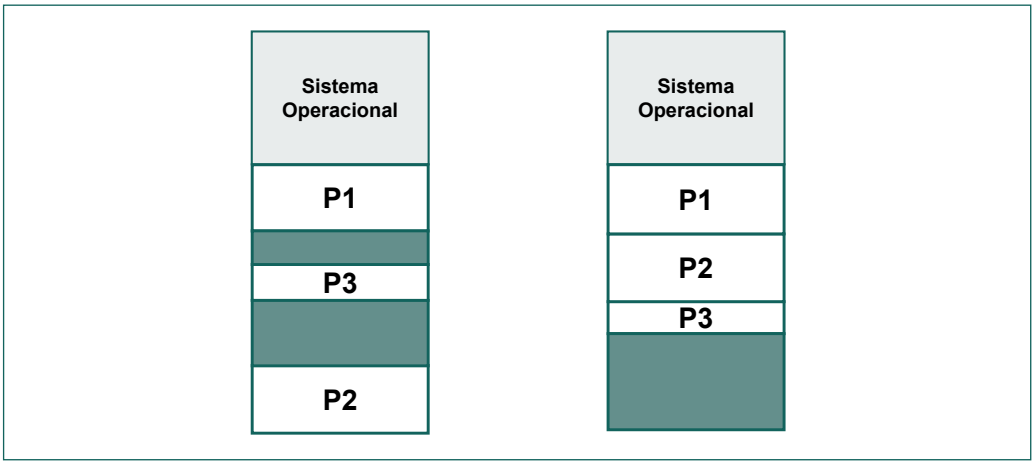


Fonte: elaborado pelo autor.

Como o espaço que sobra depois da alocação de um processo em uma determinada partição pode ser reaproveitado para a criação de uma nova partição, a estratégia de alocação *Worst Fit* passa a ser interessante, pois ela permite que o espaço que sobre depois de uma alocação continue sendo grande o suficiente para a alocação de um outro processo ou mesmo para a composição de outro espaço de alocação.

Porém, mesmo utilizando essa estratégia, ainda assim pode haver desfragmentação de memória, ou seja, podem sobrar espaços entre as alocações que processos que não podem ser aproveitados para alocação de outros processos e, para resolver esse problema, existe a possibilidade de se recorrer a uma técnica de desfragmentação, ou compactação de memória, capaz de realocar os processos em partições próximas, deixando que os espaços livres fiquem acumulados em uma posição específica da memória, permitindo sua melhor utilização e menor desperdício, conforme mostra o Quadro 4:

Quadro 04. Organização de memória em modelo multiprogramado com particionamento fixo



Fonte: elaborado pelo autor.

Com a desfragmentação, as duas partições livres que se encontravam entre os demais processos agora foram agrupadas em uma única região livre, que pode ser mais bem aproveitada pelos próximos processos que serão alocados.

Os processos podem ser alocados na memória primária, uma vez que eles precisam ser processados. Porém, a memória primária nem sempre será suficiente para a execução de todos os processos, uma vez que existem processos de tamanho bem grande e que necessitam de uma quantidade de memória, muitas vezes, maior do que a quantidade de memória primária disponível. Para isso, existe um mecanismo conhecido como memória virtual. Mas, antes de falar de memória virtual, é preciso entender que os computadores trabalham com uma abstração da memória física, ou espaço de endereçamento.

Um espaço de endereçamento é o conjunto de endereços de memória que um processo pode utilizar (TANENBAUM, 2016, p. 128). Assim, é possível trabalhar com o conceito de endereçamento lógico, ou seja, o Sistema Operacional, através de algum procedimento, gera um conjunto de endereços que são apresentados a cada processo como endereços de memória válidos e esses endereços, ao serem utilizados, são convertidos em endereços válidos pelo processo do Sistema Operacional.

2. MEMÓRIA VIRTUAL

Uma das formas de se aumentar a quantidade de memória primária disponível é através da utilização de memória virtual, que consiste em um mecanismo do Sistema Operacional de apresentar aos processos parte da memória secundária como memória primária, aumentando, assim, o espaço de endereçamento.

Segundo Berenger e Paulo (2011, p. 72):

> Memória virtual é uma técnica sofisticada e poderosa de gerência de memória em que as memórias principal e secundária são combinadas, dando ao usuário a ilusão de existir uma memória muito maior que a capacidade real da memória principal. O conceito de memória virtual fundamenta-se em não vincular o endereçamento feito pelo programa dos endereços físicos da memória principal. Dessa forma, programas e suas estruturas de dados deixam de estar limitados ao tamanho da memória física disponível, pois podem possuir endereços associados à memória secundária.

Já segundo Silberschatz e Galvin (2015, p. 214):

> A memória virtual é uma técnica que permite a execução de processos que não estão totalmente na memória. Uma grande vantagem desse esquema é que os programas podem ser maiores do que a memória física. Além disso, a memória virtual abstrai a memória principal em um array de armazenamento uniforme extremamente grande, separando a memória lógica, conforme vista pelo usuário, da memória física.

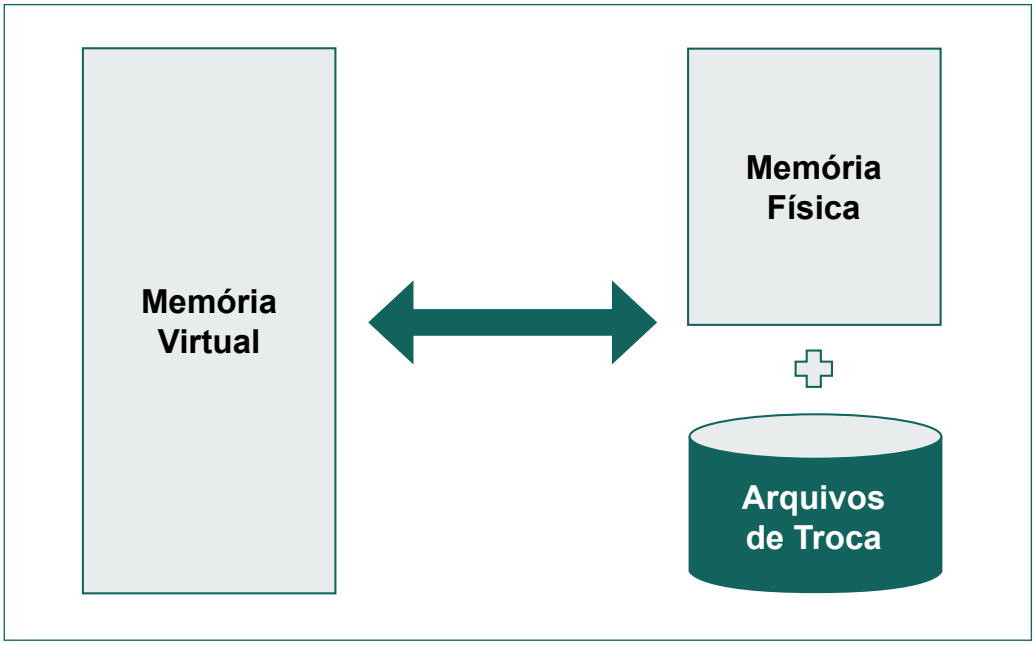
De modo geral, portanto, a memória virtual significa a possibilidade de oferecer aos processos em execução uma quantidade de memória maior do que aquela que está efetivamente e fisicamente instalada no computador.

Dentre os benefícios da utilização de memória virtual estão:

- A percepção de que a quantidade de memória disponível aos programadores é bem maior do que a memória física;
- A abstração de que a memória funcione como um grande vetor de posições, cada uma com um endereço e que pode ser utilizado na sequência, ou seja, a ideia de que o espaço de endereçamento é maior do que, efetivamente é;
- Maior eficiência, uma vez que com uma quantidade maior de memória disponível, é possível executar um número maior de processos, otimizando o uso dos recursos computacionais como um todo.

Com isso, é necessário mudar a ideia de que um computador só usa memória virtual quando ele não possui mais memória física disponível ou que o uso de memória virtual, uma vez que a memória secundária é mais lenta que a primária, prejudica o desempenho do computador. É necessário entender que a memória virtual é utilizada sempre que possível em sistemas computacionais e que isso gera eficiência.

Figura 02. Composição da memória virtual



Fonte: elaborada pelo autor.

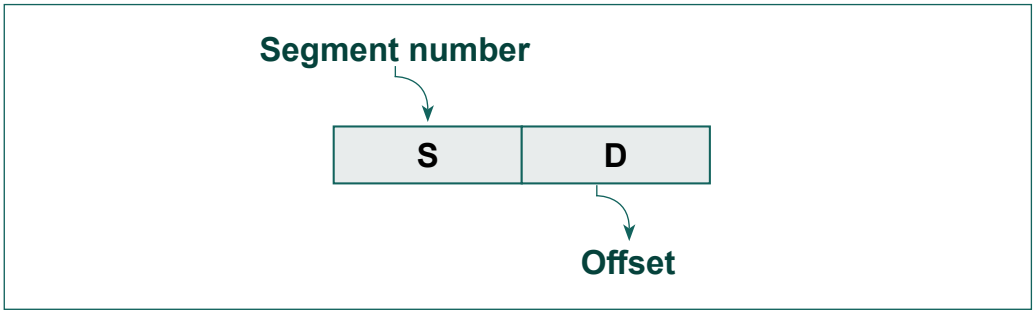
Conforme mostrado na Figura 1, a memória virtual é formada pela memória principal somada à parte da memória secundária. Chamaremos essa parte da memória secundária utilizada como memória primária de arquivos de troca, arquivos de *swap* ou simplesmente *swap.*

Existem várias técnicas para a implementação de memória virtual. As principais são a segmentação e a paginação ou então a combinação entre elas, sendo uma segmentação paginada ou uma paginação segmentada.

2.1 SEGMENTAÇÃO

Como o próprio nome sugere, a técnica de segmentação divide o espaço de endereçamento total (memória primária + arquivos de troca) em segmentos, cada um como uma finalidade. É comum haver segmentos para armazenamento do SO, segmento para armazenamento de código (programa) e segmento para armazenamento de dados separados. Dessa forma, a segmentação se apresenta como um espaço bidimensional, em que se define o endereço do segmento e depois o endereço da informação que se deseja buscar no segmento, conforme mostra a Figura 4.

Figura 03. Formação do endereço em segmentação



Fonte: elaborada pelo autor.

Fazendo uma analogia, o endereço de um segmento funciona mais ou menos como funciona o endereço de um prédio de apartamentos. Primeiramente você precisa encontrar o prédio, através de um endereço base, que no caso da segmentação seria o endereço do segmento, depois, você encontra, dentro desse prédio, qual apartamento você quer, o número do apartamento seria o *Offset* do endereço na segmentação.

Segundo Silberschatz e Galvin (2015, p. 186):

> Ao escrever um programa, um programador o vê como um programa principal com um conjunto de métodos, procedimentos ou funções. Ele também pode incluir várias estruturas de dados: objetos, arrays, pilhas, variáveis, e assim por diante. Cada um desses módulos ou elementos de dados é referenciado pelo nome. O programador fala sobre “a pilha”, “a biblioteca de matemática” e “o programa principal” sem se preocupar com os endereços que esses elementos ocupam na memória (...) A segmentação é um esquema de gerenciamento de memória que dá suporte à visão da memória desse programador. Um espaço de endereçamento lógico é um conjunto de segmentos. Cada segmento tem um nome e um tamanho.

Aqui cabe um comentário importante, a segmentação não é uma técnica de gerenciamento de memória transparente ao programador. Ele precisa conhecer a estrutura para poder endereçar corretamente cada informação que o processo utilizará.

Como cada segmento é independente, isso favorece que se possa criar segmentos que estejam na memória primária ou segmentos que estejam na memória secundária, aumentando assim, a capacidade de memória e criando vários espaços de endereçamento independentes.

Embora a segmentação seja muito útil para aumentar a eficiência do uso da memória e seja um mecanismo interessante de organização da memória, uma vez que cada tipo de informação fica fisicamente separada em segmentos criados para esse fim, ela apresenta alguns problemas.

Quando um segmento não se encontra na memória primária, deve haver a troca de um segmento que esteja na memória primária com o segmento que se deseja utilizar (lembrando que o processador só tem acesso direto à memória primária). Então, toda vez que um segmento que não esteja na memória primária for requisitado acontece uma operação de troca (*swap)*. É escolhido um segmento para sair da memória primária (geralmente o mais antigo ou aquele que está sendo menos utilizado) e o segmento que está na memória secundária é alocado em seu lugar.

O problema dessa troca, na segmentação, é ocasionado devido à natureza dos segmentos. Suponha que um segmento foi criado para armazenamento de código (armazenar os programas). Quanto mais processos são executados, maior fica esse segmento. Fazer a troca de um segmento muito grande pode ocasionar sobrecarga de processamento e lentidão no sistema.

Uma forma de minimizar esse problema é usar outra técnica de memória virtual chamada paginação.

2.2 PAGINAÇÃO

Diferentemente da segmentação, que cria um espaço bidimensional, gerando vários espaços de endereçamento separados os quais o programador precisa conhecer, na paginação o espaço de endereçamento total (memória primária mais arquivos de troca) é dividido em partes iguais, de tamanho fixo, geralmente pequenas, chamadas páginas.

Segundo Berenger e Paulo (2011, p. 77):

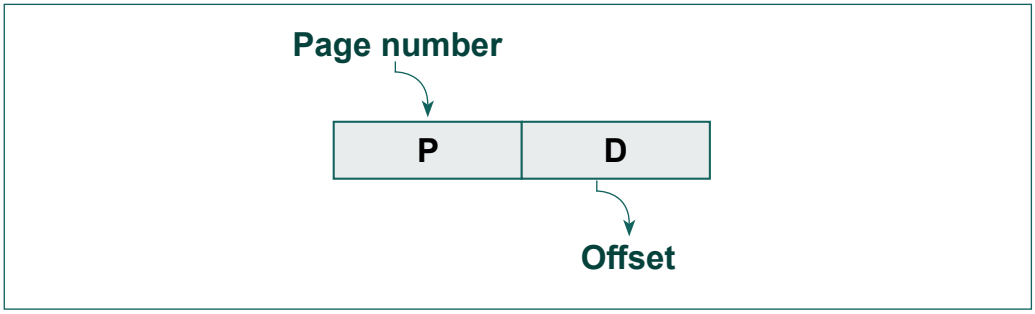
> A memória virtual por paginação é a técnica de gerência de memória em que o espaço de endereçamento virtual e o espaço de endereçamento real são divididos em blocos de mesmo tamanho chamados páginas. As páginas no espaço virtual são denominadas páginas virtuais, enquanto as páginas no espaço real são chamadas de páginas reais ou frames.

Da mesma forma que as páginas são criadas, a memória física também é dividida em espaços iguais, com o mesmo tamanho das páginas, que recebem o nome de *frames*.

Então, como todo espaço de endereçamento é dividido em páginas, algumas dessas páginas estão alocadas nos *frames*, ou seja, estão alocadas na memória primária e as demais encontram-se fora dos *frames*, ou seja, encontram-se na memória secundária.

O endereço de cada informação segue um modelo parecido com o modelo utilizado na segmentação, conforme mostrado na Figura 5.

Figura 04. Formação do endereço em paginação

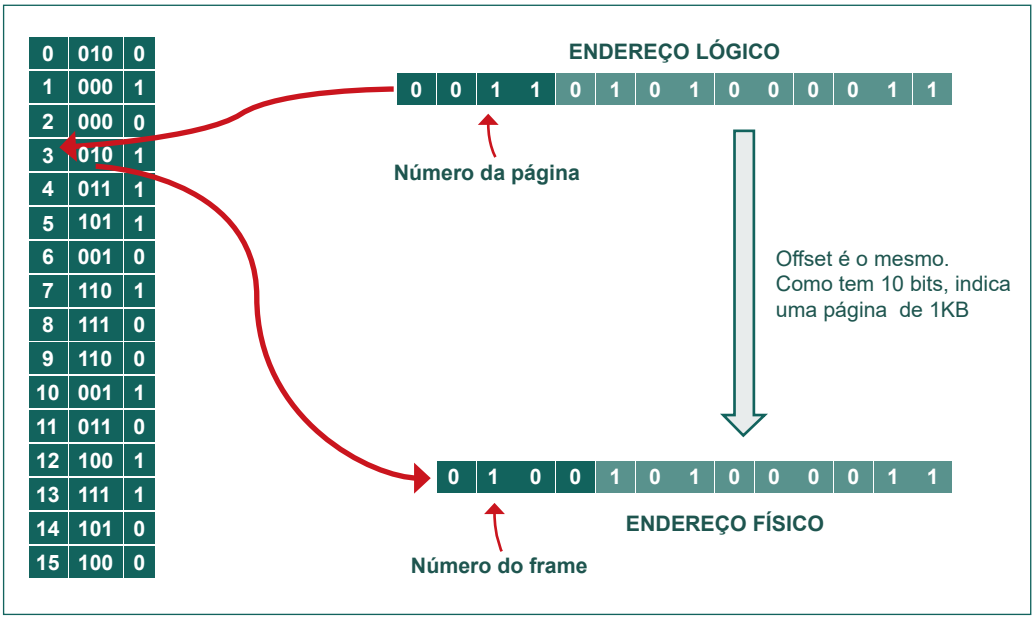


Fonte: elaborada pelo autor.

De forma análoga ao que acontece na segmentação, o endereço é composto pelo número da página seguido do deslocamento (*offset)* a partir da página. Porém, por se tratar de um espaço de endereçamento único, o programador não precisa conhecer como a paginação ocorre. Para o programador, o espaço de endereçamento é único e o endereço virtual é tratado como um endereço físico, ficando todo processo transparente.

O Sistema Operacional faz isso utilizando uma tabela de páginas, que indica em qual *frame* cada página está mapeada e se ela se encontra atualmente no *frame* ou não. A Figura 6 mostra esse processo.

Figura 05. Endereço lógico e físico na paginação



Fonte: elaborada pelo autor.

O computador gera um endereço lógico, composto pelo número de página mais o *offset*, o sistema recupera o número da página e verifica na tabela de páginas a qual *frame* essa página está mapeada e se ela está ou não presente no *frame*. Se a página estiver presente no *frame* (valor 1 na última coluna da tabela de páginas), o sistema recupera o número do *frame* em que a página está e monta, juntamente com o *offset*, o endereço físico.

Se a página não estiver presente no *frame* (valor 0 na última coluna da tabela de páginas), ocorre a falta de página (*page fault*) e o sistema realiza a troca (*swap*).

Como na paginação, diferente do que acontece na segmentação, as páginas têm tamanho fixo e pequeno, a troca de páginas não é um processo muito custoso, o que favorece o desempenho do sistema como um todo.

2.3 PAGINAÇÃO E SEGMENTAÇÃO COMBINADAS

Comparando a paginação com a segmentação, obtemos que, quanto ao programador precisar conhecer o processo, a paginação é melhor, já que oferece um espaço de endereçamento único e transparente enquanto que na segmentação, o espaço dos segmentos é bem definido, com a necessidade do programador indicar qual segmento ele pretende acessar.

Por outro lado, dependendo do que se deseja, essa pode ser uma vantagem da segmentação, que oferece múltiplos espaços de endereçamento, favorecendo a organização e a separação, por exemplo, dos dados e dos procedimentos (programas).

Na segmentação, como os dados são encapsulados em segmentos bem separados, isso facilita a segurança e o compartilhamento dos procedimentos e dos dados entre os diversos usuários do sistema.

Mas ambas as técnicas permitem que o espaço de endereçamento virtual exceda o espaço de endereçamento físico, proporcionando uma quantidade maior de processos em execução e otimizando o uso dos recursos e a produtividade do sistema.

Com isso, vemos que existem vantagens e desvantagens nas duas técnicas e, embora a paginação ainda seja a técnica mais utilizada, é possível combiná-las em técnicas de segmentação paginada ou mesmo de paginação segmentada, conseguindo-se maximizar as vantagens.

2.4 ALGORITMOS DE TROCA DE PÁGINAS

Qualquer que seja a técnica de memória virtual implementada, vimos que ela se baseia na necessidade de trocas de dados entre a memória primária e secundária. Essas trocas são comumente chamadas de *swap* e necessitam fazer a transferência da página ou segmento que está sendo requisitado e está armazenado na memória secundária para a memória primária.

Para que essa quantidade de dados seja alocada na memória primária, é preciso definir qual página ou segmento, deverá ser substituída. Essa escolha de qual página ou segmento presente na memória primária será substituída é realizada pelo algoritmo de troca. Existem várias técnicas para essa escolha e vamos discutir alguns deles:

- **Algoritmo de troca aleatória:** é um algoritmo de baixa sobrecarga de execução, uma vez que é bem simples de ser implementado. Consiste em sortear, aleatoriamente, dentre as páginas presentes na memória primária, a que será substituída. Embora simples, é pouco usado porque a página removida pode ser a próxima a ser requisitada, causando sobrecarga desnecessária;
- **Algoritmo de troca FIFO:** é um algoritmo também simples de implementar. Gera uma fila de páginas, por ordem de idade. Assim, as páginas que estão a mais tempo na memória primária são as primeiras a serem escolhidas para serem substituídas. Embora lógico, isso nem sempre é verdade, uma vez que existem processos longos que podem continuar a utilizar páginas, mesmo sendo antigas.
- **Algoritmo de troca segunda chance:** é uma variante do algoritmo FIFO. Nele, o procedimento inicial segue o mesmo do FIFO (páginas mais velhas são as primeiras a serem substituídas), porém, nesse algoritmo cria-se um valor de referência que indica se a página foi utilizada recentemente e, se esse valor indicar que a página está em uso, ela recebe uma segunda chance de permanecer na memória primária;
- **Algoritmo LRU (*Least Recently Used*):** esse algoritmo cria um relógio em cada página, indicando quanto tempo faz que a página não é utilizada. Com isso, escolhe-se para a substituição aquela página que está há mais tempo sem ser utilizada. Porém, ele impõe uma sobrecarga grande no processamento e é pouco utilizado.

3. DISPOSITIVOS DE ENTRADA E SAÍDA E SEU GERENCIAMENTO

Quando falamos de dispositivos de entrada e saída, ou dispositivos de I/O, do acrônimo inglês *Input/Output*, muitas vezes nos referimos aos periféricos como uma impressora, um mouse, um monitor de vídeo ou um teclado, por entender que são esses os dispositivos que fazem a interface do usuário com o computador e que são eles os responsáveis por enviar dados externos ao computador (entrada de dados) ou receber dados do computador e reproduzi-los de forma que chegue ao usuário (saída de dados).

Segundo Tanenbaum (2016, p. 233):

> Diferentes pessoas veem o hardware de E/S de maneiras diferentes. Engenheiros elétricos o veem em termos de chips, cabos, motores, suprimento de energia e todos os outros componentes físicos que compreendem o hardware. Programadores olham para a interface apresentada ao software — os comandos que o hardware aceita, as funções que ele realiza e os erros que podem ser reportados de volta.

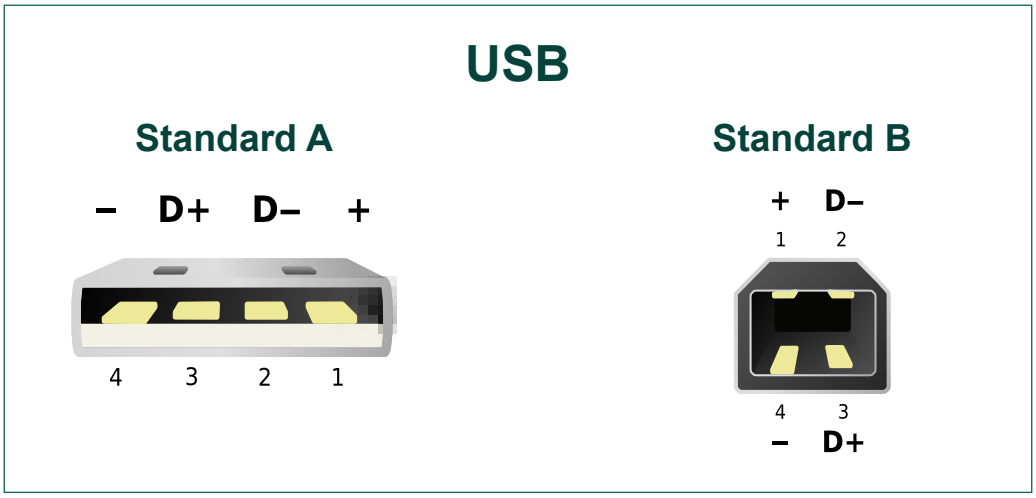
Particularmente, nos interessa essa forma de interface dos dispositivos de entrada e saída. Então, não nos preocuparemos em qual o dispositivo em si, se uma impressora ou um mouse. O que nos interessa é estudar sua interface e a forma como se comunica com o computador e como o computador se comunica com ela. Assim, se ambos forem USB (*Universal Serial Bus*), por exemplo, utilizam a mesma interface e, dessa maneira, serão tratados da mesma forma.

Não existem, portanto, um número muito grande de dispositivos de entrada e saída se pensarmos nas interfaces possíveis. No fundo, poderemos sempre agrupá-los pelo tipo de interface que utilizam. Vamos citar alguns tipos de interface e onde são comumente utilizadas:

3

USB (*Universal Serial Bus*): como o próprio nome diz, é uma interface que foi criada para ser universal, e servir para comunicação de diversos tipos de dispositivos. Isso é quase verdade. Nem toda interface foi substituída pela USB porque existem algumas particularidades, sobretudo nos discos, nas placas internas de um computador e nos dispositivos de vídeo que fazem com que outras interfaces tenham desempenho melhor. Mas, atualmente, é comum encontrar a interface USB presente em impressoras, mouses, discos internos, caixas de som, celulares entre outros. A Figura 7 mostra uma interface USB.

Figura 06. Interface USB



Fonte: https://commons.wikimedia.org/wiki/File:USB.svg. Acesso em: 8 mar. 2022.

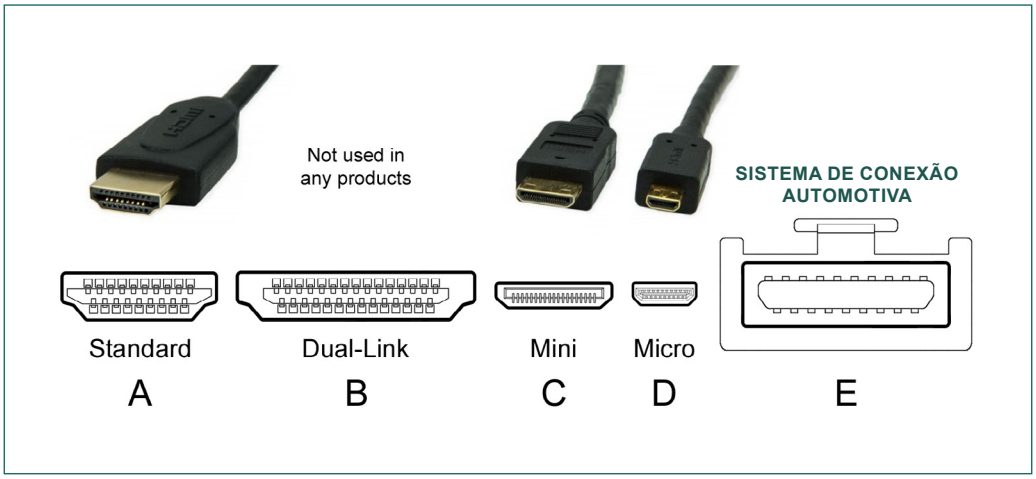
HDMI
(*High-Definition Multimedia Interface*):

Interface que se destaca especialmente para comunicação de dispositivos de vídeo. Isso se deve ao fato dessa interface conseguir transferir grandes quantidades de dados em um espaço curto de tempo. A Figura 7 mostra um exemplo de interface desse tipo.

SATA
(*Serial Advanced Technology Attachment*):

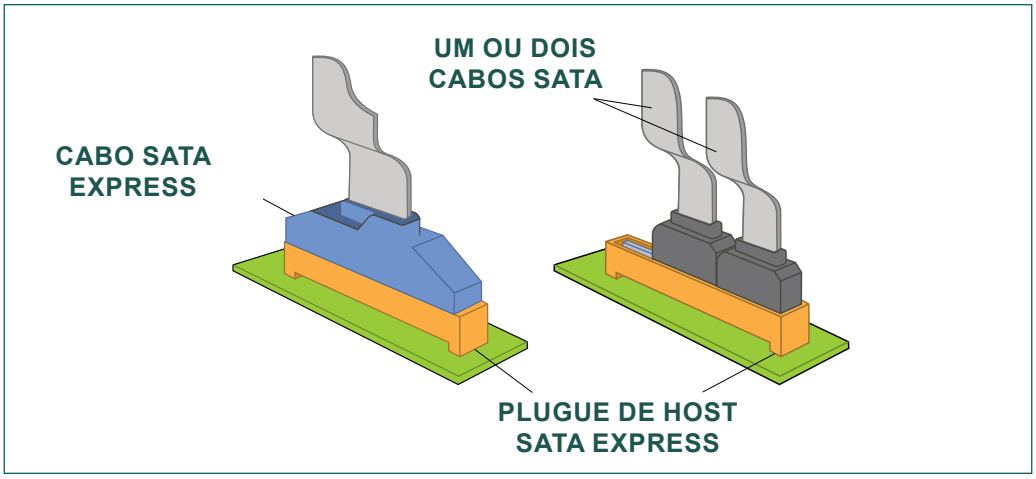
É bastante conhecido por ser a interface padrão dos discos internos do computador. Permite uma transferência de dados também eficiente e é o substituto da interface ATA, que funcionava de maneira paralela.

Figura 07. Interface HDMI



Fonte: https://commons.wikimedia.org/wiki/File:HDMI_Connector_Types.png. Acesso em: 8 mar. 2022.

Figura 08. Interface SATA



Fonte: https://commons.wikimedia.org/wiki/File:SATA_Express_motherboard_connection.svg. Acesso em: 8 mar. 2022.

Existem outras tantas interfaces que ainda são importantes e coexistem em um sistema computacional, como é o caso das placas PCI (*Peripheral Component Interconnect*) e tantas outras. Qualquer que seja a interface, nosso intuito, aqui é estudar o funcionamento básico delas, ou seja, tudo o que falaremos a partir daqui pode ser empregado em qualquer tipo de interface ou pode ser a característica de qualquer tipo de interface.

O primeiro ponto importante é saber que uma interface pode se comunicar de maneira serial ou paralela. Quando a comunicação é paralela, isso significa a existência de um barramento (cabo) com vários fios, um para cada bit a ser transferido e mais um conjunto de fios de controle. Se a interface paralela transmite 16 bits de cada vez,

deve haver um fio para cada um dos 16 bits. Já, na interface serial, os dados são transferidos por um único fio e os demais fios são fios de controle. Toda informação, portanto, é transferida pelo mesmo fio, em tempos diferentes, como se fosse uma fila de bits. Embora possa parecer meio óbvio que a comunicação paralela seja mais rápida, já que consegue transmitir vários bits simultaneamente, isso nem sempre é verdade. Isso porque vai depender da velocidade com que os bits são transferidos. Pensem em um caminhão, que consegue levar 3 carros no seu interior. Se a velocidade do caminhão for de 60km/h, por exemplo, e cada carro puder trafegar um atrás do outro a 100km/h é óbvio que os carros chegarão mais rápido se forem um atrás do outro e não dentro do caminhão. O mesmo acontece com as interfaces seriais e paralelas.

Além disso, quanto mais fios em barramentos dentro do computador, maior precisa ser o computador para que não haja aquecimento excessivo. Assim, prioriza-se, atualmente, por interfaces seriais em detrimento de interfaces paralelas, que ocupam muito espaço, podem ser mais lentas e geram muito calor.

As interfaces de entrada e saída possuem uma controladora (um hardware) que faz o controle da comunicação dos periféricos com o computador. E essas controladoras possuem um software de controle chamado *driver*. Os *drivers* fazem a interface do Sistema Operacional com a interface que o Sistema terá que interagir. Por exemplo, para que se possa utilizar a interface USB em um computador, é preciso que esse computador possua o hardware (a controladora USB) instalada e a essa controladora estejam associadas portas (interfaces de comunicação onde os periféricos serão acoplados) e o *driver* USB instalado, ou seja, é preciso que o Sistema Operacional possua o software de comunicação, que irá traduzir toda comunicação USB para uma linguagem que o Sistema Operacional conheça.

A forma como se dá a comunicação dessas controladoras com o computador podem variar, existindo duas formas básicas:

---

1) Entrada e saída programada: na entrada e saída programada, o responsável pela operação de verificação das unidades de entrada e saída é o processamento central. Assim, de tempos em tempos o sistema operacional desvia o processamento para verificar se alguma unidade de entrada e saída possui dados para ser transmitido em um processo conhecido como *handshake* (aperto de mão). Esse processo desperdiça tempo de processamento, já que muitas vezes, a unidade de entrada e saída não deseja transmitir dados.

Além disso, se a unidade de entrada e saída possui informação a ser transmitida, o processador deve aguardar até o término do processo para fazer o tratamento do dado recebido/enviado, causando mais ociosidade. Em resumo, uma operação desse tipo pode ser descrita como mostra o Código 1:

Código 1 - Operação de Entrada e Saída Programada

```
Processo chama I/O

Controladora envia comando para dispositivo de I/O

Ocorrem as operações de I/O

Dispositivo de I/O retorna status

Transferência de dados

Retorno ao processo
```

Fonte: elaborado pelo autor.

Como pode ser observado no Código 1, além da necessidade que o processador tem de fazer a verificação das unidades de entrada e saída, quando um processo inicia uma operação de entrada e saída, dos passos 2 ao 5 o processo (processador central) não faz parte da operação, que é realizada pela controladora do dispositivo. Porém, mesmo assim, o processador fica ocioso na espera do término da operação.

---

2) Entrada e saída por interrupção: na entrada e saída por interrupção, o processador não necessita esperar o término da operação de leitura/escrita no dispositivo, diminuindo a ociosidade. Nesse tipo de operação, o processo sinaliza a necessidade de utilização de um dispositivo de entrada e saída, o processador envia um comando ao dispositivo sinalizando a necessidade de operação. A partir daí, a controladora do dispositivo de entrada e saída realizada a operação de modo independente, mantendo o processador livre para realizar outras operações, isto é, para tratar outros processos que necessitem desse recurso.

Quando a controladora de entrada e saída termina o processo, interrompe o processador, que retorna ao processo original para fazer o tratamento dos resultados da operação. O Código 2 mostra como se dá esse processo:

Código 2 - Operação de Entrada e Saída Programada

```
Processo chama I/O, que realiza o processo de forma independente

Processador executa outra tarefa

Ao finalizar, controladora de I/O sinaliza uma interrupção ao
processador

Ao terminar a operação atual, processador passa a tratar o
processo que originou a interrupção
```

```
Processador salva o contexto atual

Processador trata o chamado da controladora de I/O

A rotina de tratamento da interrupção é executada

Finalizada a rotina, o processador recupera o contexto em que
estava antes da interrupção

Processador retorna ao processamento de tarefas
```

Fonte: elaborado pelo autor.

Como pode ser visto no Código 2, o processador não fica ocioso, fazendo o tratamento de outras tarefas enquanto a controladora do dispositivo de entrada e saída realiza toda a operação de leitura ou escrita dos dispositivos de entrada e saída de forma independente.

Ao finalizar a operação, a controladora sinaliza ao processador o término da operação através de uma interrupção e o processador, então, guarda o contexto da atividade que está realizando no momento para fazer o tratamento da interrupção, recuperando o contexto e voltando ao tratamento da tarefa ao término.

Com o uso de interrupção, o processo de transferência de dados entre computador e dispositivos de entrada e saída ficou muito mais eficiente, liberando o processador para realizar outras tarefas enquanto os dispositivos enviam e recebem dados.

Porém, a memória ainda era controlada exclusivamente pelo processador, então, embora a controladora do dispositivo de entrada e saída conseguisse ler e escrever dados de forma independente no dispositivo, todo dado recebido dependia do processador para ser armazenado na memória. A fim de resolver esse problema, foi criada a possibilidade dos dispositivos de entrada e saída fazerem acesso direto à memória, tornando-as ainda mais independentes na leitura e escrita de dados.

---

3) Entrada e saída com DMA (*Direct Access Memory*): Para Tanenbaum (2016, p. 238):

> A CPU pode requisitar dados de um controlador de E/S um byte de cada vez, mas fazê-lo desperdiça o tempo da CPU, de maneira que um esquema diferente, chamado de acesso direto à memória (Direct Memory Access — DMA) é usado muitas vezes.

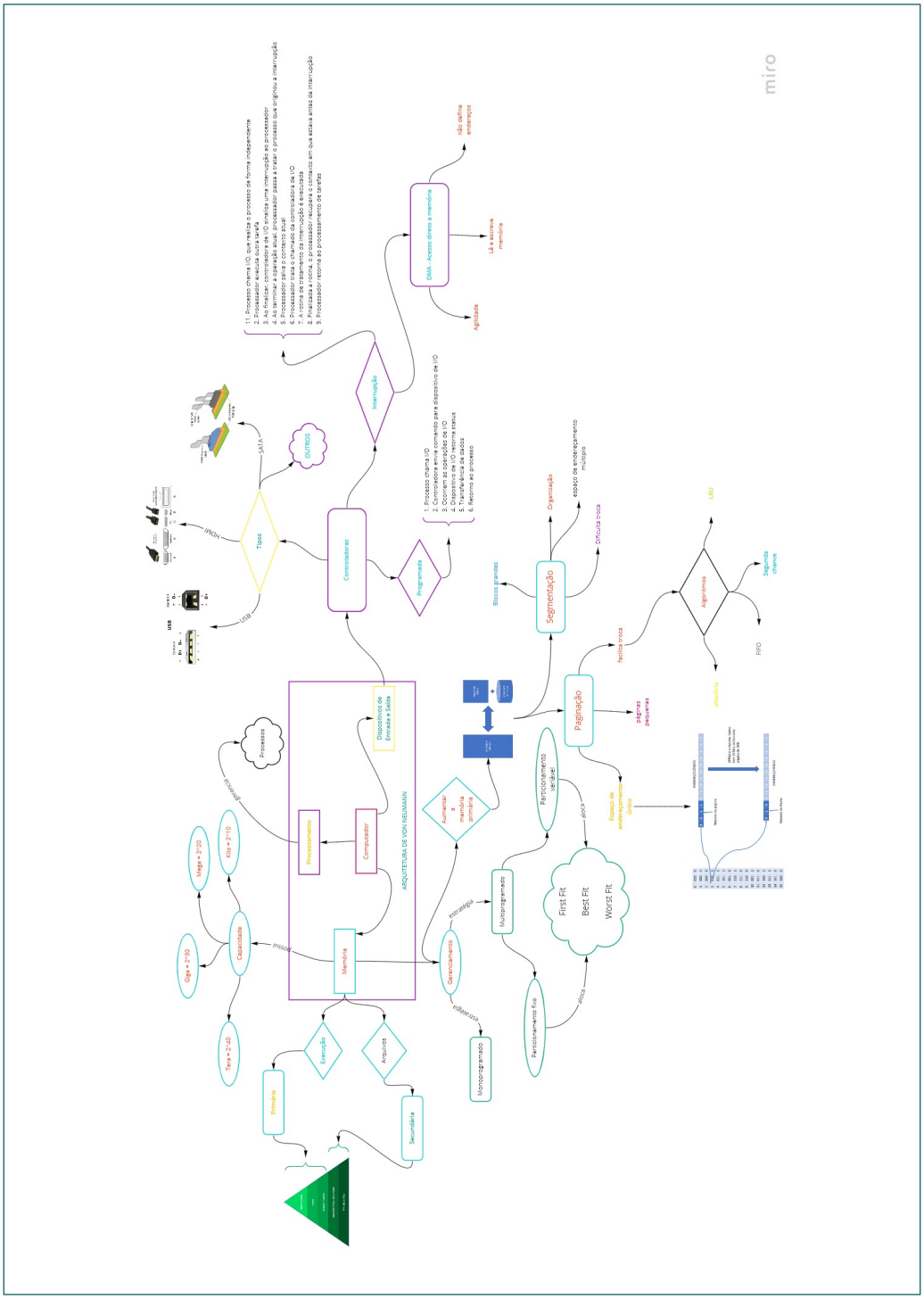
Como explica a definição acima, o processo de ler e escrever dados dos dispositivos de entrada e saída diretamente na memória, é uma atividade custosa, do ponto de vista do uso do processador. Assim, o que a entrada e saída com DMA faz é tornar as controladoras independentes nesse processo.

Para isso, toda vez que uma controladora sinaliza a necessidade de utilização da memória, seja ela de leitura ou escrita, o processador gera uma definição de quais endereços devem ser lidos ou quais endereços podem ser escritos pela controladora e, a partir daí, todo processo de leitura e escrita passa a ser realizado pela própria controladora, tornando o processador independente para realização de outras tarefas.

Como exemplo, imagine um dispositivo de rede que receberá o resultado de um *download* e deverá armazená-lo na memória. O processador gera os endereços de memória que podem receber esse arquivo e sinaliza isso à controladora da placa de rede, que a partir daí, recebe o arquivo pela rede e armazena-o na memória de forma independente do processamento central, o que diminui a ociosidade e torna o sistema como um todo muito mais eficiente.

Como pode-se observar, cada vez mais as controladoras de entrada e saída são verdadeiros computadores de uso específico, com processamento e até memória independentes. Isso faz com que um computador atual seja, na verdade, um sistema de computadores trabalhando em conjunto, maximizando a produtividade e aumentando a eficiência.

OBJETIVO DE APRENDIZAGEM



CONCLUSÃO

Desde o advento da arquitetura de Von Neumann na década de 30 que se sabe que as memórias são dispositivos primordiais no processo de execução de tarefas em meio computacional e elas continuam fundamentais até os dias de hoje.

Toda evolução pela qual esses dispositivos passaram, tanto de hardware, tornando-as mais rápidas e menores, até de software, tornando seu gerenciamento mais eficiente, visam diminuir a diferença de velocidade que existe entre a capacidade de processamento de um microprocessador e a velocidade de armazenamento ou leitura dos dados produzidos por esse processamento na memória.

As memórias continuam sendo um gargalo em se tratando de velocidade, já que sua velocidade ainda fica aquém da velocidade do processamento, porém, muito já se evolui nesse sentido, com uso de memória virtual, por exemplo, que aumenta a produtividade, possibilitando que um número maior de processos seja atendido.

Além disso, também desde Von Neumann, os dispositivos de interface com o usuário veem se aprimorando, disponibilizando os recursos do computador ao usuário de maneira cada vez mais eficiente.

Os dispositivos de entrada e saída, através de suas controladoras e drivers são verdadeiros computadores atualmente, tornando-se cada vez mais eficientes para o propósito pelos quais foram desenvolvidos, o que aumenta cada vez mais a capacidade de processamento das máquinas atuais.