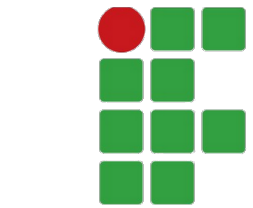
Sistemas Operacionais

Gerenciamento de Memória

Baseado no livro *Operating System Concepts* [Silberschatz et al., 2012]

Felipe Augusto Lima Reis

felipe.reis@ifmg.edu.br



INSTITUTO FEDERAL
Minas Gerais

Sumário

Introdução

Introdução

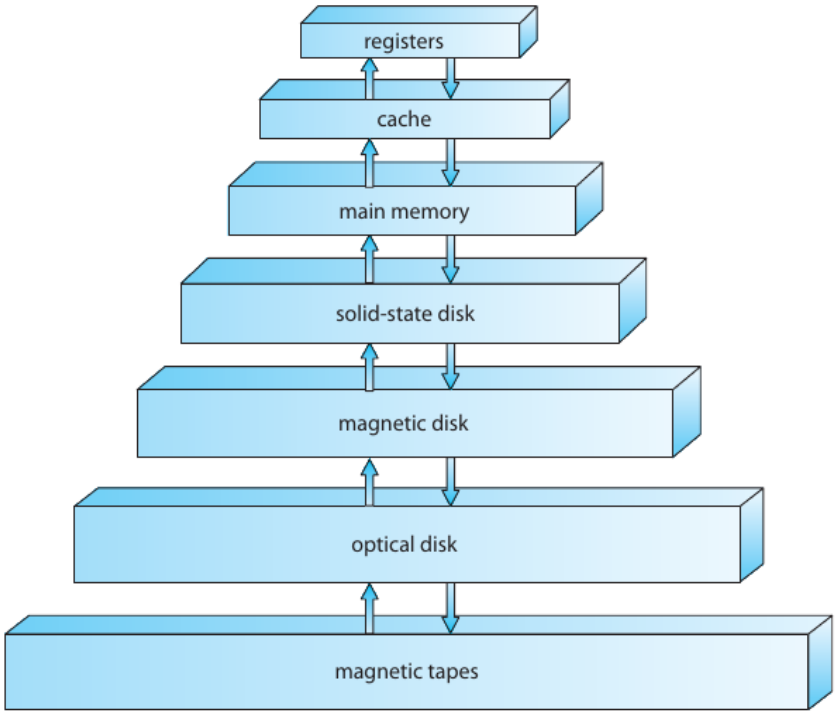
- A memória principal é fundamental para o funcionamento de um computador moderno
  - Consiste em um bloco de bytes, que pode ser dividido em partes e endereçados pelo computador;
  - Durante um ciclo de execução, a CPU pode recuperar informações na memória, decodificar operadores, processar informações e, eventualmente, armazenar os resultados novamente na memória [Silberschatz et al., 2012].
- Devido à sua importância, a memória deve ser gerenciada de forma eficiente [Tanenbaum and Bos, 2014].

Conceitos

- A memória principal e os registradores são tipos de armazenamento de uso geral que podem ser acessados diretamente pela CPU [Silberschatz et al., 2012]
  - Registradores são acessados em um (ou poucos) ciclos de clock, enquanto a memória principal requer muitos ciclos para ser acessada;
  - Para aumento do desempenho, utiliza-se uma memória intermediária, denominada cache;
  - Essa divisão produz uma hierarquia de memória, cujo objetivo é diminuir a latência de acesso.

Hierarquia de Memória

- A hierarquia de memória não é usada apenas para a memória principal.



Hierarquia de memória.
Fonte: [Silberschatz et al., 2012]

Endereço de Memória

Endereço de Memória

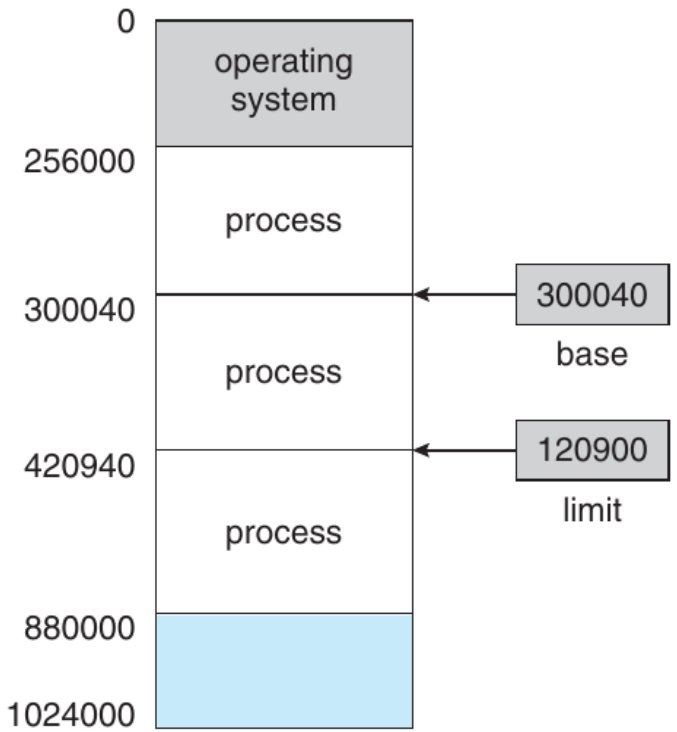
- Para garantir um bom desempenho, um sistema deve se preocupar em evitar erros
  - Em sistemas multiusuários, deve garantir que processos não interfiram uns nos outros;
  - Manter múltiplos processos separados em memória permite a execução concorrente;
  - Assim como a hierarquia de memória, esse tipo de proteção é provido por hardware [Silberschatz et al., 2012]
    - O hardware implementa esse mecanismo de diferentes formas;
    - Tal mecanismo evita que o SO interfira no acesso da memória pela CPU, uma vez que o custo seria muito elevado.

Endereço de Memória

- Para separação da memória entre processos são utilizados intervalos, limitados com auxílio de registradores
  - Registrador-base: contém o menor endereço de memória física do processo;
  - Registrador-limite: indica o tamanho do intervalo [Silberschatz et al., 2012];
- A proteção de memória é feita pela comparação de todos os endereços gerados no modo de usuário com os registradores;
- Tentativas de acesso a memória de terceiros, acidentalmente ou deliberadamente, provocam uma *trap*, interpretada como um erro fatal [Silberschatz et al., 2012].

Endereço de Memória

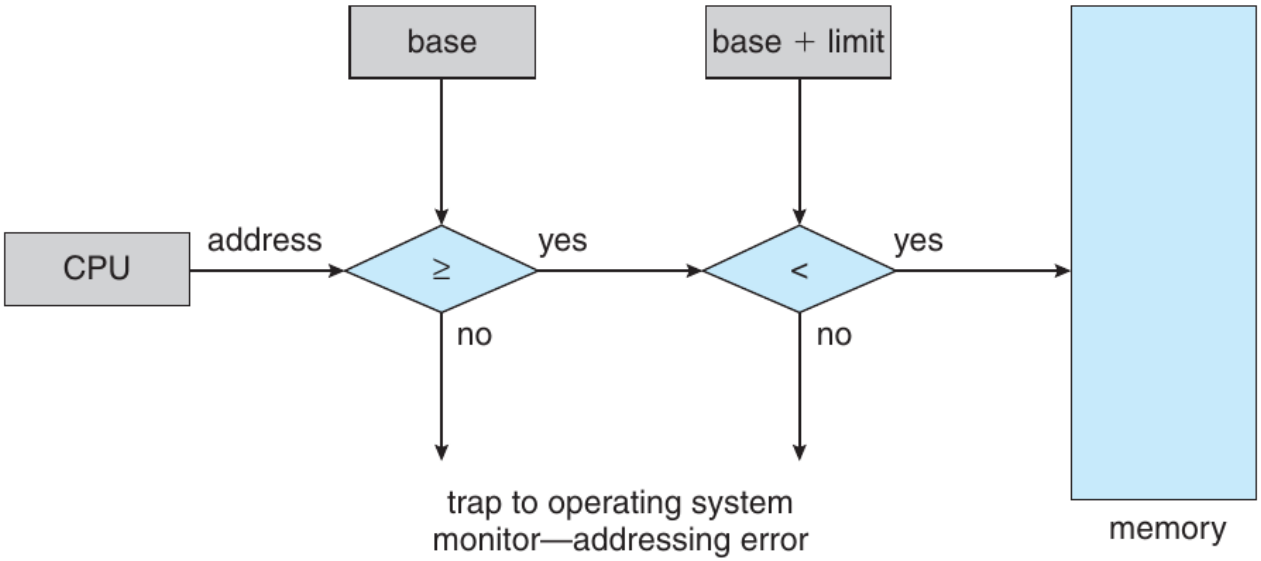
- Os registradores base e limite podem ser vistos na imagem abaixo:



Registradores Base e Limite.
Fonte: [Silberschatz et al., 2012]

Endereço de Memória

- O mecanismo de proteção de memória em hardware usando registradores base e limite pode ser visto na figura abaixo:



Mecanismo de proteção de memória com hardware.
Fonte: [Silberschatz et al., 2012]

Endereço de Memória

- Processos de sistema possuem acesso irrestrito à memória do SO e dos usuários;
- Esse poder permite que o SO:
  - Carregue programas do usuário para a memória do usuário;
  - Remova processos da memória em caso de erros;
  - Modifique parâmetros de chamadas do sistema;
  - Quaisquer outras tarefas de intervenção, para manutenção do bom funcionamento do SO [Silberschatz et al., 2012].

Código Absoluto, Realocável e Independente da Posição

Vinculação de Endereços de Memória

- A vinculação de instruções e dados a endereços de memória pode ser feita em três etapas distintas:
  - Compilação: programas são compilados para serem executados a partir de uma posição específica (fixa) de memória;
  - Carregamento: programas cujo local de carregamento não é sabido na compilação[1];
  - Execução: programas que podem ser movidos durante a execução para outros segmentos, até a etapa de mapeamento de endereços[2]
    - Utilizado pela maior parte dos SOs de propósito geral [Silberschatz et al., 2012].

[1] Se o endereço inicial mudar, o código do usuário deve ser recarregado para incorporar esse valor.

[2] Ligação do endereço de memória com o processo, permitindo que a área de memória seja identificada.

Tipos de Código

INSTITUTO FEDERAL
Minas Gerais

- A partir das etapas de vinculação das instruções temos os seguintes tipos de códigos:
  - Compilação: código absoluto;
  - Carregamento: código realocável;
  - Execução: código independente da posição.

Código Absoluto

- O código absoluto[3] somente funcionará caso seja alocado à posição que foi definida no momento de compilação
  - Deve-se saber previamente a posição em que o software será carregado na memória;
  - O acesso à memória é feito na posição estabelecida no código durante a compilação, de forma direta;
  - Se o software for carregado em um local distinto, o programa apresentará erros;
  - Tal mecanismo ocorria no MS-DOS, com os programas **.com* [Silberschatz et al., 2012].

4

[3] Nomenclatura em inglês: Absolute Code.

[4] Ao programar na linguagem Assembly, tal opção pode consistir, em muitos momentos, no método mais fácil de acesso a um determinado conteúdo.

Código Realocável

- O código realocável[5] permite que a definição da posição do código em memória seja atrasada
  - Nesse caso, a ligação final é atrasada até o tempo de carregamento;
  - Se o endereço de memória mudar, o código de usuário deve apenas ser recarregado para que funcione adequadamente;
  - Apesar das vantagens, quando o código é recarregado, ele quase se torna de natureza absoluta e fixa [Silberschatz et al., 2012].

[5] Nomenclatura em inglês: Relocatable Code ou Load-Time Locatable (LTL).

Código Independente da Posição

- O código independente da posição[6] possui todas as instruções relativas umas às outras
  - Especificam deslocamentos em relação à posição atual;
  - O SO não precisa realizar nenhuma operação pós carregamento para iniciar a execução;
  - Pode ser executado de qualquer posição da memória;
  - São comumente usados em bibliotecas compartilhadas.

[6] Nomenclatura em inglês: Position-Independent Code (PIC).

Endereço Físico e Lógico

Endereço Físico e Lógico

- Um endereço gerado pela CPU é definido como endereço lógico enquanto o endereço efetivamente visto pela unidade de memória é denominado endereço físico
  - Endereços definidos em tempo de compilação e carregamento possuem endereços lógicos e físicos iguais;
  - Endereços definidos em tempo de execução possuem endereços lógicos e físicos distintos
    - Nesse caso, o endereço lógico é denominado endereço virtual [Silberschatz et al., 2012].

As expressões "endereço virtual" e "endereço lógico" podem ser usadas, muitas vezes, com o mesmo significado.

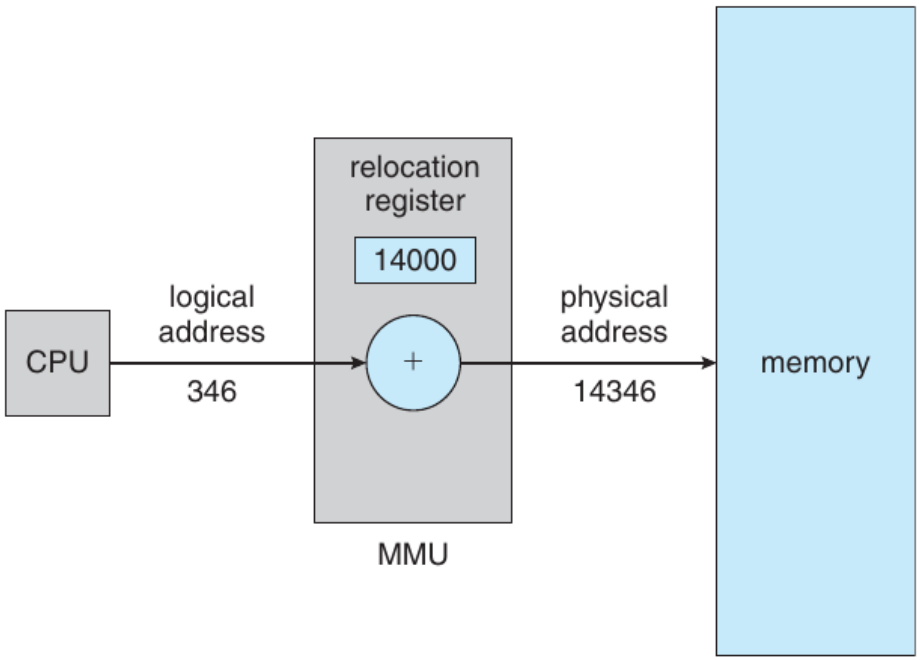
Endereço Virtual

- Para uso, o endereço virtual precisa ser mapeado em um endereço físico
  - Esse mapeamento é feito por um hardware denominado Unidade de Gerenciamento de Memória (MMU)[7];
  - Um mapeamento simples é feito com auxílio do registrador base, denominado nesse contexto de registrador de realocação
    - Um endereço sequencial é definido pelo software e somado ao registrador base, gerando o endereço físico;
    - O programa nunca acessa o endereço físico diretamente, somente utilizando esse artifício [Silberschatz et al., 2012].

[7] Tradução de Memory Management Unit (MMU).

Endereço Virtual

- O mecanismo de tradução do endereço lógico em endereço físico está representado na figura abaixo:



Mecanismo de tradução do endereço lógico em físico.
Fonte: [Silberschatz et al., 2012]

Carregamento e Linkagem

Carregamento Dinâmico

- Para execução de um processo e acesso aos seus dados, é necessário que as informações estejam em memória;
- Devido à limitação do tamanho e objetivando um uso mais eficiente, uma estratégia usada é o carregamento dinâmico
  - Nesse procedimento, uma rotina não é carregada para memória até que ela seja invocada;
  - O endereço do programa é atualizado para refletir a rotina recém carregada;
  - Tal procedimento é útil quando os códigos a serem carregados são usados com pouca frequência, evitando desperdício;
  - Carregamento dinâmico não necessita de suporte do SO, cabendo ao programador o seu uso [Silberschatz et al., 2012].

Linkagem Estática e Dinâmica (Compiladores)

- As bibliotecas compartilhadas podem ser linkadas a um software das seguintes formas: [Silberschatz et al., 2012]
  - Linkagem estática: é aquela no qual as bibliotecas do sistema são tratadas como módulos e combinadas ao programa binário;
  - Linkagem dinâmica: permite que a dependência seja postergada até o momento de execução
    - Possui como vantagem a possibilidade atualização das bibliotecas, para correção de erros e melhorias;
    - Para evitar incompatibilidades códigos podem ser associados a versões específicas de bibliotecas;
- As Dynamically Linked Libraries (DLLs)[8] são exemplos de bibliotecas de sistema vinculadas aos processos de usuários.

[8] Tradução direta: Bibliotecas Dinamicamente Vinculadas.

Swapping

Swapping

- Swapping é a técnica que permite que a soma total de memória alocada ultrapasse a quantidade de memória física de um computador [Silberschatz et al., 2012];
  - Swapping é capaz de aumentar o grau de multiprogamação de um sistema;
  - Nesse mecanismo, um processo A pode ser substituído na memória principal temporariamente, para permitir a execução de um processo B;
  - Quando o processo A precisar ser executado, ele pode voltar à memória no lugar de outro processo (não necessariamente o processo B).

A palavra *swap* pode ser traduzida como troca.

Swapping Padrão

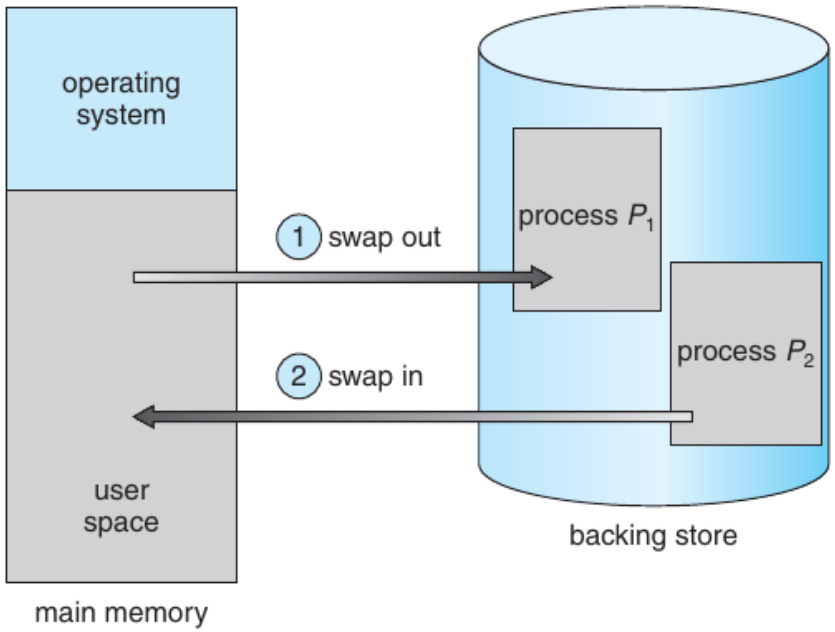
- O mecanismo de *swapping* padrão move um dado em memória principal para um dispositivo de armazenamento mais lento
  - O dispositivo pode ser um SSD ou um disco rígido[9], denominado genericamente de backing store[10];
  - A saída de dados da memória é denominada swap out enquanto a entrada de dados é denominada swap in. [Silberschatz et al., 2012].

[9] No caso dos sistemas Linux, uma partição especial de swap pode ser usada para isso.

[10] Tradução literal: unidade / loja de suporte.

Swapping Padrão

- O mecanismo de swapping padrão pode ser visto na figura abaixo:



Swapping padrão.
Fonte: [Silberschatz et al., 2012]

Nota: No swapping padrão, um processo completo é movido da memória primária para secundária. Outros mecanismos, que serão estudados à frente, pemitem a movimentação de somente partes do código-fonte.

Swapping - Mecanismo

- O sistema mantém uma fila de processos prontos, armazenados na memória ou na *backing store*;
- Quando o scheduler decide executar o processo, o dispatcher verifica se o processo está na memória
  - Caso esteja em memória, o processo é executado;
  - Caso contrário, o processo é movido para memória na troca por outro processo, segundo um critério de swap;

Nota: Para que um processo seja movido para fora da memória, ele deve estar completamente ocioso (idle). Caso esteja esperando algum dispositivo de IO, a remoção do processo de memória pode causar erros.

Swapping

- O mecanismo de swap é lento, uma vez que faz acesso a dispositivos de baixa velocidade;
  - A maior parte do tempo de swap é gasto com a transferência de dados;
  - Consequentemente a operação depende da quantidade de informação a ser movida;
- O mecanismo de swapping padrão não é utilizado em sistemas operacionais modernos
  - Variações do mecanismo são adotadas, de forma a reduzir a quantidade de movimentações e/ou trocar somente partes do processo [Silberschatz et al., 2012].

Swapping em Dispositivos Móveis

- Dispositivos móveis, em geral, não utilizam mecanismos de swapping
  - Ao invés de usarem mecanismos de swapping, os sistemas, em geral, solicitam que alguns processos liberem memória;
  - Isso ocorre devido ao uso de memórias flash, limitadas em espaço e quantidade de gravações;

Nota: O sistema iOS permite a utilização de um mecanismo conhecido como memória virual, que será visto adiante.

Alocação Contígua

Alocação de Memória

- A memória deve acomodar processos de usuários e processos do sistema operacional
  - O espaço, em geral, é dividido em uma área do SO e outra área de processos do usuário;
  - Sistemas operacionais podem ser alocados no começo ou final de memória, dependendo da estratégia do projetista do sistema [Silberschatz et al., 2012];
- Independente da posição do SO, a maior preocupação reside na forma de alocação de memória aos processos, uma vez que pode influenciar no desempenho do sistema.

Alocação de Memória

INSTITUTO FEDERAL
Minas Gerais

- Os processos podem ser alocados em memória das seguintes formas: [Silberschatz et al., 2012]
  - Alocação Contígua de Memória: aloca espaços consecutivos de memória para os processos;
  - Alocação Não Contígua de Memória: aloca espaços separados de memória para um processo;
- Para controle, podem ser utilizadas tabelas com a indicação espaços de memória disponíveis e ocupados.

Alocação Contígua de Memória

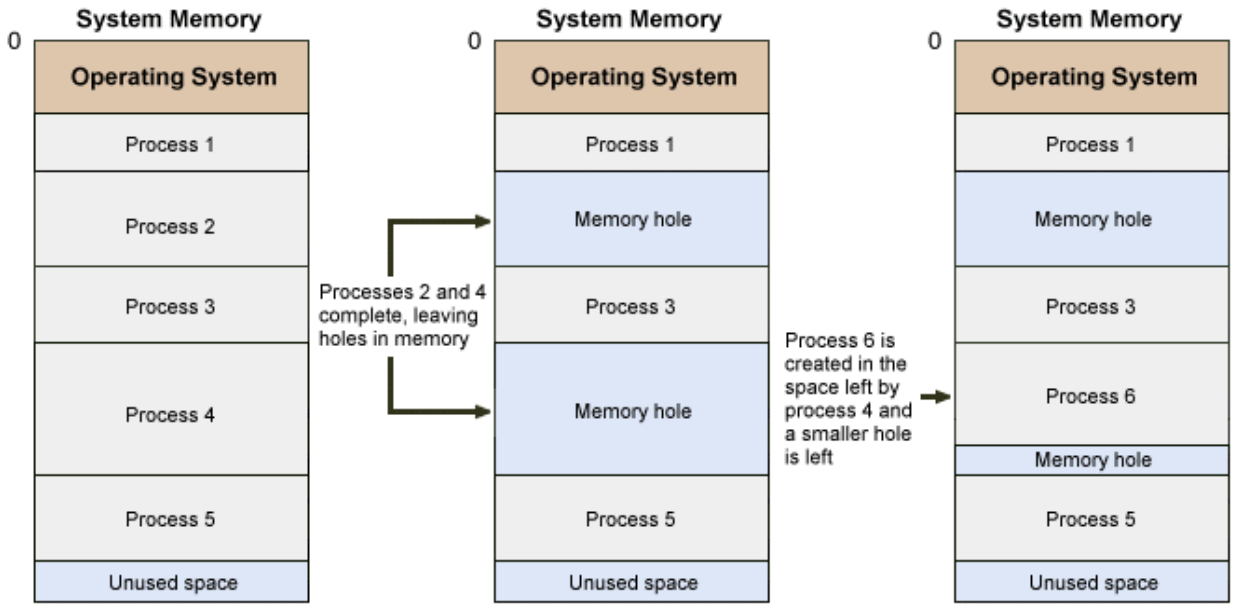
Alocação Contígua de Memória

- Na alocação contígua, a memória pode dividir o espaço disponível das seguintes formas:
  - Partições de Tamanho Fixo: memória é dividida em partições de mesmo tamanho
    - Cada partição contém exatamente um processo;
    - O número de partições define o grau de multiprogamação;
  - Partições de Tamanho Variável: cada partição possui o tamanho necessário ao processo;
    - Espaços disponíveis são denominados buracos (*holes*);
    - Processos podem ser adicionados, caso um buraco em memória tenha tamanho suficiente;
    - Caso não exista um buraco de tamanho suficiente, o processo deve aguardar [Silberschatz et al., 2012].

Alocação Contígua de Memória

INSTITUTO FEDERAL
Minas Gerais

- Devido aos diferentes tipos de softwares, requisitos de memória podem variar consideravelmente;
- À medida em que o sistema é utilizado, buracos de diferentes tamanhos são deixados na memória.



Buracos em memória devido às partições de tamanho variável.
Fonte: [Wells, 2009]

Alocação Contígua de Memória

- Buracos constituem em uma porção não utilizada da memória e podem ser considerados desperdícios;
- Para solução podem ser utilizadas as seguintes estratégias de alocação:
  - First-fit: aloca o processo no primeiro bloco de memória disponível com tamanho suficiente;
  - Best-fit: aloca o processo no menor bloco disponível com tamanho suficiente;
  - Worst-fit: aloca o processo no maior bloco disponível com tamanho suficiente [Silberschatz et al., 2012].

First, *best* e *worst fit* podem ser traduzidos, respectivamente, como primeiro encaixe, melhor encaixe e pior encaixe.

Alocação Contígua de Memória

- As estratégias de alocação *first-fit*, *best-fit* e *worst-fit* sofrem de fragmentação externa
  - À medida em que processos são adicionados e removidos da memória, pequenos pedaços são deixados vazios;
  - Em casos extremos, a soma dos fragmentos poderia ser suficiente para alocar um processo na fila, deixando parte da memória desperdiçada;
  - Pesquisas indicam que para cada n de memória alocadas, são desperdiçados $0.5n$ (50%) [Silberschatz et al., 2012].

Alocação Contígua de Memória

- Em alocação de tamanho variável, alguns fragmentos de memória podem ser muito pequenos, fazendo com que o custo de gerenciamento seja proibitivo
  - Ex.: um espaço de 4 bytes não foi utilizado; gerenciar esse espaço pode ser mais caro que deixar o espaço vago;
- Para solução desse problema, a memória é dividida em blocos de tamanho fixo e múltiplos blocos são alocados aos processos
  - O tamanho do bloco é utilizado como unidade de alocação;
  - Tal estratégia, no entanto, pode gerar a fragmentação interna, quando há desperdício de memória dentro dos blocos [Silberschatz et al., 2012].

Alocação Não Contígua de Memória

Alocação Não Contígua de Memória

- Uma alternativa para evitar o problema de fragmentação externa é permitir que os processos sejam alocados em blocos não contínuos
  - Utilizando essa técnica, um processo poderia ser dividido e ocupar muitos blocos vazios do sistema;
  - Como desvantagem, temos:
    - Necessidade de conversão de endereços durante a execução de um processo;
    - Diminuição da velocidade de execução (devido à conversão de endereços);
- Apesar das desvantagens, os benefícios obtidos são superiores, compensando as desvantagens.

Alocação Não Contígua de Memória

INSTITUTO FEDERAL
Minas Gerais

- As seguintes técnicas de Alocação Não Contígua de Memória serão estudadas nas próximas seções:
  - Segmentação;
  - Paginação;
  - Memória Virtual.

Segmentação

Segmentação

- Segmentação é o mecanismo de gerenciamento de memória que divide o espaço de endereçamento lógico em uma coleção de segmentos
  - Cada segmento possui um nome e um tamanho;
  - Um endereço pode ser identificado pelo nome[11] e pelo deslocamento (*offset*) no segmento [Silberschatz et al., 2012].
- A memória pode ser representada por um par de valores, como no modelo:

$$< \textit{número-segmento, deslocamento} >$$

[11] Por facilidade de implementação, um endereço pode ser traduzido e identificado por um número.

Nota: Segmentação é baseada na ideia de que um desenvolvedor de software entende a memória como uma coleção de estruturas (variáveis, métodos, etc) e não como um grande espaço linear para armazenamento de bytes.

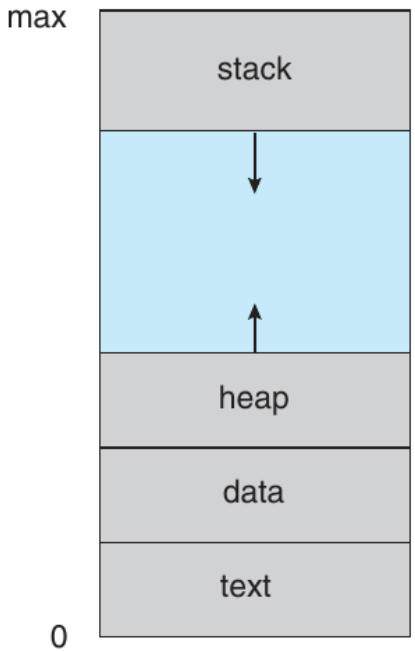
Segmentação

- O processo de compilação cria automaticamente os segmentos, de forma a refletir o código fonte do usuário;
- Um compilador C pode criar os seguintes segmentos [Silberschatz et al., 2012]:
  - Código-fonte (*text*);
  - Variáveis globais (*data*);
  - O *heap*, a partir do qual a memória é alocada;
  - Pilhas (*stack*) usadas por cada segmento;
  - A biblioteca C padrão[12].

[12] A biblioteca C normalmente é linkada ao software durante o momento de compilação.

Segmentação

- Na memória, um processo pode ser visto da seguinte forma, com os seguintes segmentos:



Organização de um processo na memória.
Fonte: [Silberschatz et al., 2012]

Segmentação

- O mapeamento de segmentos pode ser feito com uma tabela de segmentos, contendo os segmentos base e limite:

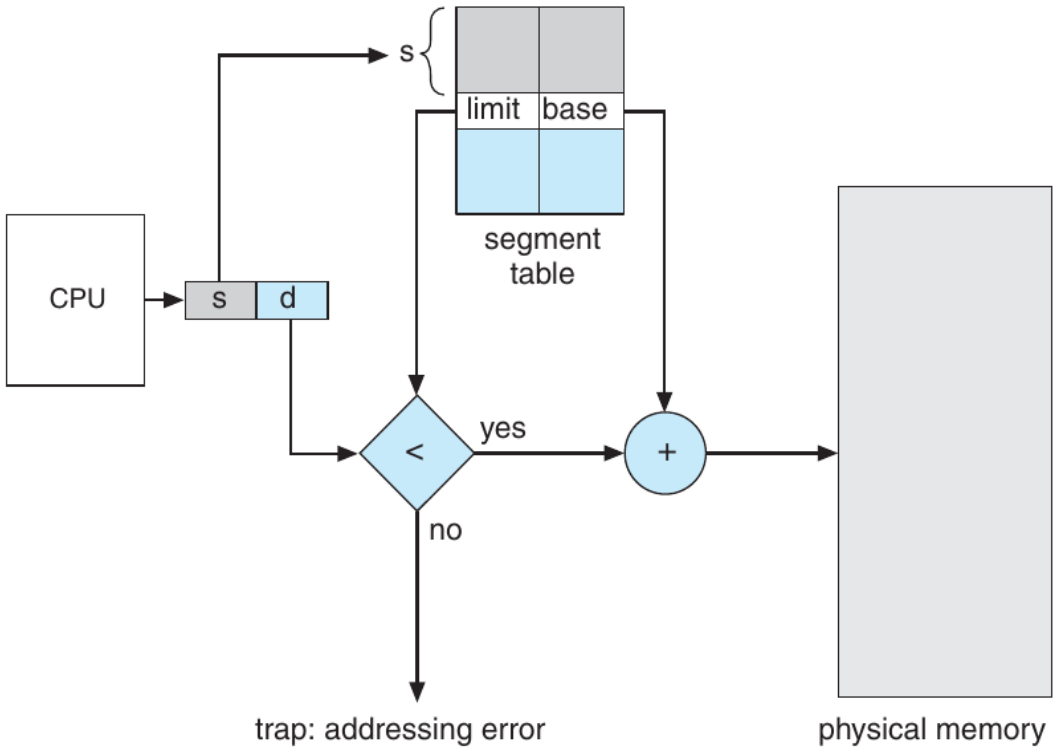
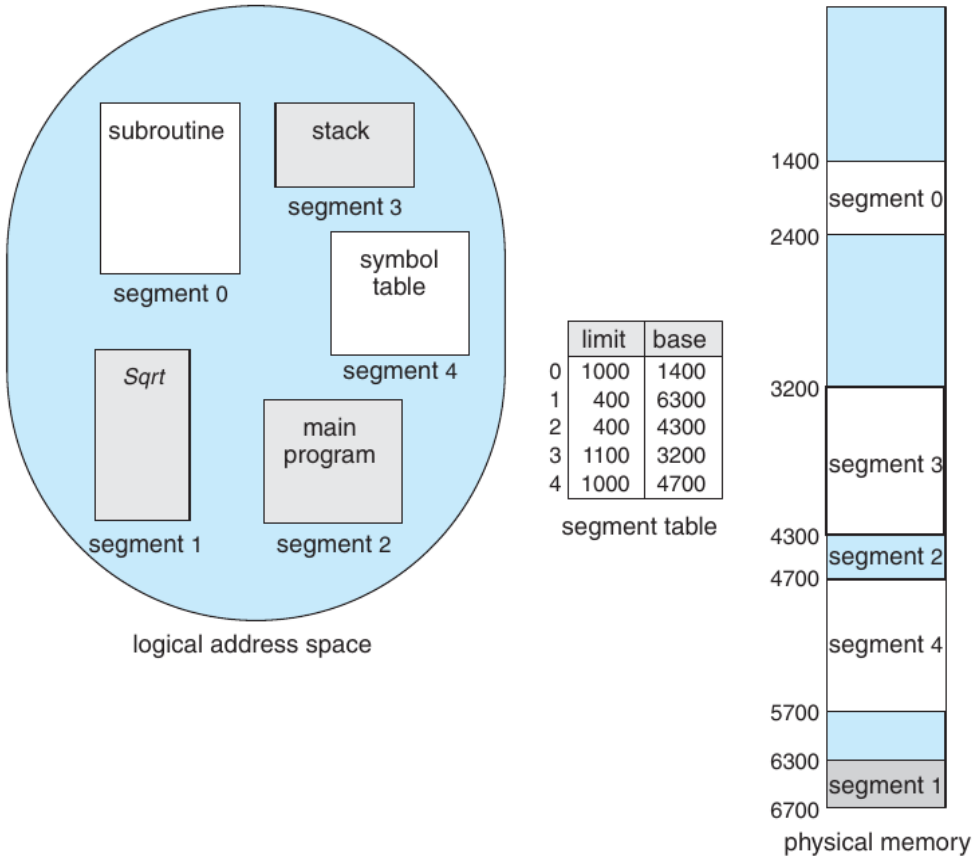


Tabela de segmentos.
Fonte: [Silberschatz et al., 2012]

Segmentação

- A segmentação pode ser resumida da seguinte forma:



Segmentação.
Fonte: [Silberschatz et al., 2012]

Paginação

Paginação

- Paginação é, assim como a segmentação, um método de gerenciamento de memória que realiza alocação não contígua
  - Paginação, no entanto, evita a fragmentação externa[13], enquanto a segmentação não evita o problema;
  - Paginação também evita problemas de fragmentação e alocação de informações de tamanho variável no *backing store* (quando ocorre swap);
- Devido às suas vantagens, é utilizada comumente em sistemas operacionais [Silberschatz et al., 2012].

[13] Apesar de evitar a fragmentação externa, a paginação pode, ainda, causar fragmentação interna.

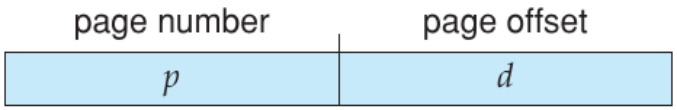
Paginação

- A paginação divide a memória física em blocos de tamanho fixo denominados quadros (*frames*) e a memória lógica em blocos do mesmo tamanho chamados páginas (*pages*)
  - Quando um processo é executado, as páginas são carregadas em quaisquer frames de memórias disponíveis;
  - Os endereços gerados pela CPU são divididos em duas partes: número da página (p) e deslocamento de página (d) [Silberschatz et al., 2012].

Número da página e deslocamento de página correspondem à tradução de *page number* e *page offset*.

Paginação

- A Tabela de Páginas contém o endereço base de cada página na memória física
  - O número de página é usado como um índice na tabela;
  - O endereço é combinado com o deslocamento de página para definir o endereço físico;
  - O tamanho de página é definido pelo hardware, variando de 512 bytes a 1GB, de acordo com a arquitetura do computador [Silberschatz et al., 2012];



Número e deslocamento de páginas.
Fonte: [Silberschatz et al., 2012]

Paginação

- O modelo de paginação de memória lógica e física está disponível na figura abaixo:

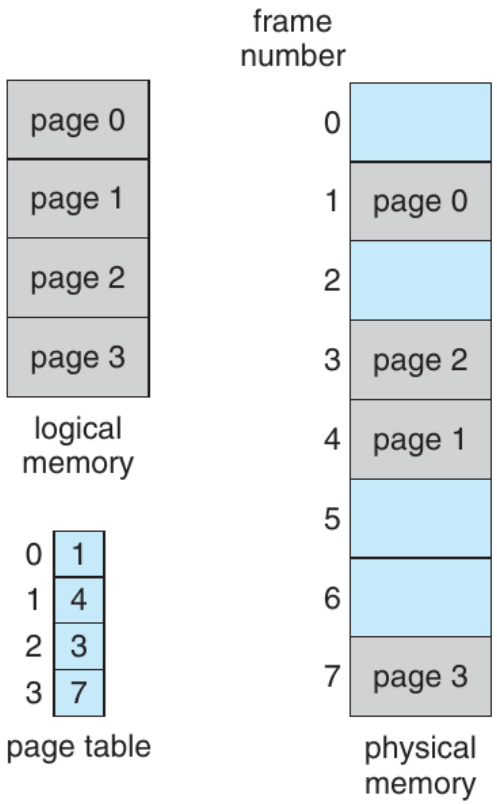


Tabela de paginação.
Fonte: [Silberschatz et al., 2012]

Paginação

- Cada página de um processo requer disponibilidade um frame;
- Se um processo precisa de *n* páginas, ao menos *n* frames devem estar disponíveis em memória;
- A paginação permite que os programadores vejam a memória como um espaço único
  - A tradução dos endereços é gerenciada pelo SO, de forma transparente para o programador;
  - Para tradução de endereços, o SO mantém uma Tabela de Frames, indicando seus status de alocação [Silberschatz et al., 2012].

Paginação

- A tabela de frames disponíveis, antes e depois da alocação de páginas, está disponível na figura abaixo:

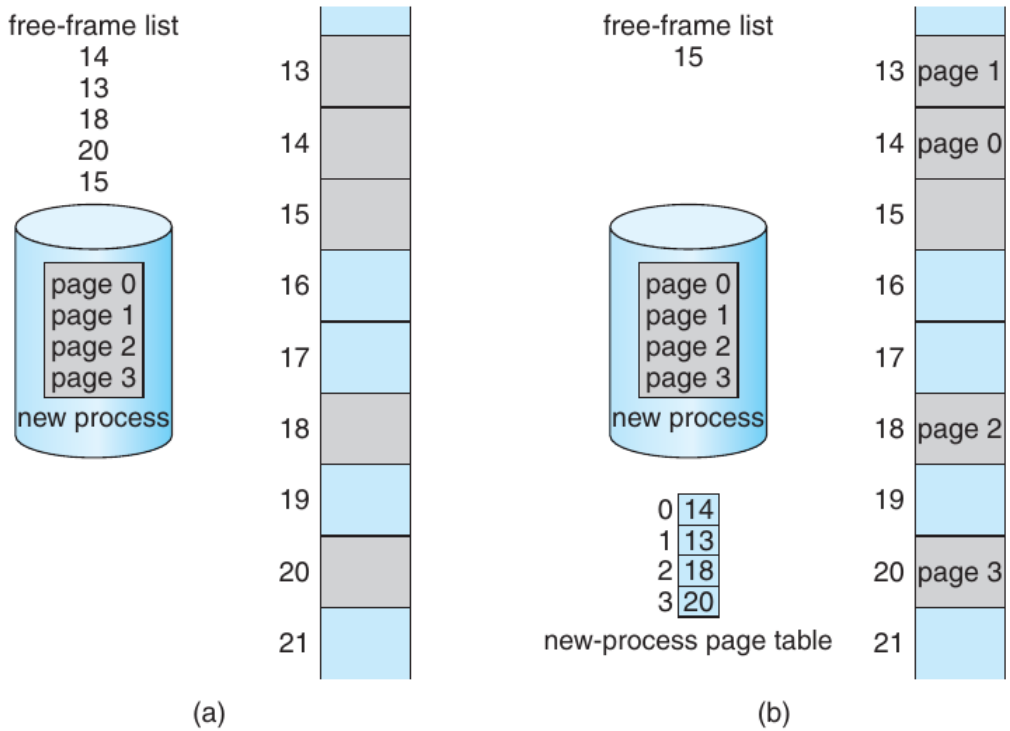


Tabela de frames antes e depois da alocação de páginas.
Fonte: [Silberschatz et al., 2012]

Paginação

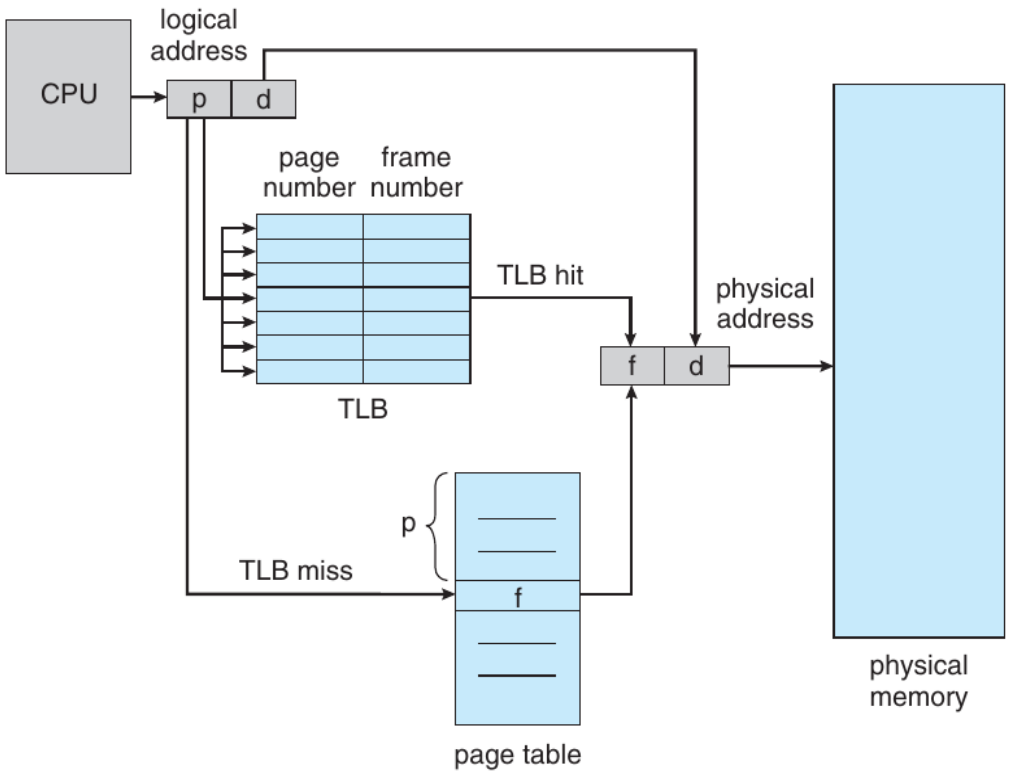
- Sistemas operacionais possuem formas distintas de armazenamento da tabela de páginas;
  - Tabelas de páginas podem ser criadas para cada processo e armazenadas em registradores;
- Devido ao aumento da memória, computadores podem ter milhares (ou até mesmo milhões) de páginas
  - Para melhor desempenho, os computadores contam com sistemas especiais para tradução de endereços;
  - A solução mais comum é uma memória associativa de alta velocidade, contendo uma chave e um valor, denominada Translation Look-Aside Buffer (TLB) [Silberschatz et al., 2012].

Paginação

- O uso da TLB é feito da seguinte forma:
  - O TLB contém poucas entradas da tabela de páginas;
  - Quando um endereço lógico é gerado pela CPU, um número de página é apresentado à TLB;
  - Se o número de página for encontrado, o número de frame é indicado para acesso à memória;
  - Caso contrário (TLB miss), uma interrupção do SO é gerada e a página e buscada na tabela de páginas do SO;
- O TLB deve ser pequeno e ao mesmo tempo deve acertar a maior quantidade de páginas possível, uma vez que acesso à tabela de páginas do SO é lento e causa um atrasos no acesso à memória [Silberschatz et al., 2012].

Paginação

- O esquema de paginação usando TLB está disponível na figura abaixo:



Paginação usando TLB.
Fonte: [Silberschatz et al., 2012]

Memória Virtual

Memória Virtual

- "Memória Virtual é a técnica que permite a execução de processos que não estão completamente na memória primária" [Silberschatz et al., 2012]
- O uso de memória virtual:
  - Permite que programas possam utilizar uma quantidade maior memória que a memória física disponível;
  - Abstrai a memória física em uma unidade de memória lógica extremamente grande;
  - Possibilita que programadores não se preocupem com limitações de espaço de memória disponível [Silberschatz et al., 2012].

Memória Virtual

- Conforme visto previamente, um programa deve estar completamente em memória para que possa ser executado
  - Tal abordagem, no entanto, limita o tamanho do programa em memória e a disponibilidade do recurso;
  - Trechos de código que raramente (ou nunca) serão executados podem ocupar espaço em memória sem necessidade;
  - Além disso, mesmo trechos necessários ao programa, podem não ser importantes em um dado momento;
  - Tal situação desperdiça um recurso fundamental (memória) [Silberschatz et al., 2012].

Memória Virtual

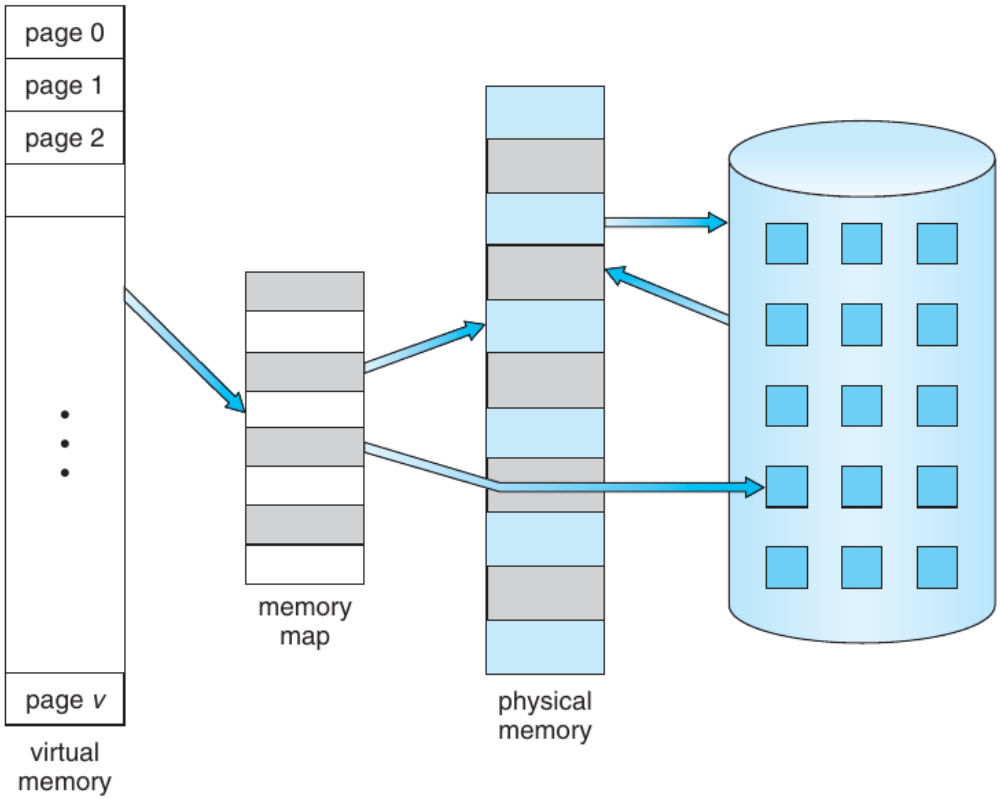
- A capacidade de execução de programas que não estão completamente em memória possibilita:
  - Simplificação da tarefa de programação, pois não é necessário avaliar a existência de memória disponível;
  - Mais programas podem ser executados ao mesmo tempo, uma vez que programas completos não precisam ser colocados em memória;
  - Com aumento do número de programas, há possibilidade de maior uso da CPU;
  - Menor quantidade de I/O necessário para carregar ou trocar programas do usuário [Silberschatz et al., 2012].

Memória Virtual

- Para endereçamento de informações, a memória virtual utiliza um espaço de endereçamento virtual
  - Esse mecanismo corresponde ao modo como um processo é armazenado em memória;
  - Os endereços iniciam tipicamente em 0, de forma contígua;
  - O mapeamento das páginas lógicas para os frames de páginas físicos é feito pelo MMU [Silberschatz et al., 2012].

Memória Virtual

- A disponibilidade de memória virtual em comparação com a memória física está disponível na figura abaixo:



Memória virtual.
Fonte: [Silberschatz et al., 2012]

Memória Virtual

- Além da separação da memória física e lógica, a memória virtual permite que múltiplos processos compartilhem páginas em memória[14]
  - Bibliotecas podem ser compartilhadas entre diferentes processos;
  - Processos podem criar regiões que serão compartilhadas com outros processos [Silberschatz et al., 2012].

[14] Tal procedimento é conhecido com *Page Sharing*.

Referências I

Silberschatz, A., Galvin, P. B., and Gagne, G. (2012).
Operating System Concepts.
Wiley Publishing, 9th edition.

Tanenbaum, A. S. and Bos, H. (2014).
Modern Operating Systems.
Prentice Hall Press, USA, 4th edition.

Wells, C. J. (2009).
Memory management.
[Online]; acessado em 07 de Dezembro de 2021. Disponível em: `https://www.technologyuk.net/computing/computer-software/operating-systems/memory-management.shtml`.