

Sistemas Operacionais

Cap.7 - Memória Principal



Prof. MSc. Renzo P. Mesquita
renzo@inatel.br

Capítulo 7

Memória Principal

7.1. Introdução

7.2. Swapping

7.3. Esquemas de Alocação de Memória

7.4. Alocação de Memória Contígua

7.5. Paginação

7.6. Segmentação

7.7. Paginação com Segmentação



Objetivos

- Apresentar como um SO pode gerenciar o hardware de memória de um computador;
- Discutir várias técnicas de gerenciamento de memória, como Alocação Contígua, Paginação e a Segmentação;



7.1. Introdução

Como já discutimos, a memória consiste em um grande Array de palavras ou bytes, cada um com seu próprio endereço. A CPU obtém instruções a partir da memória de acordo com o valor do PC (Program Counter).

A CPU pode acessar diretamente somente seus registradores, a memória cache e a memória principal.

Por conta disso, a Memória Principal deve ser protegida de acessos indevidos por parte dos processos dos usuários.

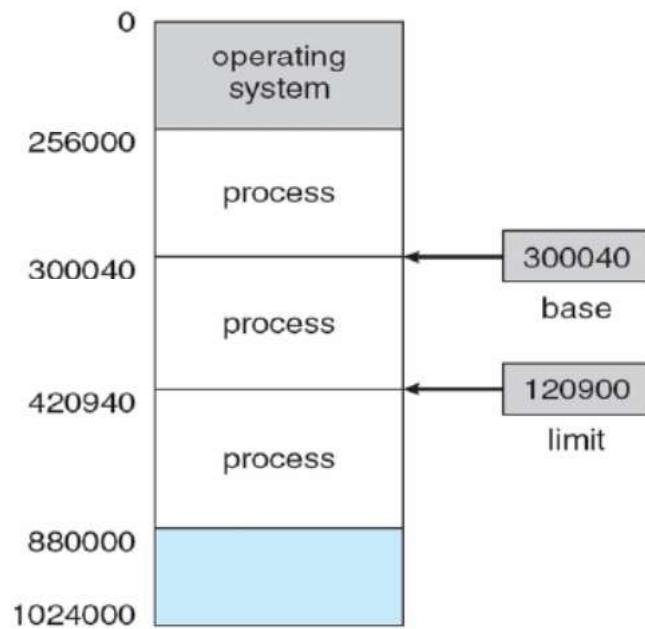
No nível mais básico, o que podemos fazer para prover tal proteção?

7.1. Introdução

Essa proteção utiliza de dois valores de registradores para cada processo, um **registrador base** e um **registrador limite**.

- O Registrador Base contém o menor endereço válido da memória física;
- O Registrador Limite especifica o tamanho do Intervalo;

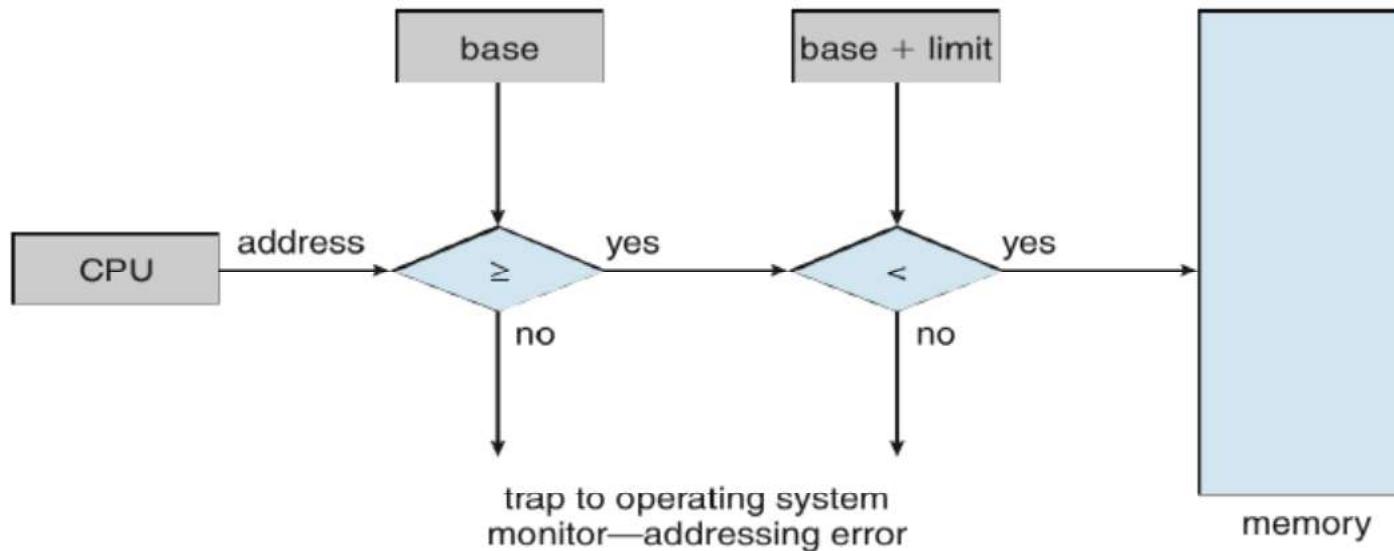
Ex:



Neste exemplo, se o registrador base contiver 300040 e o registrador limite especificar 120900, o programa poderá acessar legalmente todos os endereços de 300040 (inclusive) a 420939.

7.1. Introdução

Qualquer tentativa de acessar a memória do Sistema Operacional ou a memória de outros usuários feita por um programa em execução na modalidade de usuário resulta em uma interrupção que trata a tentativa como um erro fatal.



Apenas o Sistema Operacional, executando na modalidade de kernel, tem acesso irrestrito tanto à sua própria memória quanto à memória dos processos de usuários.

7.1. Introdução

7.1.1 Vinculação de Endereços

Um Programa para ser executado deve ser inserido na memória, se tornando um Processo. Durante sua execução, o processo acessa instruções e dados na memória

A vinculação de instruções e dados a endereços de memória pode ser realizada de 3 (três) maneiras:

1) Tempo de Compilação

Quando se sabe, em tempo de compilação, onde o processo irá residir na memória. A vinculação de endereços em tempo de compilação gera um código absoluto;

2) Tempo de Carga

Quando não se sabe em tempo de compilação onde o processo vai residir na memória, então o processo gera um código relocável na memória;

3) Tempo de Execução

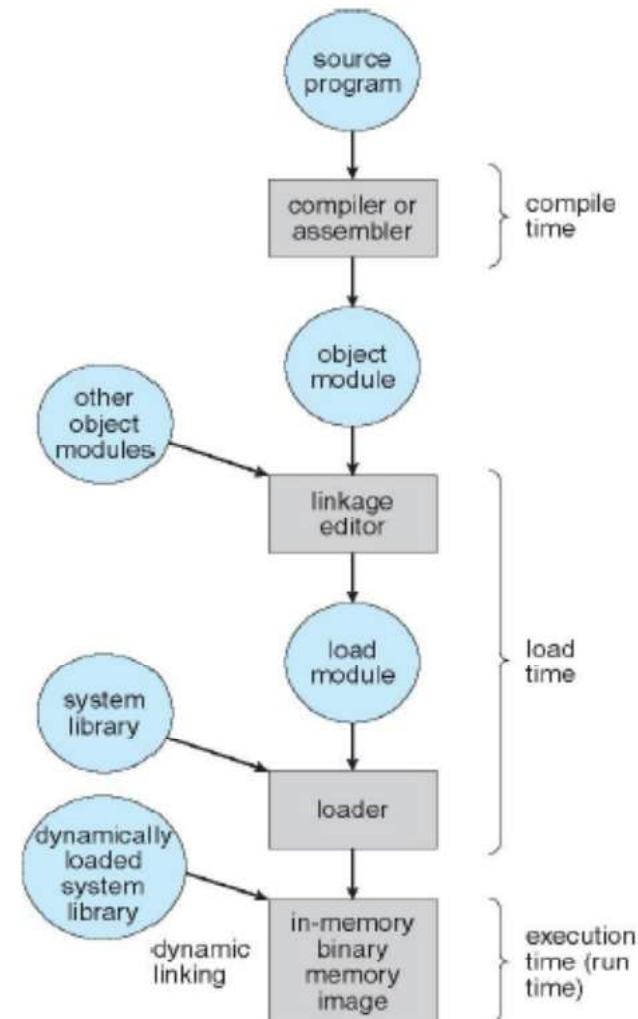
Quando o processo pode ser movimentado de um segmento de memória para outro durante sua execução. Utilizado pela maioria dos SO's.



7.1. Introdução

7.1.1 Vinculação de Endereços

- Se a **Vinculação de Endereços** (Address Binding) for feita em **Tempo de Compilação (Compile Time)** ou em **Tempo de Carga (Load Time)**, o **processo não poderá mover-se** de lugar, na memória, durante sua execução;
- Se a **Vinculação de Endereços** for feita em **Tempo de Execução (Execution Time)**, o **processo poderá mover-se** de lugar na memória durante sua execução;



7.1. Introdução

7.1.2 Espaços de Endereçamento

1) Endereços Lógicos (Endereços Virtuais)

- Os programas do usuário JAMAIS enxergam os Endereços Físicos reais. O programa do usuário lida com Endereços Lógicos;

2) Endereços Físicos

- Correspondem a uma posição real da memória;
- Implementado pelos circuitos integrados da memória;

O conjunto de todos os endereços lógicos gerados por um programa constitui-se no seu Espaço de Endereçamento Lógico;

Os correspondentes Endereços Físicos formam o seu espaço de endereçamento físico

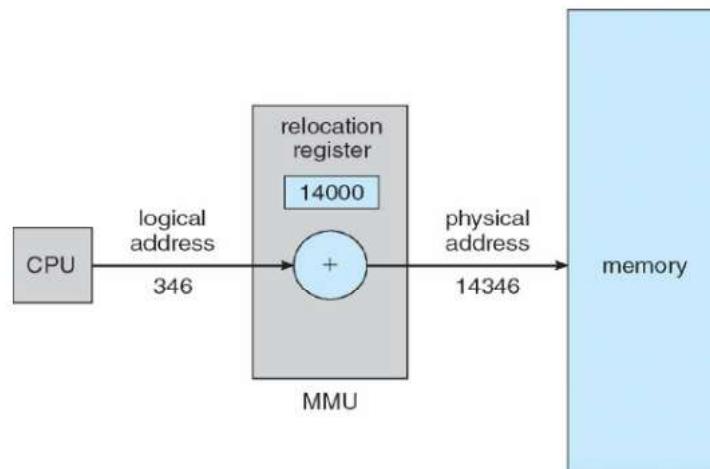
Mas afinal, como podemos transformar um Endereço Lógico em um Endereço Físico?

7.1. Introdução

7.1.2 Espaços de Endereçamento

A MMU (Memory Management Unit) é um dispositivo de Hardware que mapeia endereços lógicos em endereços físicos.

- Ela contém um registrador de relocação (relocation register), que, na verdade, é o registrador base que vimos anteriormente;
- Ela realiza o Address Binding (Vinculação de Endereços) em tempo de execução, ou seja, mapeia cada endereço lógico gerado pela CPU em um endereço físico correspondente;



7.1. Introdução

7.1.3 Carga Dinâmica

Até o momento, temos considerado que o programa inteiro e todos os dados de um processo estejam na memória física para que o processo possa ser executado (Carga Estática). Para obter melhor utilização do espaço de memória, podemos usar a Carga Dinâmica.

1) Carga Dinâmica

- Só o programa principal é carregado na memória pelo loader;
- Todas as sub-rotinas do programa são mantidas em disco;
- Sub-rotinas nunca chamadas não são carregadas desnecessariamente na memória;

2) Carga Estática

- O Programa inteiro e os dados do processo têm que estar em memória;
- O programa é integralmente carregado pelo loader;
- O tamanho máximo de um processo é limitado pelo tamanho da memória;

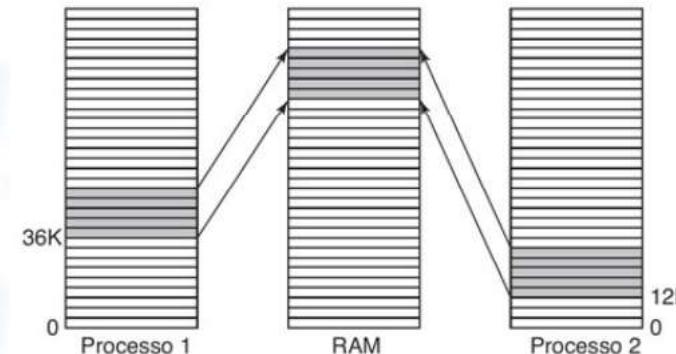


7.1. Introdução

7.1.4 Vinculação Dinâmica e Bibliotecas Compartilhadas

Cada programa pode usufruir de várias bibliotecas comuns. Mas o que poderíamos fazer para **compartilhar bibliotecas entre programas** a fim de maximizar o uso da Memória Principal? Podemos usar de uma técnica chamada **Vinculação Dinâmica**.

- Com este esquema, todos os processos que utilizam uma biblioteca de linguagem executam apenas uma cópia do código da biblioteca;
- Os **Programas utilizam de Stubs** (pequenos fragmentos de código) para **referenciar as bibliotecas comuns**;
- Evita o desperdício tanto de espaço em disco quanto em memória principal;

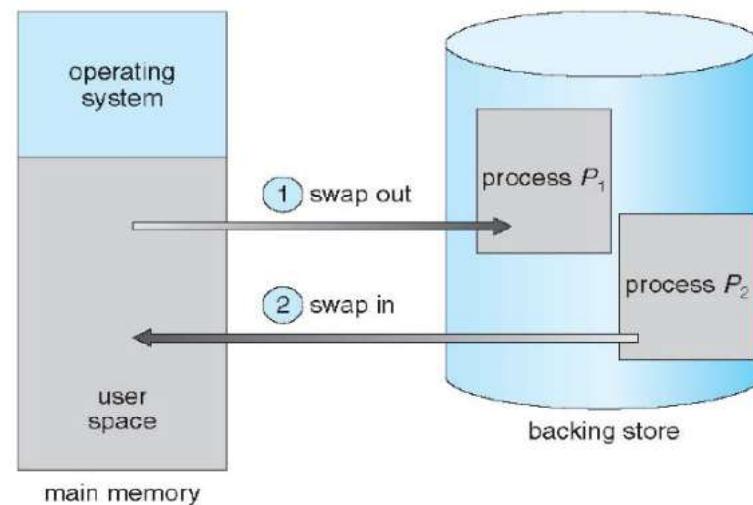


Obs: Nem todos Sistemas Operacionais aceitam Vinculação Dinâmica, mas apenas Estática, onde cada biblioteca deve ser incorporada em cada programa.

7.2. Swapping

Um processo precisa estar na memória para ser executado. Entretanto, um processo pode ser transferido temporariamente da memória principal para uma memória secundária e retornar quando necessário. Esta técnica é chamada Swapping.

- **Swapping Out** é a remoção integral de um processo da memória para o disco;
- **Swapping In** é a volta integral de um processo do disco para a memória;
- Se o CPU Scheduler selecionar um processo não residente na memória para executar, então ele deve trazer para a memória este processo.



7.2. Swapping

Swapping é uma **operação custosa**, por isso, deve ser evitado.

Que tipos de Processos são fortes candidatos a realizarem Swapping?

- Processos em hibernação;
- Processos suspensos;
- Processos em Espera (Wait) de I/O;

Swapfiles são arquivos em disco com a finalidade de **armazenar Swapped Processes**. O acesso ao SwapFile deve ser otimizado por questões de desempenho.



7.3. Esquemas de Alocação de Memória

Como já vimos, a Memória Principal deve acomodar tanto o SO quanto os diversos processos de usuário. Portanto, precisamos alocar a memória principal da maneira mais eficiente possível.

A memória é normalmente dividida em duas partições: uma para o Sistemas Operacional e outra para os Processos do Usuário.

Como podemos alocar a memória disponível aos processos que estão na fila de entrada esperando para serem carregados na memória?

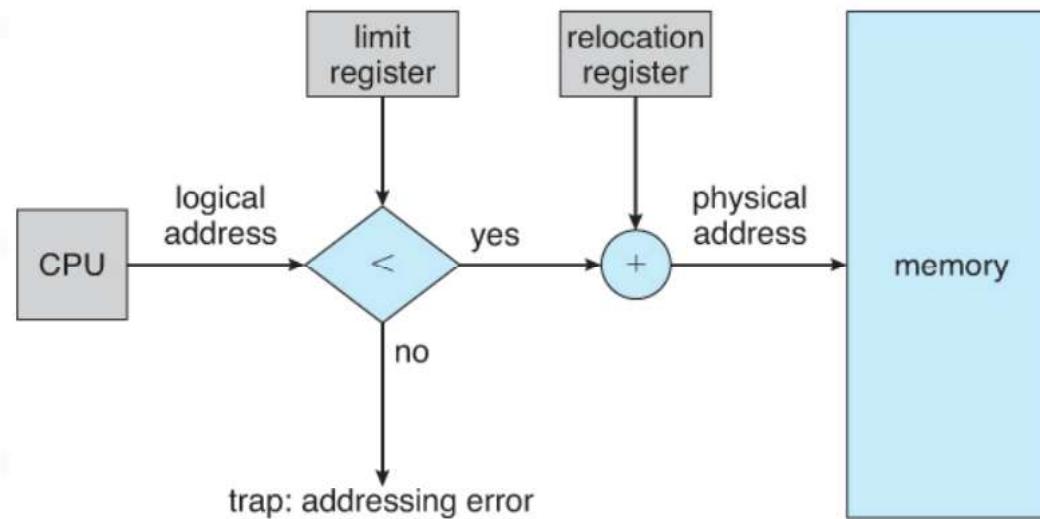
Podemos utilizar de 3 (três) técnicas:

- 1) *Alocação Contígua (Contiguous Allocation);*
- 2) *Paginação (Paging);*
- 3) *Segmentação (Segmentation);*



7.4. Alocação Contígua

Neste esquema de alocação, como o **espaço de endereçamento do processo é contíguo**, o mapeamento de um Processo na memória pode ser definido apenas pelos valores dos registradores **Relocation** e **Limit**:



Vale lembrar que quando o Scheduler da CPU seleciona um Processo para execução, o **despachante carrega os registradores de relocação e limite** com os valores corretos no momento de context switching.

7.4. Alocação Contígua

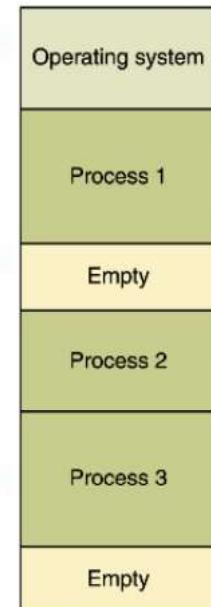
No método de alocação contígua, duas estratégias podem ser utilizadas para alocar processos na memória, são elas:

1) Partições de Tamanho Fixo

- Divide toda a memória em várias **partições de tamanho fixo**;
- Nível de **multiprogramação** fica **limitado** pela quantidade de partições;
- Apesar de simples, este método já caiu em desuso por não ser muito viável;

2) Partições de Tamanho Variável

- O **Sistema Operacional** mantém uma **tabela** indicando que partes da memória estão disponíveis e quais estão ocupadas;
- **Partições são criadas conforme a necessidade** dos Programas;
- Com a entrada de programas, a memória passará a ter um conjunto de intervalos de vários tamanhos;



7.4. Alocação Contígua

Programas entram o tempo todo na memória. Como alocar um bloco de memória (hole) que satisfaça o tamanho de um programa? (Ou seja maior ou igual a n bytes). Podemos usar de 3 métodos:

1) FIRST FIT

Aloca o programa na primeira partição grande o bastante para alocá-lo;

2) BEST FIT

Aloca o programa na menor partição grande o bastante para alocá-lo;

3) WORST FIT

Aloca o programa na maior partição grande o bastante para alocá-lo;

Obs: Essas diferentes alocações são possíveis pois o Sistema Operacional guarda uma lista de partições disponíveis.

7.4. Alocação Contígua

7.4.1. Fragmentação

Conforme processos são carregados na memória e dela removidos, o espaço de memória livre fica dividido em pequenos pedaços, gerando a **Fragmentação Externa**.

A Fragmentação Externa ocorre quando existe espaço total de memória suficiente para atender uma solicitação, mas os espaços disponíveis não são contíguos.

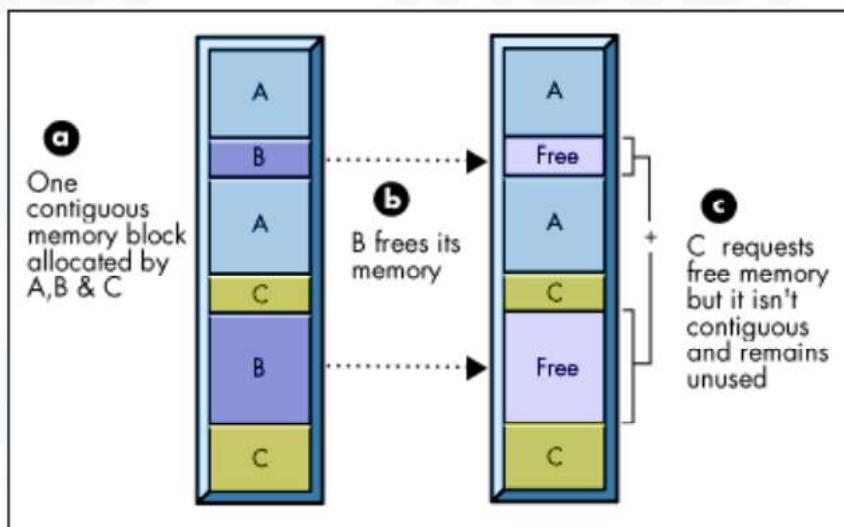
Solução: Utilizar de uma técnica chamada **Compactação**.

- Que no modo mais simples **move todos os processos** para uma das **extremidades da memória** a fim de liberar espaço;
- Ou permite que o espaço de endereçamento lógico dos processos não seja contíguo (Paginação e Segmentação);
- Apesar de resolver o problema, é uma operação custosa para o SO;

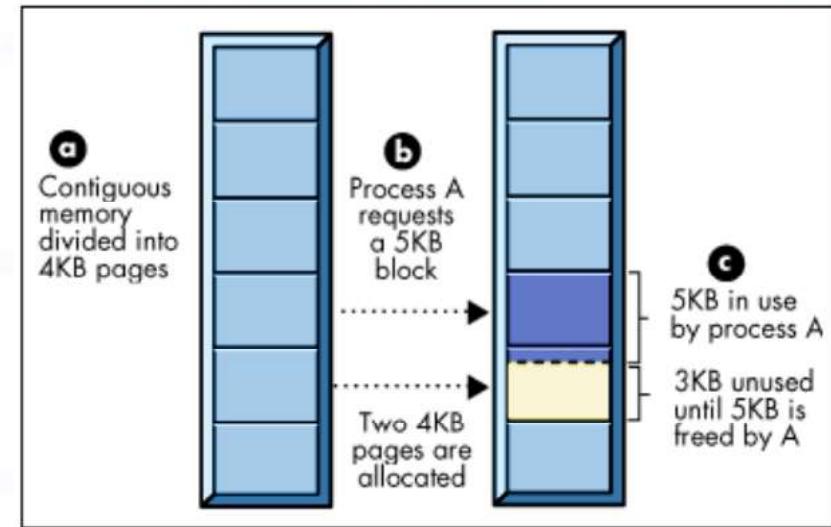
7.4. Alocação Contígua

7.4.1. Fragmentação

A fragmentação também pode ser **Interna**, quando o bloco de memória dado a um processo é maior que o necessário.



Fragmentação Externa



Fragmentação Interna

Fonte imagens: <http://support.novell.com/techcenter/articles/ana19970301.html>

HORA DO

Kahoot!



ACESSSE:

<https://kahoot.it>

7.5. Paginação

7.5.1. Método Básico

O grande problema do uso de Alocação Contígua é que ela fragmenta a memória.

A Paginação é um esquema de gerenciamento de memória que permite que o espaço de endereçamento físico de um processo não seja contíguo, consequentemente, evitando a fragmentação externa e a necessidade de compactação.

- A Memória Física é dividida em blocos de tamanho fixo chamados quadros;
- A Memória Lógica é quebrada em blocos do mesmo tamanho chamados páginas;

Vamos ver como este processo funciona?

7.5. Paginação

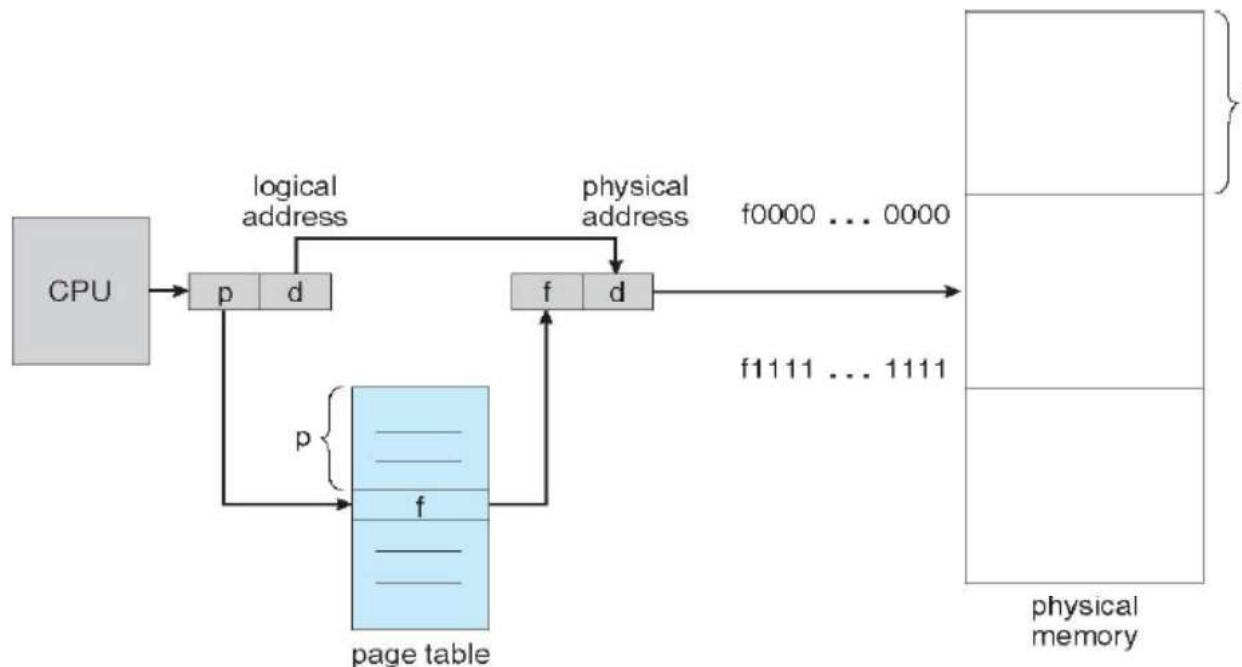
Cada endereço gerado pela CPU é dividido em duas partes:

Número de Página (p):

Usado como um índice na Tabela de Páginas que contém o endereço base de cada página na memória física;

Deslocamento de Página (d):

É combinado com o endereço base para definir o endereço na memória física;



7.5. Paginação

EXEMPLO 1

Considere a memória ao lado.
Usando um tamanho de página de 4 bytes e uma memória física de 32 bytes (8 páginas), encontre:

A) O Endereço Físico do Endereço Lógico 0;

B) O Endereço Físico do Endereço Lógico 4;

C) O Endereço Físico do Endereço Lógico 13;

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

logical memory

0	5
1	6
2	1
3	2

page table

0	
4	i j k l
8	m n o p
12	
16	
20	a b c d
24	e f g h
28	

physical memory



7.5. Paginação

EXEMPLO 1

Considere a memória ao lado.
Usando um tamanho de página de 4 bytes e uma memória física de 32 bytes (8 páginas), encontre:

A) O Endereço Físico do Endereço Lógico 0;
 $(5 \times 4) + 0 = 20$

B) O Endereço Físico do Endereço Lógico 4;
 $(6 \times 4) + 0 = 24$

C) O Endereço Físico do Endereço Lógico 13;
 $(2 \times 4) + 1 = 9$

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

logical memory

0	5
1	6
2	1
3	2

page table

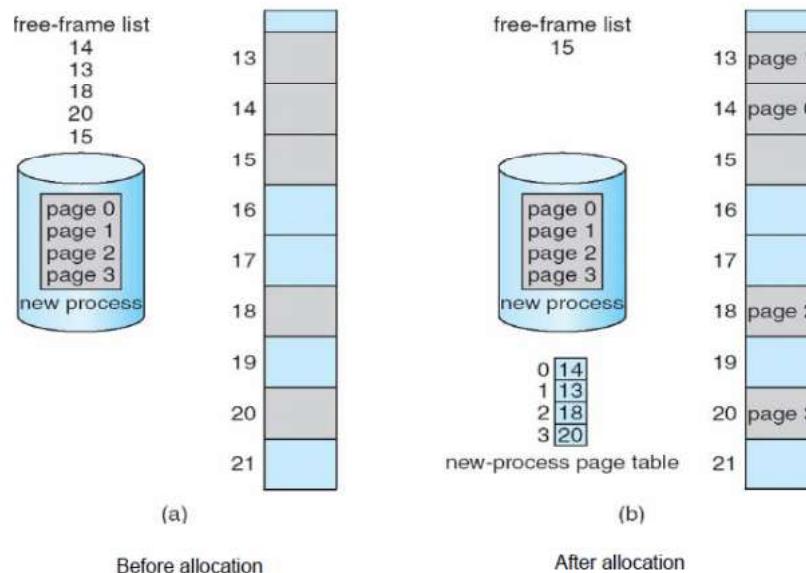
0	
4	j k l
8	m n o p
12	
16	
20	a b c d
24	e f g h
28	

physical memory

7.5. Paginação

Um aspecto importante da paginação é a separação clara entre a visão que o usuário tem da memória e a memória física real.

- O programa do usuário vê a memória como um espaço único, mas na verdade, o programa é espalhado por toda a memória física, que também contém outros programas;
- Todo mapeamento fica oculto do usuário e é realizado pelo Sistema Operacional;
- O Sistema Operacional mantém uma cópia da Tabela de Páginas para cada processo;
- Do modo que quadros vão sendo liberados, novas páginas vão sendo alocadas;



7.5. Paginação

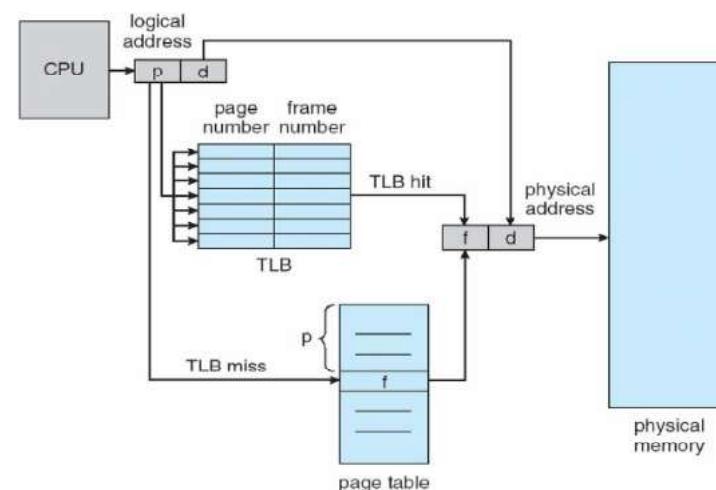
7.5.2. Suporte de Hardware

Antigamente

- A page table de cada processo era gravada em registradores;
- O PCB de cada processo armazenava os valores destes registradores;

Atualmente

- A page table de cada processo é muito grande, e fica, portanto, no espaço de endereçamento do SO;
- Um registrador chamado PTBR (Page-table base register) aponta para a page table;
- O Processador utiliza de uma Cache de memória chama TLB (Translation look-aside buffer) que guarda as páginas de diferentes processos mais acessadas recentemente;
- Cada entrada contém um address space identifier (ASID) que especifica o processo a qual a entrada se refere;



7.5. Paginação

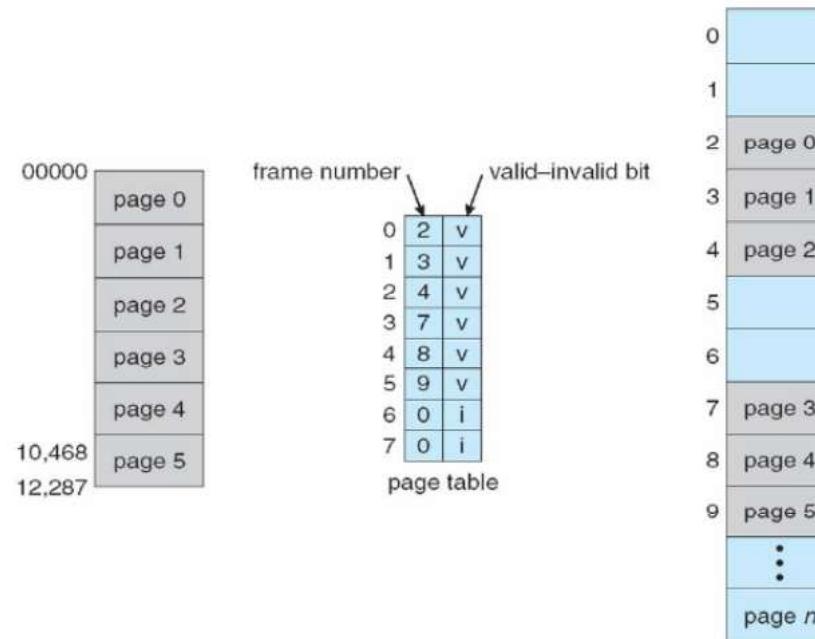
7.5.3. Proteção

O processo de Paginação deve incluir um **mecanismo para proteção de memória**;

Geralmente um **bit válido-inválido** e um **bit de proteção read-write** deve ser associado a cada posição na tabela de páginas para prover esta proteção;

O **Bit válido-inválido** caso esteja em 'i', indica que uma página ainda não se encontra na memória - page fault;

O **Bit read-write** indica se uma página é de leitura-gravação ou somente de leitura;



7.5. Paginação

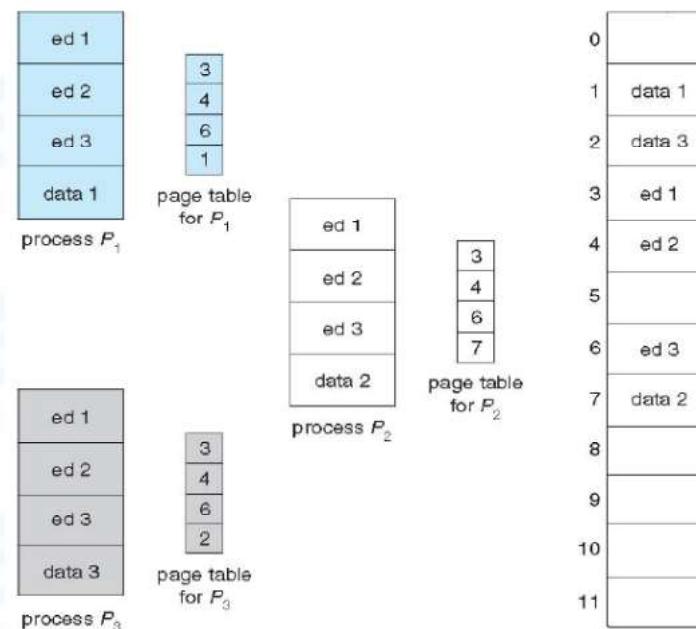
7.5.4. Páginas Compartilhadas (Shared Pages)

Uma grande vantagem de paginação é a facilidade em compartilhar código reentrante. Mas o que seria códigos reentrantes?

Código Reentrante é aquele que não se modifica por si mesmo, ou seja, nunca muda durante a execução (Read-Only)

Portanto pode ser executado ao mesmo tempo por vários Processos.

Ex:

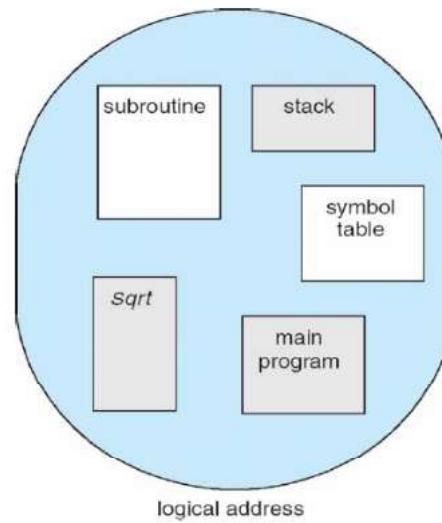


7.6. Segmentação

7.6.1. Método Básico

A visão que usuários preferem ter da memória não é como uma sequência de bytes;

Usuários preferem ter uma visão da memória como uma coleção de **Segmentos** de tamanho variável;



A Segmentação é um esquema de gerenciamento de memória que suporta a visão que o usuário tem da memória

7.6. Segmentação

Um espaço de endereçamento lógico passa a ser uma Coleção de Segmentos;
Cada Endereço Lógico consiste em uma tupla dupla:

<índice do segmento, deslocamento>

Normalmente, o programa do usuário é compilado e o compilador constrói automaticamente segmentos para este programa. Por exemplo, o Compilador C pode criar segmentos separados para os elementos a seguir: 1. Código;

2. Variáveis Globais;
3. Heap
4. Stacks (Pilhas)
5. Bibliotecas

...

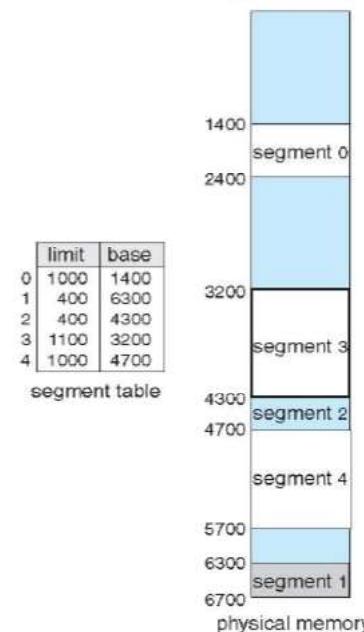
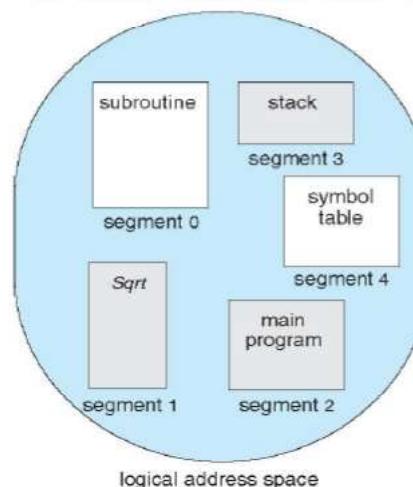
7.6. Segmentação

7.6.2. Hardware

O mapeamento de segmentos lógicos em endereços físicos é feito por meio de uma Segment Table;

Cada entrada da Segment Table possui uma base e um limite;

Logo, a Segment Table é um array de pares de registradores base-limite;



7.7. Paginação com Segmentação

Se os segmentos são grandes, não dá para mantê-los na memória em sua totalidade.

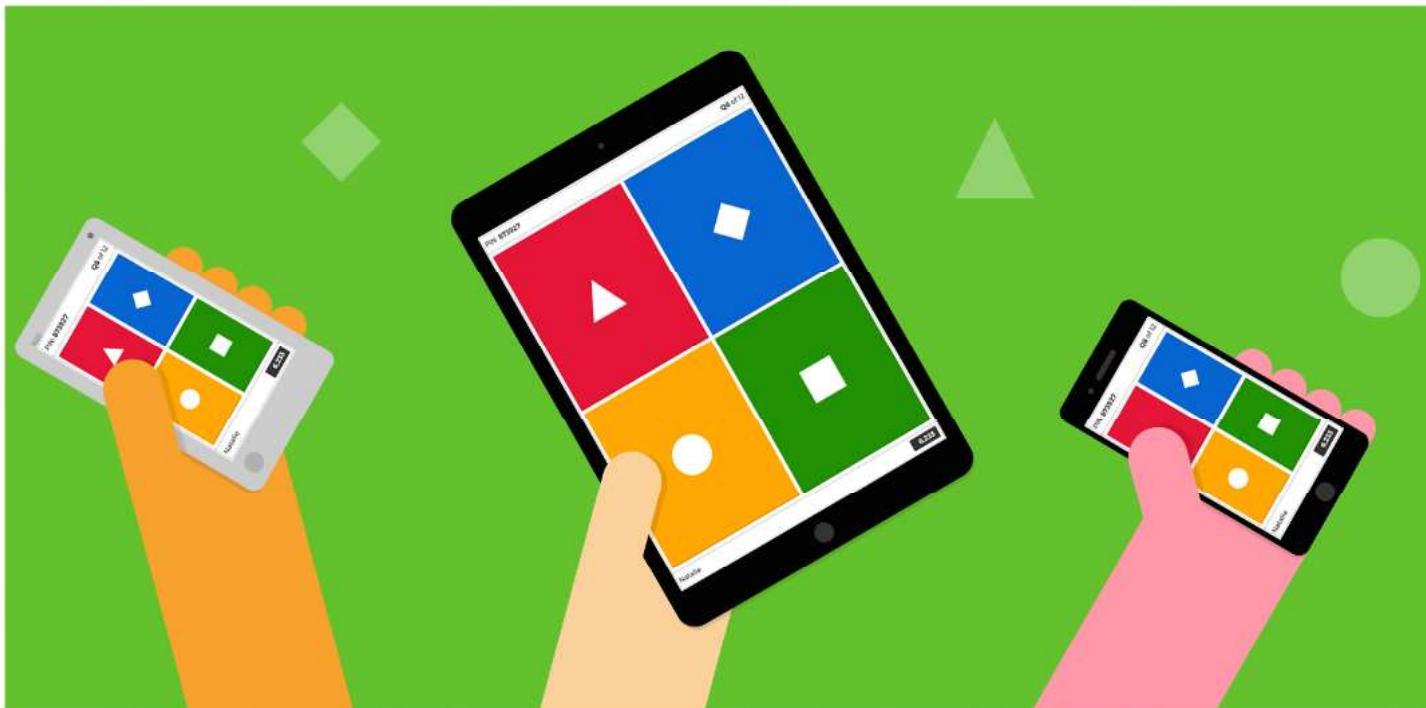
O que fazer para resolver este problema?

Aplicar a idéia de **Paginação nos Segmentos**, só deixando na memória as páginas realmente necessárias para cada segmento. Logo, o endereço lógico das páginas de um segmento com paginação seria formado pela combinação:

<Nº do Segmento, Nº da Página, Deslocamento da Página>

HORA DO

Kahoot!



ACESSE:

<https://kahoot.it>

**FIM
DO
CAPÍTULO 7**



EXERCÍCIOS