Capítulo 3

Gerenciamento de Memória

- Gerenciamento básico de memória
- Troca de processos
- Memória virtual
- Algoritmos de substituição de páginas
- Modelagem de algoritmos de substituição de páginas
- Questões de projeto para sistemas de paginação
- Questões de implementação
- Segmentação

Gerenciamento de Memória

Principais funções:

- manter o controle de quais partes da memória estão em uso e quais não estão
- alocar e liberar memória aos/dos processos
- foco do estudo de gerenciamento: ***memória principal***
- tratar da hierarquia de memórias

Gerenciamento de Memória

- Desejo de todo programador. Ter uma memória:
  - grande
  - rápida
  - não volátil
  - custo baixo

- Hierarquia de memórias
  - pequena quantidade de memória rápida, volátil e de alto custo - cache
  - quantidade considerável (gigabytes) de memória principal de velocidade média, volátil e de custo médio
  - terabytes de armazenamento em disco de velocidade e custo baixos
  - além do **armazenamento removível:** DVD, USB e HD externo

Sem abstração de memória

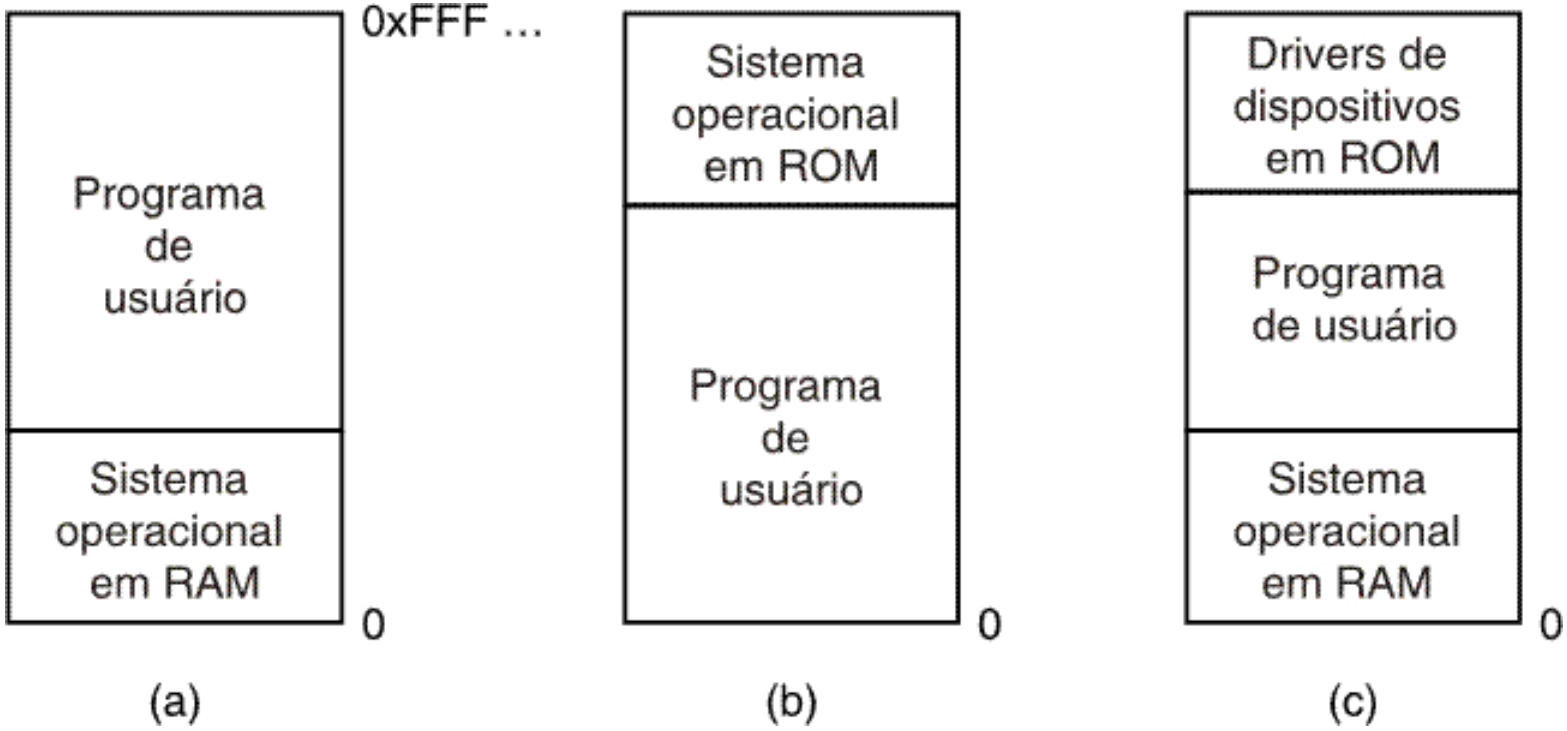
- Até 1980 os computadores não tinham abstração de memória
  ⇒ ***cada programa via apenas a memória física.***
  Mesmo assim, 3 opções de organização da memória eram possíveis

1) Um sistema operacional e um processo de usuário.
   Ex: **MOV REGISTER1, 1000**
   move conteúdo da posição 1000 (mem física) p/ registrador1

Neste cenário, não é possível executar 2 programas ao mesmo tempo
⇒ **um programa poderia apagar o conteúdo do outro**

Gerenciamento Básico de Memória

Monoprogramação sem Troca de Processos ou Paginação



a) computadores de grande porte antigos e minicomputadores;

b) sistemas embarcados

c) primeiros computadores pessoais – ROM: BIOS (basic input output system)

Gerenciamento Básico de Memória

Monoprogramação sem Troca de Processos ou Paginação

- ✓ para cada commando do teclado: SO copia o programa equivalente ao comando do disco para a memória. Para um novo commando, um novo programa é carregado, sobrescrevendo o programa anterior;

- ✓ como obter paralelismo? Usando threads

  problemas: 1. programas não relacionados não podem se executados ao mesmo tempo

  2. improvável que o SO sem abstração de memória forneça abstração de *threads*

- ✓ outra forma de obter "paralelismo":
  swapping (troca de processos entre a memória e o disco)

Múltiplos programas sem abstração de memória

Hardware especial: registrador PSW com uma chave de 4 bits

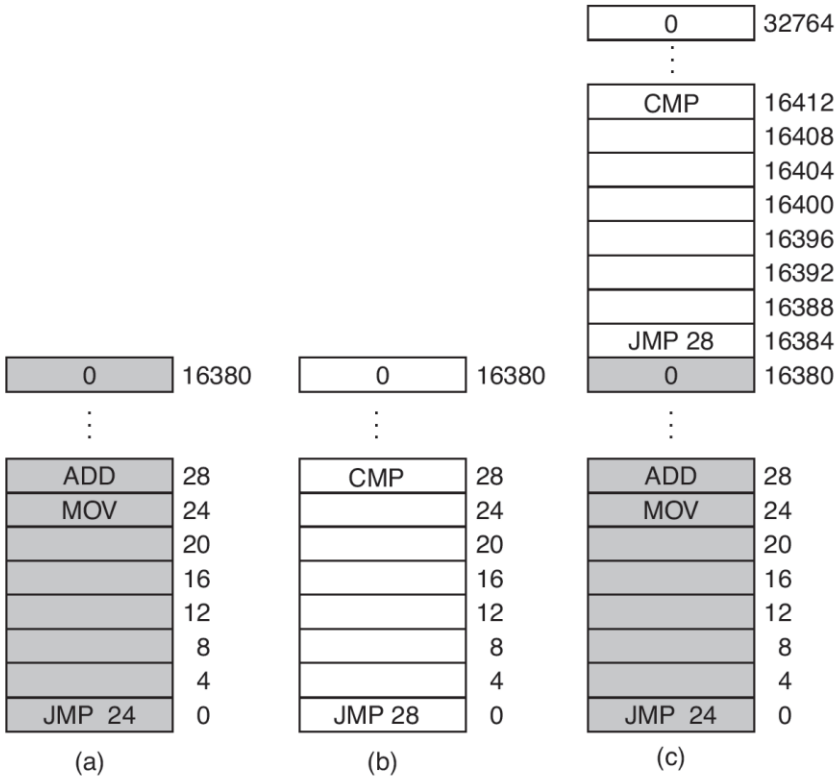


Figura 3.2 Ilustração do problema de realocação. (a) Um programa de 16 KB. (b) Outro programa de 16 KB. (c) Os dois programas carregados consecutivamente na memória.

Múltiplos programas sem abstração de memória

Problema:

JMP 28 faz um salto para um endereço físico

Solução:

somar 16384 a todos os endereços no carregamento do programa (realocação estática).

$\Rightarrow$ mais lento

$\Rightarrow$ como distinguir entre endereços e constantes?

Ex: MOV R1, 28

Sem abstração de memória

- A falta de uma abstração de memória ainda é comum em sistemas embarcados e de cartões inteligentes.

- Exemplos nos quais o software endereça a memória absoluta: rádios, máquinas de lavar roupas e fornos de micro-ondas. Nestes casos:
  - todos os programas são conhecidos antecipadamente
  - usuário não pode executar seu próprio software

- Smartphones possuem sistemas operacionais mais elaborados.

Abstração de memória: espaço de endereçamento

Problemas a serem resolvidos: ***realocação e proteção***

- ***Proteção*** Ex: programas defeituosos: pode ser resolvido com as chaves (IBM-360)
- ***Realocação*** à medida que são carregados

Problema: ***solução lenta porque não se sabe, a princípio, onde o programa será realocado na memória***

Espaço de endereçamento

Definição:

Conjunto de endereços que um processo pode usar para endereçar a memória

Cada processo tem seu próprio espaço de endereçamento

Registrador base e registrador limite

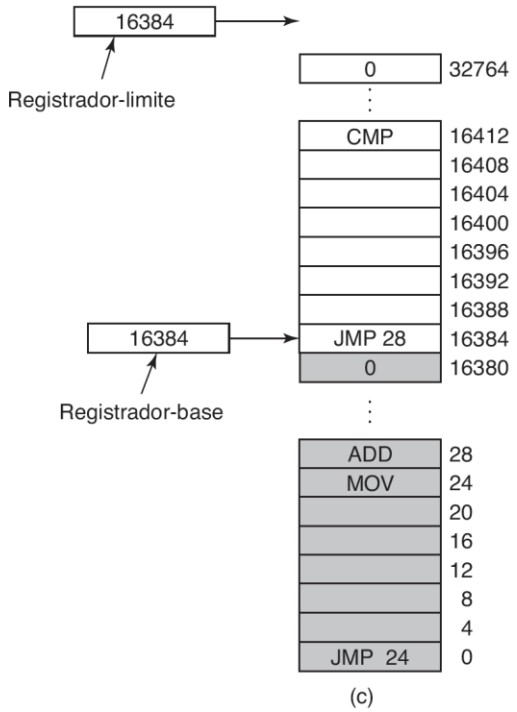


Figura 3.3 O registrador-limite e o registrador-base podem ser usados para dar a cada processo um espaço de endereçamento independente.

Desvantagem:

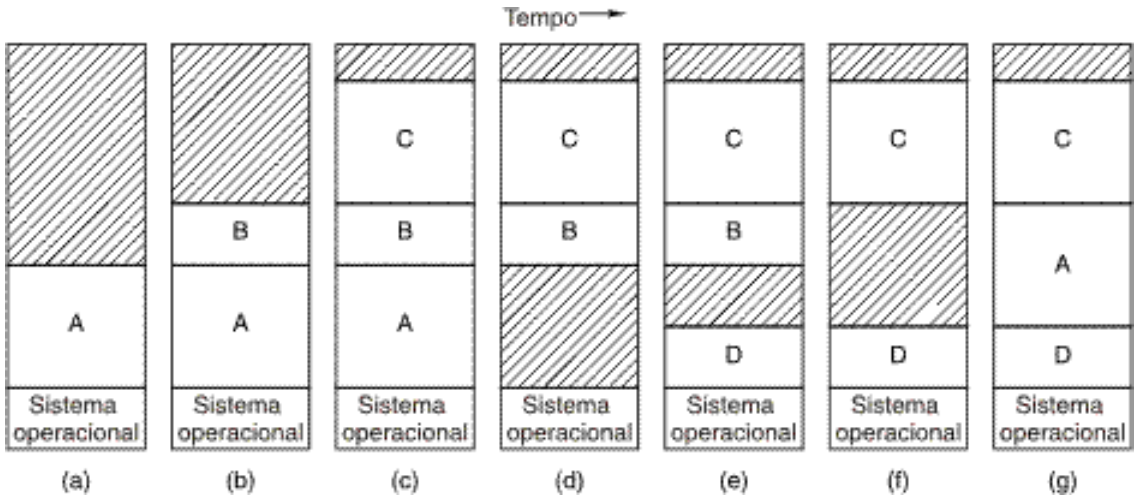
– ter que realizar uma soma e uma comparação a cada acesso à memória

Problema:

– normalmente o tamanho da memória não é suficiente para comportar todos os processos (Linux e Windows podem iniciar de 50 a 100 processos quando o computador é ligado)

⇒ ***manter todos os processos na memória exigiria uma quantidade enorme de memória***

Troca de Processos (Swapping)



- Alterações na alocação de memória à medida que processos entram e saem da memória

- Regiões sombreadas correspondem a regiões de memória não utilizadas naquele instante ⇒ compactação (gasta tempo)

Troca de Processos (Swapping)

- Processos criados com tamanho fixo ⇒ alocação é simples
  ⇒ só aloca exatamente o que é necessário
  ⇒ problema quando um processo tenta crescer.

- Existe espaço adjacente ao processo? Pode ser alocado
  ⇒ processo pode crescer naquele espaço.

- Se processo adjacente a outro
  ⇒ 1. movido para onde haja espaço suficiente para ele
  2. um ou mais processos terão de ser trocados para criar um espaço suficiente.
  3. ou.....

Troca de Processos (Swapping)

3. Se processo não pode crescer (área de *swap* no disco cheia)

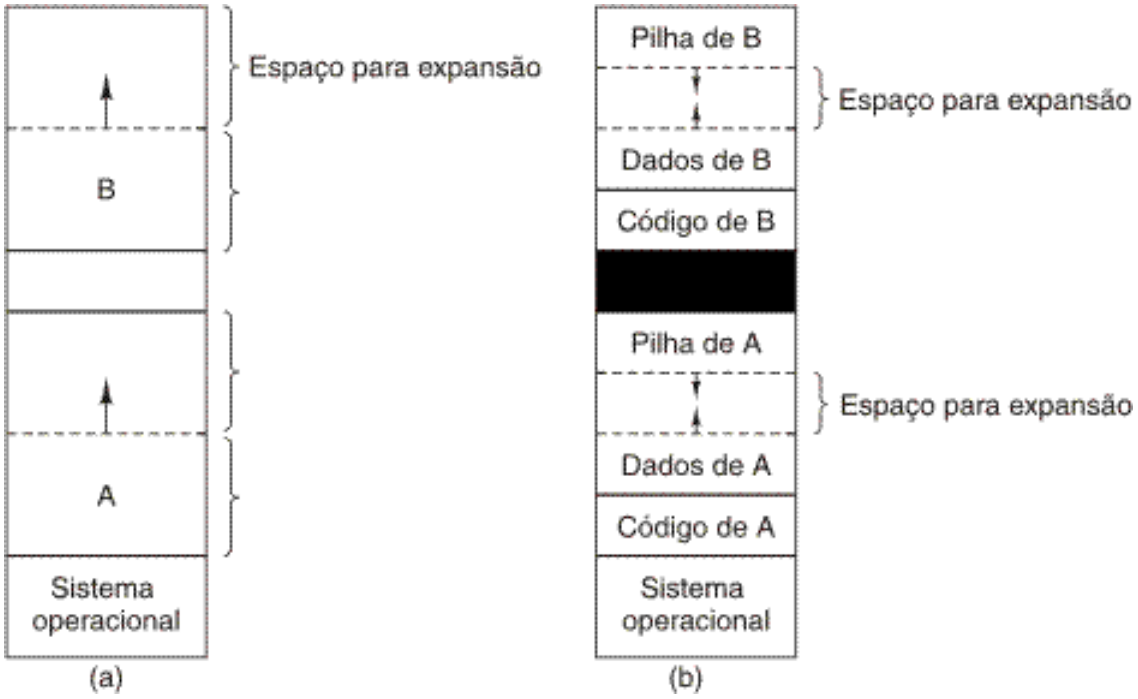
   ⇒ suspenso até que algum espaço seja liberado (ou, na pior das hipóteses, ser morto).

Se maioria cresce:

Alocar memória extra sempre que processo trocado ou movido

⇒ reduz sobrecarga associada com a troca e movimentação dos processos que não cabem mais na memória alocada.

Troca de Processos

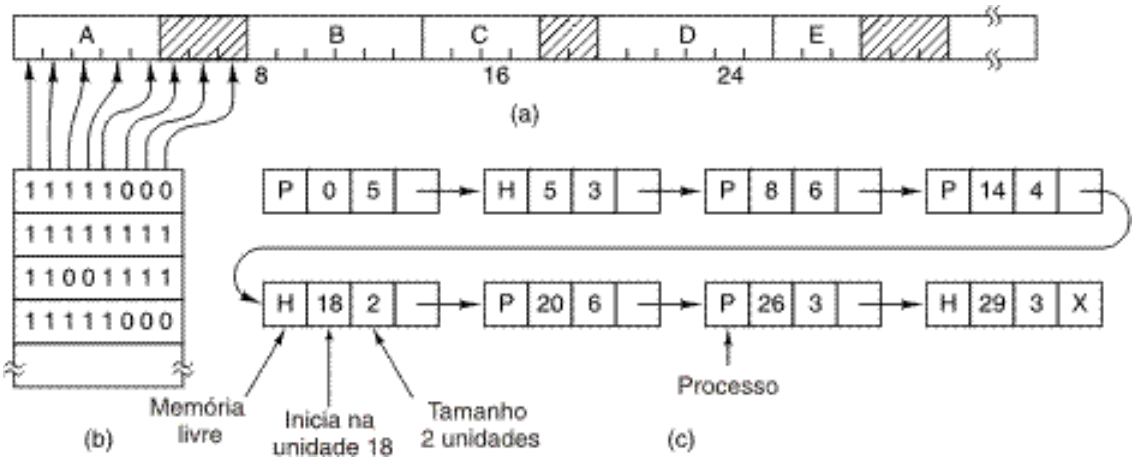


a) Alocação de espaço para uma área de dados em expansão

b) Alocação de espaço para uma pilha e uma área de dados, ambos em expansão

Obs: se espaço se esgotar, mover processo p/ outra área ou p/ disco

Gerenciamento dinâmico de memória com **mapa de bits**

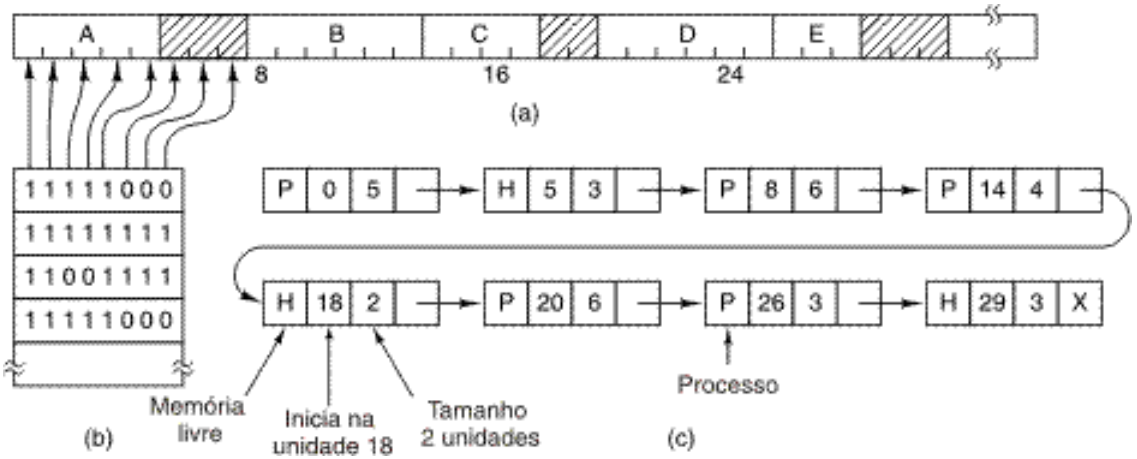


a) Ex.: parte da memória com 5 segmentos de processos e 3 segmentos de memória livre
   - memória é dividida em **unidades de alocação**
   - regiões sombreadas denotam segmentos livres

b) mapa de bits correspondente

c) mesmas informações em uma lista encadeada

Gerenciamento dinâmico de memória com **mapa de bits**



a) tamanho do mapa de bits: varia com o tamanho da memória e com o tamanho da unidade de alocação

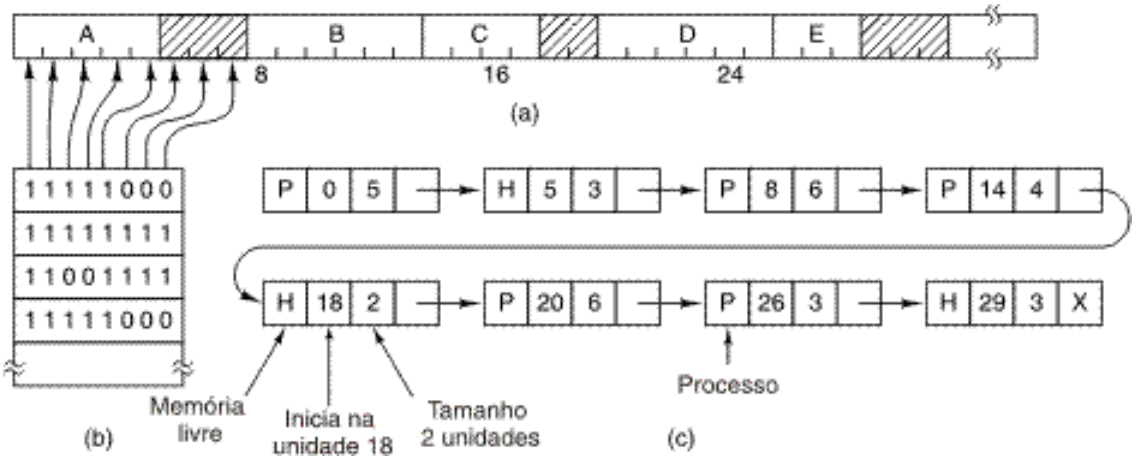
⇒ será maior quando unidade de alocação pequena, mas…

Quando a unidade de alocação é grande

⇒ existirão espaços desperdiçados

(processo não é múltiplo da unidade de alocação)

Gerenciamento dinâmico de memória com **mapa de bits**



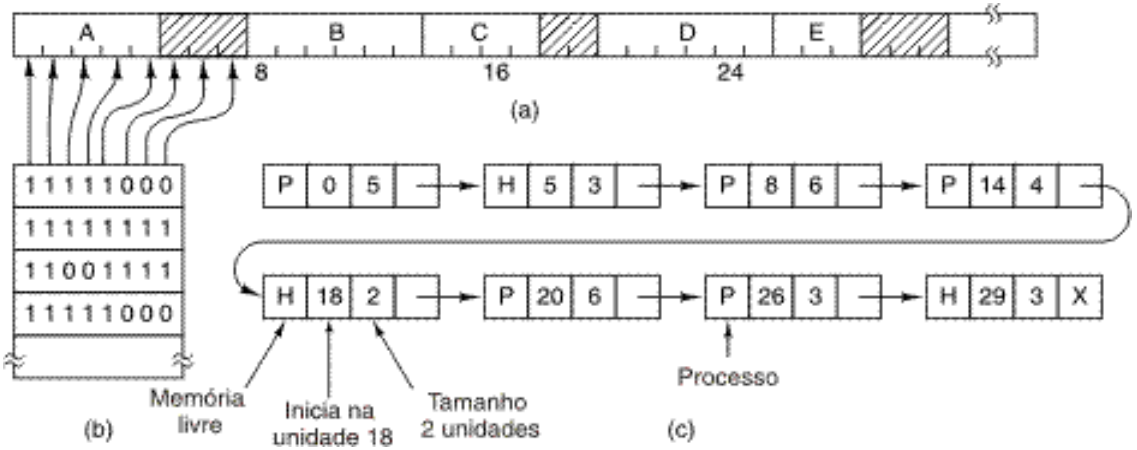
Principal problema:

Alocar espaço para um processo que ocupe k unidades de alocação

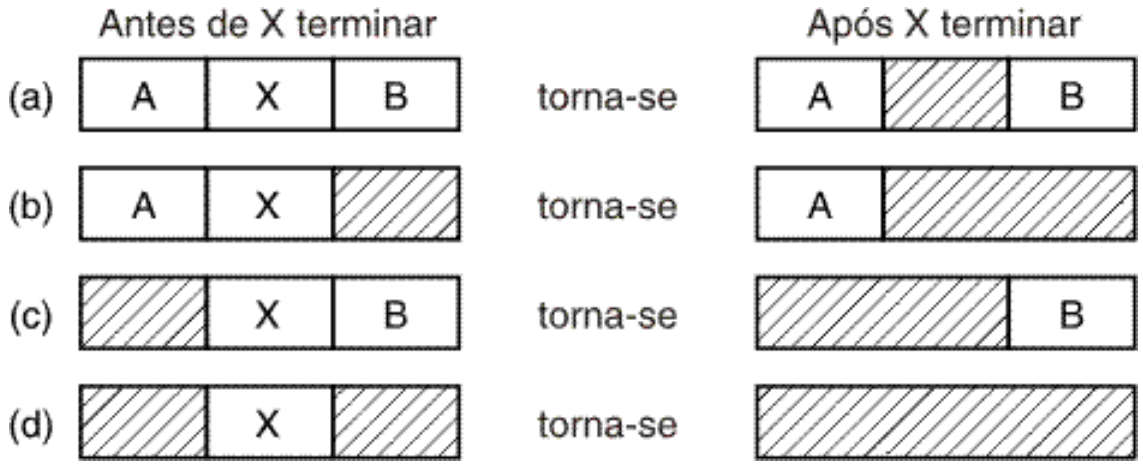
⇒ procurar um sequência de k bits “0” no mapa de bits

⇒ Processo de busca é lento!!!

Gerenciamento dinâmico de memória com **listas encadeadas**



Gerenciamento dinâmico de memória com **listas encadeadas**



Quatro combinações de vizinhança para o processo X em término de execução.

Nos itens b,c e d existe uma concatenação de segmentos

Gerenciamento dinâmico de memória com **listas encadeadas**

Algoritmos para alocação de memória c/ listas encadeadas

Algoritmo ***first fit*** – procura o primeiro segmento disponível cujo tamanho é suficiente para alocar o processo desejado

⇒ **menor busca possível**

Algoritmo ***next fit*** – semelhante ao anterior, com a diferença de que a procura é feita a partir do último segmento alocado

Algoritmo ***best fit*** – pesquisa toda a lista e escolhe o menor segmento no qual se encaixa o processo. Além de **mais lento**,

gera segmentos minúsculos inúteis

⇒ **desperdiça mais memória!**

Algoritmo ***worst fit*** – escolhe o maior segmento de memória disponível.

Ideia: segmentos que sobram serão úteis

Gerenciamento dinâmico de memória com **listas encadeadas**

Os algoritmos seriam mais rápidos na alocação se fossem mantidas duas listas separadas: ***segmentos disponíveis e segmentos alocados.***

Desvantagem: **complexidade adicional -** segmento liberado deve ser removido da lista de alocados e inserido na lista de disponíveis, provocando uma queda no desempenho na liberação de um segmento. Lista espaço livre deve ser ordenada por tamanho (***best fit*** teria um desempenho melhor, assim como o ***first fit***)

Algoritmo ***quick fit*** – mantém uma **tabela com ponteiros para várias listas**, cada uma com os tamanhos de segmentos disponíveis mais usados. Ex: listas de 4KB, 8KB, 12 KB, …

Memória Virtual / Paginação

Motivação:

- ✓ Tamanho dos programas cresce mais rápido que o o crescimento do tamanho da memória;
- ✓ Tendência de crescimento de aplicações multimídia;

  ⇒ **maior demanda de memória**
- ✓ Necessidade de executar programas maiores que a memória;
- ✓ Necessidade de executar vários programas simultaneamente (somados > memória)
- ✓ Troca de processos entre disco e memória é lenta. Ex: disco Sata 100 MB/s

Programa de 1 GB ⇒ 10 s p/ retirá-lo da memória e 10 s p/ iniciá-lo

Memória Virtual / Paginação

Abstração criada pelo Sistema Operacional:

- Cada programa tem seu espaço de endereçamento ***virtual***, que é dividido em blocos denominados ***páginas***

- Nem todas as páginas precisam estar na memória física

Paginação:

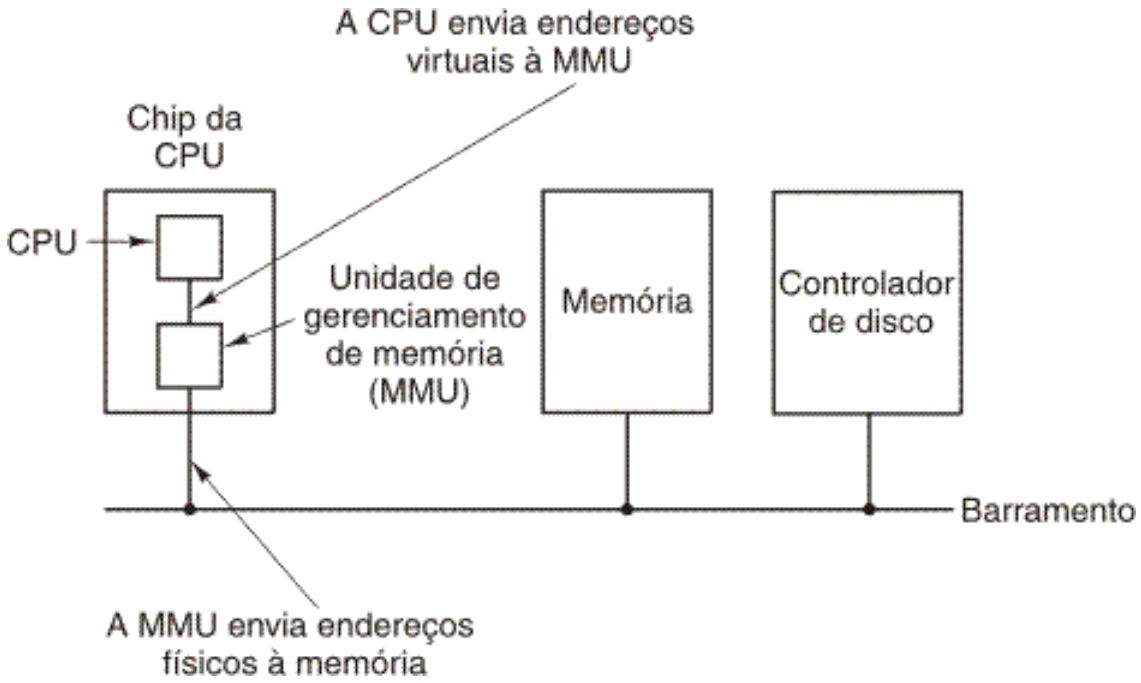
Técnica que permite o mapeamento de ***endereços virtuais*** em ***endereços físicos*** da memória

Memória Virtual / Paginação

- Funciona bem em um sistema de multiprogramação, com partes de muitos programas na memória simultaneamente.

- Enquanto um programa está esperando que partes de si mesmo sejam lidas (páginas que estão no disco), a CPU pode ser dada para outro processo.

Memória Virtual / Paginação



Localização e função da MMU:

Mapear endereços virtuais em endereços físicos da memória

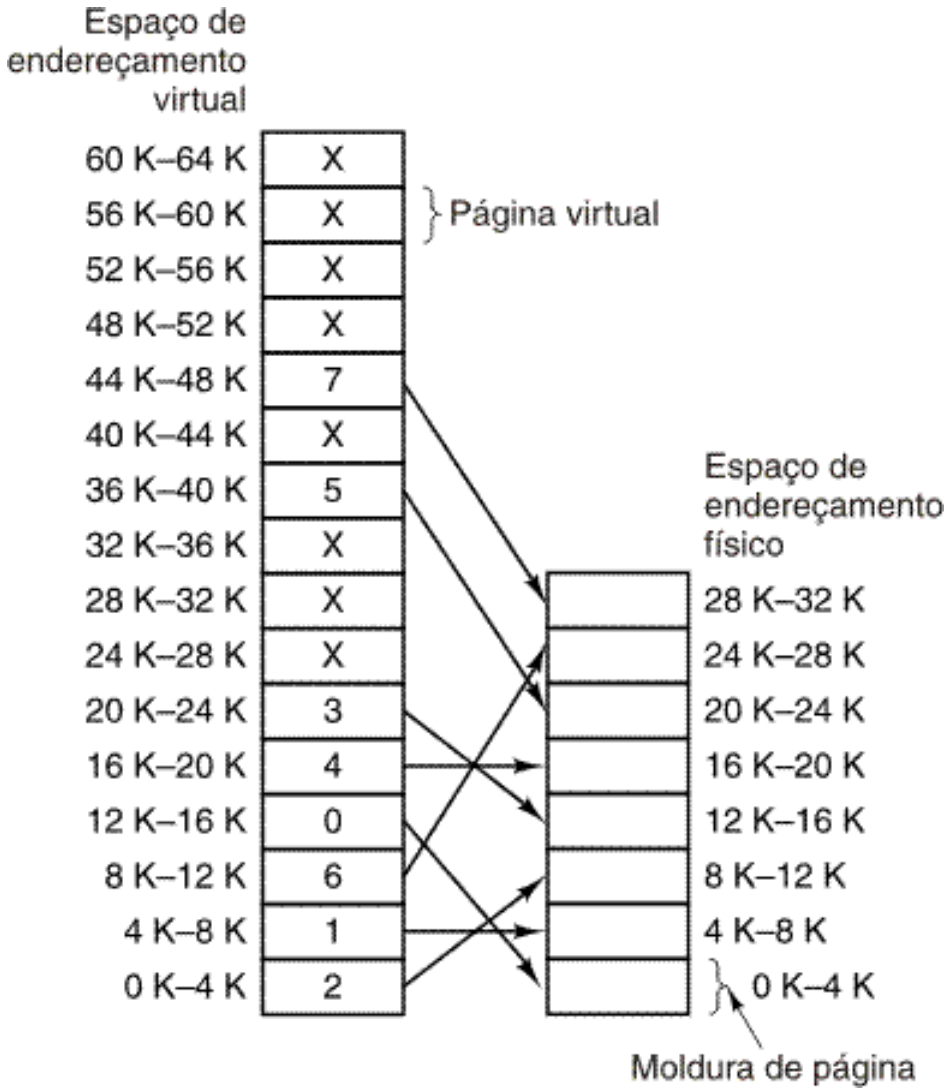
Memória Virtual / Paginação

Paginação:

Técnica usada pela maioria dos sistemas de memória virtual.

- As unidades correspondentes na memória física são chamadas de ***quadros de página*** (sempre do mesmo tamanho que a página virtual)
- Transferências entre a memória física e o disco são sempre em páginas inteiras.
- Muitos processadores dão suporte a múltiplos tamanhos de páginas que podem ser combinados e casados como o sistema operacional preferir.

Memória Virtual / Paginação



Relação entre endereços virtuais (disco) e endereços físicos da memória dada pela tabela de páginas

Memória Virtual / Paginação

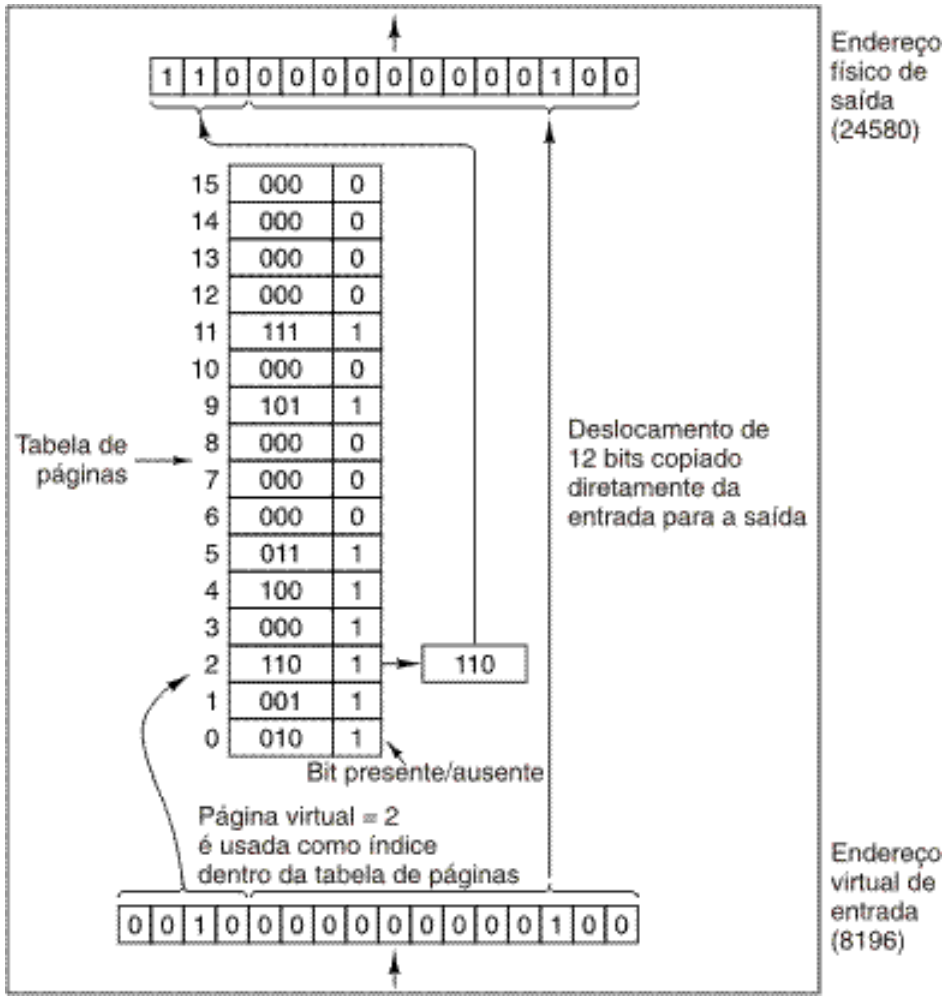
- Hardware real, um **bit Presente/Ausente** controla quais páginas estão fisicamente presentes na memória.

- A **falta de página** (*page fault*) gera uma interrupção, uma chamada ao SO.

- SO escolhe um ***quadro de página*** pouco usado e escreve seu conteúdo de volta para o disco ⇒ SO carrega (do disco) a página recém-referenciada no quadro recém-liberado, muda o mapa e reinicia a instrução que causou a interrupção.

- O número da ***página virtual*** é usado como um **índice** para a ***tabela de páginas***, resultando no **número** do ***quadro de página*** correspondente àquela página virtual.

Tabela de Páginas



Operação interna de uma MMU com 16 páginas de 4KB

Tabela de páginas

- Objetivo da tabela de páginas: mapear as páginas virtuais em quadros de páginas.

- Matematicamente:

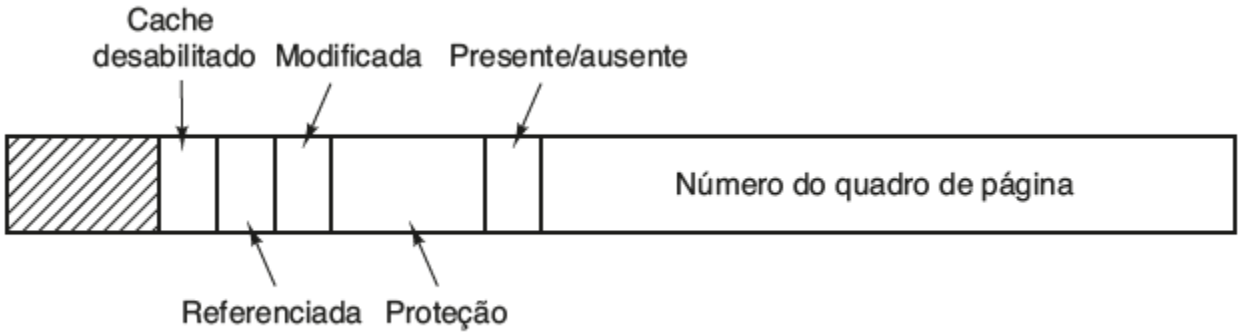
Função onde o número da página virtual é o argumento* e o *número do quadro de página é o resultado.

- Resultado dessa função:

O campo da página virtual em um endereço virtual é mapeado em um campo de quadro de página, formando um endereço de memória física.

Tabela de páginas

Estrutura de uma entrada da tabela de páginas:



- O tamanho varia de computador para computador, mas 32 bits é um tamanho comum.

- O campo mais importante é o ***número do quadro de página***.
  ⇒ Meta do mapeamento é localizar esse valor.

Acelerando a paginação

Problemas na implementação da paginação:

Maioria das instruções:

- ***Um acesso à memória (a própria instrução)***
- ***Um acesso a dados***
  - ⇒ ***2 acessos à memória, duas traduções***
  - ⇒ ***Se cada instrução demora 1 ns, cada tradução deve ser menor do que 0,2 ns para evitar que o mapeamento se torne um gargalo!!!***

Logo, mapeamento deve ser muito rápido!!!

Acelerando a paginação

Duas questões fundamentais precisam ser abordadas:

- Mapeamento rápido do endereço virtual para o endereço físico.
- Espaço do endereço virtual grande ⇒ tabela de páginas grande.

Um Projeto simples:

- Tabela de páginas consiste de uma série de registradores de hardware rápidos

  vantagem: carregada só uma vez, não necessitando de outros acessos

  desvantagem: caro se a tabela for grande!!

  Com uma entrada para cada página virtual indexada pelo número da página virtual...

  Ex: barramento de 32 bits ⇒ 2^{32} endereços

  tamanho de pág. - 4K

  ⇒ ~ 1 milhão de entradas!!

Acelerando a paginação

Problemas na implementação da paginação (cont.):

Se tabelas de páginas estão totalmente na memória principal

$\Rightarrow$ mais acessos à memória

$\Rightarrow$ ***reduz desempenho à metade, no mínimo***

Porém, a maioria dos programas faz referências a um pequeno número de páginas

TLB (Translation Lookaside Buffer)

TLB (Translation Lookaside Buffers) ou Memória Associativa

Dispositivo de hardware que mapeia endereços virtuais em endereços físicos sem ter de passar pela tabela de páginas.

Válida	Página virtual	Modificada	Proteção	Quadro de página
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

TLB (Translation Lookaside Buffers)

- Localizado dentro da MMU
- Possui de 8 a 256 entradas
- Campos têm correspondência de um para um com a tabela de páginas, exceto o número da página virtual (índice da tabela de páginas)

TLB (Translation Lookaside Buffers)

Funcionamento:

- Endereço virtual é apresentado à MMU
- Hw verifica se nº da página está na TLB (acesso paralelo a todas às entradas)
- Se correspondência encontrada e acesso não viola a proteção
  ⇒ moldura obtida
- Se acesso viola proteção ⇒ falha por violação
- Se página não se encontra na TLB ⇒ ***page miss***
  ⇒ busca na tabela de páginas (busca comum)
- MMU troca alguma entrada da TLB por essa que acabou de ser buscada

Gerenciamento da TLB pelo SO (SW)

Ausência leve: qdo a página referenciada não se encontra na TLB, mas está na memória $\Rightarrow$ MMU gera uma chamada de sistema. SO faz busca na tabela de páginas, remove uma entrada da TLB, insere a nova e reinicia a instrução que gerou a ausência (ns)

Ausência completa: qdo a página em si não está na memória (e, é claro, também não está na TLB) $\Rightarrow$ busca no disco (~ms)

Casos especiais:

- ✓ ***Falta de página menor*** **–** página não está na tabela, mas está na memória, trazida do disco por outro processo (no caso em que a memória é compartilhada)
- ✓ ***Programa acessou um endereço inválido:*** nenhum mapeamento é necessário ser acrescentado à TLB. SO mata o processo gerando uma ***falha de segmentação (segmentation fault)***

Memórias grandes: tabelas de páginas multinível

Problema:

Espaço de endereçamento virtual muito grande

$\Rightarrow$ tabela de páginas muito grande

Objetivo:

Evitar manter todas as tabelas de páginas na memória, mas somente as necessárias.

Exemplo:

Endereço virtual de 32 bits (4GB): para um tamanho de página de 4K, como vimos, tem-se um milhão de páginas (2^{20})

Divindo a tabela de páginas em dois níveis: 10 bits p/ o campo PT1, 10 bits para o campo PT2 e 12 bits para o deslocamento. (página de 4K bytes)

PT1 + PT2 $\Rightarrow$ 2^{20} páginas.

Ex: processo precisa de 12 MB: 4MB para código, 4MB para dados e 4MB para pilha (espaço enorme entre dados e pilha. 12 MB de 4GB de memória!)

Memórias grandes: tabelas de páginas multinível

Funcionamento:

1. Endereço apresentado à MMU;
2. MMU extrai o campo PT1 e o usa como índice da tabela de nível 1;
3. Cada entrada representa 4MB;
4. A entrada da tabela de nível 1 aponta para o endereço da tabela de páginas de nível 2;

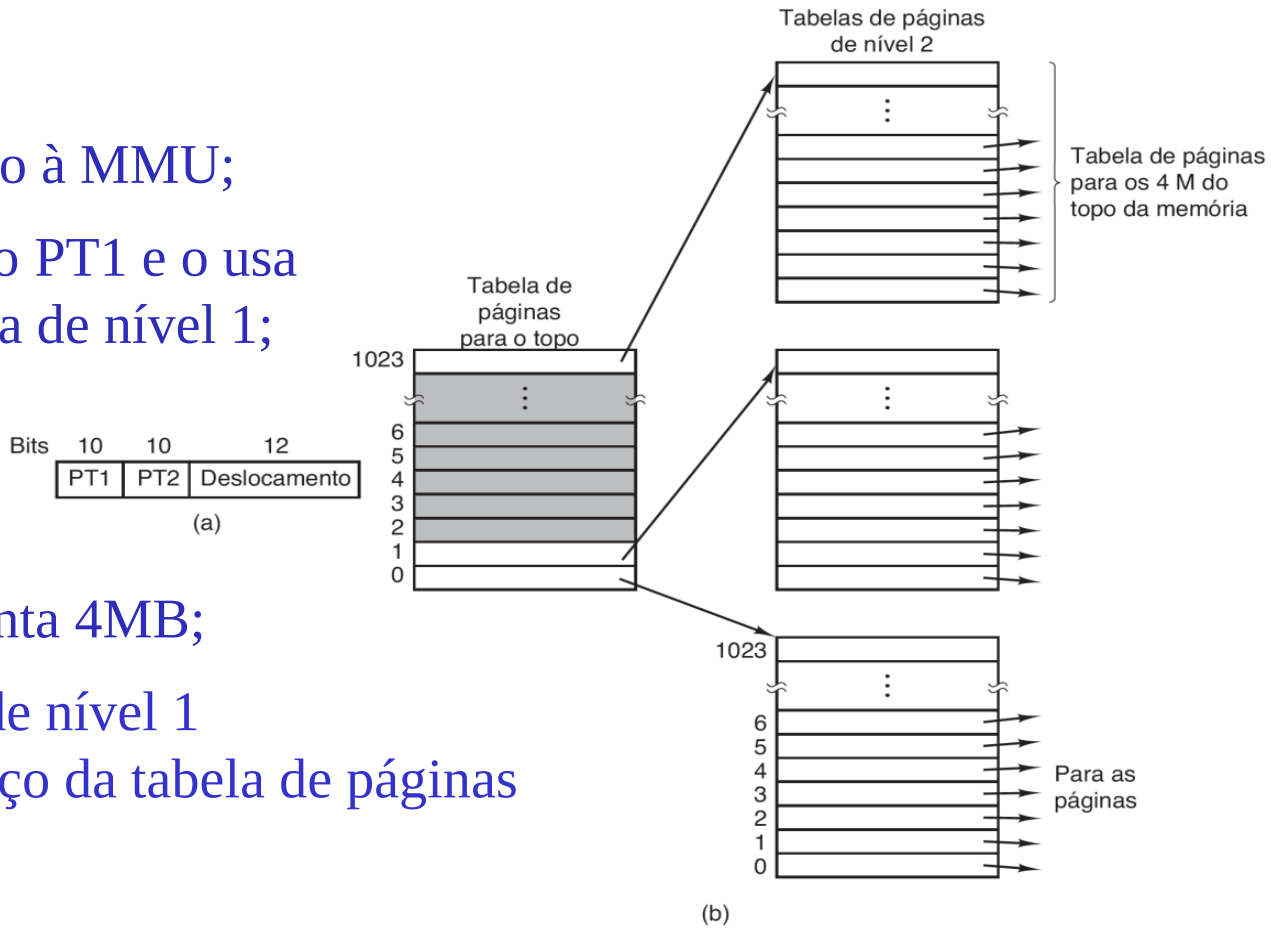


Figura 3.12 (a) Um endereço de 32 bits com dois campos de tabela de páginas. (b) Tabelas de páginas de dois níveis.

Memórias grandes:
Tabelas de páginas multinível

Funcionamento:

5. No exemplo, a entrada 0 apontará para a tabela referente ao código do programa, a entrada 1 para a tabela referente aos dados e a entrada 1023 para a tabela referente à pilha;
6. As outras tabelas não ficam na memória
7. O campo PT2 é, então, usado como índice da da tabela de nível 2 escolhida para encontrar o número do quadro de página correspondente

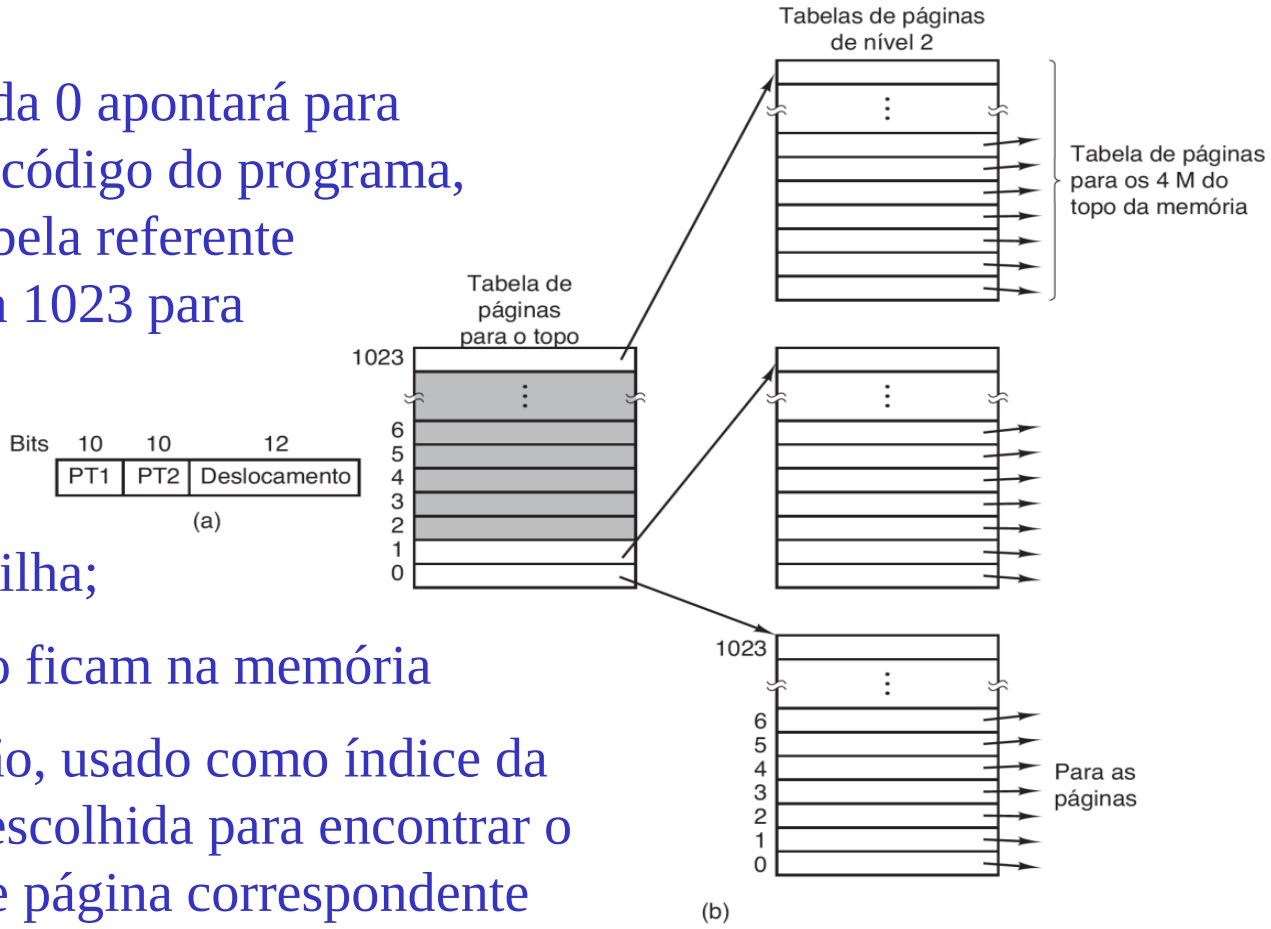


Figura 3.12 (a) Um endereço de 32 bits com dois campos de tabela de páginas. (b) Tabelas de páginas de dois níveis.

Memórias grandes:
Tabelas de páginas multinível

Obs:

1. apenas 4 tabelas são necessárias!
2. sistema pode ser expandido para mais níveis

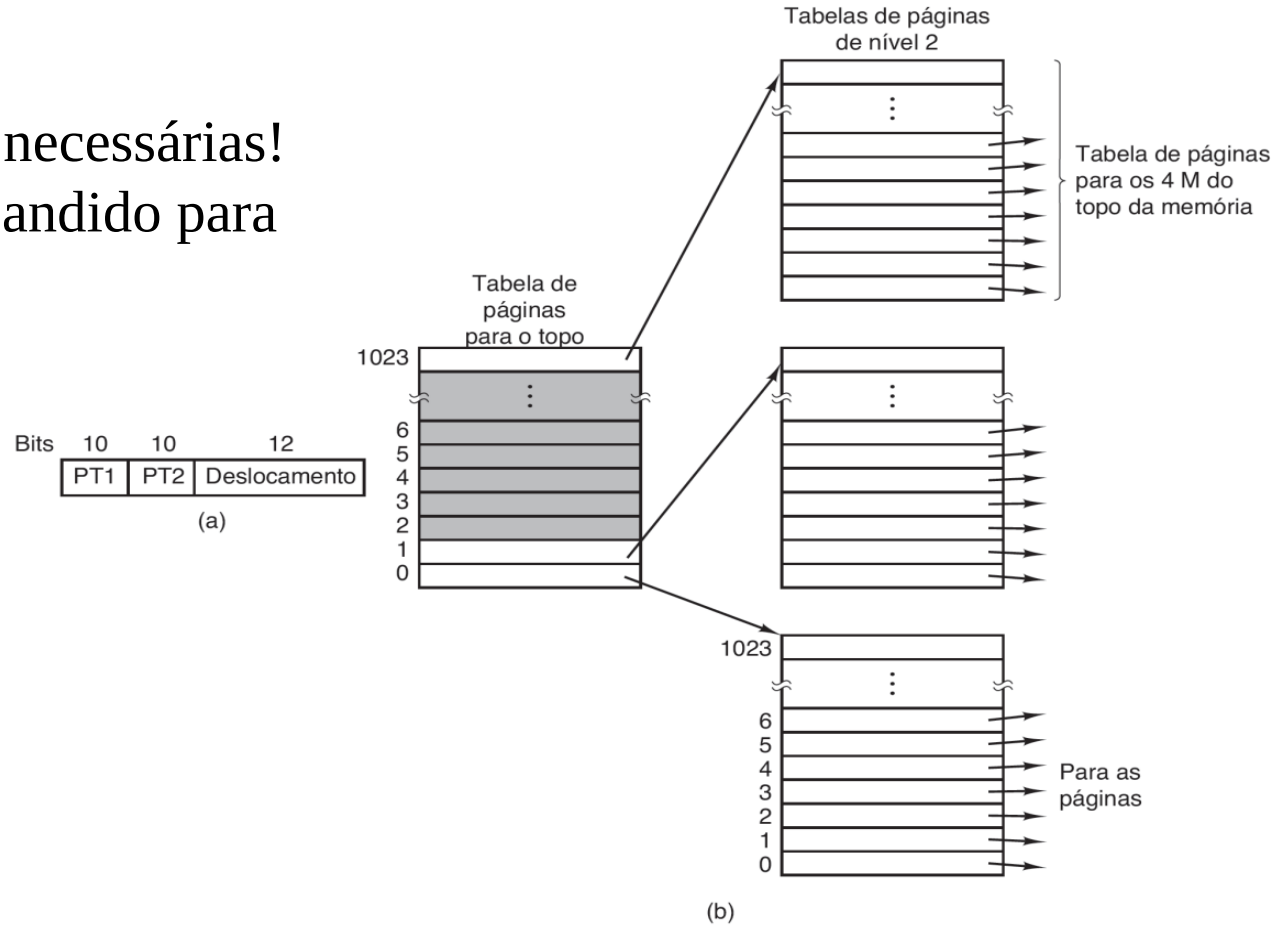


Figura 3.12 (a) Um endereço de 32 bits com dois campos de tabela de páginas. (b) Tabelas de páginas de dois níveis.