Arquitetura de Computadores

MiEI – 2016/17

DI-FCT/UNL

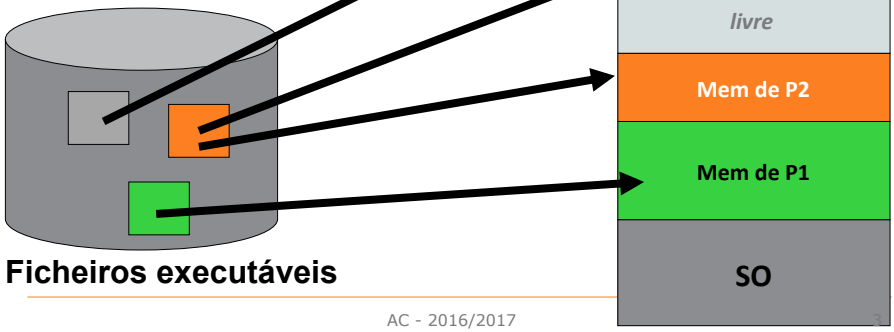
Aula 20

Conceito de processo

- Um SO executa um conjunto de programas:
  - Múltiplos serviços numa máquina; o mesmo serviço pode ter vários clientes em simultâneo
  - Os utilizadores têm vários programas carregados em memória
- **Processo** – um programa em execução; um processo executa um dado programa de forma independente de todos os outros programas em execução
  - → **SO dá a virtualização de memória, CPU e IO**

Criação dos processos

- Programas carregados a partir de ficheiros em disco para a memória central
- Necessidade de espaços de memória independentes



Processos em execução

vad — top — 80×24

```
Processes: 215 total, 2 running, 13 stuck, 200 sleeping, 805 threads   09:17:42
Load Avg: 1.09, 2.25, 4.84  CPU usage: 1.14% user, 3.43% sys, 95.42% idle
SharedLibs: 141M resident, 20M data, 16M linkedit.
MemRegions: 30438 total, 1461M resident, 78M private, 463M shared.
PhysMem: 4064M used (1018M wired), 31M unused.
VM: 585G vsize, 535M framework vsize, 2223897(0) swapins, 3347224(0) swapouts.
Networks: packets: 2160696/1245M in, 960395/121M out.
Disks: 1619469/44G read, 800243/38G written.

PID   COMMAND      %CPU TIME     #TH   #WQ  #PORT MEM    PURG   CMPRS  PGRP PPID
5876  mdworker     0.0  00:00.05 3     0    44    2988K  0B     0B     5876 1
5875  mdworker     0.0  00:00.06 3     0    53    3464K  0B     0B     5875 1
5874  Grab         4.2  00:00.49 7     4    186+  7972K+ 176K+  0B     5874 1
5873  top          8.1  00:01.09 1/1   0    24    2208K  0B     0B     5873 5860
5860  bash         0.0  00:00.01 1     0    17    800K   0B     0B     5860 5859
5859  login        0.0  00:00.02 3     1    30    1216K  0B     0B     5859 5800
5844  bash         0.0  00:00.02 1     0    17    804K   0B     0B     5844 5843
5843  login        0.0  00:00.02 3     1    30    1112K  0B     0B     5843 5800
5830  bash         0.0  00:00.02 1     0    17    832K   0B     0B     5830 5829
5829  login        0.0  00:00.03 3     1    30    1152K  0B     0B     5829 5800
5828  mdworker     0.0  00:00.06 3     0    43    3068K  0B     0B     5828 1
5825  mdworker     0.0  00:00.06 3     0    43    3024K  0B     0B     5825 1
5824  Activity Mon 0.0  00:06.98 7     4    217   19M    9944K  0B     5824 1
5823  com.apple.Sa 0.0  00:00.45 4     2    42    11M    0B     0B     5823 1
```

Também pode ser visto no *task manager* do Windows (CTRL+ALT+DEL)

Espaço de endereçamento de um processo

- Para ser executado, um programa tem de ser trazido para memória central – carregamento do ficheiro executável
- Esta imagem em memória é a "Imagem do Processo":
  - o conteúdo da memória inclui o código, dados vindos do ficheiro executável
  - Também inclui dados e pilha criados durante a execução
- A imagem é constituída por um conjunto de endereços "privados"
  - Todos os endereços do programa são **internos** à sua imagem

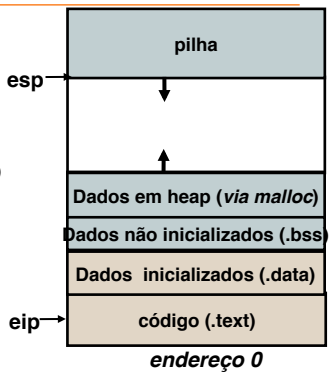


Imagem de memória de um processo Linux/x86 (simplificado)

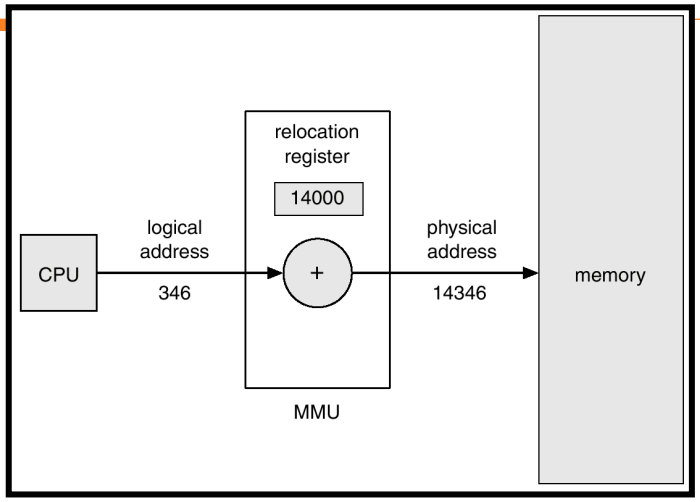
O SO assegura uma máquina virtual para cada processo

- Para um processo é como se ocupasse a máquina sozinho
- O estado da computação é preservado nas trocas de contexto: CPU, memória e IO
- As interações com periféricos e ficheiros são realizadas pelo SO
- Um processo só consegue escrever na memória que lhe está reservada. Mais nenhum processo consegue aceder a essa memória.

Problemas

- O programa tem de ser carregado sempre na mesma posição de memória?
  - Como podemos ter mais de um programa na memória?
  - Se o ficheiro executável for carregado em diferentes endereços, os endereços usados no programa têm de ser alterados?
- Não podemos ter programas (código+dados +pilha) maiores que a memória instalada?

1ªsolução: Registo de recolocação

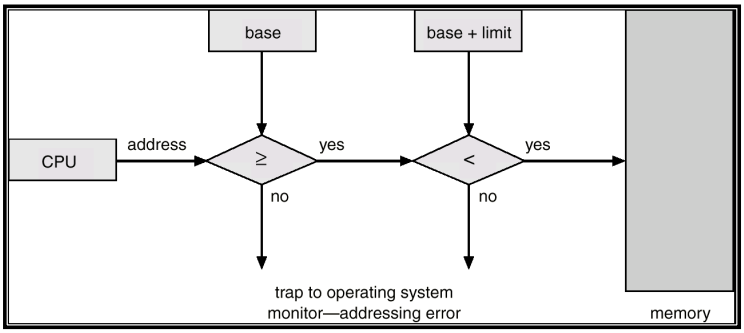


Esta técnica permite carregar o programa em qualquer posição da memória física

Protecção de Memória - Exemplo

- Definição da zona de memória de um processo:
  - **Registo Base** e **Registo Limite** para cada processo
  - Estes fazem parte duma unidade de transformação de endereços (**MMU**-*Memory Management Unit*)
  - Os endereços efetivos do programa são **virtuais** e ajustados pelo registo base para obter o **endereço real**
  - Determinam a faixa legal de endereços
  - A memória fora dessa zona é proibida ao programa.
  - Estes são guardados/repostos nas trocas de contexto
- O SO, sabendo as zonas de memória livres, define estes registos para cada processo

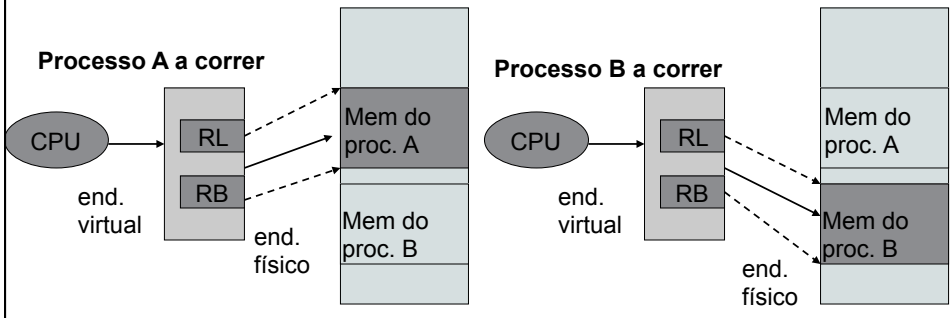
Protecção de memória



- As instruções para alterar os registos *base* e *limite* são privilegiadas.
  - Só o SO as pode usar

Troca de segmento de memória física

- Suporte *hardware*: registos base e limite
  - O valor corrente dos registos base (RB) e limite (RL) são específicos de cada processo
  - O S.O. gere estes valores, usando instruções privilegiadas



Espaço físico gerido por afetação de segmentos contíguos

- Cada executável indica a necessidade máxima de memória
- A imagem de um processo é criada num segmento de memória de dimensão igual ou superior às necessidades do processo
- Suporte *hardware*: registos base e limite
  - Asseguram a recolocação do programa e a protecção entre processos;
  - Definidos pelo SO para cada processo criado
- Transformação dos endereços:
  - *End.físico = End.Virt + End.Base*
  - *Se dentro do limite válido*

Espaços de endereços lógicos (ou virtuais) e físicos (ou reais)

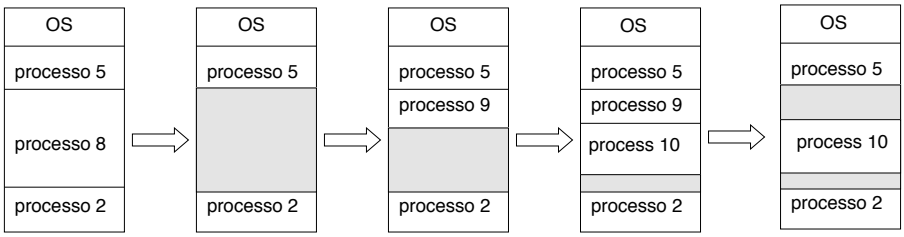
- Cada processo tem um espaço de endereçamento lógico que é separado do dos outros processos
- O conceito de um espaço de endereçamento lógico é independente do espaço de endereços físico:
  - Endereços lógicos – usados no programa (definidos pelo programador, compilador e *linker*); na execução são os vistos pelo CPU; também conhecidos por endereços virtuais.
  - Endereços físicos – obtidos a partir dos endereços lógicos e recebidos pela cache e pela memória física

Espaços de endereços lógicos (ou virtuais) e físicos (ou reais)

- Existe uma Unidade de Transformação de Endereços ou MMU
- Os endereços virtuais e físicos diferem, claro!
  - E os físicos dependem de onde o processo é colocado em cada execução
- Em cada momento, estão carregados em memória imagens (código+dados+pilha) de vários processos
  - Torna independentes as organizações dos dois espaços
  - o SO gere a memória física disponível em cada instante

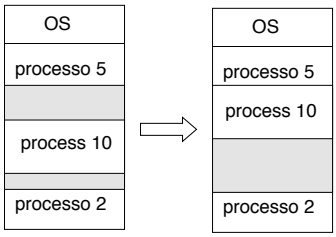
Inconvenientes da atribuição de memória por segmentos contíguos

- Zonas livres de diferentes dimensões estão espalhadas pela memória física.
- Quando um processo é criado, é necessário atribuir-lhe uma zona livre contígua suficientemente grande
- As zonas de memória livre podem-se encontrar dispersas pela memória → **Fragmentação**



Combatendo a fragmentação

- A fragmentação pode ser eliminada juntando a memória ocupada
  - demorado: exige movimentar os vários segmentos em memória
- Todos os processos usam segmentos do mesmo tamanho
  - um processo pode necessitar de mais espaço (ou vários segmentos)
  - ou então desperdiçar espaço → **fragmentação interna**



2ªsolução: Paginação

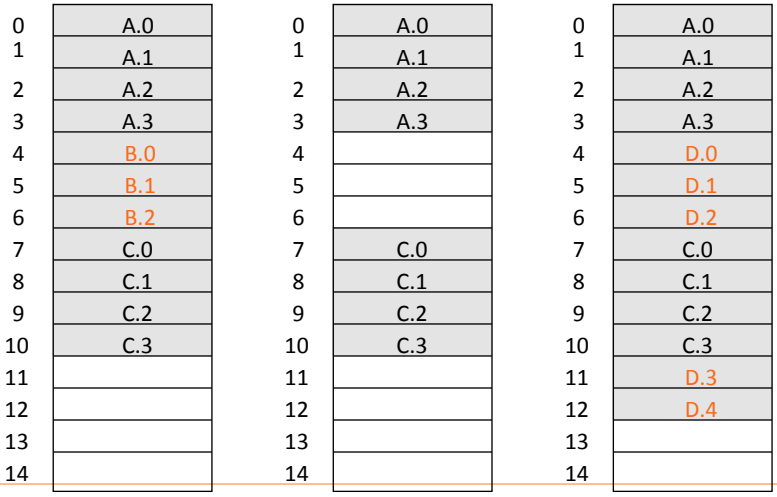
- Divide-se a memória física em pequenos segmentos de tamanho fixo chamados de **páginas físicas** (ou **frames**)
  - o tamanho é uma potência de 2 entre 0,5Kbytes e 8Kbytes.
- Divide-se o espaço de endereçamento lógico em blocos do mesmo tamanho, chamados de **páginas lógicas** (ou páginas virtuais).
- Cada processo pode ocupar várias páginas, para acomodar imagens maiores que uma só página
- Para um processo executar tem de existir uma **tabela** que relaciona páginas virtuais com os frames físicos
  - Esta está na unidade de transformação de endereços

Carregando os processos A e B em memória

Número da página física (frame)					
0		0	A.0	0	A.0
1		1	A.1	1	A.1
2		2	A.2	2	A.2
3		3	A.3	3	A.3
4		4		4	B.0
5		5		5	B.1
6		6		6	B.2
7		7		7	
8		8		8	
9		9		9	
10		10		10	
11		11		11	
12		12		12	
13		13		13	
14		14		14	

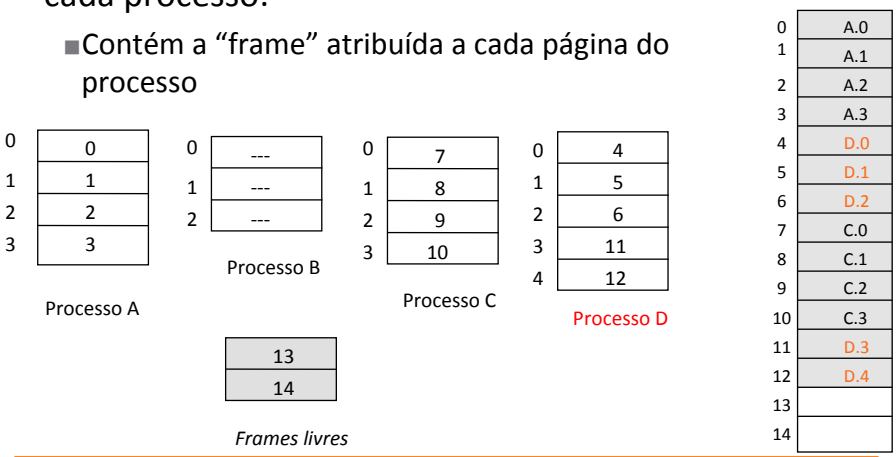
Paginação: atribuição não contígua

- o processo B termina e inicia-se o processo D:



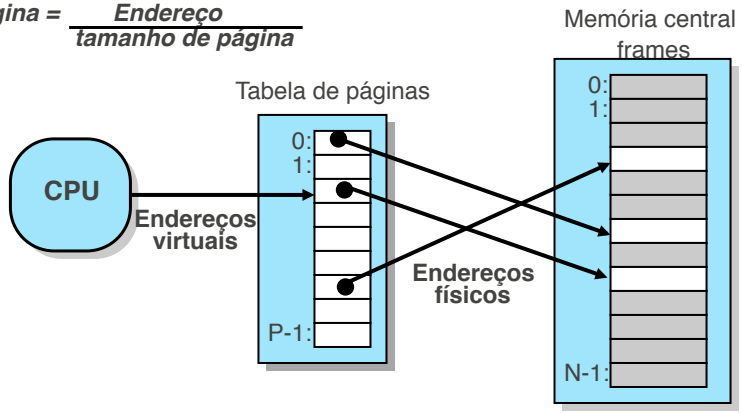
Tabelas de páginas para o exemplo

- O OS mantém uma tabela de páginas para cada processo:
  - Contém a "frame" atribuída a cada página do processo



Transformação de endereços (1)

$$\textit{Num página} = \frac{\textit{Endereço}}{\textit{tamanho de página}}$$

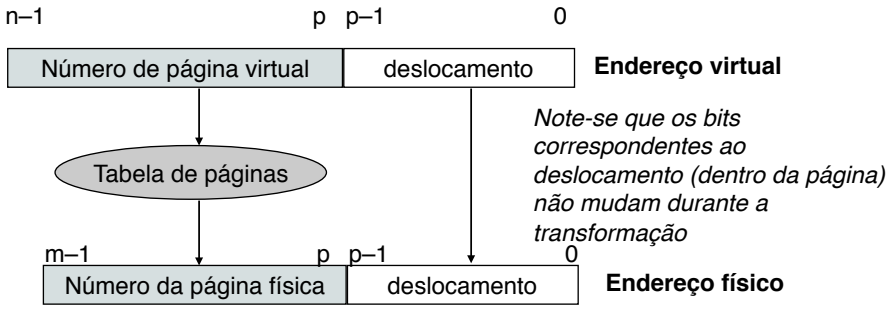


Transformação de endereços: Unid. Transf. (MMU – *Memory Management Unit*) converte páginas em frames físicos através da tabela de páginas (gerida pelo SO)

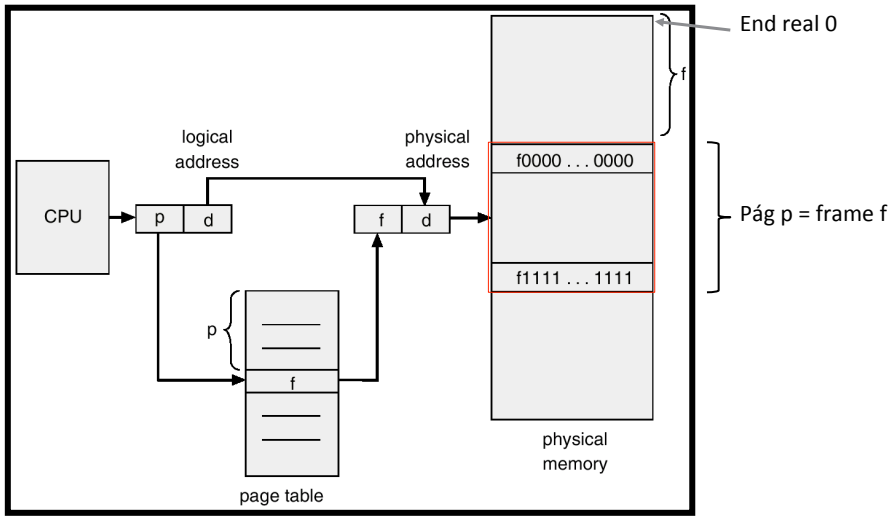
Transformação de endereços (2)

$$\textit{Num página} = \frac{\textit{Endereço}}{\textit{tamanho de página}}$$

- Parâmetros
  - P = 2^p = tamanho da página (usa p bits).
  - N = 2^n = espaço virtual (endereço tem n bits)
  - M = 2^m = espaço físico (endereço tem m bits)



Transformação de endereços (3)



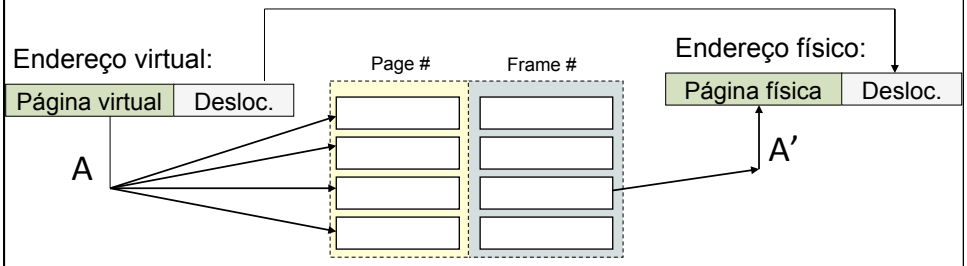
Suporte da tabela de páginas

- Onde se guarda a tabela de páginas de um processo?
- No caso do Pentium, end. 32bits e páginas de 4Kbytes:
  - tabela tem 2^{20} entradas (2^{32} /2^{12}); se cada uma tiver ~3 bytes, são ~3 Mbytes!
  - E há uma tabela de páginas por processo …
  - Não é tecnicamente viável guardar a tabela de páginas na MMU
- Solução:
  - Guardar a tabela de páginas em memória

Suporte da tabela de páginas

- Neste esquema cada acesso a dados ou código obriga a **dois acessos à memória**:
  - um para ler a tabela de páginas obtendo o endereço físico
  - e outro para ler/escrever o dado ou ler a instrução.
- O problema dos dois acessos à memória pode ser mitigado com uma **cache** da tabela de páginas:
  - ***Translation Look-aside Buffer*** (TLB)
  - *hardware* especial para consulta rápida, baseia em memória associativa

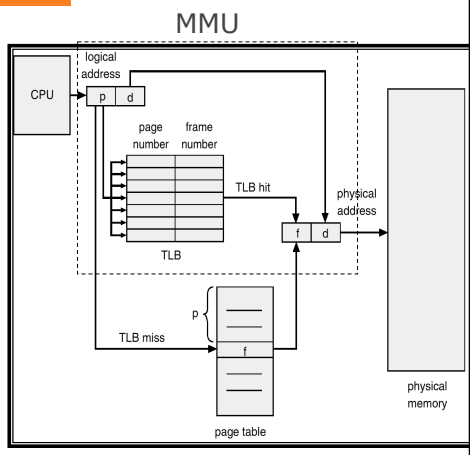
TLB – memória associativa para frames



- Transformação de endereços:
  - Relaciona números de páginas com números de frames
  - Se A está na memória associativa, TLB responde com o número da "frame" A'
  - Senão, temos um TLB miss…

MMU com TLB

- Transformação de endereços: A→ A'
  - Se A está na TLB responde logo com o número da "frame" A'
  - Se não está, obtém o número da "frame" da tabela de páginas na memória e atualiza a TLB
    - Podendo ter de eleger uma vítima na TLB para dar lugar à nova página



Tempo de acesso efectivo

- Se tempo de acesso à RAM é T unid. de tempo
- Se *TLB Hit rate* = α (percentagem das vezes em que o número da *frame* está num registo do TLB)
  - E se acesso à TLB desprezável
- Tempo de acesso efetivo (TAE)

  TAE = α T + (1 – α) 2T

  = (2 – α)T

Exemplo: para α = 90% e T=10ns

TAE = 11ns

Neste exemplo os endereços virtuais e sua transformação está a custar 1ns por acesso

Protecção de memória com páginas

- A protecção de memória está associada ao preenchimento que o SO faz da tabela de páginas de cada processo
- A MMU tem de referenciar a tabela do processo em execução (controlado pelo SO)
- Mais informação pode estar associada à tabela ou a cada página. Exemplos:
  - Pode limitar o espaço de endereçamento limitando o tamanho da tabela de páginas
  - Pode haver entradas interditas ao processo → não mapeadas na memória real
  - Cada página pode ter associada informação descrevendo os acessos permitidos (por exemplo: só leitura, pode executar)

Usando menos memória central

- Num programa há partes que só são usadas em alguns momentos
  - Exemplos: no início, no fim, em casos de erro, ...
- Durante a execução só partes da imagem do processo é que estão realmente em uso (*working set*)
- Poderá o SO trazer para memória só as partes realmente necessárias a cada processo?
- Pode o espaço de endereçamento de um processo ser maior que a memória real de um computador?
  - As zonas que num dado momento não são necessárias estão em disco e não em memória. **A memória funciona como cache da imagem em disco.**

Páginas da imagem de um processo

- Nem todo o espaço de endereços necessita de ter logo atribuído memória física.
- Valid-invalid bit (bit V) associado a cada entrada da tabela de páginas:
  - "válido" indica que a página está no espaço de endereçamento lógico do processo – é uma página válida
  - "inválido" indica que a página não está no espaço de endereçamento virtual do processo.

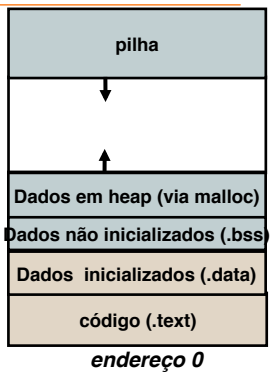
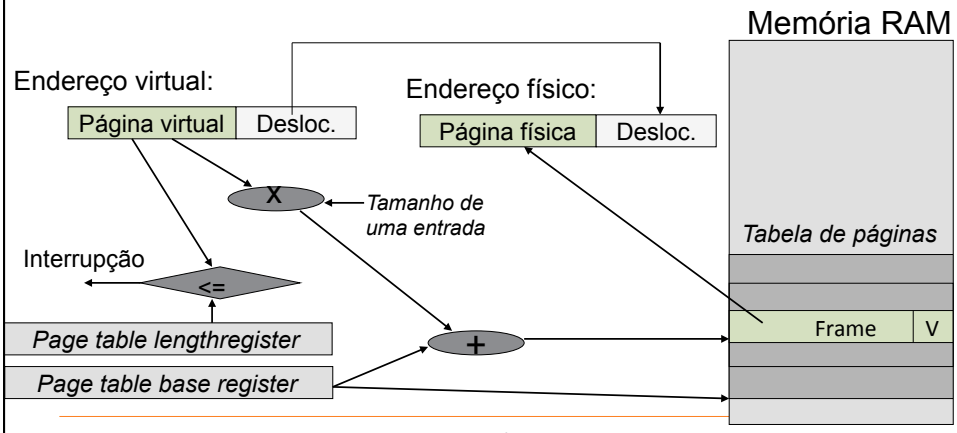


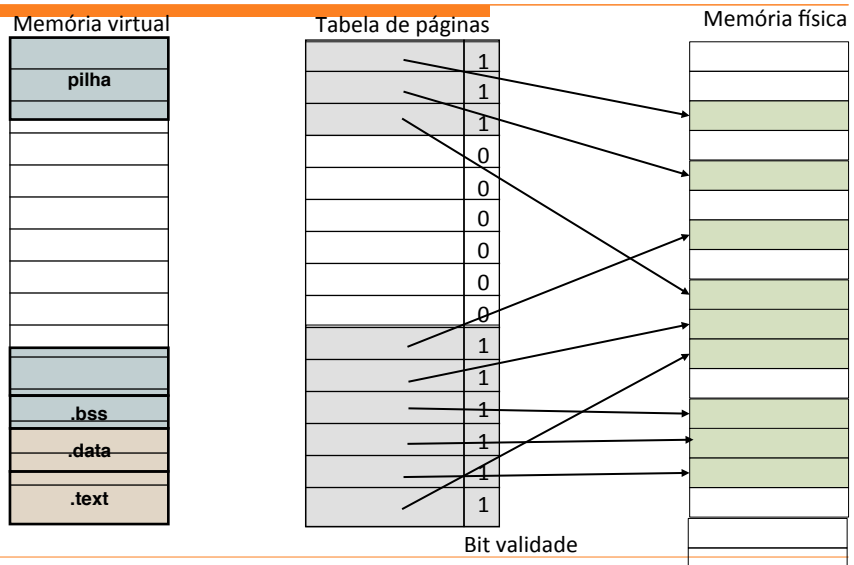
Imagem de memória de um processo Linux/x86 (simplificado)

Tabela de páginas com de bit validade

- Para cada processo:
  - *Page-table base register* (PTBR) aponta para a base da tabela.
  - *Page-table length register* (PRLR) indica o tamanho da tabela de páginas.

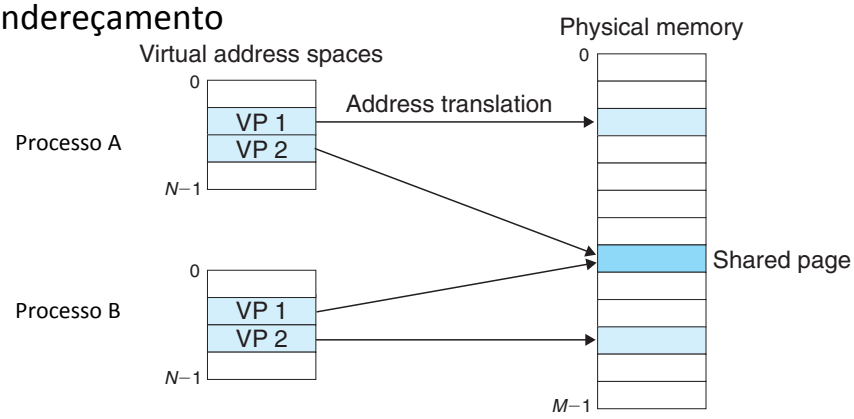


Páginas da imagem de um processo



Partilha de frames entre processos

- As páginas fisicas (os endereços reais) podem ter vários endereços virtuais, dependendo do espaço de endereçamento



Partilha de páginas

- Pode ser vantajosa a partilha de páginas:
  - Os processos incluem partes de código comum (bibliotecas, ou podem ser o mesmo programa)
  - Pode servir de mecanismo de comunicação entre processos
- Para manter a proteção, há que controlar o tipo de acesso autorizado:
  - Se código: pode ser lido e executado (o fetch é possível)
  - Dados para lêr: só pode ser lido e não escrito
  - Dados para lêr e escrever: pode ser lido e escrito

Mais informação para cada página

- A entrada *p* da tabela de páginas contém para a página *p* a seguinte informação:
  - Bit de validade V (página pertence ou não ao espaço de endereçamento do processo)
  - Número da frame atribuída F (se V==1)
  - Bits que definem as possibilidades de acesso: READ-ONLY, WRITE, EXECUTE
    - Páginas de código têm EXECUTE e READ-ONLY
    - Páginas de dados têm READ/WRITE
    - ...
  - Outros bits usados para gestão

Memória virtual com paginação

- Memória virtual (VM) – separação da memória lógica (virtual) da memória física
  - MV definida pelo espaço de endereços lógicos (ou virtuais)
  - Só parte da imagem do processo precisa de estar em memória
  - É possível ter memória real < soma das imagens de todos os processos
  - Se espaço de endereços virtuais > memória real num computador, um único processo pode "usar" uma memória virtual > memória real
- É completamente transparente para programador/ programa
  - O SO fica responsável por oferecer a visão de memória virtual
  - Usa o disco para “estender” a memória real, gerindo a memória real como uma **cache** para todas as imagens dos processos