FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Fundamentos de Sistemas de Operação

LEI - 2023/2024

Vitor Duarte
Mª. Cecília Gomes

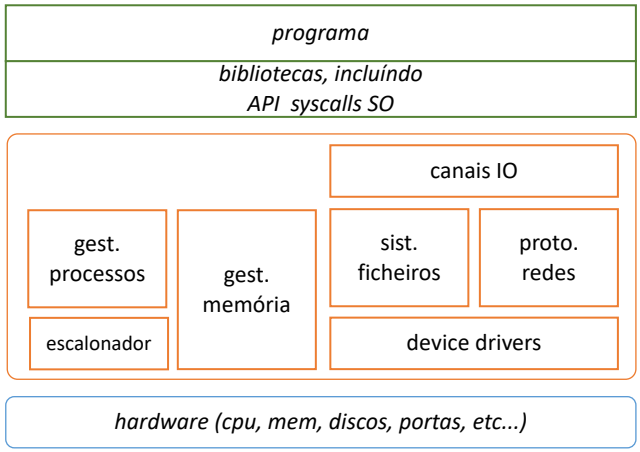
1

Aula 11

- Introdução à gestão de memória
- OSTEP: cap. 13, revisões de AC: cap. 15, 16, 18, 19

2

Principais componentes dos sistemas



a realidade é bem mais complicada!

Atributos de um processo no SO

- Descritor de processo ou *Process Control Block* (PCB) ou *Task Structure:*
  - Identificador (*process identifier - PID*)
  - Identificador do utilizador (UID)
  - **Localização da imagem do programa em memória**
    - **Código, dados e pilha (e.g. tabela de páginas)**
  - Estado do CPU (quando não executa)
    - Conteúdo dos registos
  - Estado do I/O
    - Tabela de descritores canais de I/O: ficheiros abertos e posição em que vai a leitura / escrita
  - Estado de execução (exemplo: READY, RUNNING, BLOCKED, ZOMBIE, ...)
- O gestor de processos gere um conjunto destes PCB (por exemplo num vetor) com todos os processos existentes

Espaço de endereçamento virtual

- Cada processo tem um espaço de endereços virtual
  - Todos os endereços possíveis estão nesse espaço
  - inclui o código, dados, heap e pilha
  - A imagem é "privada"
- O CPU executa com endereços virtuais
  - Uma unidade hardware (MMU) transforma cada endereço virtual em físicos
- Garante o isolamento entre processos
  - os endereços físicos têm de estar no espaço atribuído pelo SO

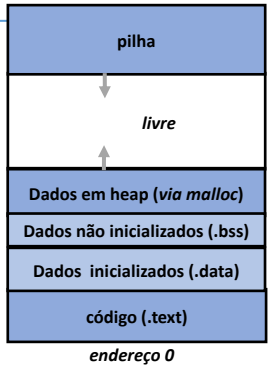
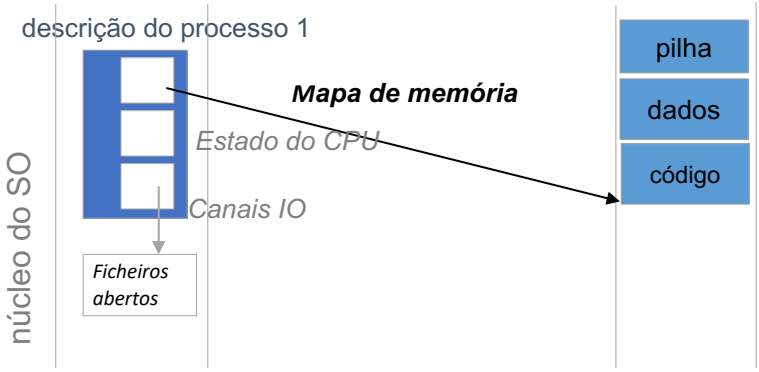


Imagem de memória de um processo Linux/x86 (simplificado)

Processo e sua memória (simplificado)

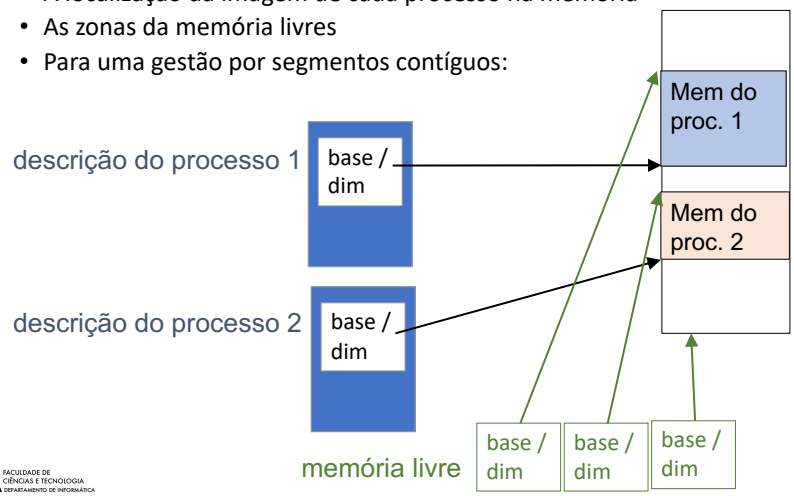


Se gestão por segmentos contíguos:

- A imagem do processo está numa zona contígua da memória física
- Esta pode ser descrita por endereço base e dimensão
- A transformação de endereços virtuais em reais é trivial:
  - End.Base + End.Virtual = End.Físico

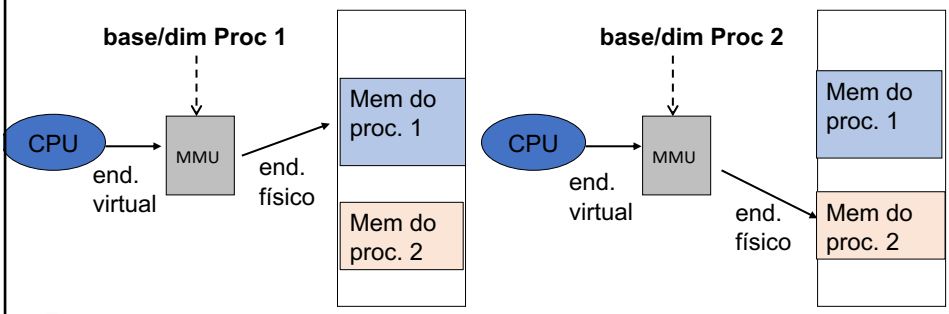
Virtualização de memória

- O kernel do SO mantém:
  - A localização da imagem de cada processo na memória
  - As zonas da memória livres
  - Para uma gestão por segmentos contíguos:



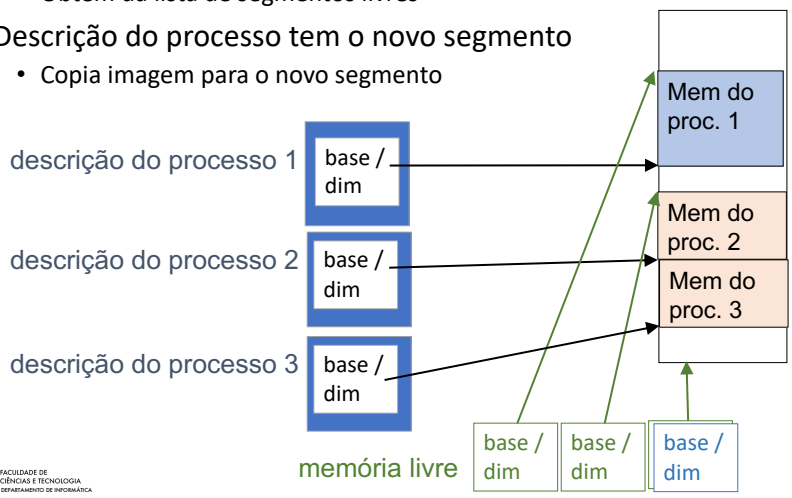
Troca de processos

- Suporte hardware: transformação de endereços
  - O CPU executa com endereços virtuais
  - O S.O. gere a memória e define a transformação necessária para cada processo
  - Só o SO (modo supervisor) pode alterar a MMU
- A troca de contexto de processo inclui trocar a configuração na MMU



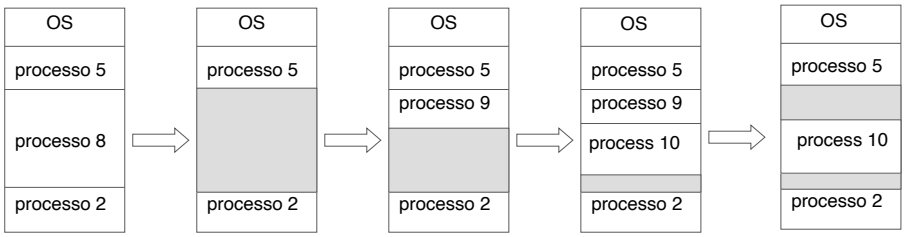
Novo processo (fork)

- O kernel do SO atribui memória livre ao novo processo
  - Obtém da lista de segmentos livres
- Descrição do processo tem o novo segmento
  - Copia imagem para o novo segmento



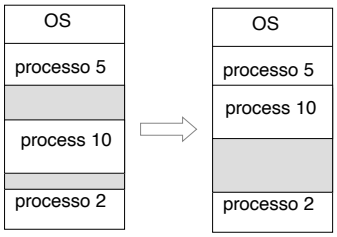
Gestão de memória por segmentos contíguos

- Zonas livres de diferentes dimensões estão espalhadas pela memória física → SO mantém uma lista de zonas livres
- O Espaço do processo definido por endereço base e dimensão
  - Quando um processo é criado, é necessário atribuir-lhe uma zona livre contígua suficientemente grande (como escolher?)
- Com a continuação as zonas livres tendem a ser pequenas e a ficar dispersas pela memória → Fragmentação

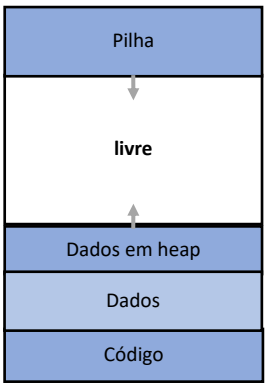


Combatendo a fragmentação?

- A fragmentação pode ser eliminada juntando a memória ocupada, passando a um só bloco livre
  - demorado: exige movimentar os vários segmentos em memória



Espaço livre em cada processo

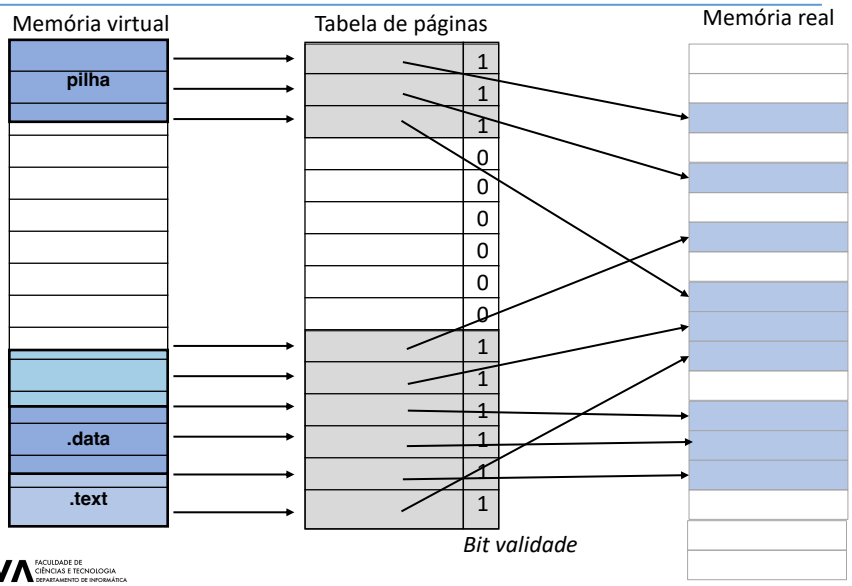


- O espaço livre já faz parte do segmento atribuído ao processo
- heap e pilha usam esse espaço
- No conjunto de todos os processos, muito espaço pode estar livre e nunca ser necessário → fragmentação interna

Eliminando fragmentação: páginas

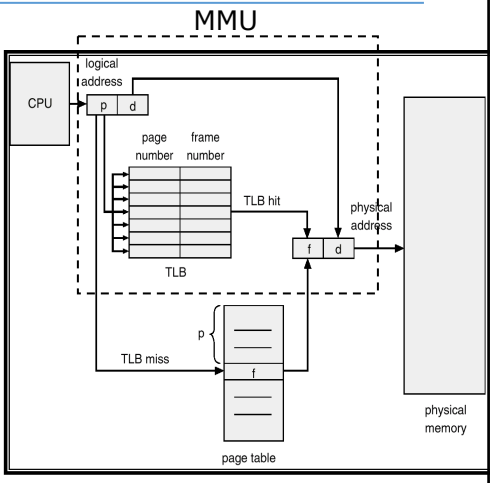
- Divide-se a memória física em pequenos segmentos de tamanho fixo chamados de páginas físicas (ou frames)
  - o tamanho é uma potência de 2 entre 0,5Kbytes e 8Kbytes.
- Divide-se o espaço de endereçamento lógico em blocos do mesmo tamanho, chamados de páginas lógicas (ou páginas virtuais).
- A transformação de endereços é conseguida usando uma tabela que relaciona a base de cada página virtual com o respetiva página física
  - Esta tabela é usada pela unidade de transformação de endereços (MMU)

Páginas da imagem de um processo

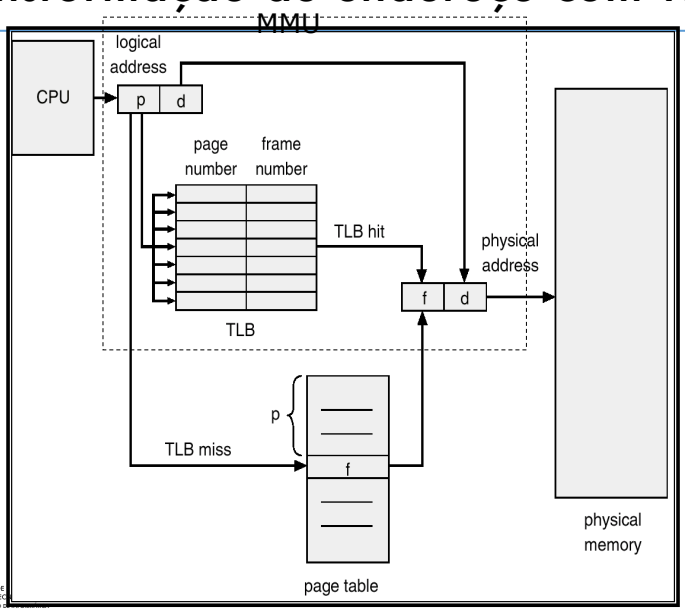


TLB: cache de tabela de páginas

- Transformação de página 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

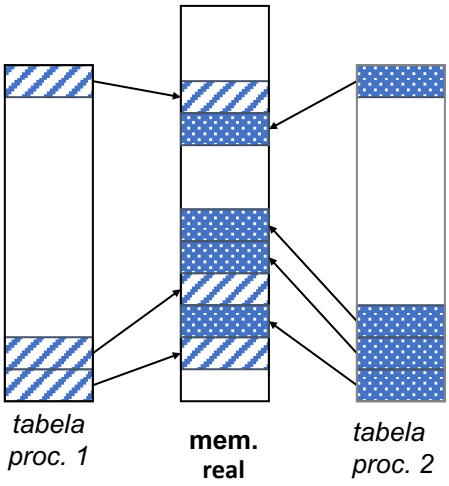


Transformação de endereço com TLB



Gestão de memória em páginas

- Cada processo tem a sua tabela de páginas apontando frames privadas
- Descrição do processo indica a tabela de páginas
- O SO mantém uma tabela de frames livres
- Atribuir memória consiste em escolher frames livres e colocá-las na tabela do processo respectivo
- Na troca de contexto troca a referencia para a tabela activa na MMU

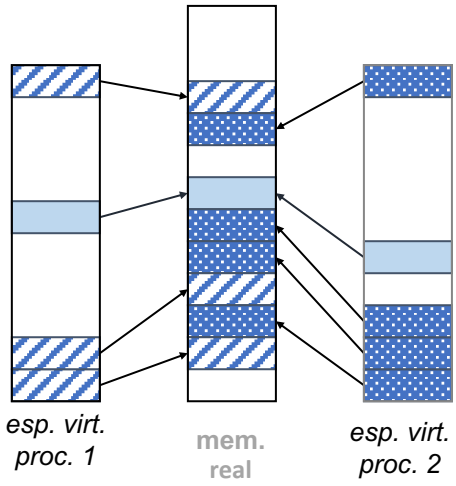


Protecção de memória com páginas

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

Memória Partilhada entre processos

- Se páginas virtuais de diferentes processos forem mapeadas (pelo SO) nos mesmos frames?
- Temos **memória partilhada** entre processos
- Endereços de diferentes processos referem a mesma memória física

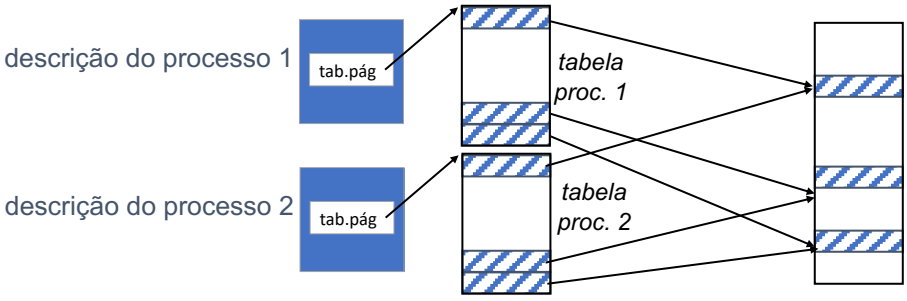


Partilha de páginas/frames

- 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 (como nos threads)
- É conveniente controlar o tipo de acesso autorizado a cada página:
  - Executável: pode ser lido e executado (o fetch é possível)
  - Só leitura: (constantes) só pode ser lido e não escrito (load)
  - Leitura e Escrita: pode ser lido e escrito (load + store)

Revisitando o fork()

- processo filho → espaço virtual distinto mas que começa com o mesmo conteúdo
- Em vez de copiar as frames físicas, o SO pode optar pela partilha das frames do pai
- Como cada processo pode ter memória privada?



memória do filho

- processo filho → espaço virtual distinto mas que começa com o mesmo conteúdo
- Em vez de copiar as frames físicas, o SO pode optar pela partilha das frames do pai
- Usando as permissões por página:
  - Código: executável, só leitura
  - Dados para lêr (constantes): só leitura
  - Dados privados para lêr e escrever: devem permitir leitura e escrita
  - De início, enquanto partilhadas, ficam marcadas só para leitura nas tabelas dos processos
  - Quando um processo tenta escrever, usa-se e técnica de Copy-on-Write

Copy-on-Write

- Só quando se tenta modificar a página partilhada é que se procede à sua duplicação (cópia)
  - Para a MMU a página não pode ser escrita
  - Quando se tenta um escrita (exemplo: mov para memória) é levantada uma interrupção pela MMU
  - O SO atende e atribui uma nova frame ao processo que toma o lugar da página que deu lugar à interrupção
    - copia conteúdo se necessário
    - atualiza nas tabelas de páginas dos processos para leitura e escrita
- Esta técnica permite que o pai e o filho partilhem todas as páginas em memória (incluindo os dados e a pilha) que não sejam escritas.

Gestão das páginas dos processos

- As páginas permitem maior eficiência na gestão da memória central e na criação de processos:
  - O novo processo começa com a mesma memória do pai
  - Ocupa menos espaço real
  - Na tabela, as páginas de código são marcadas como Exec, Read-Only e partilhadas
  - A copia da restante memória será "lazy": em ambas as tabelas, as áreas de dados e "stack" são marcadas como páginas Read-Only para COPY-on-WRITE