Sistemas Operativos
7ª parte - Gestão de Memória

Prof. José Rogado
jrogado@ulusofona.pt
Prof. Pedro Gama
pedrogama@gmail.com
Universidade Lusófona
Revistos para a LIG por Dr Adriano Couto

Gestão de Memória

- Background
- Noção de Swapping
- Alocação de Memória Contígua
- Paginação
- Estrutura da Tabela de Páginas
- Segmentação
- Exemplo: O Pentium Intel

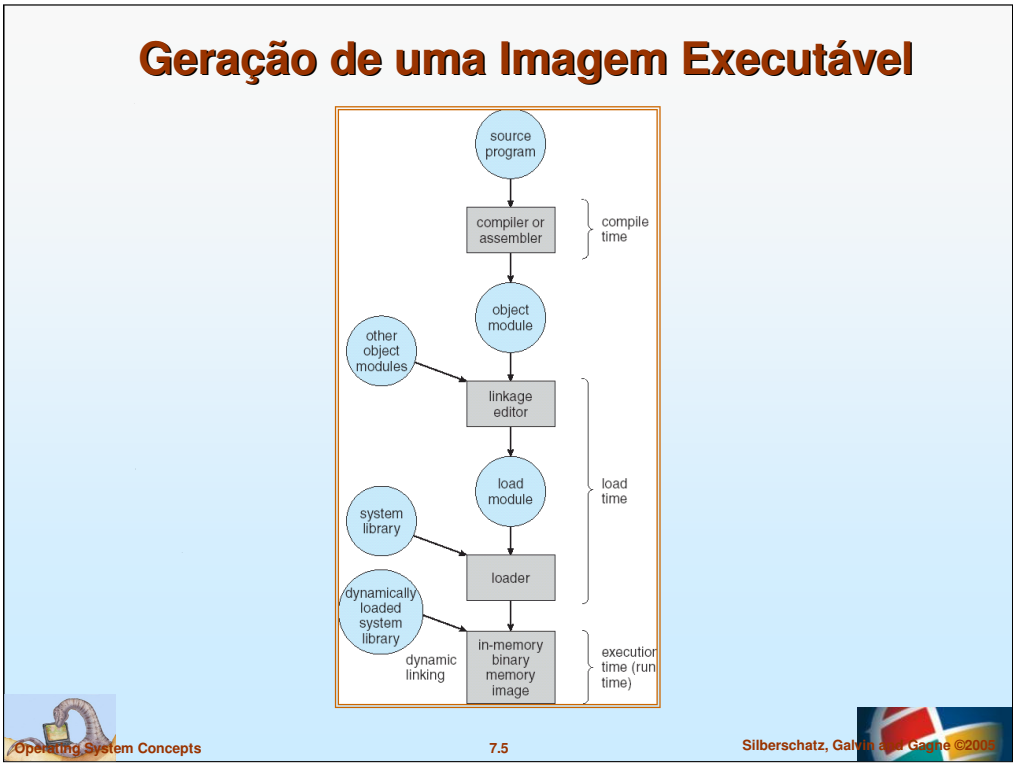
Objectivos

- Apresentar uma descrição das várias formas de organização da memória
- Compreender o mecanismo de swapping
- Comparar várias técnicas de gestão de memória, nomeadamente paginação e segmentação

Introdução

- A memória principal e os registos e o cache são os únicos dispositivos de armazenamento que o CPU pode aceder directamente
- Quando um programa é executado, o seu código tem portanto de ser trazido de disco para a memória central
  - A imagem do processo é **carregada** em memória
- Para que o processo possa ser carregado em memória é necessário atribuir-lhe um espaço
  - Pode ser em qualquer zona da memória física que esteja livre
  - Designado por **espaço de endereçamento físico** do processo
- Uma vez que existem inúmeros processos carregados simultaneamente em memória (*residentes*):
  - Os processos residentes em memória devem ter espaços físicos distintos
  - Têm de existir mecanismos de protecção memória para assegurar a separação entre eles
  - Como é que cada processo sabe qual é o seu espaço de endereçamento físico? => Ligação entre Instruções e Memória

A imagem é uma descrição da organização do programa em memória antes da execução. Localização e dimensão dos dados, das instruções, etc.



O diagrama ilustra as diveras etapas desde o momento em que as instruções são compiladas até ao carregamento para posterior execução. Durante estas etapas não é possível, num computador moderno, determinar as posições **finais,** em memória, das instruções e dados.

Ligação de Instruções e Dados à Memória

- A **ligação** das Instruções e Dados a Endereços Físicos da Memória podem ser feitos em três alturas distintas:
- **Na compilação**: se os endereços de memória são conhecidos à priori, o código pode gerado de forma **absoluta**
  - Mas se o endereço de início muda, terá de ser recompilado
- **No carregamento**: se os endereços da memória onde o processo vai ser carregado não são conhecidos no momento da compilação, o código é gerado em modo **relativo**
- **Na execução**: se o processo pode ser carregado em várias zonas de memória durante a sua execução, a ligação aos endereços não pode ser realizada de forma definitiva.
- Nesse dois últimos casos é necessário suporte do hardware para realizar o *mapeamento* de endereços
  - Registos de base e limite
  - Unidade de Gestão Memória

Carregamento Dinâmico

- Designa o facto de só carregar código executável em memória quando é invocado
  - Permite uma melhor aproveitamento do espaço memória
  - Se não for invocado, nunca é carregado
  - Muito útil quando grandes quantidades de código são necessários para gerir casos que ocorrem pouco frequentemente
- A ligação de endereços é deixada para o momento da execução
  - É implementada por um módulo (*stub*) utilizado para localizar o código e invocar o seu carregamento dinâmico
  - Depois de uma rotina ser localizada e/ou carregada, o *stub* invoca-a no endereço em que se encontra *mapeada*
- É necessária a intervenção do sistema operativo para carregar a rotina e *mapeá-la* no espaço de endereçamento do processo
- O carregamento dinâmico pode ser utilizado com bibliotecas partilhadas por vários processos, ou com módulos do Sistema Operativo

As imagens que preparadas para carregamento dinâmico são as .DLL (Dinamicaly Linked Library) em Windows e os .SO (Shared Object) ou SA (Shared Archive)

Espaços de Endereçamento Lógico e Físico

- Nos sistemas actuais, quando a imagem de um processo é gerada nunca é conhecido o local onde vai ser carregado um processo
- Durante a execução de um processo, este pode vir a ser carregado em zonas distintas de memória
- A separação entre o espaço de endereçamento **lógico** e espaço de endereçamento **físico** de um processo é essencial para a Gestão de Memória
  - **Endereços lógicos** – gerados pelo CPU, também conhecidos por endereços virtuais.
  - **Endereços físicos** – endereços recebidos pela unidade de memória
- O *mapeamento* (correspondência) entre o espaço lógico e físico é realizado por um dispositivo hardware
  - Unidade de Gestão Memória ou *Memory Management Unit*

Essencial perceber isto e as suas implicações.

Exemplo de Endereçamento Lógico

O programa em C:

```
int i;
int j;
int k;

main()
{
  printf("&main = %x\n", &main);
  printf("&printf = %x\n", &printf);
  printf("&i = %x\n", &i);
  printf("&i = %x\n", &j);
  printf("&i = %x\n", &k);
  scanf("%d", &i);
}
```

Produz sempre o mesmo resultados, mesmo executado em simultâneo:

```
&main = 8048394
&printf = 80482e4
&i = 8049658
&i = 8049650
&i = 8049654
```

```
&main = 8048394
&printf = 80482e4
&i = 8049658
&i = 8049650
&i = 8049654
```

```
&main = 8048394
&printf = 80482e4
&i = 8049658
&i = 8049650
&i = 8049654
```

Conclusão: os endereços lógicos de um processo são sempre os mesmos, os endereços físicos é que mudam

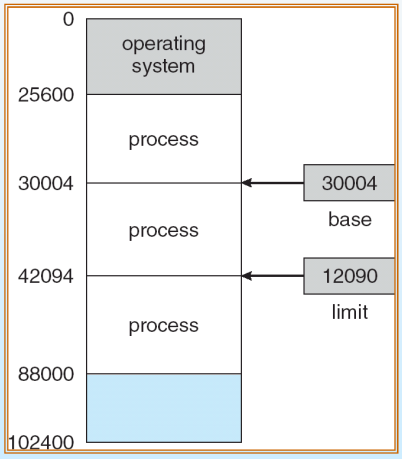
MMU - Memory Management Unit

- Unidade de Gestão de Memória (UGM)
- Dispositivo hardware que mapeia endereços lógicos para físicos
- Num sistema com MMU, o valor do registo de relocação é adicionado a todos os endereços gerados por um processo utilizador antes de serem enviados para a memória física
- O programa utilizador só conhece endereços lógicos, nunca conhece o endereço físico real
- Existem inúmeros tipos de MMU
  - Inicialmente simples, baseada em *registos de relocação*
  - Constituída depois por um *chip* dedicado
  - Actualmente um dos dispositivos do *chipset* de um processador mais complexos de utilizar e programar

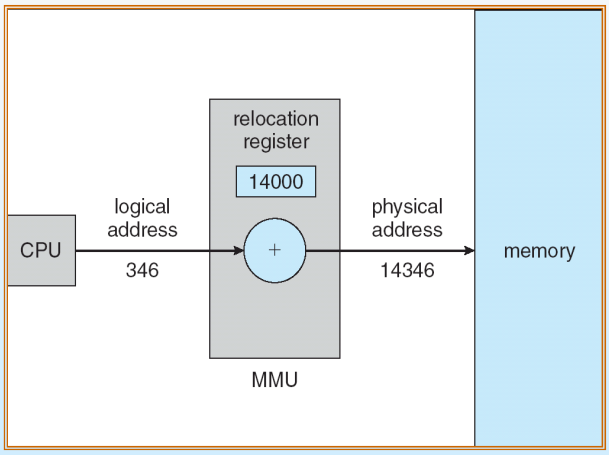
O aluno compreender o papel deste dispositivo de hardware para a realização da conversão entre endereços, nomeadamente o mecanismo básico que está no slide 7.12..

Registos de Relocação

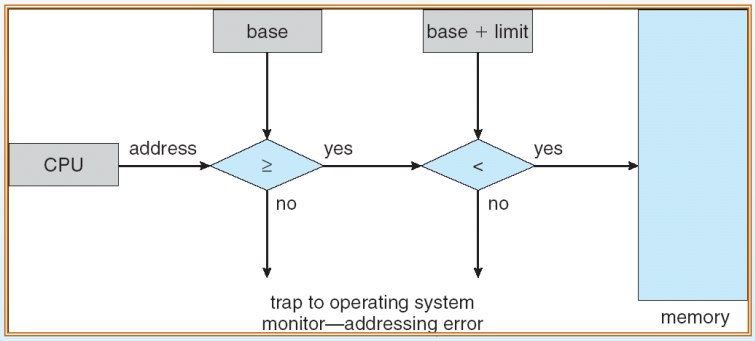
- O espaço de endereçamento lógico de um processo pode ser estabelecido através de dois registos: os registo de relocação **base** e **limite**



Mecanismo do Mapeamento



Protecção de Endereços



Registos de Base e Limite:

- Esquema de protecção muito simples
- Quando o endereço gerado pelo CPU é inferior à base ou superior ao limite é gerada uma excepção para o sistema operativo
- Para cada processo são carregados valores diferentes nos registos

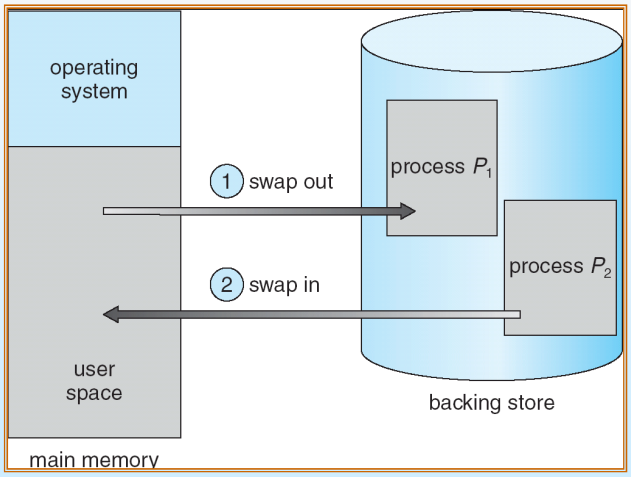
Organização da Memória

- A memória principal está geralmente dividida em duas partições:
  - Sistema Operativo residente, geralmente carregado a partir do início da memória, contendo o vector de interrupções
  - Processos dos utilizadores carregados na restante zona
- Os registos de relocação da MMU são utilizados para proteger os processos uns dos outros e de acederem ao espaço do sistema (código e dados)
  - O Registo de base contém sucessivamente o endereço de início de cada processo activo
  - O registo de limite contém um intervalo de endereços lógicos dentro do qual o processo está contido
    - Todos os endereços acedidos pelo processo devem estar contidos entre o valor de base e o de limite
  - Os endereços são mapeados pela MMU dinamicamente

Swapping

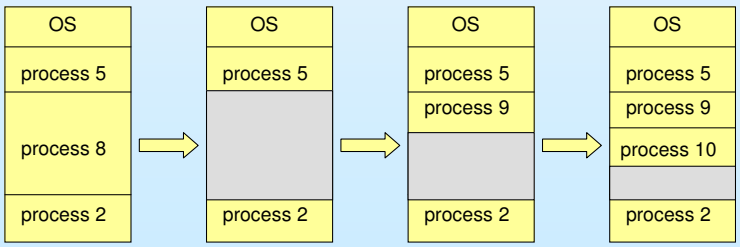
- Um processo pode ser temporariamente retirado da memoria central caso haja falta de espaço para manter todos os processo, e mais tarde de novo carregado para continuar a execução
- **Memória Secundária** (*Backing Store*) – espaço em disco rápido suficientemente grande para guardar cópias das imagens memória de todos os processos residentes; deve permitir acesso rápido e directo a essas imagens
- **Roll out, roll in** – variante do swapping utilizada para algoritmos de scheduling baseados na prioridade: os processos com baixa prioridade podem ser retirados da memória para deixar espaço a processos com maior prioridade
- O tempo de transferência de e para memória secundária é directamente proporcional ao tamanho do processo transferido
- Versões modificadas de **swapping** existem na maior parte dos sistemas operativos (i.e., UNIX, Linux, e Windows)
- O sistema mantém uma na **ready queue** informação dos processos que têm as suas imagens guardadas em disco
  - Pode ser uma fila específica (swap queue)

Vista Esquemática do Swapping



Alocação Contínua

- Para carregar um processo em memória, atribui-se-lhe uma zona de memória de tamanho igual ou superior ao seu espaço físico total
- A alocação de Memória é feita através do Particionamento Múltiplo do espaço físico disponível em Blocos
  - Bloco livre – espaço contínuo de memória livre que pode estar localizado em qualquer sítio da memória física
  - Quando um processo está para ser carregado em memória, é-lhe atribuída memória de um bloco livre
  - O Sistema Operativo mantém informações sobre os
    - Blocos livres
    - Blocos ocupados



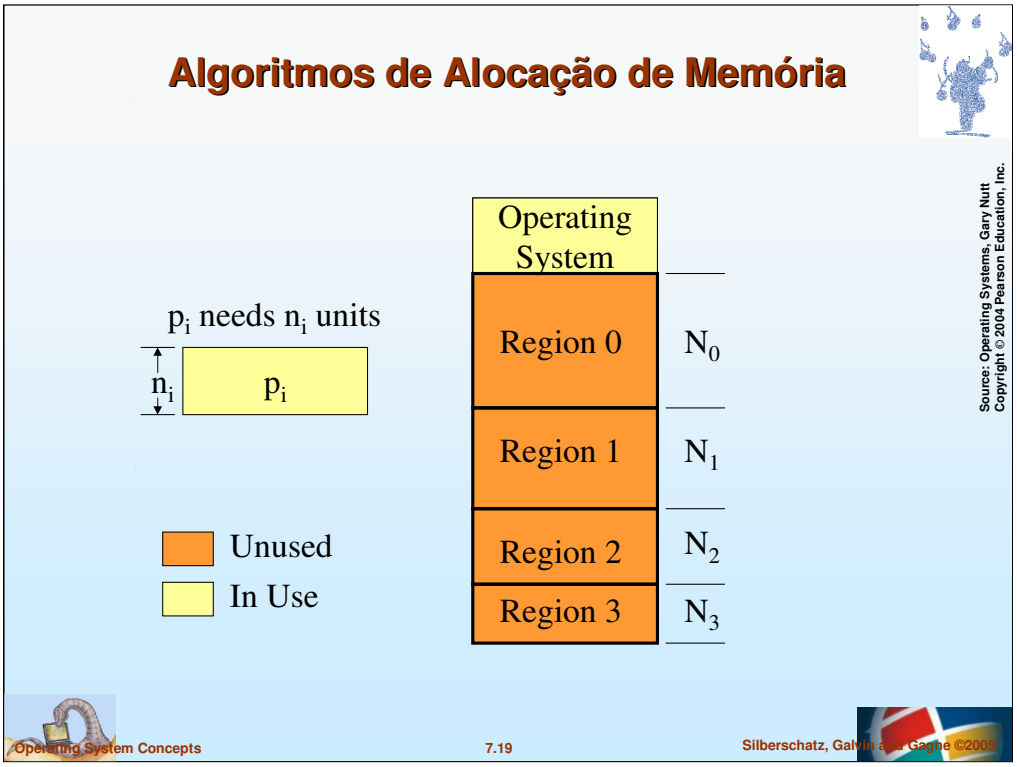
Numa primeira abordagem, procura-se que o espaço de memória atribuido a um processo seja contínua. Veremos adiante de que forma os modernos SO, com a assitência do hardware, operam fora deste cenário ideal.

Problema da Alocação Contínua de Memória

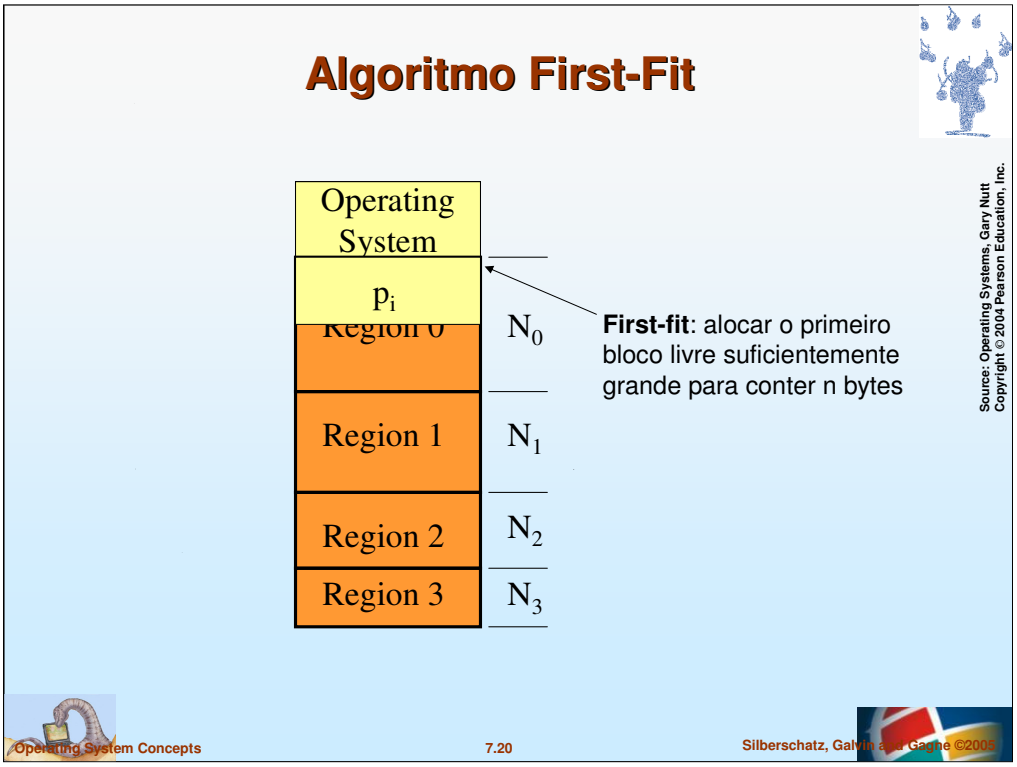
Como responder a um pedido de alocação de N bytes a partir de uma lista de blocos livres:

- **First-fit**: alocar o primeiro bloco livre suficientemente grande para conter N bytes
- **Best-fit**: alocar o mais pequeno bloco que seja suficientemente grande para conter N bytes
  - É preciso procurar a lista toda, a menos que esteja ordenada por tamanhos
  - Produz os melhores resultados pois provoca menos desperdícios
- **Worst-fit**: alocar o maior bloco que seja suficientemente grande
  - Necessita procurar a lista toda
  - Produz fragmentos maiores

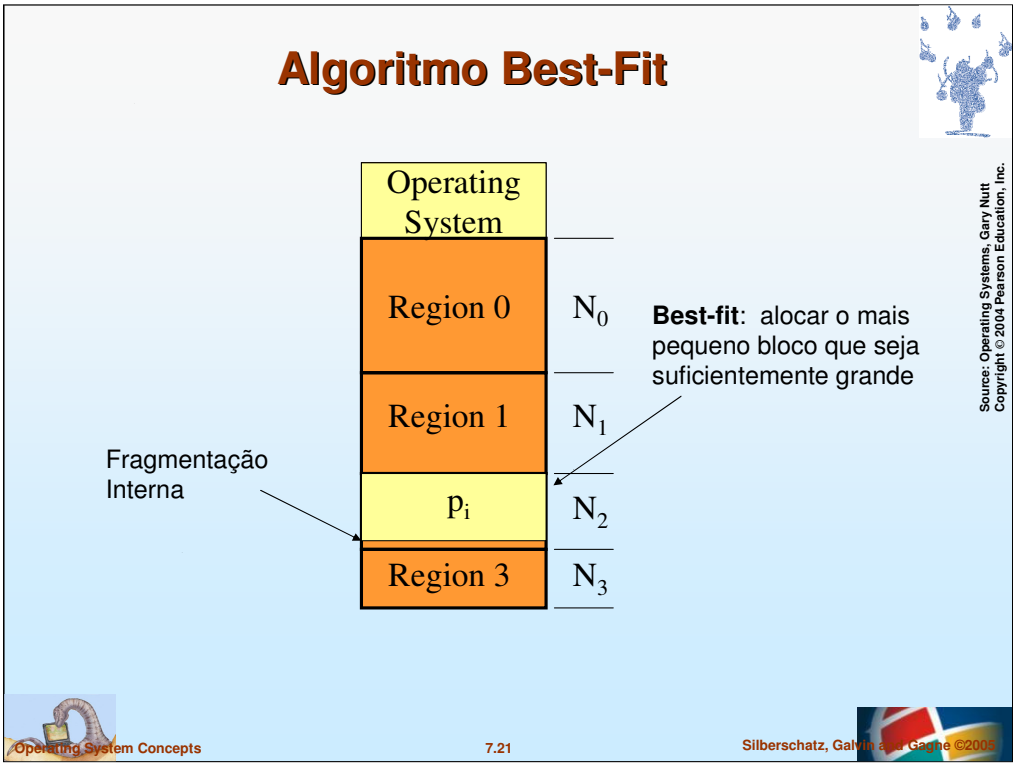
Meramente ilustrativo. O conceito é trivial. O importante a reter é que, à medida que os processos se inciiam e terminam, iremos ter a memória física fragmentada!



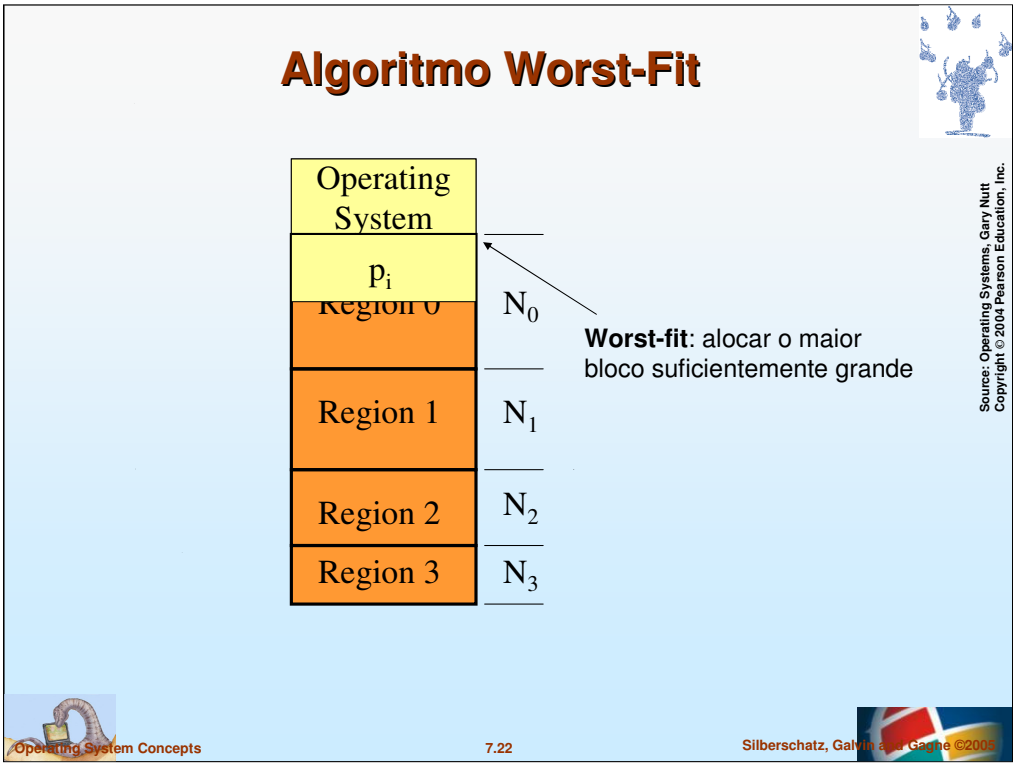
Só para ilustração.



Só para ilustração.



Só para ilustração.



Só para ilustração.

Partições Fixas e Variáveis

- Tradicionalmente os SO utilizavam partições de memória de tamanho fixo
  - A memória é dividida em blocos de vários tamanhos que não podem mudar
  - Os processos grandes são alocados aos blocos maiores
  - A memória não alocada numa partição provoca fragmentação interna (a cada bloco)
- No sistemas interactivos, o espaço memória utilizado pelos processos pode variar muito no decorrer da sua execução
  - Nos sistemas mais recentes, a partição da memória é realizada em blocos de tamanho variável
  - Os blocos podem ser agregados ou separados segundo as necessidades dos processos
  - Neste caso a fragmentação é externa aos blocos

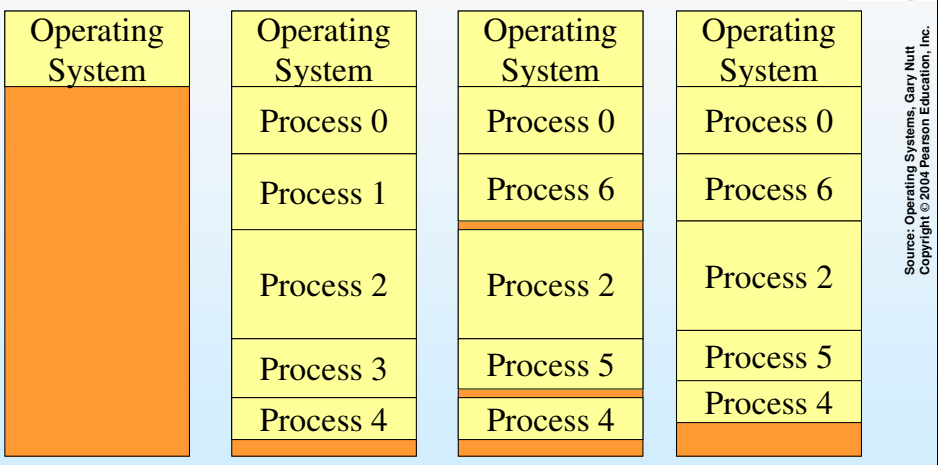
A utilização de partições de memória fixa levava a uma má gestão deste escasso recurso.

Fragmentação

- **Fragmentação Interna** – o espaço alocado é ligeiramente maior do que o necessário; esta diferença está no interior de uma partição mas não é utilizado pois o seu tamanho não permite utilizá-lo para carregar nenhum processo
- **Fragmentação Externa** – o espaço memória total livre que existe é suficientemente grande para satisfazer um pedido, mas não é contínuo
- A fragmentação externa pode ser reduzida através de compactação
  - Reorganizar o espaço memória de forma a colocar toda a memória livre num único bloco contínuo
  - A compactação só é possível se a relocação for dinamica, e é feita durante a execução
  - Problema de I/O durante a recompactação
    - Um processo é impedido de realizar I/O durante a recompactação
    - Utilizar só os buffers do cache

Aqui detalha-se um pouco a forma os buracos (que podem levar ao desperdício) estão localizados: na partição/bloco do processo ou nas região livre.

Partições Memória Variáveis



Source: Operating Systems, Gary Nutt
Copyright © 2004 Pearson Education, Inc.

Fragmentação Externa: a compactação permite mover os processos em memória e juntar os espaços livres num único bloco livre

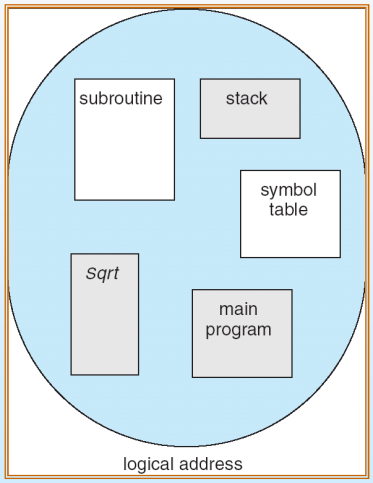
Segmentação

- Um esquema de gestão de memória que implementa uma visão mais próxima da do utilizador
- Um programa é um conjunto de segmentos que são entidades lógicas tais como:
  - programa principal,
  - subrotinas (funções)
  - objectos (métodos)
  - variáveis locais,
  - variáveis globais,
  - pilha,
  - tabela de símbolos,
  - bibliotecas

Em Windows, o programa dumpbin.exe (vem com o Visual Studio) permite listar informação sobre um executável, biblioteca, etc.

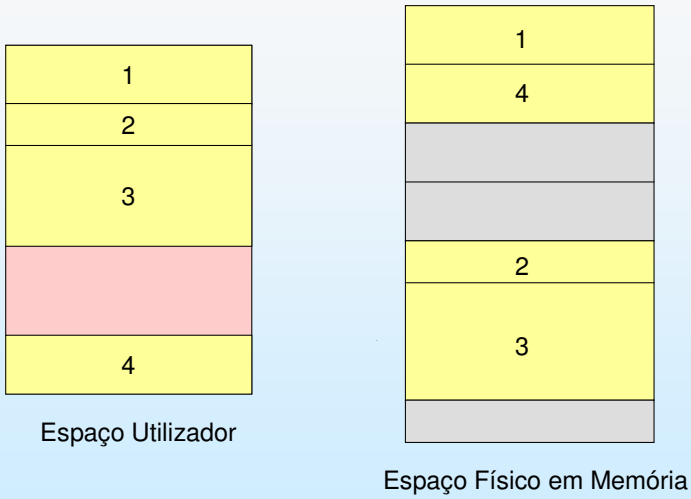
Em UNIX ver objdump(1). No Linux além deste existe o readelf(1)

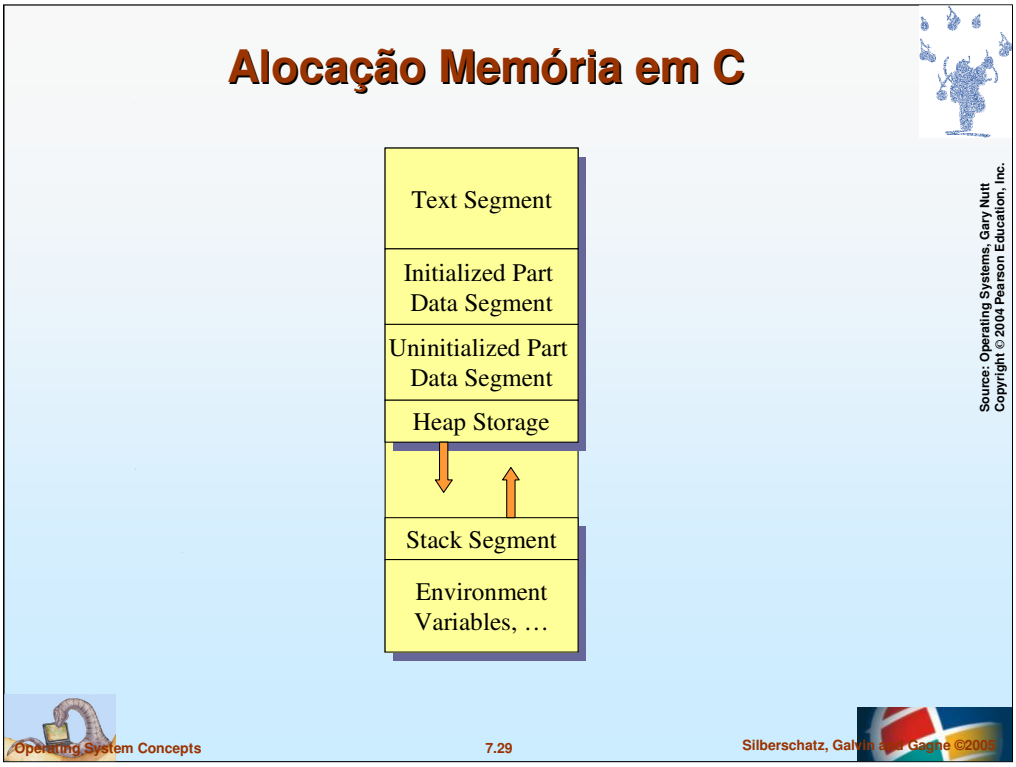
Visão do Programa Utilizador



As várias secções de um processo não precisam de ser contínuas, mesmo do espaço de endereçamento lógico, pois são independentes. São segmentos.

Visão Lógica da Segmentação





Para reter: a forma como os segmentos heap e stack crescem.

Arquitectura Segmentada

- Um endereço lógico é constituído pelo por:

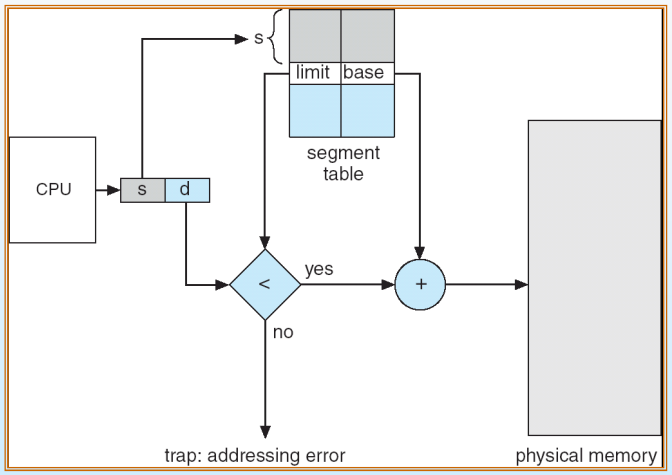
  <segment-number, offset>

- **Tabela de Segmentação** – mapeia endereços lógicos em físicos; cada entrada na tabela tem:
  - **base** – contém o endereço de memória física onde o segmento está carregado
  - **limite** – indica o comprimento do segmento

- **Segment-table base register (STBR)** indica o endereço da tabela de segmentação em memória
- **Segment-table length register (STLR)** indica o número de segmentos que um programa utiliza;

  o número de segmento ***s*** é legítimo se ***s*** < **STLR**

Hardware de Segmentação

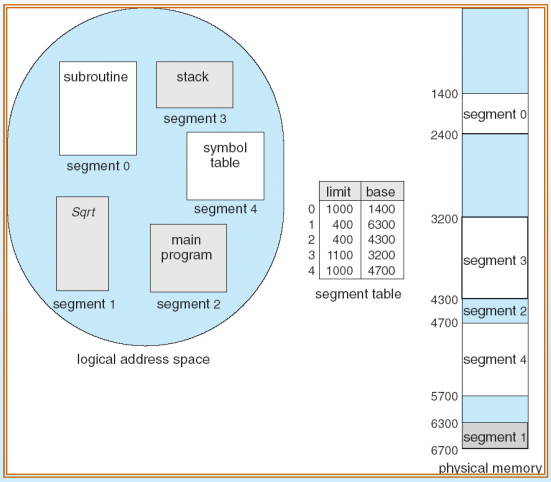


Como o hardware usa os segmentos. Ver por exemplo, os manuais da Intel (online)

Protecção e Partilha de Segmentos

- Protecção
  - A cada elemento da tabela de segmentos está associado
    - Um bit de validação = 0 ⇒ segmento ilegal
    - Privilégios de leitura/escrita/execução
- Esta funcionalidade permite partilhar zonas de memória entre vários processos
  - Ex. segmento de texto (código) partilhado entre várias instâncias da mesma aplicação
- A partilha de código entre vários processos é feita a nível dos segmentos
  - A granularidade da protecção é por segmento
- Como os segmentos têm tamanhos variáveis, a alocação memória é feita dinamicamente
  - O tamanho do segmento define o mais pequeno bloco de memória endereçável
  - Incorre nos problemas de fragmentação vistos atrás
    - Na realidade, os processadores actuais utilizam um misto de segmentação e paginação

Exemplo de Segmentação



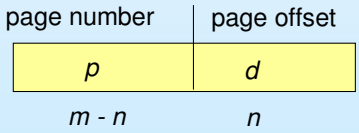
Paginação

- Para resolver os problemas de fragmentação introduzidos pela alocação contínua em arquitecturas segmentadas
  - O espaço de endereçamento físico de um processo pode ser descontínuo
  - A memória livre é utilizada onde estiver disponível
- A memória física é dividida em blocos de tamanho fixo chamados ***frames*** ("molduras") cujo tamanho é uma potência de 2, geralmente compreendido entre 512 e 8192 bytes.
- A memória lógica é dividida em blocos do mesmo tamanho chamados **páginas**
- O sistema guarda a lista de todas as *frames* livres
- Para executar um programa com ***n*** páginas é necessário encontrar *n frames* livres
- A localização física de cada *frame* é guardada numa tabela de páginas para realizar a tradução de endereços lógicos em endereços físicos
- A única fragmentação que existe é no interior das páginas

Este é um mecanismo mais fino que existe por baixo dos segmentos, nos processadores modernos.

Tradução de Endereços

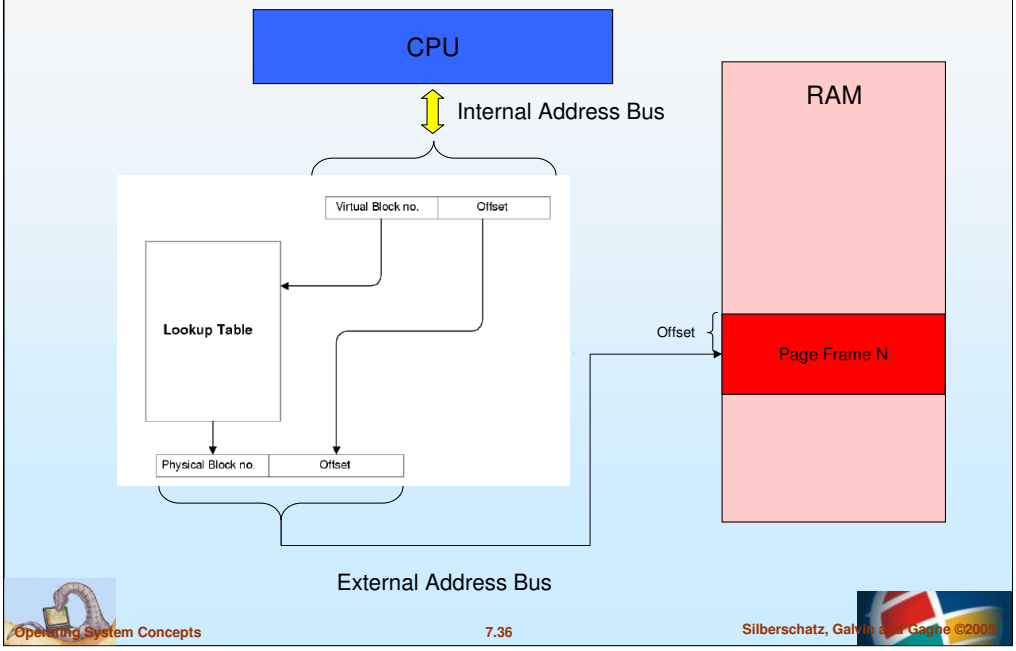
- O endereço gerado pelo CPU é dividido em:
  - **Número de página (*p*)** – usado como índice numa tabela de páginas que contém o endereço de base da frame correspondente em memória física
  - **Offset de Página (d)** – é combinado com o endereço de base para formar o endereço físico enviado para a memória
- Para um espaço de endereçamento 2^m e tamanho de página 2^n (m>n), o endereço físico é da forma:



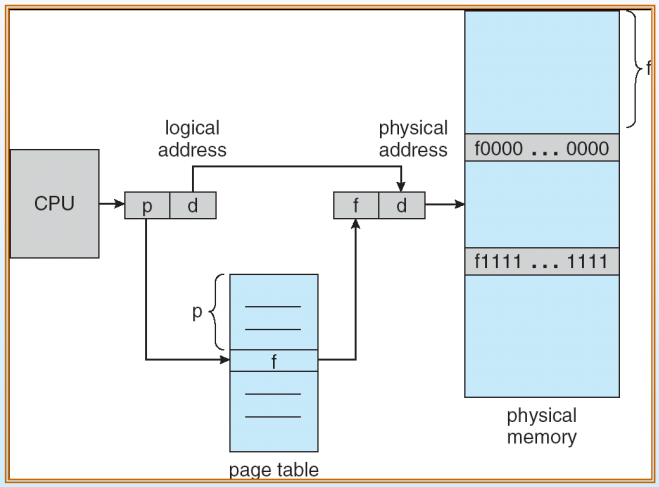
Perceber genericamente a tradução de endereços, semelhante aos segmentos.

A diferenças é que um processo pode ter milhares de páginas.

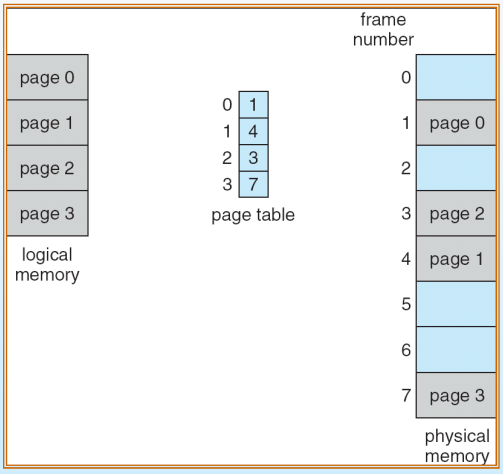
Mecanismo de Paginação



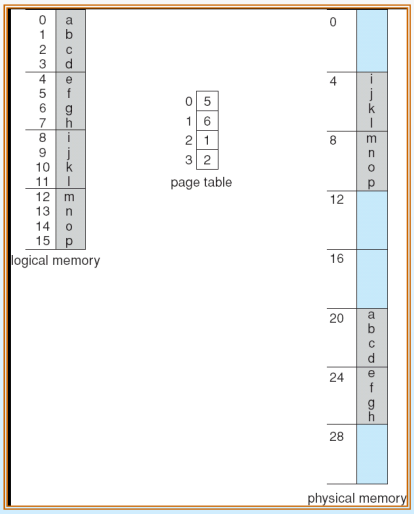
Hardware de Paginação



Conteúdo da Tabela de Páginas

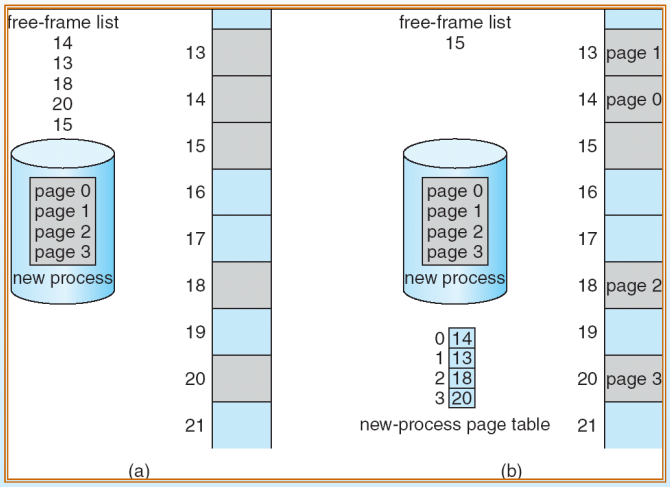


Exemplo de Alocação Paginada



Memória de 32 bytes e páginas de 4 bytes

Alocação de Memória por Frames



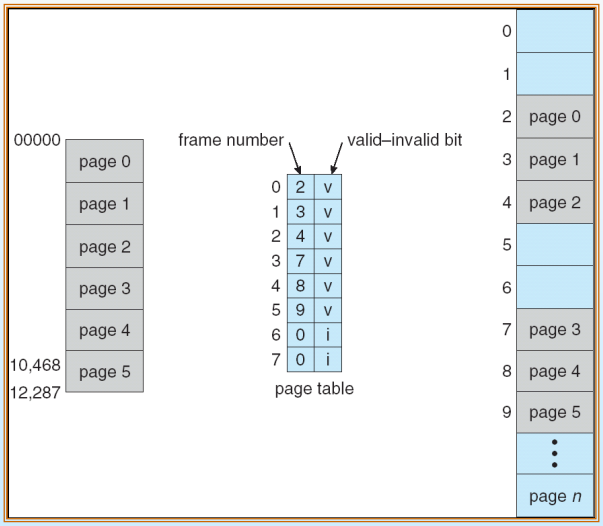
Antes da alocação

Depois da alocação

Protecção Memoria

- A protecção memória é implementada para garantir uso exclusivo de processos a espaço memória
- Bit **Válido/inválido** associado a cada entrada na tabela de páginas:
  - "válido" indica que a página associada está no espaço de endereçamento do processo e é portanto um acesso legal
  - "inválido" indica que a página não está acessível no espaço de endereçamento de um processo
  - Um acesso de um processo a uma página com o bit inválido provoca uma excepção (trap) **Page Fault**
- Cada processo tem um conteúdo específico da tabela de páginas que faz parte do seu contexto de execução
- Mesmo as páginas não utilizadas pelo processo têm de ser declaradas inválidas para prevenir acessos fora do espaço autorizado

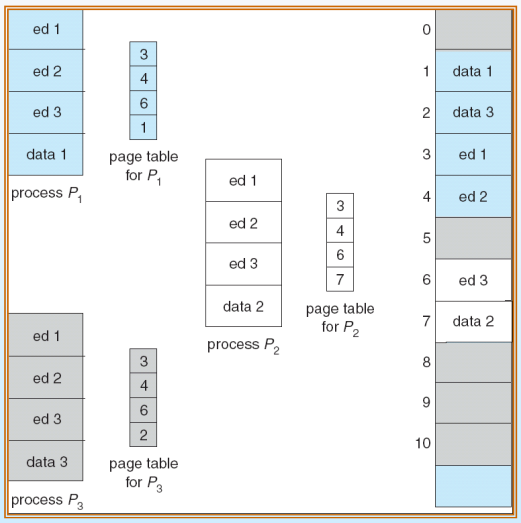
Bit Válido (v) / Inválido (i)



Páginas Partilhadas

- **Código partilhado**
  - Uma cópia de código *read-only* pode ser partilhada por vários processos com múltiplas instâncias (ex: editores de texto, compiladores, browsers, etc…)
  - O código partilhado tem de aparecer no mesmo local no endereço lógico do processo
- **Dados e código privado**
  - Cada processo tem uma cópia privada separada dos dados e de código que não é partilhado
  - As páginas de código privado e de dados podem estar em qualquer lugar do espaço de endereçamento lógico do processo
  - Os descritores de páginas contêm informação adicional
    - Código / dados
    - Leitura / Escrita

Exemplo de partilha de Páginas



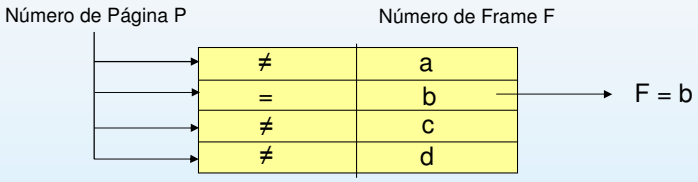
Implementação da Tabela de Páginas

- A tabela de paginas pode estar guardada em memória principal
  - O Registo de Base da Tabela de Páginas aponta para o início da tabela de páginas (**Page-table base register – PTBR**)
  - O Registo de Comprimento da Tabela de Páginas indica o tamanho da tabela de páginas (**Page-table length register PRLR**)
- Assim, cada ciclo memória implica dois acessos à memória
  - Um para aceder à tabela de páginas, outro para aceder ao conteúdo
- Nos processadores mais recentes, a tabela de páginas é implementada em hardware especial de acesso rápido, designado por **Memória Associativa** ou **Translation Look-aside Buffer (TLB)**
- Alguns TLBs armazenam também Identificadores de Espaço de Endereçamento que identificam univocamente cada processo para fornecer protecção de espaço de endereçamento entre processos

Só para ilustração.

Memória Associativa

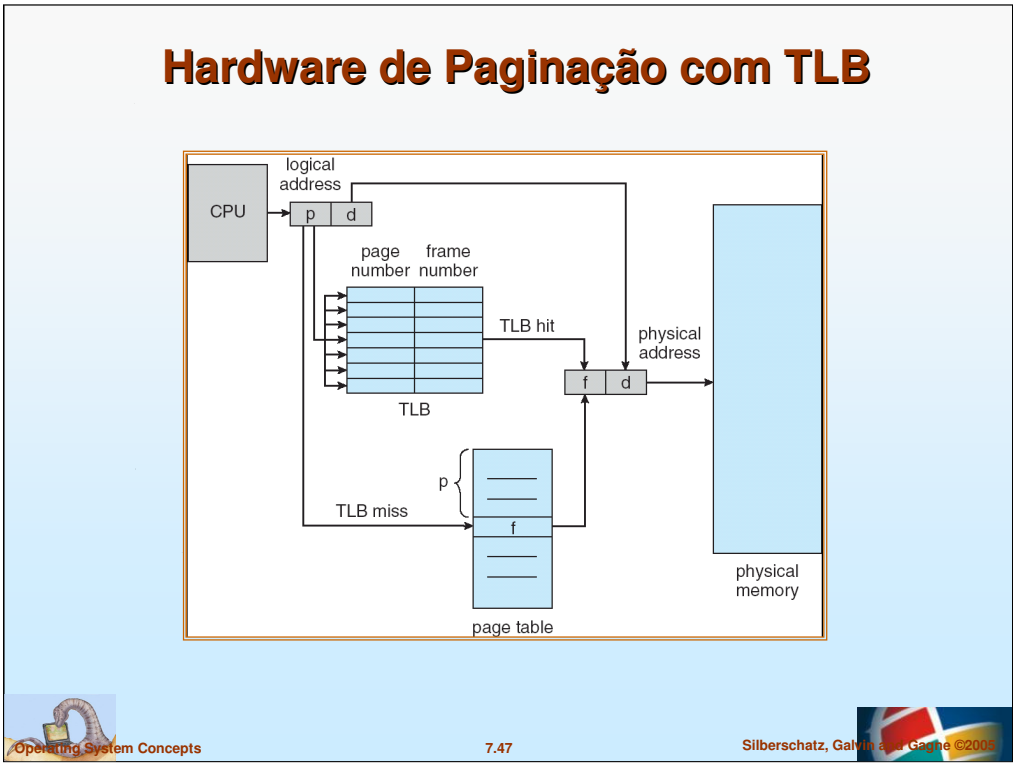
- Memória Associativa – busca em paralelo



Tradução de endereço (p, d)

- Se P está num registo, utilizar a frame F associada
- Senão, indexar a tabela de páginas com P e obter F
- P e F são colocados em dois registos associados da memória associativa

Só para ilustração.



Só para ilustração.

Problema das Tabelas de Paginação

- Para CPUs com espaços de endereçamento de 32 bits (4 Gb), o número de entradas da tabela de páginas torna-se grande demais:
  - Para páginas de 4Kbytes (2^{12}), o número de entradas necessárias na tabela de páginas é de $2^{32}/2^{12} = 2^{20}$
    - 1 Milhão de entradas por processo !

- Mesmo as páginas não utilizadas por processos precisam de ser preenchidas para impedir acessos fora do espaço permitido
- Por isso utilizam-se outras formas de particionamento do endereço lógico, sobretudo em arquitecturas de 64 bits
  - Tabelas de Páginas Hierárquicas
  - Tabelas de Páginas com Hashing
  - Tabelas de Páginas Invertidas

Só para ilustração. Perceber os problemas envolvidos.

Tabelas de Páginas Hierárquicas

- O espaço de endereçamento é dividido em várias zonas, correspondendo a vários níveis de tabelas de páginas
  - As páginas não utilizadas são invalidadas no nível superior
- Um técnica simples é utilizar um a tabela de páginas com dois níveis em endereços de 32 bits:

10	10	12

- Mas para 64 bits ainda não é suficiente !

20	20	10	14

Soluções encontradas. Só para ilustração.

Exemplo de Paginação a 2 Níveis

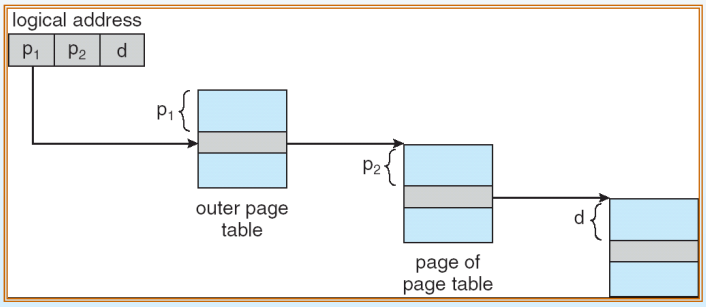
- Um endereço lógico (CPU de 32-bits com páginas de 4K) é dividido em :
  - Número de tabela de páginas representado por 10 bits
  - Num número de página na tabela representado por 10 bits
  - Offset na página representado por 12 bits
- Assim um endereço lógico é particionado da seguinte forma:

tabela	página	offset
p_i	p_2	d
10	10	12

  - onde p_i o índice na tabela de páginas externas e p_2 é o offset dentro dessa tabela

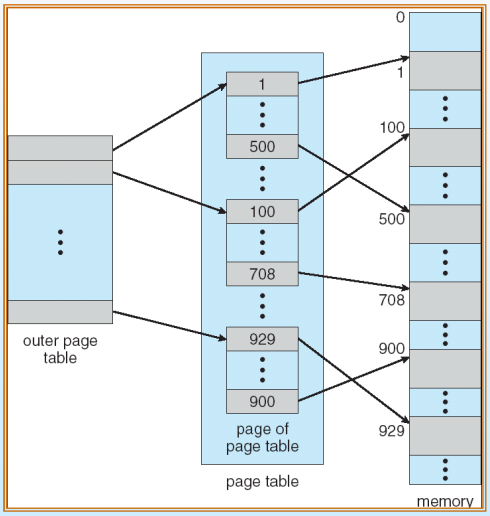
Soluções encontradas. Só para ilustração.

Esquema de Tradução de Endereços



Soluções encontradas. Só para ilustração.

Implementação de Paginação a 2 Níveis



Soluções encontradas. Só para ilustração.

Paginação a N níveis

Dois níveis de paginação em 64 bits não é suficiente (242 = 4 Tb !)

outer page	inner page	offset
p_1	p_2	d
42	10	12

Necessidade de mais níveis em arquitecturas de 64 bits

2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

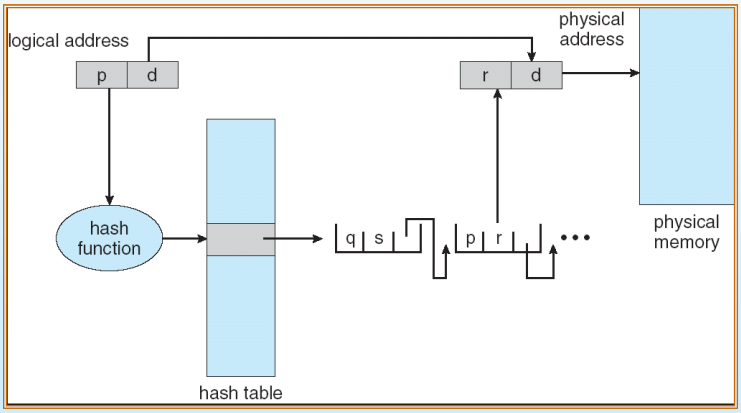
Soluções encontradas. Só para ilustração.

Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location.
- Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.

Soluções encontradas. Só para ilustração.

Hashed Page Table

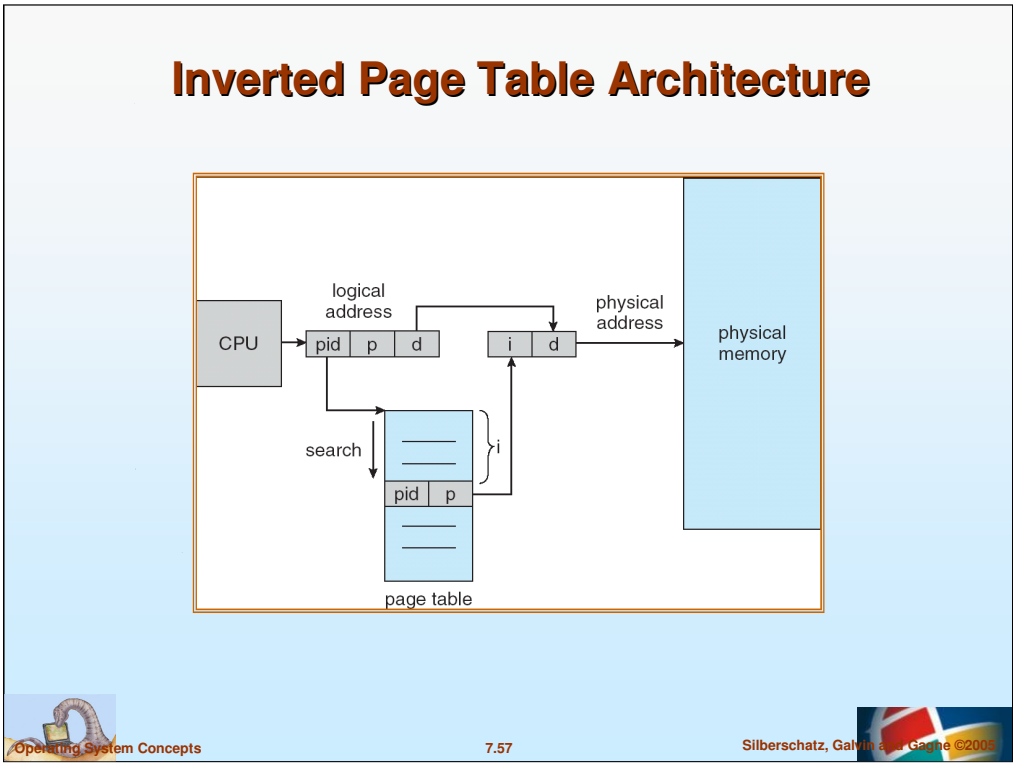


Soluções encontradas. Só para ilustração.

Inverted Page Table

- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries

Soluções encontradas. Só para ilustração.



Soluções encontradas. Só para ilustração.

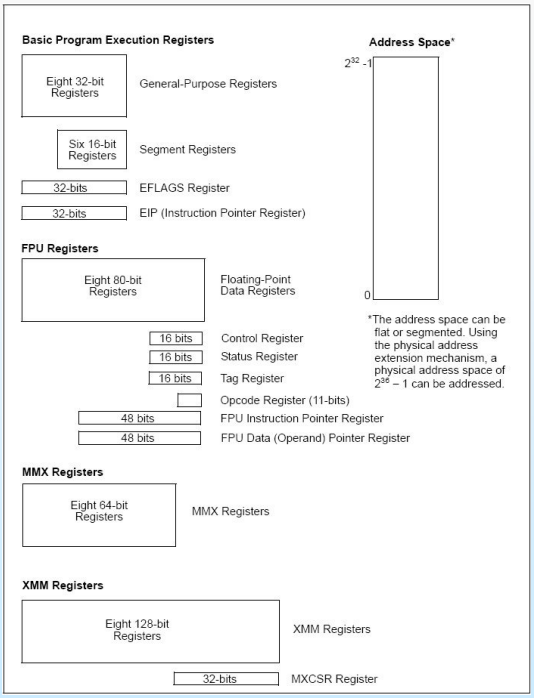
Exemplo: o Pentium Intel

- Suporta segmentação e paginação
- O CPU gera um endereço lógico
  - Entra na unidade de segmentação
    - Produz um endereço linear
  - Endereço linear entra na unidade de paginação
    - Gera um endereço físico para a memória
  - O conjunto das duas unidades de segmentação e paginação é equivalente a uma MMU

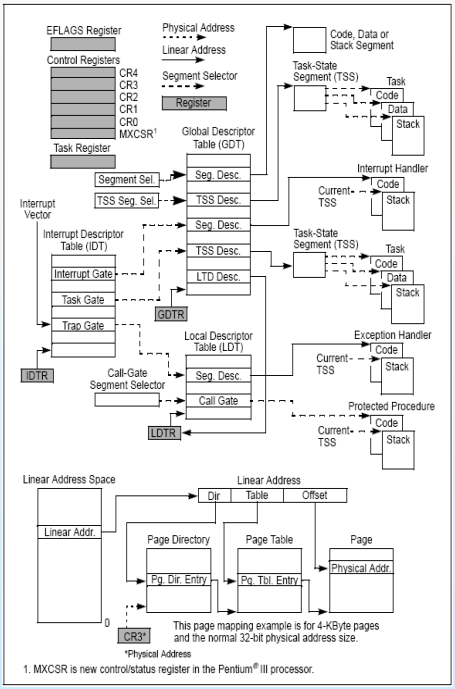
Só para ilustração (todos os restantes slides).

Mas convem ler para cimentar os conhecimentos☺

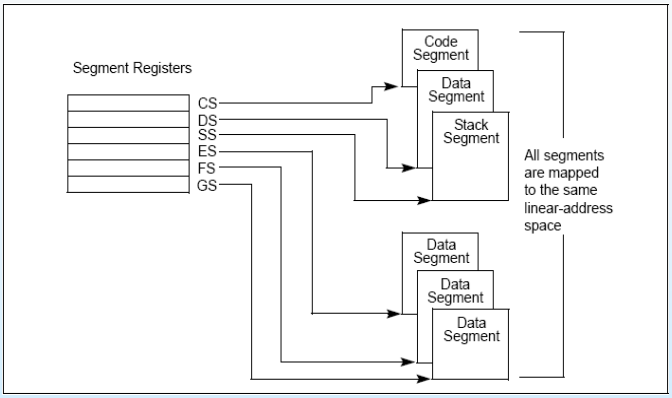
Registos do Pentium Intel



Registos de Gestão Memória do Pentium

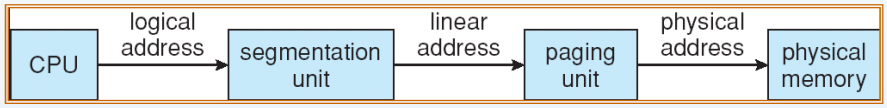


Registos de Segmentação

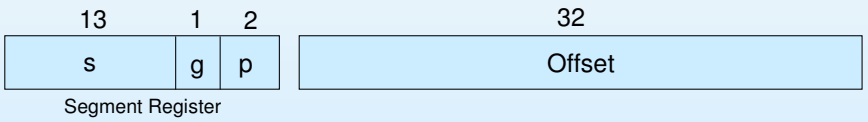


Segment Register

Conversão de Endereços no Pentium



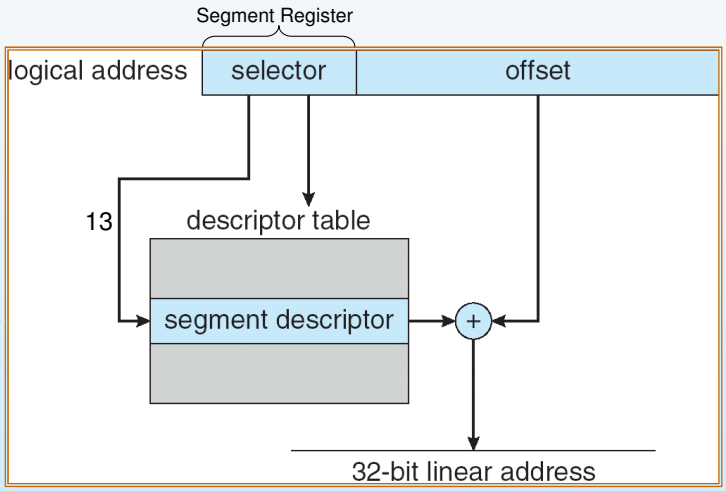
Logical address



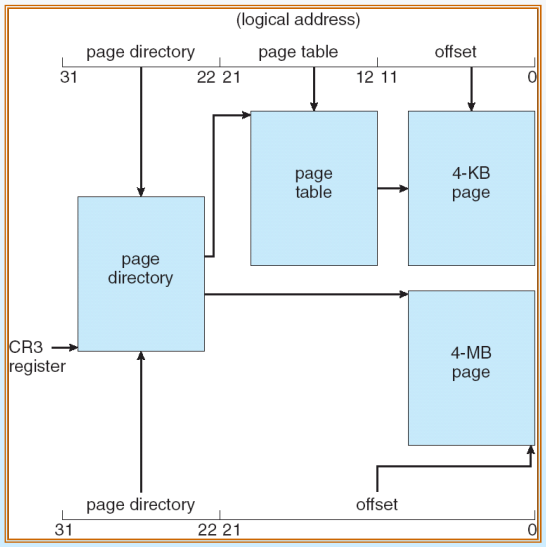
Linear address

page number		page offset
p_1	p_2	d
10	10	12

Segmentação no Pentium Intel

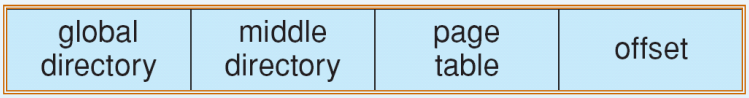


Arquitectura de Paginação no Pentium



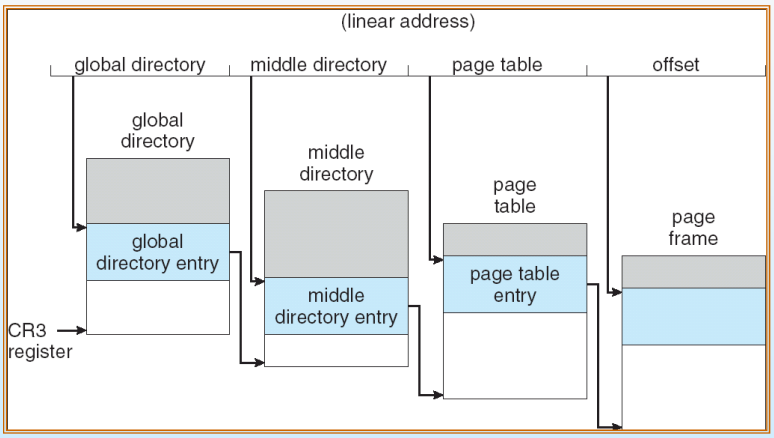
Endereço Linear em Linux

Dividido em quatro partes:



No Pentium, o ***middle directory*** não é utilizado

Paginação a 3 Níveis no Linux



Fim