GESTÃO DE MEMÓRIA

- **Conceitos introdutórios**
  - Criação de um programa executável
  - Recolocação
  - Ligação (*linking*)
  - Carregamento (*loading*)
  - Endereços reais e endereços virtuais
  - *Swapping*
  - Protecção de memória
- **Técnicas de gestão de memória**
  - Alocação contígua e alocação não-contígua
  - Partição fixa e partição dinâmica
  - Paginação e segmentação
  - Memória virtual

Sistemas c/ Monoprogramação e Multiprogramação

Sistemas c/ monoprogramação

- um processo em memória de cada vez
- o processo pode acupar toda a memória disponível p/ o utilizador
- técnica de sobreposição (***overlay)*** permitia correr processos que ocupavam mais memória do que a memória física

Sistemas c/ multiprogramação

- necessidade de **recolocação**
  - » carregar o programa numa zona arbitrária da memória
- necessidade de **protecção**
  - » isolar os espaços de endereçamento do S.O. e das aplicações
- necessidade de **partilha**
  - » cooperação entre processos
- técnica de memória virtual permite correr processos que ocupam mais memória do que a memória física

Criação de um Programa Executável

Programa-fonte
- **endereços simbólicos**
  - » **ex: nextstep**

Compilador

Módulo-objecto
- **endereços recolocáveis**
  - » **ex: 92 *bytes* a partir do início do módulo**

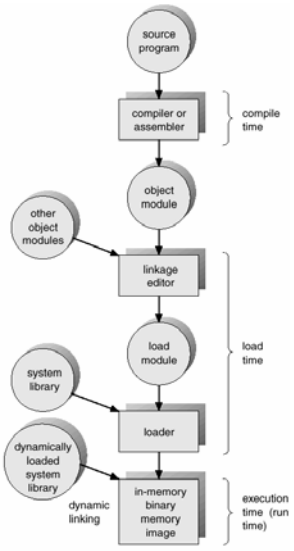
Linker

Módulo-carregável
- **módulo-objecto + rotinas da biblioteca**
- **endereços recolocáveis referidos ao início do módulo**

Loader

Imagem binária em memória

Criação de um Programa Executável



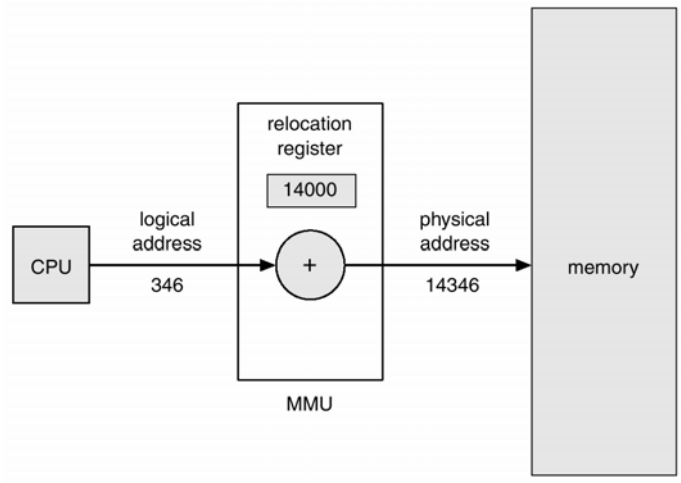
Recolocação

Capacidade de carregar e executar um dado programa num lugar arbitrário da memória

Formas de recolocação:

- **estática**
  - » **antes ou durante o carregamento do programa**
    - **antes → na compilação ou na ligação (*linking*)**
      ⇒ conhecer *a priori* onde o processo vai ser carregado
      (ex: programas com a extensão .COM em MS-DOS)
    - **durante → feita pelo *loader***
      o compilador gera código recolocável
  - » **o programa não pode deslocado na memória (difícil o *swapping*)**
- **dinâmica**
  - » **durante a execução do programa**
  - » **o processo pode ser deslocado na memória durante a execução**
  - » **⇒ suporte de *hardware* → *base register* (*s)***
    conteúdo = endereço físico inicial do programa ou de um segmento do programa

Recolocação



Ligação *(Linking)*

Juntar um conjunto de módulos-objecto produzindo um módulo contendo o programa global e os dados a serem passados ao *loader*.

A ligação *(linking)* pode ser feita:

- **estaticamente**
- **dinâmicamente**

Ligação estática:

- **cada módulo-objecto, compilado ou assemblado é criado com referências relativas ao início do módulo;**
- **todos os módulos são colocados num único módulo recolocável com referências relativas ao início do módulo global.**

Ligação *(Linking)*

Ligação dinâmica

- **durante o carregamento *(load-time)***
  - » **O módulo carrregável é lido para memória e qualquer referência a um módulo externo faz com que o *loader* carregue este módulo e altere as referências à memória necessárias.**
  - » **Vantagens:**
    - **fácil incorporar versões novas ou alteradas do módulo externo sem recompilar**
    - **fácil partilhar código entre várias aplicações (basta um cópia de cada módulo)**
- **durante a execução *(run-time)***
  - » **Parte da ligação é adiada até à altura da execução.**
  - » **Quando é feita referência a um módulo ausente, o S.O. localiza-o, carrega-o e liga-o ao módulo que o invocou.**
  - » **Vantagens:**
    - **permite alterar rotinas da biblioteca sem recompilar os programas ;**
    - **permite que uma única cópia de uma rotina seja partilhada por diferentes processos .**

Carregamento *(Loading)*

Operação de colocação de um módulo carregável em memória.

Vários tipos de carregamento

- **Absoluto**
  - » **O programa é carregado sempre no mesmo endereço inicial .**
  - » **Todas as referências à memória devem ser absolutas .**
  - » **A atribuição de endereços às referências de memória e feita pelo programador, pelo *assembler* ou pelo compilador.**
- **Recolocável**
  - » **A decisão quanto ao sítio onde se carrega o programa é tomada na altura do carregamento.**
  - » **O *assembler* / compilador não produz endereços absolutos mas relativos ao início do programa.**
- **Dinâmico em *run-time***
  - » ***Swapping* ⇒ possibilidade de carregar o mesmo processo em zonas diferentes da memória .**
  - » **A geração de endereços absolutos não pode ser feita por altura do carregamento inicial.**
  - » **O endereço absoluto só é calculado quando a instrução é efectivamente executada ⇒ suporte de *hardware*.**

Técnica de *overlays*

Técnica que permitia (actualmente caiu em desuso) correr processos que ocupavam mais memória do que a memória física disponível.

Ideia:

- **Dividir o programa numa parte residente (sempre em memória) e em *overlays* que são módulos independentes que são carregados em memória a pedido do programa.**
- **A comunicação entre os *overlays* é feita através da parte residente.**

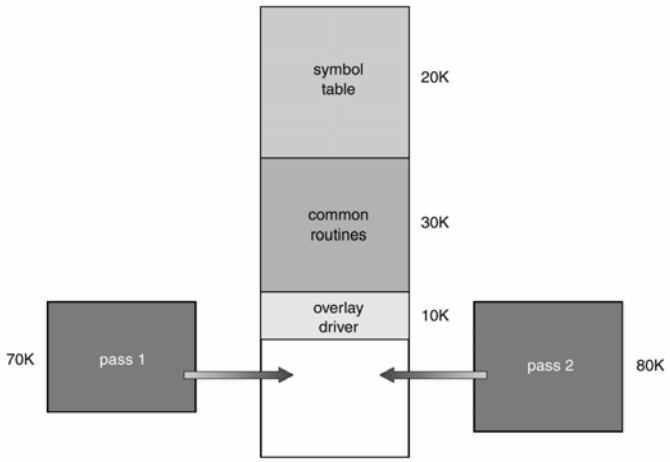
⇒ *overlay driver*

Dificuldades

- **(dimensão da parte residente + dimensão dos *overlays*) < dimensão da memória**
- **Dividir certos programas (compete ao programador fazê-lo)**

Alguns compiladores facilitavam a tarefa do programador.

Técnica de *overlays*



***Assembler* de 2 passos**

Endereços reais e Endereços virtuais

Endereçamento real

- **O endereço indicado pelo programa é exactamente o que é acedido na memória do computador (endereço físico), sem qualquer transformação operada pelo *hardware*.**
- **Desvantagens:**
  - » **dimensão dos programas limitada à dimensão da memória física (técnica de *overlay* permitia ultrapassar esta limitação)**
  - » **o programa só pode funcionar nos endereços físicos para que foi escrito**
  - » **multiprogramação difícil / impossível**

Endereçamento lógico ou virtual

- **Os endereços gerados pelo programa são convertidos pelo processador (pela *MMU-Memory Management Unit*), durante a execução, em endereços físicos.**
- **A palavra referenciada pelo endereço virtual pode estar em memória principal ou secundária (⇒carregá-la previamente)**

Endereços reais e Endereços virtuais

Endereço lógico / virtual
- endereço gerado pela *CPU*

Endereço físico / real
- endereço visto pela unidade de memória (carregado no *memory address register*)

Espaço de endereçamento lógico
- conjunto de todos os end.os lógicos gerados por um programa

Espaço de endereçamento físico
- conjunto de todos os end.os físicos correspondentes àqueles end.os lógicos

O mapeamento entre os 2 espaços é feito em *run-time* pela *Memory Management Unit - MMU* .

Swapping

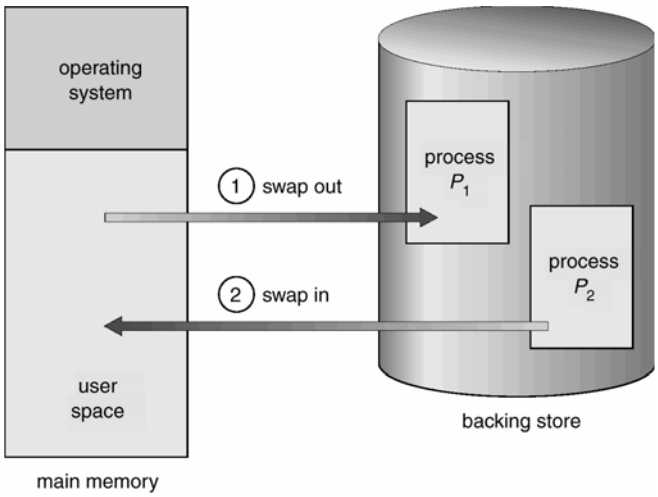
Swapping
- transferência de programas entre a memória principal e o disco

***Swapping* ⇒**
- possibilidade de recolocação
- comutação de contexto mais demorada

Swapper
- processo do S.O.
  - » selecciona processo(s) que vai sofrer *swap-out* (processos bloqueados, processos c/baixa prioridade, …)
  - » selecciona processo que vai sofrer *swap-in* (baseado no tempo que passou em disco, prioridade,…)
  - » aloca e gere o espaço de *swapping*

Swap-file
- onde é guardada a imagem do processo *swapped-out*
  - » uma única p/ todos os processos, com tamanho fixo e acesso sem ser através do *file system* (mais rápido)
  - » uma p/ cada processo

Swapping



Protecção de memória

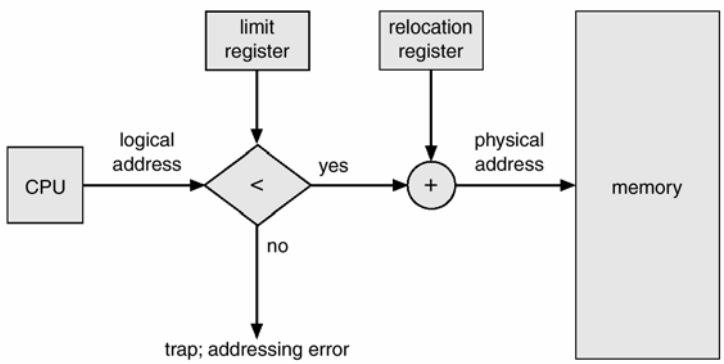
Cada processo (do S.O / do utilizador) deve ser protegido contra interferências indesejáveis de outros processos, acidentais ou intencionais.

Todas as referências de memória têm de ser verificadas em tempo de execução *(run-time)* ⇒

Suporte de *hardware*

- ***limit register***
  - » **conteúdo = endereço virtual mais elevado referenciado no programa**

Protecção de memória



Partilha de memória

Permitir que vários processos acedam à mesma zona de memória.

- **Processos que executam o mesmo programa devem ter a possibilidade de partilhar o código do programa.**
- **As bibliotecas partilhadas e as bibliotecas com ligação dinâmica são partilhadas por vários processos.**
- **Os processos também podem ter necessidade de partilhar dados.**

O *swapping* complica a partilha.

Para facilitar a partilha
as regiões de memória partilhada podem ser reservadas
no espaço de endereçamento do S.O.
e este passa a cada aplicação o endereço dessas regiões.

Técnicas de gestão de memória

Alocação contígua

- **Partição fixa**
  - » **partições de tamanho igual**
  - » **partições de tamanho diferente**
- **Partição dinâmica**

Alocação não-contígua

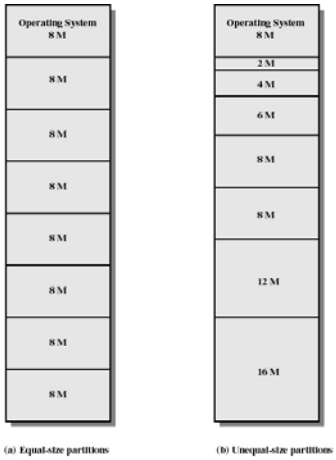
- **Paginação**
- **Segmentação**
- **Segmentação c/paginação**

Partição fixa

- **A memória destinada aos processos do utilizador é dividida em partições de tamanho fixo (eventualmente diferentes entre si).**
- **O S.O. mantém uma tabela com indicação das partições ocupadas.**
- **Inicialmente ... os programas eram compilados p/uma determinada partição ⇒**
  - **uma partição podia ter uma fila de programas à espera de poder executar enquanto outras filas estavam vazias**
- **Posteriormente ... possibilidade de recolocação**
  - **um programa pode ser carregado em qualquer partição**
- **Se necessário recorrer a *swapping*.**

Partição fixa

- **Partições de tamanho igual**
  - **Dificuldades**
    - » **um programa pode não caber nas partições ⇒ usar *overlays***
    - » **<u>fragmentação interna</u> - utilização ineficiente da memória quando o programa não ocupa a partição toda**
- **Partições de tamanho diferente**
  - **Ex.:**
    - » **1 partição de 1 MB**
    - » **1 partição de 768KB**
    - » **2 partições de 512KB**
    - » **1 partição de 320 KB**



Exemplo de partição fixa numa memória c/ 64 MB

Partição fixa

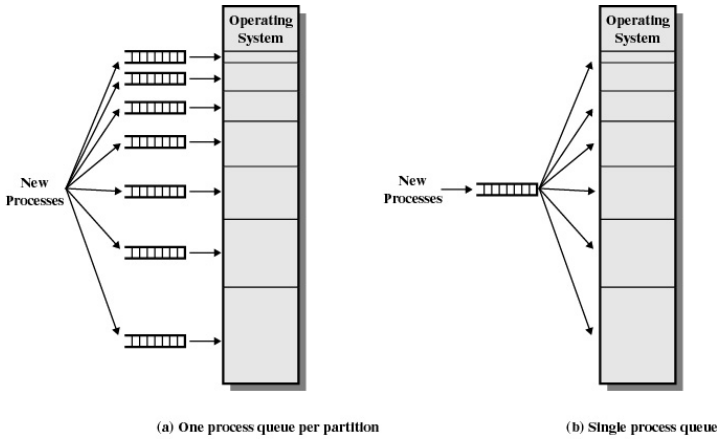
- **Mecanismo de protecção**
  - **par de registos onde são carregados os endereços máx. e min. da partição actual**
- **Algoritmo de colocação**
  - **tamanho igual - carregar o processo em qualquer partição disponível**
  - **tamanho diferente**
    - » **atribuir o proc. à menor partição em que ele cabe (mínimo desperdício) → uma fila por partição**

      <u>ou</u>
    - » **escolher a menor partição disponível capaz de conter o processo quando ele for carregado → uma fila única para todas as partições**

Desvantagens:

- **o nº de partições limita o nº de processos activos**
- **o tamanho das partições é fixado por ocasião da geração do sistema ⇒ utilização ineficiente da memória, q.do os processos são pequenos.**

Partição fixa



(a) One process queue per partition (b) Single process queue

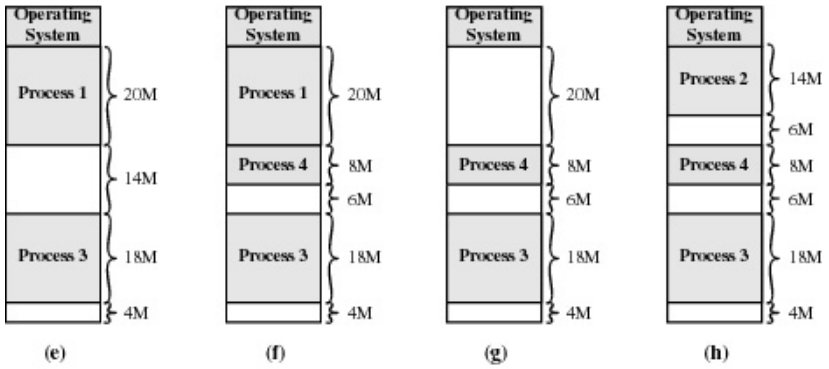
Algoritmos de colocação para partições fixas

Partição dinâmica

- **Inicialmente (q.do não há nenhum processo carregado) ...**
  → existe uma única partição, ocupando toda a memória.
- **Quando é executado um programa ...**
  → alocar zona de memória para o colocar.
- **Idem, para os programas seguintes.**

- **O nº da partições e o seu tamanho é variável**
- **Quando um processo termina, a memória é libertada e pode ser usada para carregar outro programa.**
- **Ao fim de algum tempo existirão fragmentos de memória não utilizada espalhados pela memória do computador (fragmentação externa).**
- **De tempos a tempos a memória terá de ser compactada.**

Partição dinâmica



Fragmentação externa

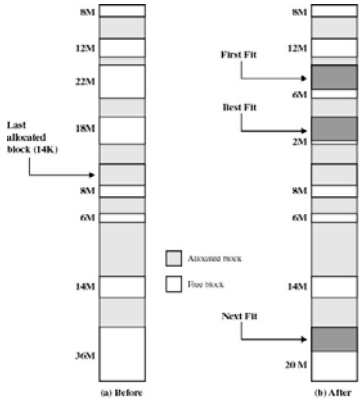
Partição dinâmica

- **Dimensão dos programas**
  - **(+) limitada pela memória física**
  - **(+) não é necessário parar o sistema p/ reconfigurar as partições**
- **Mecanismo de protecção**
  - **semelhante ao da partição fixa**
- **Algoritmo de colocação**
  - ***first-fit*** **- alocar o 1º bloco livre c/ tamanho suficiente**
    (começar pesquisa no 1º bloco livre)
  - ***next-fit*** **- alocar o 1º bloco livre c/ tamanho suficiente**
    (começar pesquisa no 1º bloco livre a seguir àquele em que terminou a últ. pesq.ª)
  - ***best-fit*** **- alocar o bloco livre mais pequeno que tenha tamanho suficiente**
    ⇒ pesquisar a lista de blocos livres toda
  - ***worst-fit*** **- alocar o bloco livre maior**
    (na expectativa de que o que sobra ainda tenha tamanho suficiente p/ ser útil)
  - ***buddy-system*** **- ir dividindo a memória livre, sucessivamente,**
    em blocos de tamanho 2^k (*buddies*- blocos em que se divide o bloco anterior)
    até ter um bloco livre em que o procº caiba c/ menor desperdício

Partição dinâmica

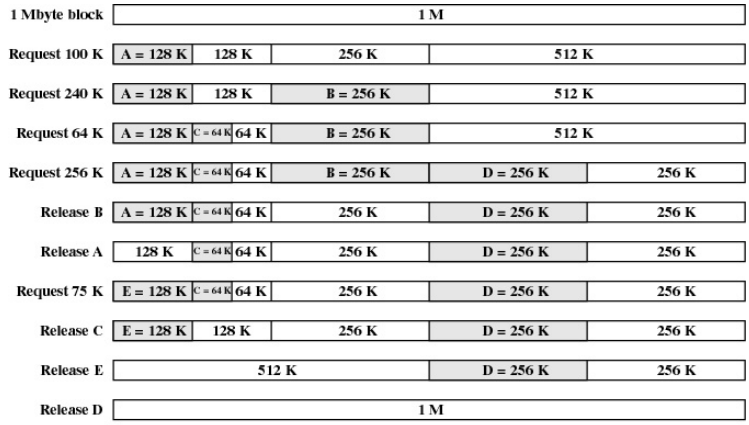
- **Algoritmo de colocação (cont.)**
  - **Qual o melhor ?**
    - » **Depende da sequência de *swapping* de processos e do seu tamanho.**
  - ***first-fit***
    - » **(+) o mais simples**
    - » **(+) usualmente o melhor e o mais rápido**
    - » **(-) usualmente dá origem a muitos blocos livres de pequena dimensão no início da memória**
  - ***next-fit*** **(Stallings,Tanenbaum)**
    - » **resultados de simulação indicam que é ligeiramente pior que o *first-fit* (Tanenbaum)**
  - ***best-fit***
    - » **(-) lento**
  - ***worst-fit***
    - » **(-) em geral, dá maus resultados (simulação)**
  - ***buddy-system*** **(Stallings,Tanenbaum)**
    - » **(+) fácil fazer a junção de 2 *buddies* livres contíguos**
    - » **(-) ineficiente em termos de utilização de memória (ex.: um proc. de 33kB ocupa um bloco de 64KB)**

Partição dinâmica



Configuração da memória antes e depois da alocação de um bloco de 16 MB usando diversos algoritmos de colocação

Partição dinâmica



Exemplo do *buddy-system*

Partição dinâmica

- **Algoritmo de substituição**
  - **quando não há memória livre para carregar um processo, (mesmo após compactação) que processo retirar da memória para ganhar espaço livre ?**
    (v. adiante, a propósito da memória virtual)
- **Problemas:**
  - **fragmentação externa, qualquer que seja o algoritmo de alocação usado**
  - **perda de tempo na gestão de buracos livres muito pequenos (=sem utilidade)**
    - » **ex.: bloco livre de 20000 bytes**
      um processo precisa de 19998 bytes
      solução: alocar pequenos buracos, juntamente c/ o pedido
  - **necessidade de compactação**
    - » **consome tempo**
    - » **⇒ capacidade de recolocação dinâmica**
    - » **difícil arranjar estratégia óptima**
      - **Compactar num único bloco ou em vários blocos grandes ?**
      - **Concentrar os blocos livres num dos extremos da memória ou minimizar os deslocamentos ?**
      - **Quando compactar, sempre que um processo termina ou só quando for necessário ?**

Estruturas de dados usadas na gestão de memória

Mapas de *bits*

- **Dividir a memória em blocos.**
- **A cada bloco é associado um *bit* que indica se ele está ocupado ou não.**
- **Tamanho da memória e dos blocos determinam o nº de *bits* necessários.**
- **Dificuldade : *overhead* necessário p/encontrar o nº de blocos livres consecutivos necessários p/carregar um programa.**

Listas ligadas

- **Manter uma lista duplamente ligada de blocos livres e ocupados (em geral por ordem crescente de endereços)**
- **Vantagem: fácil fazer a junção de 2 blocos livres contíguos, quando um processo liberta memória.**

Paginação

Mecanismos de gestão de memória anteriores:

- **A memória alocada a um processo é contígua.**
- **Problema : utilização ineficiente da memória**
  - » **partição fixa → fragmentação interna**
  - » **partição dinâmica → fragmentação externa**

<u>Paginação</u> :

- **o espaço de endereçamento físico de um processo pode ser não-contíguo**

<u>Objectivos da paginação</u>:

- **facilitar a alocação**
- **facilitar o *swapping***
- **reduzir a fragmentação da memória**

Paginação

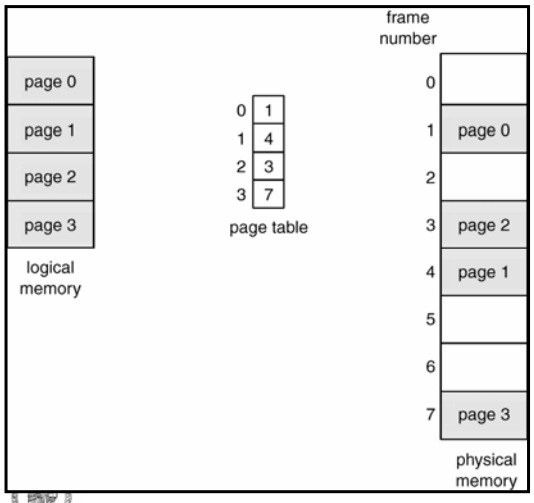
Método básico:

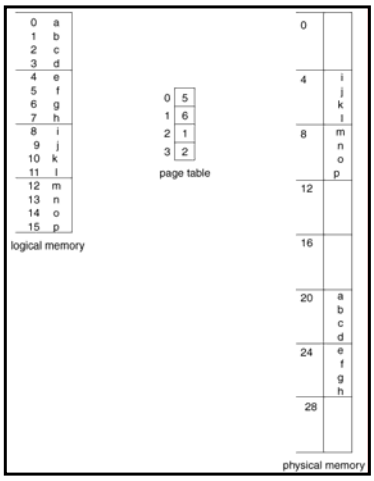
- **Dividir a memória física em blocos de tamanho fixo chamados quadros *(frames)*.**
- **Dividir a memória lógica em blocos de tamanho fixo chamados páginas.**
  - » **a dimensão das páginas é igual à dos quadros**
  - » **a dimensão das páginas depende da arquitectura da máquina**
  - » **algumas máquinas suportam vários tamanhos de página**
- **As páginas constituintes de um processo são carregadas em quaisquer quadros livres.**
- **O S.O. mantém uma tabela de páginas (*page table*), por cada processo, que estabelece a correspondência entre páginas e *frames*.**

***Nota*:**

- **O utilizador continua a ver a memória como um único espaço contíguo.**
- **O mapeamento entre os espaços de endereçamento lógico e físico está escondido do utilizador, sendo feito sob controlo do S.O. c/ o auxílio de *hardware* especial (*MMU*).**

Paginação





Exemplos

Paginação

***Hardware* de suporte**

- **Método geral de tradução de endereços**
  - » **Dividir o endº lógico pelo tamanho da página p/ determinar o nº da página**
  - » **Aceder à tabela de páginas p/ determinar o endº-base do quadro**
  - » **Adicionar o deslocamento (*offset*) dentro da página (resto da divisão anterior) ao endº-base do quadro, para obter o endº físico**

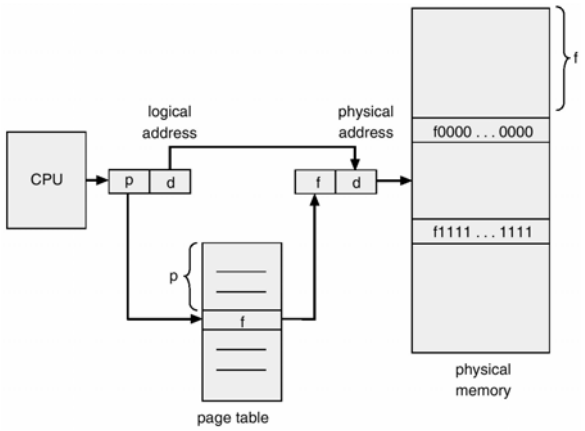
```
EndereçoFísico = TabelaPáginas [ EndereçoLógico DIV TamanhoPágina ] +
               + EndereçoLógico MOD TamanhoPágina
```

- **Na prática**
  - » **Usar tamanhos de página que sejam potências de 2**

    → possibilidade de usar *shifts* para fazer DIV e MOD ou (melhor !)
    → extrair directamente do endereço lógico os *bits* que formam o nº da página e o deslocamento

***Nota:* a paginação é uma forma de recolocação dinâmica**

Paginação



Tradução de endereços lógicos em endereços físicos

Paginação

Multiprogramação com paginação

- O S.O. mantém uma tabela de quadros (*frame table*) c/ a indicação dos quadros ocupados e livres.
- A partir do tamanho do ficheiro executável é determinado o nº de páginas necessário.
- O *long term scheduler* verifica se esse nº de páginas está disponível. Se estiver, constrói uma tabela de páginas p/ o novo processo à medida que carrega o programa.

Vantagens da paginação:

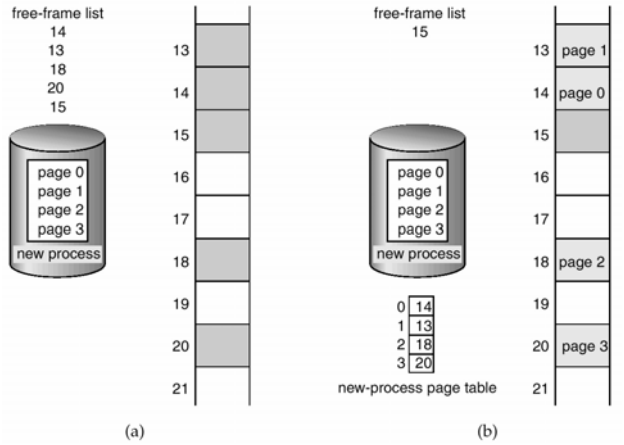
- A alocação é fácil
  - » manter uma lista de quadros livres e alocá-los por qualquer ordem;
  - » facilidade de *swapping* dado que tudo tem o mesmo tamanho
- Elimina a fragmentação externa

Paginação

Dificuldades da paginação

- Eficiência de acesso (*overhead* por cada referência à memória)
  - » Tabelas de página (mesmo q.do pequenas) são, em geral, demasiado grandes p/ carregar na memória rápida da *MMU*.
  - » Pode acontecer que as tab.s de página sejam mantidas em mem. principal e a *MMU* só tenha o endº-base da tabela.
- Espaço ocupado pela tabela
  - » Se as páginas forem pequenas o tamanho da tabela pode ser enorme
    - ex.: espaço de endereçamento de 32 *bits* (4GB= 2^{32}) c/ páginas de 4KB (=2^{12}) e 4 *bytes* por elemento da tabela tabelas de páginas c / 4MB (= $2^{32}/2^{12}*4$)
- Fragmentação interna
  - » Quando o tamanho do processo não é múltiplo do tamanho da página.
  - » Quanto maior for a página maior a fragmentação.
  - » Fragmentação média = 1/2 página.
- Aumento do tempo de comutação de contexto
  - » Necessário carregar a tabela de páginas do processo que vai correr e actualizar certos registos do *hardware*.

Paginação

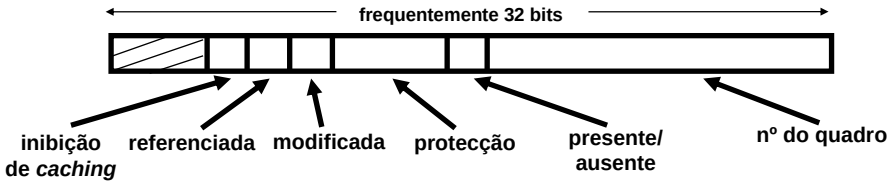


***Frames* livres**

Paginação

Implementação da tabela de páginas

- **Estrutura dos elementos da tabela de páginas**
  - » **Os campos de cada elemento variam, consoante o S.O. mas o tipo de informação presente é sensivelmente o mesmo.**



- **Nº do quadro**
  - **É o campo mais importante (imprescindível)**
- **Presente / ausente (1 *bit*)**
  - **Indica se esta entrada da tabela é válida ou não, isto é, se a página está ou não em memória (↔ memória virtual)**
  - **Uma tentativa de referenciar uma página inválida dá origem a um *trap* p/o S.O.**

Paginação

- **Modificada** (1 *bit*)
  - Indica se a página foi modificada ou não.
  - Importante p/ saber se a pág. tem de ser escrita em disco (↔ memória virtual)
- **Referenciada** (1 *bit*)
  - Indica se a página foi referenciada p/ leitura ou escrita
  - Importante p/ a gestão de memória virtual
- **Protecção** (1 ou mais *bits*)
  - Indica que tipo de acesso é permitido
  - Só 1 *bit* → 0 = Read/Write ; 1 = Read only
  - 3 *bits* → 1 *bit* = Read (enable/disable) ;
    1 *bit* = Write (enable/disable) ;
    1 *bit* = Execute (enable/disable) ;
- **Inibição de *caching*** (1 *bit*)
  - Importante p/ páginas que são mapeadas em registos de dispositivos e não em memória.
    Se o S.O. necessitar de aceder a um dispositivo de I/O é necessário inibir o *caching* de modo a que ele vá buscar a informação ao dispositivo e não à *cache* de memória.

Paginação

Implementação da tabela de páginas

- Varia de sistema operativo para sistema operativo.
- Muitos sistemas operativos usam uma tab. de páginas por processo. No *PCB* é guardado um apontador para a tabela de páginas.
- Se a dimensão da tab. de páginas for pequena
  - » Usar um conjunto de registos para manter a tabela de páginas.
  - » Ex.: DEC PDP-11 (anos 70)
    16 bits de endereço (64KB de memória)
    tamanho da página = 8KB
    ⇒ 8 entradas / tabela de páginas mantidas em registos de acesso rápido
- Se a dimensão da tab. de páginas for grande (a maior parte dos computadores contemporâneos)
  - » A tab. de páginas é mantida em memória principal.
  - » O *Page-Table Base Register* (*PTBR*) aponta p/ a tab. de páginas.
  - » Problema: cada acesso a uma posição de memória implica 2 ref.as físicas à memória.
  - » Solução : usar uma *cache* de acesso rápido onde é mantida informação acerca das páginas acedidas mais recentemente chamada *Translation Look-aside Buffer (TLB).*

Paginação

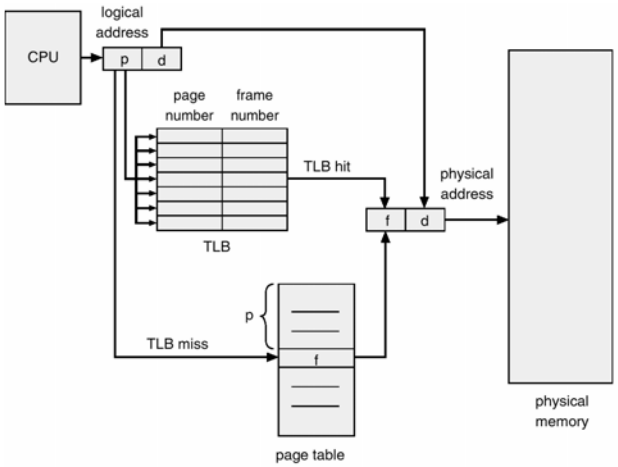
Translation Look-aside Buffer (*TLB*)

- Memória de tipo associativo.
- Cada registo da memória associativa tem 2 campos: uma chave e um valor.
- Quando é apresentado um item aos registos associativos ele é comparado com todas as chaves simultâneamente.
- Se o item for encontrado no campo chave, o valor correspondente é apresentado na saída.
- Se não for encontrado, isso é assinalado ao *hardware* de gestão de memória.

- De facto, a pesquisa na *cache* é lançada em paralelo c/ o acesso à tab. de páginas.
- Se a chave for encontrada na memória associativa, é interrompido o acesso à tabela de páginas.

- *Hit ratio* - percentagem de vezes que o nº da página é encontrada nos registos associativos. Esta percentagem, indicada pelo fabricante do processador é, em geral, muito próxima de 100% (ex: 98%)

Paginação



Paginação* com *TLB

Paginação

Quando as tabelas de páginas são muito grandes ...

» **Ex: endereços lógicos de 32 *bits* (4GBytes de memória) e páginas de 4Kbytes**
⇒ 2^{20} entradas na tabela de páginas

... existem várias soluções:

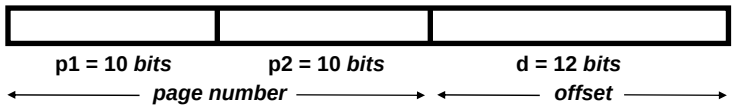
- **Armazenar as tabelas de páginas em memória virtual (cap. seguinte).**
- **Usar paginação multinível.**
- **Usar uma tabela de páginas *hashed.***
- **Usar uma tabela de páginas invertida.**

- **Armazenar as tabelas de páginas em memória virtual (cap. seguinte)**
  - **As próprias tab.s de páginas estão sujeitas a paginação (!)**
  - **Quando um processo está a executar apenas parte da tab. de páginas estará, em geral, em mem. principal.**

Paginação

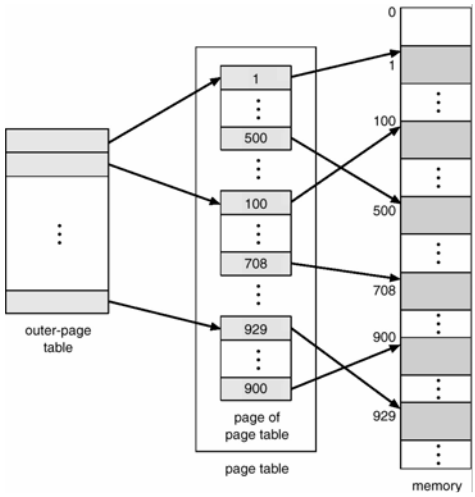
- **Usar paginação multinível**

» **Ex.:**



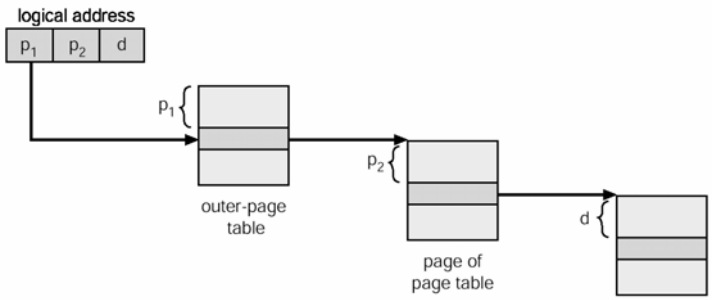
- **Existe um directório de tabelas de páginas com 2^{p1} elementos.**
- **Cada elemento aponta para uma tabela de páginas com 2^{p2} elementos.**
- **Em geral, o comprimento máximo de cada tabela de páginas não pode ser superior à dimensão de uma página.**
- **Vantagem : evitar ter todas as tab.s de páginas em memória, simultâneamente.**

Paginação



Paginação multinível (tabela de páginas c/ 2 níveis)

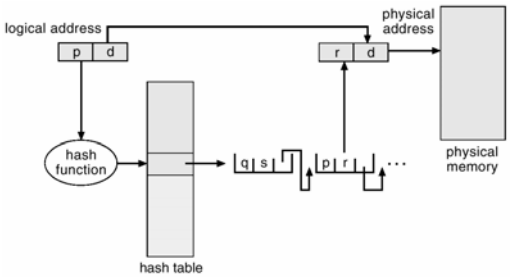
Paginação



Tradução de endereços em paginação multinível

Paginação

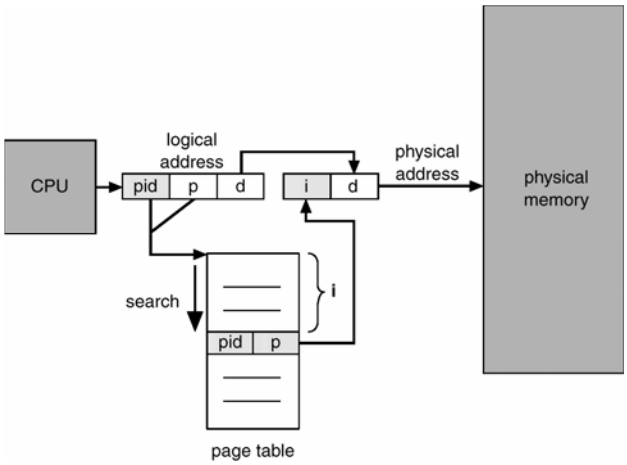
- **Usar uma tabela de páginas *hashed***
  - **Comuns quando o espaço de endereçamento é > 32 bits.**
  - **O número da página é convertido num índice da tabela de páginas.**
    Cada elemento da tabela de páginas aponta para uma lista de páginas que deram origem ao mesmo índice.
    Cada elemento da lista contém, para cada página o nº do quadro respectivo.
  - **A lista é percorrida até encontrar a página, obtendo-se então o nº do quadro**



Paginação

- **Usar uma tabela de páginas invertida**
  - **A tabela tem uma entrada por cada quadro da memória física.**
  - **A informação contida em cada elemento da tabela é:**
    - » **a *PID* do processo a que pertence o quadro**
    - » **o endº virtual da página que está actualmente no quadro.**
  - **Usa-se uma tabela de *hash* p/aceder aos elementos da tab. de pág.s invertida de forma rápida (alternativa: pesquisa sequencial)**
  - **Vantagem :**
    - » **só existe uma tabela de páginas no sistema e a sua dimensão é fixa e mais pequena do que a das tabelas convencionais.**
  - **Desvantagens :**
    - » **A tabela de páginas deixa de conter informação acerca do espaço de endereçamento lógico de um processo (necessária q.do a página referenciada não está em memória) ⇒ manter uma tab. de pág.s convencional, por cada processo, em mem. secundária**
    - » **Aumento do tempo de acesso à memória (devido ao acesso intermédio à tabela de *hash* ou a pesquisa sequencial) Solução : usar memória associativa p/ manter informação acerca dos acessos mais recentes.**

Paginação



Tradução de endereços usando uma tabela de páginas invertida

Paginação

Tamanho da página

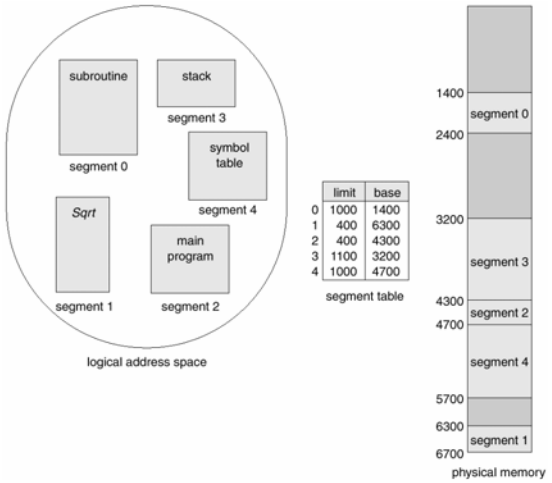
- **Decisão do *designer* do *hardware*.**
- **Factores a considerar:**
  - » **Fragmentação interna**
    - **diminui quando o tamanho da página diminui**
  - » **Nº de páginas / processo ↔ dimensão da tabela de páginas**
    - **o nº de pág.s / proc.º aumenta quando o tamanho da página diminui**
    - **(em sistemas c/ mem. virtual)**
      tab.s de pág.s muito grandes podem implicar uma dupla falta de página
      - uma por falta da parte da tab. de páginas necessária
      - outra por falta da página necessária
  - » **Taxa de falta de páginas (↔ mem. virtual, princípio da localidade)**
    - **pág.s pequenas - taxa baixa**
    - **pág.s grandes - taxa elevada (mas ...pág.s muito grandes albergam o proc.º todo → taxa nula !)**
    - ***Nota*:**
      a taxa depende não só da dimensão das páginas
      mas também do nº de quadros / processo
      (q.do este aumenta a taxa de falta de páginas diminui)

Segmentação

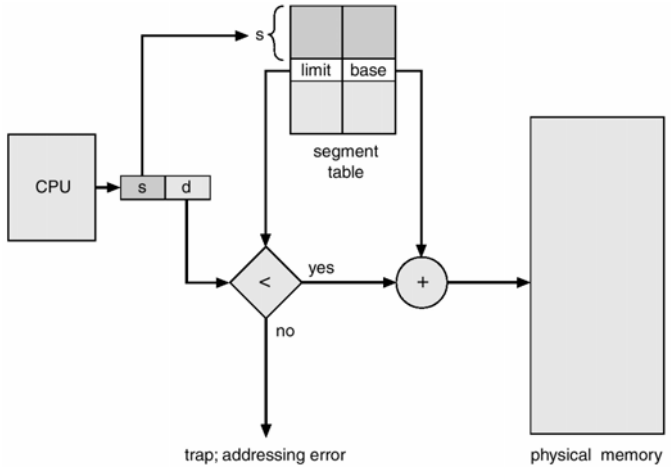
Método básico:

- Dividir o programa e os dados em partes de tamanho diferente (segmentos).
- Um segmento é uma unidade lógica.
  - » Ex.: uma função, um procedimento, as variáveis globais, a *stack*, ...
- Um endereço lógico é constituído por um par <nº do segmento, deslocamento> .
- Os segmentos são carregados em blocos de memória livres, não necessariamente contíguos.
- A tabela de segmentos (uma por cada processo) faz o mapeamento entre os endereços lógicos e os endereços físicos.
- Cada entrada da tabela de segmentos contem:
  - » endereço inicial do segmento
  - » comprimento do segmento
- A tradução de um endereço lógico num endereço físico é feita do seguinte modo:
  - » Extrair, do endereço lógico, o número do segmento (*bit*s mais significativos).
  - » Aceder à tabela de segmentos, usando este número, p/ obter o endereço físico do início do segmento
  - » Comparar o deslocamento (*bit*s menos significativos do endereço lógico) com o comprimento do segmento; se aquele for maior do que este o end.º é inválido.
  - » Endereço físico = endereço físico inicial do segmento+ deslocamento.

Segmentação



Segmentação



Tradução de endereços lógicos em endereços físicos

Segmentação

- A alocação de memória pode ser feita usando um dos métodos estudados na alocação contígua, dinâmica (*first-fit*, *best-fit*, ...).
- A recolocação é feita dinamicamente, recorrendo à tabela de segmentos.
- A paginação é invisível para o programador. A segmentação é usualmente visível.
  O programador ou o compilador coloca o programa e os dados em segmentos diferentes.

Fragmentação da memória:

- Evita a fragmentação interna
- Conduz a fragmentação externa.

Segmentação

Vantagens:

- **Elimina a necessidade de alocação contígua de todo o espaço de endereçamento de um processo (também a paginação).**
- **Facilita a protecção, através de *bits* de protecção associados a cada segmento.**
- **Facilita a partilha.**
  - » **Segmentos partilhados (ex. código) podem ser mapeados no espaço de endereçamento de todos os processos que estão autorizados a referenciá-los.**

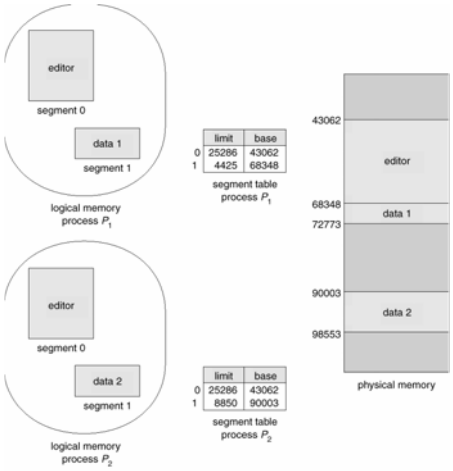
    ***Nota*: é preciso cuidado c/o *swaping* de um segmento partilhado p/vários processos.**
  - » **Partilha de código → poupança de memória**

Desvantagens:

- **Necessidade de compactação.**
- **Necessidade de acessos adicionais à memória p/ obter os endereços físicos (também na paginação).**

A utilização de segmentação simples é cada vez mais rara.

Segmentação



Partilha de segmentos

Segmentação com Paginação

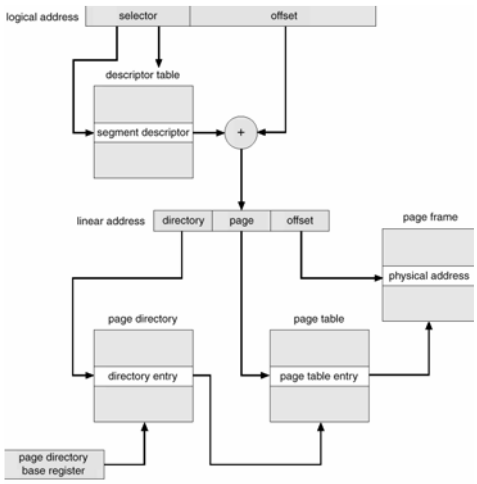
A paginação foi usada para resolver os problemas da partição dinâmica. Porque não aplicar a paginação aos segmentos ?

Método básico:

- **O programador / compilador divide o espaço de endereçamento em segmentos.**
- **Cada segmento é dividido em páginas de tamanho fixo (=tamanho dos quadros da memória física) .**
- **O deslocamento dentro do segmento traduz-se em nº de página + deslocamento dentro da página**
- **Cada segmento tem uma tabela de páginas associada.**

Combina vantagens da paginação e da segmentação.

Segmentação com Paginação



Tradução de endereços no Intel 80386