MEMÓRIA VIRTUAL

- **Introdução**
- ***Demand paging* / Paginação a pedido**
- **Performance da paginação a pedido**
- **Substituição de páginas**
- **Algoritmos de substituição de páginas**
- **Alocação de *frames***
- ***Thrashing***
- ***Demand segmentation* / Segmentação a pedido**

Memória virtual

- **Limitação importante dos mecanismos de gestão de memória descritos anteriormente:**
  - **todo o espaço de endereçamento lógico de um processo deve estar em memória física, simultaneamente.**
- **Isto pode ter um efeito adverso no grau de multiprogramação dado que pode limitar o nº de processos que podem correr simultaneamente.**
- **No entanto, os programas não necessitam de aceder a todo o s/espaço de endereçamento simultaneamente:**
  - **há partes do programa que raramente são executadas**
  - **há dados que raramente/nunca são acedidos**
- **Além disso verifica-se normalmente que as referências ao programa (instruções) e dados tendem a ser localizadas em períodos curtos de tempo (princípio da localidade de referência).**

Memória virtual

Memória virtual:

- **Técnica que permite a execução de processos que podem não estar completamente em memória principal.**
  - » **Um processo pode executar com apenas parte do s/espaço de endereçamento lógico carregado na memória física**
- **O espaço de endereçamento lógico pode pois ser muito maior do que o espaço de endereçamento físico.**
  - » **O utilizador / programador "vê" uma memória potencialmente muito maior - memória virtual - do que a memória real.**

O que é necessário:

- **Divisão de um processo em páginas ou segmentos.**
- **Tradução dos endereços virtuais em endereços reais executada pelo (S.O.+*HARDWARE*) em *run-time*.**
- **Mecanismo de transferência do conteúdo da memória lógica (em disco) para a memória física, à medida que for necessário (*swapping* incremental).**

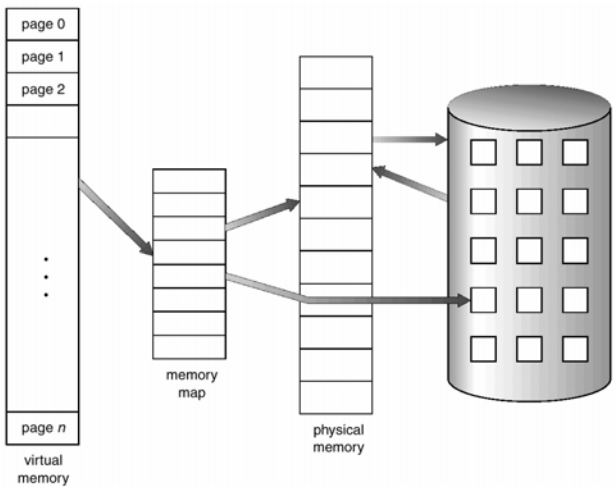


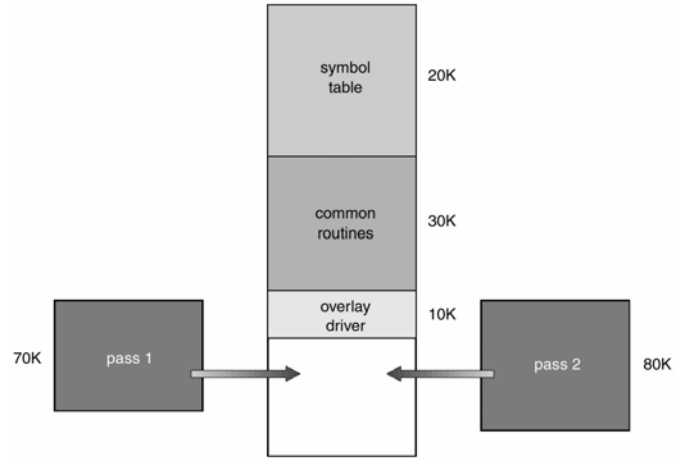
Diagrama mostrando memória virtual maior do que a memória física

Memória virtual

Overlays - técnica de memória virtual usada antigamente

- **Parte da memória era reservada p/uma secção de *overlay*.**
- **Partes do programa, identificadas pelo programador, são compiladas e *linkadas* de modo a poderem correr nos endereços da secção de *overlay*.**
- **Um *overlay driver* (sob controlo do programa) carrega diferentes *overlays* da memória secundária p/ a secção de *overlay*.**
- **O carregamento é feito dinâmicamente: os procedimentos e dados são trazidos p/ memória quando necessário, através de código gerado pelo compilador**
  (ex: a chamada a uma função testa primeiro se ela está em memória)
- **Problema:**
  - » **Os *overlays* não podiam referenciar-se mutuamente.**

Técnica de *overlays*



***Assembler* de 2 passos**

Demand paging / Paginação a pedido

Paginação a pedido

- A maior parte dos S.O's modernos são baseados em paginação a pedido (por necessidade ou por exigência).
- Semelhante à paginação convencional excepto que as páginas só são transferidas p/ a memória principal quando são necessárias.
  Pode conduzir a
  - » redução de *I/O*
  - » resposta mais rápida (só se carregam as páginas necessárias)
  - » redução da memória necessária por processo
  - » maior grau de multiprogramação
  - » mais utilizadores
- Quando é referenciada uma página (um endereço de memória)
  - » se a referência é inválida ⇒ abortar
  - » se a referência é válida e a página não está em memória ⇒ trazê-la p/ memória

Paginação a pedido

Bit de página válida/inválida (ou presente/ausente)

- Cada entrada da tabela de páginas tem um *bit* que indica se a página em questão está ou não em memória (ex.: 1 = presente / 0 = ausente).
- Durante a tradução de endereço (lógico→físico) se o *bit* estiver a 0 ⇒ falta de página .

Falta de página (→ *trap* p/ o S.O.) ⇒

- Verificar na tabela de páginas se a referência é válida ou inválida.
- Referência inválida ⇒ abortar
  Referência válida ⇒ continuar (trazer a página p/ mem. principal) .
- Obter um *frame* livre .
- Ler a página necessária .
- Actualizar a tabela de páginas c/ indicação de que a pág. está em memória, e em que *frame* está.
- Recomeçar a instrução interrompida devido à falta de página.

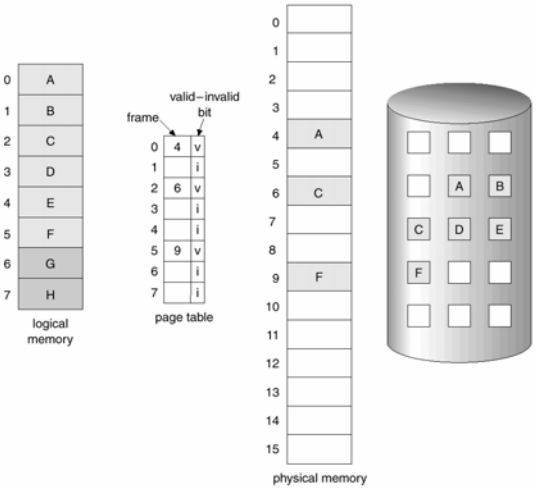
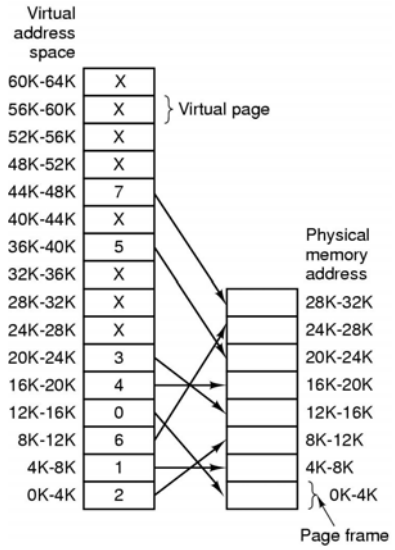
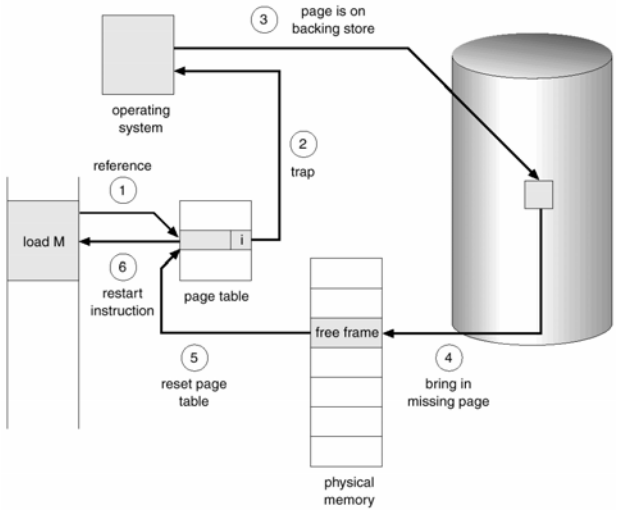


Tabela de páginas quando algumas páginas não estão na memória principal



Relação entre endereços virtuais e endereços físicos dada pela tabela de páginas



Passos no tratamento de uma falta de página

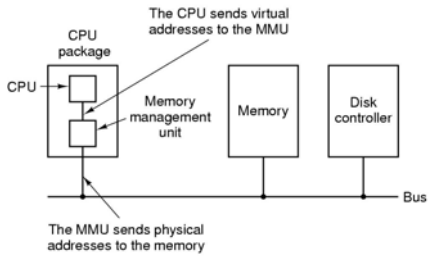
Paginação a pedido

O que acontece se não houver *frames*/quadros livres ?

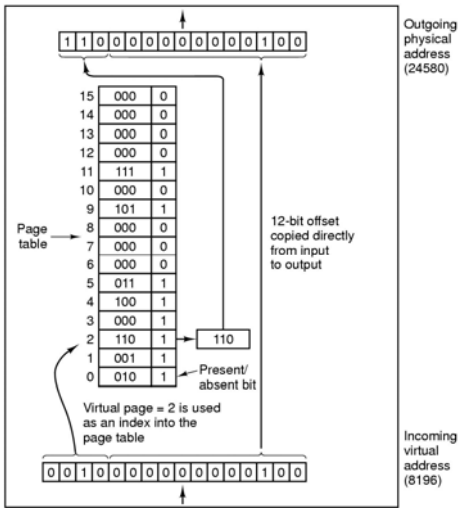
- **Substituição de página -**
  encontrar uma página em memória que não esteja a ser utilizada e fazer o *swap out* dessa página ⇒
  - » **algoritmo de substituição de página que resulte num número mínimo de faltas de página (ver adiante)**

Suporte de *hardware* necessário

- **Tabela de páginas c/ *bit* de página válida / inválida.**
- **Possibilidade de recomeçar uma instrução que falhou devido a uma falta de página ⇒**
  - » **Guardar o estado inicial de uma instrução e repô-lo após o *trap*.**
  - » **Por vezes os processadores guardam um estado de execução parcial e continuam a instrução onde ela foi interrompida.**
  - » **Dificuldade principal: instruções que movimentam blocos de dados.**
- **Disco rápido p/ guardar as páginas que não estão em memória.**



Posicionamento e função da *MMU - Memory Management Unit*



Funcionamento interno de uma *MMU* num sistema com 16 páginas de 4 KB

Performance da paginação a pedido

- **A paginação a pedido pode implicar uma degradação significativa da performance do computador.**

Tempo efectivo de acesso à memória, T_{eam}

$$T_{eam} = (1-p) \times T_{am} + p \times T_{fp}$$

p = taxa de falta de páginas $0.0 \leq p \leq 1.0$
p=0, não ocorrem;
p=1, todas as referências conduzem a falta de página

T_{am} = tempo de acesso à memória

T_{fp} = tempo que o S.O. demora a processar uma falta de página

- **O princípio da localidade de referência indica que p deve ser próximo de 0.**
- **Contudo T_{fp} pode ter um efeito significativo, pois é muito superior a T_{am} .**

Performance da paginação a pedido

Estimação de T_{fp} (tempo de processamento de uma falta de página):

- **tempo de processamento de uma interrupção**
- **tempo de carregamento da página**
- **custo do recomeço da instrução**

Exemplo:

p = 1/1000 (1 falta de pág. em cada 1000 ref.as à memória); T_{am} = 100 ηs; T_{fp} = 25 ms

$\Rightarrow T_{eam} = (1-1/1000)\times 100 + 1/1000\times 25000000\ \eta s \approx 100 + p \times 25000000\ \eta s$

Degradação < 10% ⇒

$100 + 10 > 100 + 25000000\ p \Rightarrow p < 4\times 10^{-7}$ (uma falha em 2.5×10^6 ref.as)

T_{am} $10\%\times T_{am}$ $(1-p)\times 100 \approx 100$

Performance da paginação a pedido

Tempo de acesso a disco

- **O acesso ao espaço de *swap* é em geral mais rápido do que o acesso a um ficheiro normal**
  - » **O acesso é feito directamente, e não através do sistema de ficheiros.**

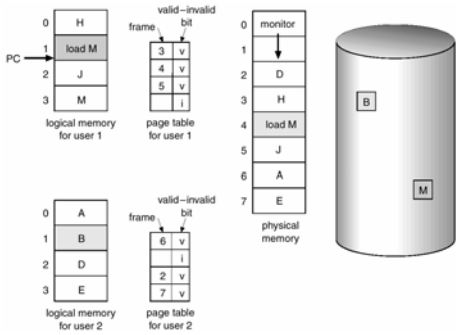
Pode-se melhorar a velocidade de acesso às páginas copiando a imagem do ficheiro p/ o espaço de *swap*, inicialmente, e executar o carregamento das páginas a partir daí.

Outras opções:

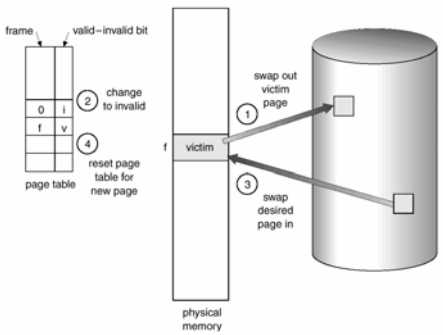
- » **Se o espaço de *swap* for limitado o código do programa pode ser sempre carregado a partir do ficheiro. Quando houver necessidade de substituir uma página de código não há necessidade de a escrever.**
- » **Inicialmente, ir buscar as páginas usando o sistema de ficheiros. As páginas que forem substituídas vão p/ o espaço de *swap*. Se estas forem necessárias posteriormente são carregadas do espaço de *swap*.**

Substituição de páginas

- **Como proceder quando uma página necessitar de ser carregada em memória e não houver nenhum *frame* livre ?**
  - **proceder ao *swap out* de todo um processo**
  - **libertar um *frame* (a melhor opção)**
- **Se a página que está no *frame* libertado tiver sido modificada desde o s/ último carregamento ⇒ escrever a página no disco**
  - **a escrita no disco pode duplicar o T_{fp} (⇒ 1 escrita + 1 leitura)**
- **Se não tiver sido modificada (página de código ou *read only*) o *frame* pode ser usado à vontade.**
- **Um *dirty bit / modified bit*, na tabela de páginas, pode ser usado para saber se a página a retirar foi modificada**
  - **activado pelo *hardware* por cada escrita no *frame***



Necessidade de substituição de página



Substituição de página

Substituição de páginas

A substituição de páginas completa a separação entre a memória lógica e a memória física:

- **uma memória virtual grande pode ser fornecida com base numa memória física mais pequena**

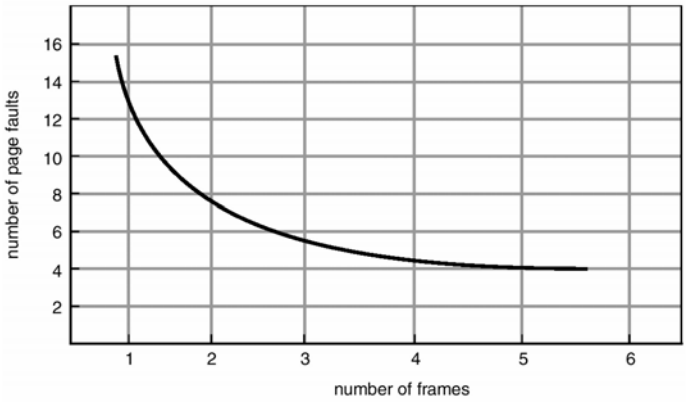
Decisões fundamentais a tomar

- **Quantos *frames* atribuir a um processo ?**
  (ALGORITMO DE ALOCAÇÃO DE *FRAMES*)
- **Que páginas substituir e quando ?**
  (ALGORITMO DE SUBSTITUIÇÃO DE PÁGINAS)

Algoritmos de substituição de páginas

- **Que algoritmo usar p/ seleccionar a página a substituir ?**
  - ***FIFO***
  - **Óptimo**
  - ***LRU* e *LRU* aproximado**
  - **Baseados em contagens (*LFU* e *MFU*)**
- **Como avaliar um algoritmo ?**
  - **Avaliam-se os algoritmos aplicando-os a uma determinada sequência de referências à memória e determinando o nº de faltas de página nessa sequência.**
  - **Sequência (basta o nº das páginas referenciadas)**
    - » **aleatória**
    - » **seguimento das referências de um sistema real**
  - **Determinação do número de faltas de página ⇒ conhecer o nº de *frames* disponíveis**

    Em geral, se
    nº de *frames* ↑ ⇒ nº de faltas de página ↓



Evolução do nº de faltas de página
em função do nº de *frames* atribuídos a um processo

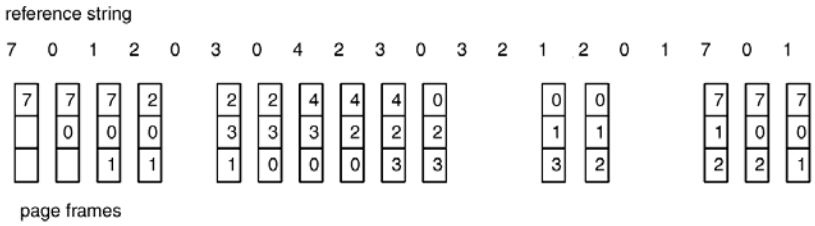
Algoritmos de substituição de páginas

ALGORITMO *FIRST-IN FIRST-OUT* (*FIFO*)

- **A página substituída é a que estiver há mais tempo em memória.**
- **Implementação eficiente ⇒**
  - » **manter uma lista *FIFO* com as páginas que estão em memória**
  - » **substituir a página à cabeça da lista**
  - » **inserir a página carregada no fim da lista**
- **Problema:**
  - » **uma página frequentemente referida pode ser retirada, se tiver sido carregada há muito tempo**

ALGORITMO *FIRST-IN FIRST-OUT* (*FIFO*)

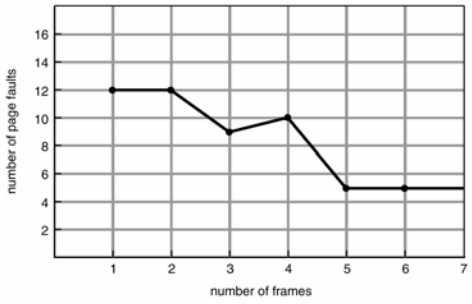
- **Exemplo:**
  - » **sequência: 70120304230321201701**
  - » ***frames*: 3**
  - » **faltas de página: 15**



Algoritmos de substituição de páginas

ALGORITMO *FIRST-IN FIRST-OUT* (*FIFO*) (cont.)

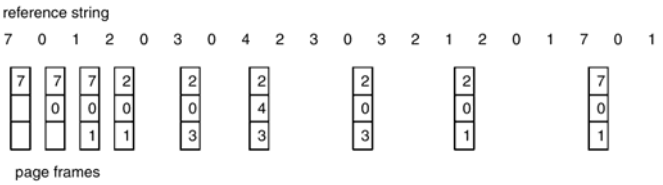
- **Performance**
  - » **Altamente influenciada pelo nº de frames disponíveis.**
  - » **Exemplo:**
    - **sequência anterior**
    - ***frames* = 4**
    - **faltas de página = 10**
- **Anomalia de Belady** →
  - » **Mais *frames* ⇏ Menos faltas de página**
  - » **Exemplo:**
    - **sequência: 123412512345**
    - **3 *frames* ⇒ 9 faltas de página**
    - **4 *frames* ⇒ 10 faltas de página**



Algoritmos de substituição de páginas

ALGORITMO ÓPTIMO

- **Substituir a página que não será usada por um período de tempo mais longo, no futuro (!!!).**
- **Problema:**
  - » **Como saber isso ?!**
- **Exemplo:**

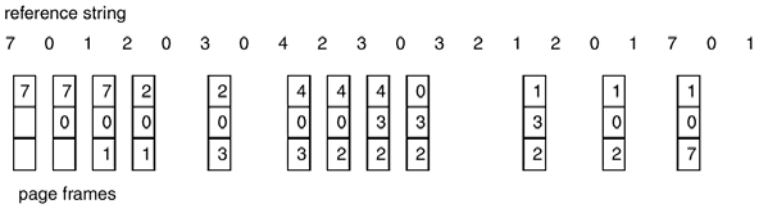


- **Algoritmo frequentemente usado para avaliar a performance de outros algoritmos.**

Algoritmos de substituição de páginas

ALGORITMO *LEAST-RECENTLY-USED* (*LRU*)

- **A página substituída é a que não foi referenciada há mais tempo (a página usada menos recentemente)**
- **É uma aproximação ao algoritmo óptimo**
  - » **usar o comportamento passado p/ prever o futuro**
- **Exemplo:**



Algoritmos de substituição de páginas

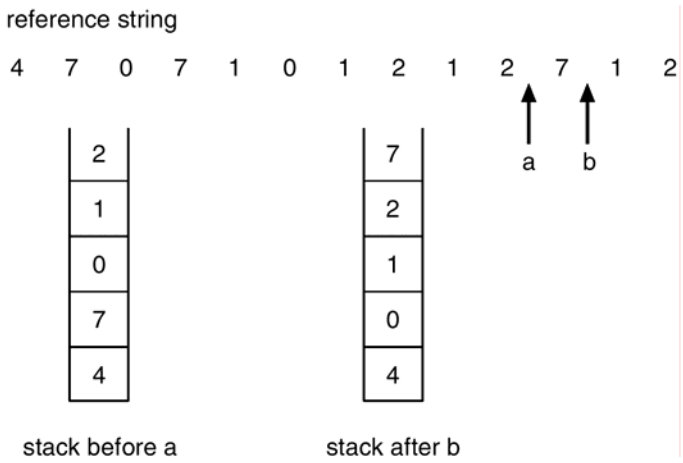
ALGORITMO *LEAST-RECENTLY-USED* (*LRU*)

- **Algoritmo usado frequentemente (/ algoritmos que o aproximam). Comportamento bastante bom.**
- **Problema:**
  - » **manter e actualizar o tempo em que uma página foi referenciada.**
    ⇒ gastos de tempo e de espaço

Algoritmos de substituição de páginas

ALGORITMO *LEAST-RECENTLY-USED* (*LRU*) **(cont.)**

- **Implementação**
  - » **Baseada em contadores**
    - **Associar um campo "tempo" a cada página da tabela de páginas.**
    - **De cada vez que a página é referenciada, copiar um *clock* p/ aquele campo.**
    - **Substituir a página com o "tempo" mais baixo (referênciada há mais tempo).**
    - **Problemas:**
      - – **Pesquisa p/ encontrar a página a substituir.**
      - – **Actualização do campo "tempo" por cada referência à memória.**
  - » **Baseada numa *stack***
    - **Manter uma lista duplamente ligada c/ os números das páginas referenciadas.**
    - **De cada vez que uma página é referenciada, colocá-la no topo da lista**
    - **Problema: actualização de diversos apontadores por cada ref.a à memória.**
    - **Vantagem: a decisão da página a substituir é rápida (pág. do fim da lista).**



Utilização de uma *stack*
para guardar as páginas referênciadas mais recentemente

Algoritmos de substituição de páginas

ALGORITMOS QUE APROXIMAM O ALGORITMO *LRU*

- **A implementação do algoritmo *LRU* é "pesada".**
  Poucos sistemas implementam o *LRU* real.
- **Aproximações:**
  - » **Utilizando o *bit* de referência.**
  - » **Utilizando o algoritmo da 2ª oportunidade ou do relógio.**
  - » **Utilizando o algoritmo da 2ª oportunidade melhorado.**

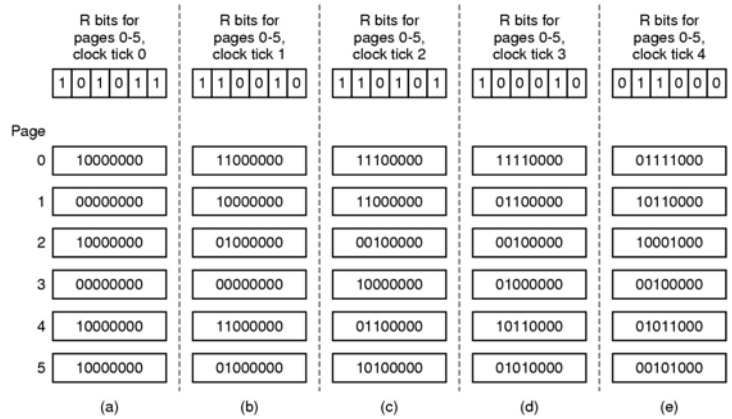
APROXIMAÇÃO AO *LRU* USANDO O *BIT* DE REFERÊNCIA

- **Algoritmo:**
  - » **Associar a cada página um *bit* de referência (inicialmente=0).**
  - » **Quando uma página é referenciada, colocar o *bit* =1 .**
  - » **Substituir uma página com o *bit* = 0 (se existir)**
    mas impedir a substituição de uma pág. carregada recentemente
- **Dificuldade:**
  - » **Não se sabe a ordem pela qual as páginas foram referenciadas.**

Algoritmos de substituição de páginas

APROXIMAÇÃO AO *LRU* USANDO O *BIT* DE REFERÊNCIA (cont.)

- **Variante (melhoria do algoritmo anterior):**
  - » **Usar um registo de *bits* de referência (ex: 8 *bits*) por cada página da tabela de páginas.**
  - » **Com intervalos regulares introduzir o *bit* de referência de cada página no *bit* mais significativo do registo respectivo, deslocando o s/ conteúdo p/ a direita, e limpar todos os *bits* de referência da tabela de páginas.**
  - » **Considerando o conteúdo do registo como um número em binário a página com o número mais baixo é a que foi acedida há mais tempo.**
- **Dificuldades desta variante:**
  - » **Perde-se o que se passa entre 2 *ticks* de *clock*.**
  - » **Perde-se as referências feitas há mais tempo do que *N×Intervalo*, em que N=nº de *bits* do registo**

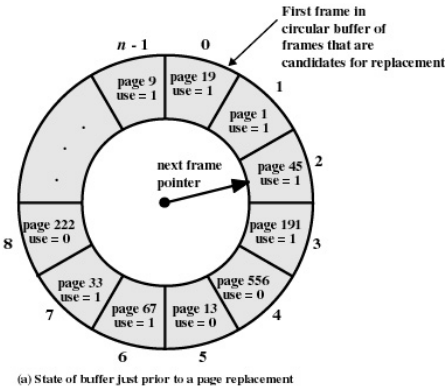


Aproximação ao *LRU*
utilizando um registo de *bits* de referência

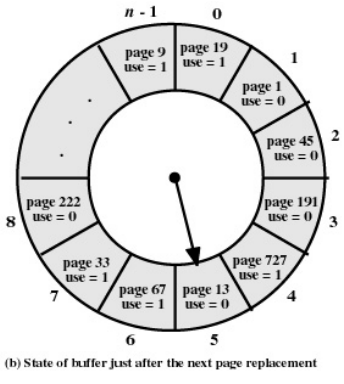
Algoritmos de substituição de páginas

APROXIMAÇÃO AO *LRU* USANDO O ALGORITMO DA 2ª OPORTUNIDADE OU DO RELÓGIO

- **Algoritmo:**
  - » **Manter os *frames* numa lista circular.**
  - » **Quando uma página é carregada pela 1ª vez o *bit* de referência é colocado em 1.**
  - » **Quando é necessário substituir uma página, percorrer os *frames* circularmente e a qualquer *frame* que tenha o *bit* de referência igual a 1 é dada uma 2ª oportunidade, colocando o *bit* de referência igual a 0, e avançando p/ o *frame* seguinte.**
  - » **Se o *bit* de referência ainda estiver em 0 na 2ª passagem, a página é substituída.**



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Algoritmo da 2ª oportunidade ou do relógio

Algoritmos de substituição de páginas

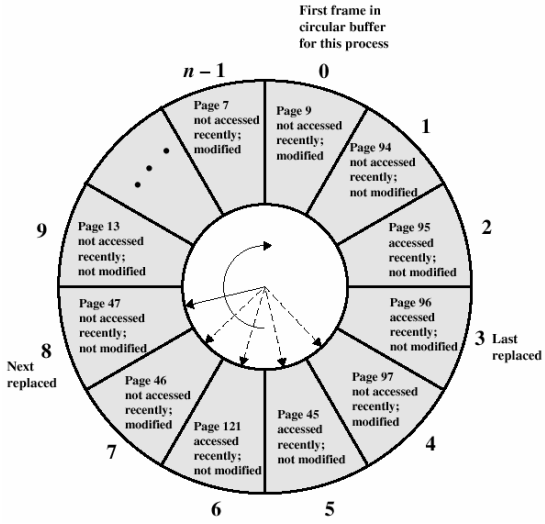
APROXIMAÇÃO AO *LRU* USANDO
O ALGORITMO DA 2ª OPORTUNIDADE MELHORADO

- **Usar 2 *bits* por cada *frame***
  - » **R - *frame* referenciado**
  - » **M - *frame* modificado**
- **4 classes possíveis de *frames***
  - » **R=0, M=0**
    - **página nem referenciada recentemente, nem modificada**
    - **corresponde a uma das melhores páginas para substituir**
  - » **R=0, M=1**
    - **esta página não é muito boa p/ substituir pois tem de ser escrita**
  - » **R=1, M=0**
    - **esta página, provavelmente (?), vai ser referenciada outra vez, a seguir**
  - » **R=1, M=1**
    - **esta página, provavelmente, vai ser referenciada outra vez, a seguir**
    - **e tem de ser escrita no disco, se for substituída**

Algoritmos de substituição de páginas

APROXIMAÇÃO AO *LRU* USANDO
O ALGORITMO DA 2ª OPORTUNIDADE MELHORADO (cont.)

- **Algoritmo**
  - » **1-**
    Percorrer a lista circular, começando na posição actual do apontador.
    Durante este percurso, não alterar o *bit* R.
    O 1º *frame* encontrado c/ (R=0,M=0) é seleccionado p/ ser substituído.
  - » **2-**
    Se o passo 1 falhar,
    percorrer a lista à procura de um *frame* c/ (R=0,M=1).
    O 1º *frame* encontrado é seleccionado p/ ser substituído.
    Durante esta passagem,
    fazer R=0 em todos os *frames* por onde passar.
  - » **3-**
    Se o passo 2 falhar,
    o apontador deve ter retornado à sua posição original.
    Voltar ao passo 1. Desta vez encontrar-se-á um *frame* p/ substituir.



Algoritmo da 2ª oportunidade melhorado

Algoritmos de substituição de páginas

ALGORITMOS BASEADOS EM CONTAGENS

- Manter um contador do nº de referências feitas a cada página.
- ALGORITMO *LFU - Least Frequently Used*
  - » Substituir a página c/ a contagem mais pequena
  - » Problema:
    - as páginas que foram muito usadas há muito tempo são mantidas em memória
  - » Solução:
    - técnica de envelhecimento: dividir a contagem por 2 c/ intervalos regulares
- ALGORITMO *MFU - Most Frequently Used*
  - » Substituir a página com contagem mais elevada (!!!)
  - » Pressuposto:
    - talvez as páginas c/ contagens mais baixas sejam mais recentes e venham a ser necessárias (?!)
- Estes algoritmos são pouco utilizados.

Buffering de páginas

- Usado c/ um qualquer dos algoritmos de substituição de páginas.
- Uma página substituída
  não é imediatamente retirada da memória interna
  é colocada numa de 2 listas (a página não é deslocada na memória):
  - lista de páginas livres (*FIFO*), se a página não foi modificada
  - lista de páginas modificadas (*FIFO*), se a página foi modificada
- Quando é preciso carregar uma página em memória
  usa-se o *frame* do início da lista de páginas livres.
- As páginas da lista de páginas modificadas
  são escritas em disco, quando o dispositivo de paginação estiver livre.
  - » Vantagem: podem ser escritas em grupos (mais rápido)
- Vantagem do *buffering*:
  - Possibilidade de evitar ir buscar uma página ao disco
    se ela ainda estiver no *buffer* (*cache*) de páginas substituídas.
  - Permite ler uma pág. do disco sem que a pág. substituída seja escrita no disco.

Alocação de *frames*

- Como distribuir o número fixo de *frames* da memória
  pelos vários processos ?
- Questões a resolver:
  - Nº mínimo de *frames* por processo ?
  - Algoritmo de alocação ? Alocação fixa ou variável ?
  - Alcance da substituição ? Local ou global ?
- Alguns factores a ter em conta:
  - Quando a quantidade de memória alocada a cada processo diminui
    - nº de processos em memória pode aumentar
    - probabilidade de encontrar um processo pronto a correr pode aumentar
    - *swapping* pode diminuir
  - Quando o nº de páginas de um processo em memória diminui
    - a taxa de falta de páginas pode aumentar
  - A partir de um certo tamanho,
    o aumento da memória alocada a um processo
    não tem nenhum efeito notável na taxa de falta de páginas.

Alocação de *frames*

NÚMERO MÍNIMO DE *FRAMES* POR PROCESSO

- **Cada processo necessita de um nº mínimo de *frames* em memória.**
- **O nº mínimo depende da arquitectura do processador:**
  - » **É o nº máximo de páginas que podem ser acedidas numa única instrução**
  - » **Exemplo:**
    - **Uma instrução MOVE,**
      que permite copiar um bloco de dados de uma zona de memória p/ outra,
      podendo os endereços ser indicados de forma indirecta
    - **a instrução ocupa 6 bytes …….. ⇒ pode abranger 2 páginas**
      end.º de origem indicado indirectamente ⇒ pode abranger 2 páginas
      endº de destino indicado indirectamente ⇒ pode abranger 2 páginas
      ----
      6 páginas
- **O nº máximo é limitado pela memória física.**

Alocação de *frames*

ALGORITMOS DE ALOCAÇÃO

- **Alocação fixa**
  - **O nº de *frames* alocados a um processo é fixo.**
  - **Quando é necessário substituir uma página,**
    é escolhido um dos *frames* pertencentes ao processo.
  - **O nº de *frames* é decidido quando o processo é carregado,**
    podendo ser determinado com base em:
    - » **tipo de processo (interactivo, *batch*, tipo de aplicação, …)**
    - » **informação do utilizador ou do gestor do sistema**
  - **Variantes:**
    - » **partição igual: cada processo recebe *int(NFrames/NProcs) frames***
    - » **partição proporcional: baseada no tamanho ou na prioridade dos processos**
- **Alocação variável**
  - **O nº de *frames* alocados a um processo pode variar**
    durante a sua existência, de acordo com
    - » **grau de multiprogramação (grau de multiprog. ↑ ⇒ nº pág.s / processo ↓)**
    - » **tamanho e/ou prioridade dos processos**
    - » **taxa de falta de páginas (taxa de falta de pág.s ↑ ⇒ nº pág.s / processo ↑)**

Alocação de *frames*

ALCANCE DA SUBSTITUIÇÃO

- **Ao seleccionar uma página p/ substituir, o algoritmo de substituição pode fazê-lo c/ alcance local ou global.**

- **Alcance local**
  - **Escolher a página a substituir entre as páginas residentes do processo que originou a falta de página.**
  - **Desvantagem:**
    - » **um processo pode "embargar" outros processos ao não ceder *frames* de que pode não estar a precisar.**

- **Alcance global (mais comum)**
  - **Qualquer uma das páginas residentes pode ser substituída, mesmo que pertença a outro processo.**
  - **Vantagem:**
    - » **um processo prioritário pode retirar páginas a outros.**
  - **Inconveniente:**
    - » **o conjunto de páginas de um processo em memória depende do comportamento dos outros processos (o tempo de execução pode variar muito de uma vez p/ outra).**

Thrashing

Thrashing

- **acontece quando um processo passa mais tempo em actividades de paginação do que a executar.**

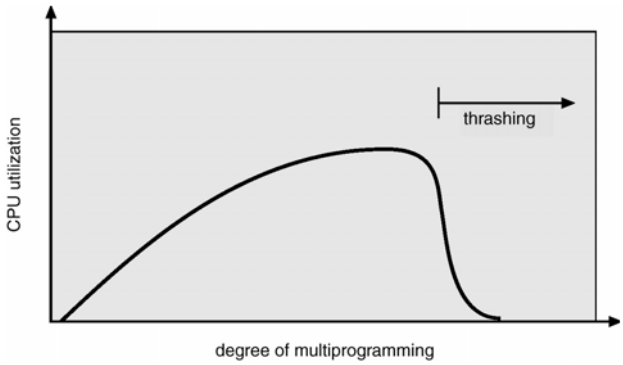
Causa

- **Uma má política de paginação.**
- **Se um processo não tiver páginas suficientes em memória a taxa de falta de páginas é muito elevada ⇒**
  - » **elevada actividade de transferência de páginas**
  - » **baixa utilização do processador**
  - » **baixa utilização de outros dispositivos**

  ← **sintomas de *thrashing***

  - » **o S.O. "pensa que" pode aumentar o grau de multiprogramação**
  - » **outro processo é acrescentado ao sistema**
  - » **a utilização do processador baixa ainda mais, …etc**
  - » **… colapso !!!**

- **O *thrashing* ocorre porque o número de páginas que são activamente usadas é superior ao tamanho total da memória.**



Thrashing

Thrashing

Solução

- **Fazer o *swap out* de um ou mais processos.**
  (decisão do *medium term scheduller*)
- **Retomar a sua execução quando houver mais memória livre.**
- **Por vezes, impõe-se limites mínimos ao tempo que um processo tem de estar em memória ou em disco.**
  - » **Um processo em estado executável tem de permanecer pelo menos T1 segundos em memória antes de ser *swapped out*.**
  - » **Um processo em disco tem de permanecer aí pelo menos T2 segundos antes de ser *swapped in*.**

Limitar os efeitos do *thrashing*

- **Algoritmo de substituição local de páginas**
  - » **Evita que se um processo entrar em *thrashing*, outros também entrem.**
  - » **No entanto, basta que um processo entre em *thrashing* para que o tempo efectivo de acesso à memória dos outros processos aumente.**

Como evitar o *thrashing* ?

- **Estratégia do conjunto de trabalho (*working-set strategy*).**
- **Estratégia da frequência de falta de página.**

Estratégia dos conjuntos de trabalho
(*Working-set strategy*)

Trata de determinar simultâneamente
- **Quantos *frames* alocar a um processo.**
- **Que páginas manter nesses *frames.***

Objectivo:
- **Evitar o *thrashing*,**
  mantendo um grau de multiprogramação tão alto quanto possível.

A ideia:
- **Usar as necessidades recentes de um processo**
  para adivinhar as necessidades futuras (reduzir a taxa de falta de páginas)
  baseado no princípio da localidade de referência.

Estratégia dos conjuntos de trabalho

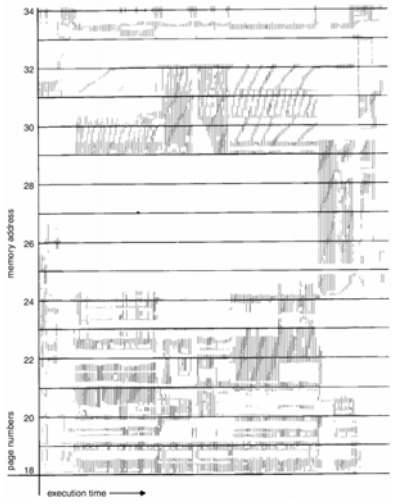
Conjunto de trabalho *(working set)* de um processo
num instante t → W(t,Δ)
- **conjunto de páginas referenciadas nas últimas Δ referências à memória**

O nº ideal de *frames* a atribuir a um processo é
o necessário para guardar o seu conjunto de trabalho.

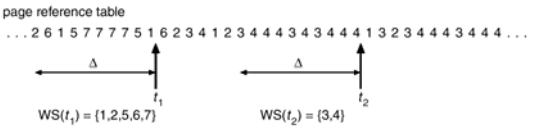
Procedimento:
- **Monitorizar o conjunto de trabalho de cada processo.**
- **Um processo nunca será executado a não ser que**
  o seu conjunto de trabalho esteja em memória principal.
- **Uma página não pode ser removida da memória**
  se fizer parte do conjunto de trabalho de um processo.

  WSSi = dimensão do conjunto de trabalho do processo Pi
  D=Σ WSSi = necessidade total de *frames;*
  N = nº total de *frames* existentes

- **Se D>N suspender um processo para evitar o *thrashing.***
- **Se D<N pode-se iniciar um novo processo.**



Localidade num conjunto de referências à memória



Estratégia dos conjuntos de trabalho

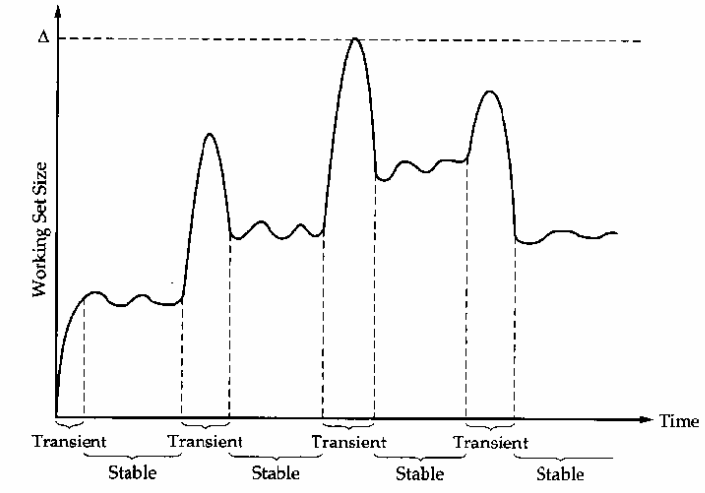
Estratégia dos conjuntos de trabalho

Problemas:

- **O passado nem sempre ajuda a prever o futuro**
  - » **O conjunto de trabalho pode variar com o tempo alternando períodos de alguma estabilidade com períodos de mudança brusca.**
- **Dificuldade em manter actualizado o conjunto de trabalho**
  - » **Necessária uma "janela móvel" sobre a lista de referências à memória.**
  - » **Aproximação:**
    - ***timer*** **que gera interrupções +** ***bit*** **de referência + registo de n** ***bits*****, por página.**
    - **Copiar o** ***bit*** **de referência para o registo, a intervalos regulares.**
    - **Se um dos** ***bits*** **do registo estiver activado a página pertence ao conjunto de trabalho do processo**
    - **<u>Dificuldade</u>: não se consegue saber o que se passou entre interrupções.**
    - **<u>Solução (…)</u> : aumentar o tamanho do registo e reduzir o intervalo entre interrupções.**
- **Determinar o valor óptimo de Δ**
  - » **Δ demasiado pequeno**
    - **pode não englobar toda uma** ***localidade*** **(pág.s activamente usadas, em conjunto)**
  - » **Δ demasiado grande**
    - **pode englobar várias** ***localidades*** **e abranger mais pág.s do que o necessário**

Sequence of Page References	Window Size, Δ			
	2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	18 24	•	24 17 18
17	18 17	24 18 17	•	•
17	17	18 17	•	•
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	•
17	24 17	•	•	17 15 24
24	•	24 17	•	•
18	24 18	17 24 18	17 24 18	15 17 24 18

Evolução do conjunto de trabalho de um processo em função do tamanho da janela



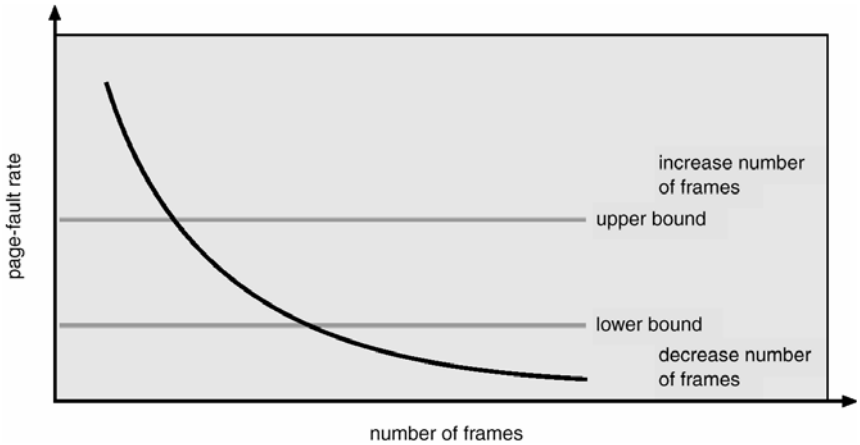
Evolução típica do conjunto de trabalho de um processo

Estratégia da frequência de falta de página

FREQUÊNCIA DE FALTA DE PÁGINA:
Estratégia para evitar o *thrashing*
mais simples do que a dos conjuntos de trabalho.

Procedimento:

- **Monitorizar a frequência de falta de páginas de um processo.**
- **Estabelecer um gama de frequências aceitáveis.**
- **Acima de uma certa frequência atribuir mais um *frame* ao processo; se não houver *frames* disponíveis, suspender o processo.**
- **Abaixo de uma certa frequência, retirar um *frame* ao processo.**



Estratégia da frequência de falta de página

Outras considerações

Além dos
algoritmos de substituição de páginas e
da estratégia de alocação de *frames*
há outros factores a ter em conta:

- **Deve usar-se pré-paginação ?**
- **Qual o tamanho de página mais adequado ?**
- **Como é que a estrutura de um programa pode influenciar a sua performance, tendo em atenção a existência de paginação ?**
- **Será conveniente proceder à fixação de algumas páginas em memória ?**

Pré-paginação

A paginação a pedido "pura"
conduz a elevado nº de faltas de página

- **quando um processo começa a correr;**
- **quando o processo retoma a execução, após um *swap out*.**

Pré-paginação

- **Procura evitar este elevado nº de faltas de página carregando mais páginas do que as exigidas pela falta de página, procurando aproveitar o facto de, o carregamento consecutivo poder ser mais rápido do que o individual (se as pág.s estiverem em posições consecutivas do disco)**

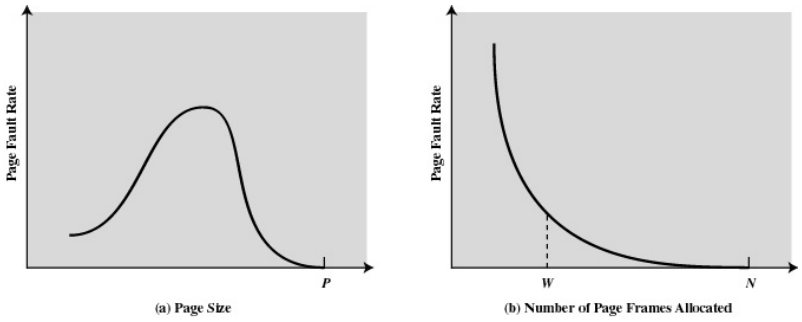
Interesse duvidoso

- **Pode ser vantajoso em algumas situações.**
- **Será o seu custo menor do que servir as faltas de página ?**
- **Pode acontecer que muitas das páginas carregadas não venham a ser usadas !**

Tamanho da página

- **Usualmente determinado pelo *hardware*.**
  (Alguns processadores admitem tamanhos de página variáveis.)
- **Não existe um tamanho ideal.**
- **Argumentos a favor de <u>páginas pequenas</u>:**
  - **Reduz a fragmentação interna.**
  - **Permite isolar mais facilmente a memória que é efectivamente necessária**
    (↔princípio da localidade de referência)
  - **Reduz a *I/O* necessária.**
  - **Permite que a memória ocupada por um processo possa ser reduzida (relativamente a quando as páginas são grandes)**
- **Argumentos a favor de <u>páginas grandes</u>:**
  - **Reduz o tamanho da tabela de páginas.**
  - **A *I/O* é mais eficiente.**
    (o *overhead* devido ao posicionamento da cabeça do disco pode pesar significativamente no tempo total de *I/O* de uma pág. pequena)
  - **Reduz o nº de faltas de página, a partir de certa dimensão das páginas.**

Tendência histórica: aumentar o tamanho das páginas



P = size of entire process
W = working set size
N = total number of pages in process

Evolução típica da taxa de falta de páginas de um programa em função: a) do tamanho das páginas; b) do nº de quadros alocados

Estrutura de um programa

- A paginação a pedido é transparente para o programador.
  No entanto, a performance de um programa pode ser melhorada se o programador estiver consciente do modo como é feita a paginação.
- Exemplo:
  - Programa em C
  - int A[1024][1024]; ← armazenado linha a linha
  - 1 inteiro = 4 bytes
  - *Frames* de 4KB; 1 *frame* pode conter 1 linha da matriz (1024×4 bytes)
  - Programa 1

```
for (j=0; j<1024; j++)
    for (i=0; i<1024; i++)
            A[i][j] = 0;
```

⇒ O 1º elemento A[i][j] acedido está numa página, o 2º está noutra página, etc.

  - Programa 2

```
for (i=0; i<1024; i++)
    for (j=0; j<1024; j++)
            A[i][j] = 0;
```

⇒ Os primeiros 1024 elementos A[i][j] podem estar todos na mesma página, os segundos 1024 elementos podem estar todos noutra página, etc.

Estrutura de um programa

- Uma selecção cuidadosa das estruturas de dados e das estruturas de programação pode reduzir o nº de faltas de página e o nº de páginas no conjunto de trabalho.
  - *stack* - boa localidade de referência
  - tabela de *hash* - má localidade de referência
  - utilização de apontadores - tende a introduzir má localidade de referência
- A linguagem de programação utilizada também pode influenciar.
  - ex.: certas linguagens fazem uso intensivo de apontadores

Estrutura de um programa

- A paginação a pedido é transparente para o programador.
  No entanto, a performance de um programa pode ser melhorada se o programador estiver consciente do modo como é feita a paginação.
- Exemplo:
  - Programa em Pascal
  - Var A: Array [1024,1024] of integer; ← armazenado linha a linha
  - 1 inteiro = 4 bytes
  - *Frames* de 4KB; 1 *frame* pode conter 1 linha da matriz (1024×4 bytes)
  - Programa 1

```
For j:=1 to 1024 do
   For i:=1 to 1024 do
         A[i,j] := 0;
```

⇒ O 1º elemento A[i,j] acedido está numa página, o 2º está noutra página, etc.

  - Programa 2

```
For i:=1 to 1024 do
   For j:=1 to 1024 do
         A[i,j] := 0;
```

⇒ Os primeiros 1024 elementos A[i,j] podem estar todos na mesma página, os segundos 1024 elementos podem estar todos noutra página, etc.

Fixação de páginas

- Alguns *frames* podem ser "fechados" (*locked*)
  isto é, as páginas neles contidas não podem ser substituídas
  ⇒ usar um ***lock bit***.
- Exemplo:
  - Grande parte do núcleo do S.O. e das estruturas de dados do S.O. .
  - Processos com tempos de execução críticos.
  - *Buffers* de *I/O* (em memória do utilizador)
    - » Objectivo: evitar que aconteça o seguinte
      - um processo pede *I/O*
      - a seguir, bloqueia
      - o processo que entra em execução gera uma falta de página
      - a página do *buffer* de *I/O* é substituída
      - o pedido de *I/O* é executado p/ uma pág. que não pertence ao processo que fez o pedido
    - » Soluções:
      - Nunca fazer *I/O* para a memória do utilizador mas p/ a memória do S.O. ou
      - Permitir que as páginas com *I/O* pendente sejam fixadas.

Fixação de páginas

- **Outra utilização da fixação de páginas:**
  - **Impedir que uma página recentemente carregada seja substituída antes de ser usada pelo menos uma vez.**

- **Exemplo:**
  - **Um processo de baixa prioridade tem uma falta de página.**
  - **A página é carregada.**
  - **Enquanto a página é carregada o procesador é atribuído a um processo de prioridade mais elevada.**
  - **Entretanto, a página pedida pelo processo de baixa prioridade é carregada.**
  - **O processo de alta prioridade também sofre uma falta de página.**
  - **A página substituída pode ser a que foi carregada a pedido do processo de baixa prioridade.**

Segmentação a pedido

- **Alguns processadores podem não suportar paginação mas suportar segmentação (ex.: Intel 80286).**

- **A segmentação a pedido é semelhante à paginação a pedido:**
  - **Um processo não precisa de ter todos os segmentos em memória para executar.**
  - **O descritor de cada segmento tem um *bit* de segmento válido / inválido.**
    (Os descritores contêm informação acerca do tamanho, protecção e localização dos segmentos)
  - **Quando um segmento referenciado não está em memória (→ *trap* p/ o S.O) é necessário carregá-lo.**
  - **Se houver necessidade de substituir um segmento p/ carregar outro usa-se um dos algoritmos de substituição descritos anteriormente.**
  - **Usa-se um *bit* de referência p/ saber os segmentos que foram acedidos.**

- **Maior diferença relativamente à paginação a pedido:**
  - **Necessidade de compactação p/ reduzir a fragmentação externa**
    - » **Se o espaço livre total for suficiente mas não houver nenhum bloco livre de tamanho suficiente ⇒ compactação**
    - » **Se o espaço livre total não for suficiente ⇒ substituição de segmentos e compactação (eventualmente)**