FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Fundamentos de Sistemas de Operação

LEI - 2023/2024

Vitor Duarte
Mª. Cecília Gomes

1

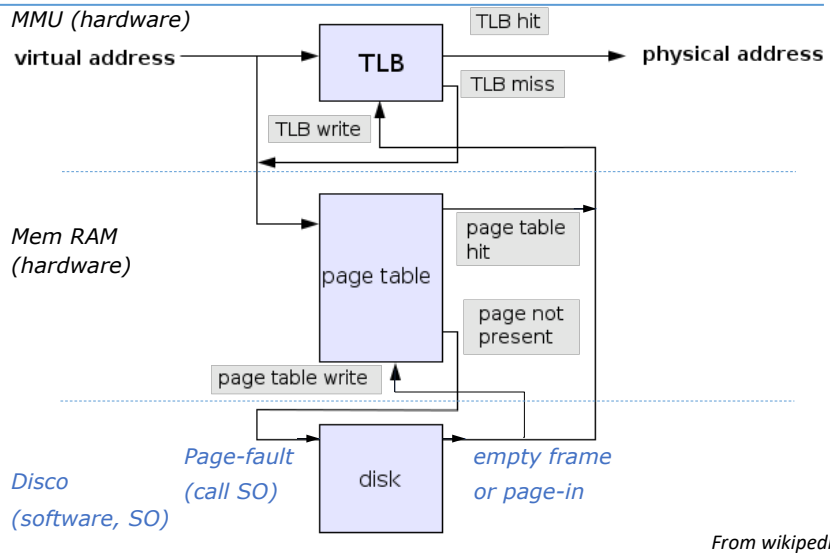
Aula 14

- Implementações da tabela de páginas
- Algoritmos de substituição de páginas

- OSTEP: cap. 20, 21, 22

2

Revendo Transformação de End Virtual



From wikipedia

Desempenho da paginação a pedido

- Uma página pode não estar em memória porque nunca esteve ou foi entretanto despejada para dar lugar a outra
- *Page Hit Rate* $0 \leq p \leq 1.0$
  - Se p == 1 não há faltas de página
  - Se p == 0 todas as referências provocam faltas
- Tempo de acesso efetivo (TAE)

  TAE = p x Tm_RAM + (1-p) x Tm_DISCO

  - Tm_RAM é o tempo médio tendo em conta a TLB
    - Depende do TLB *hitratio*. Será entre T_acessoRAM e 2 x T_acessoRAM
  - Tm_DISCO é o tempo médio tendo em conta eventual page-in e que algumas vezes é também necessário swap-out de uma página
  - *(e ainda podíamos contar com as caches – assumimos que está incluído no Tm_RAM)*

Impacto do page-in e page-out

- Se tempo médio de acesso à RAM = 30 ns
- Se 50% das vezes a frame escolhida foi modificada e portanto precisa de ser escrita no disco (swapped out)
- Se tempo de leitura/escrita no disco 10 ms = 10 000 000 ns

$$TAE = p \times 30 + (1-p) \times (10\,000\,000 + 0.5 * 10\,000\,000)$$

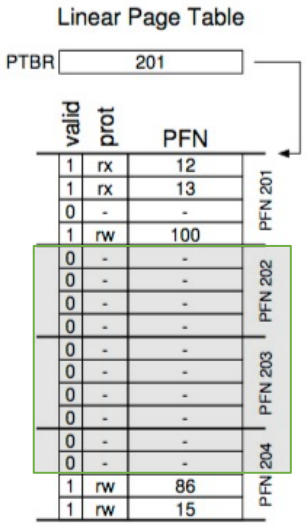
- Para p = 99%: TAE=150 029,7 ns

- Nas páginas espera-se ter hits mais perto de 100%
  - a localidade dos acessos nos programas costuma ser muito grande
  - page faults no início do programa (on-demand)
  - depois raramente ocorrem page faults → > 99,99 %
    (exceto se falta memória RAM)

Tamanho da Tabela de páginas linear

- Implementação simples:
  - nº página usado como índice para um vetor que tem nº frame, bit validade, permissões, etc
- A tabela de páginas é uma estrutura de dados como qualquer outra
  - Com algum suporte no hardware para melhor desempenho
- Problema: exemplo
  - Endereços de 32 bits (4GB de end); páginas de 4KB; necessitamos de 1M páginas); supondo 4 bytes por entrada: espaço ocupado por cada tabela (aprox.) 4MB
  - Cada processo necessita da sua; podem existir centenas de processos . . .
- Alternativas? Podemos diminuir o espaço? Sem perder desempenho?

Tabela de páginas linear



Grande zona de entradas livres mas que ocupam espaço

PTBR=page table base register
PFN=page frame number

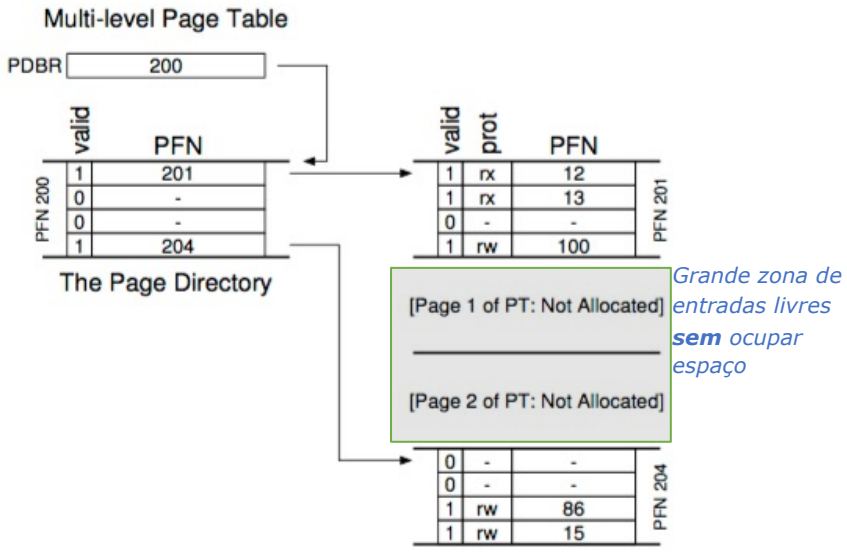
Impacto do tamanho da página

- Quanto maior cada página, menos páginas são precisas para um processo; logo menos entradas nas tabelas de páginas
- Tabelas mais pequenas logo menos uso de memória
- O disco lida eficientemente com blocos contíguos; seria bom usar páginas grandes
  - Mas cada swap-in ou swap-out vai demorar mais
  - Quanto maior é o tamanho da página, maior é a memória desperdiçada por fragmentação interna. Menos memória para os processos

Tabela de páginas com mais de um nível

- Tabela de páginas ocupa vários frames em memória RAM
- Podemos paginar a tabela de páginas… passar a uma árvore.
- O espaço de endereçamento virtual é dividido em várias partes, correspondendo a cada uma à sua sub-tabela de páginas.
- Uma caso simples é utilizar dois níveis
  - Exemplo: se cada página (4KB) permite 1024 entradas, 2 níveis permite 1024*1024 (1M) entradas, 4GB de endereçamento

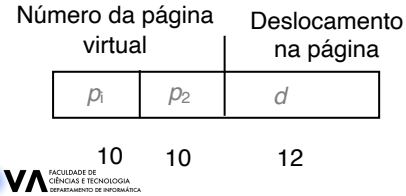
Tabela "paginada" (dois níveis)



Grande zona de entradas livres ***sem*** *ocupar espaço*

Exemplo de tabela de páginas com 2 níveis: Intel 32 bits

- 32 bits de endereçamento, páginas de 4 Kbytes:
  - Número da página com 20 bits.
  - Deslocamento dentro da página (page offset) com 12 bits.
- A própria tabela de páginas ocupa várias páginas de 4 Kbytes, logo o número da página é dividido em:
  - a 10-bit para índice na tabela de páginas "principal".
  - a 10-bit para índice numa tabela de páginas "secundária".
- O endereço virtual é:

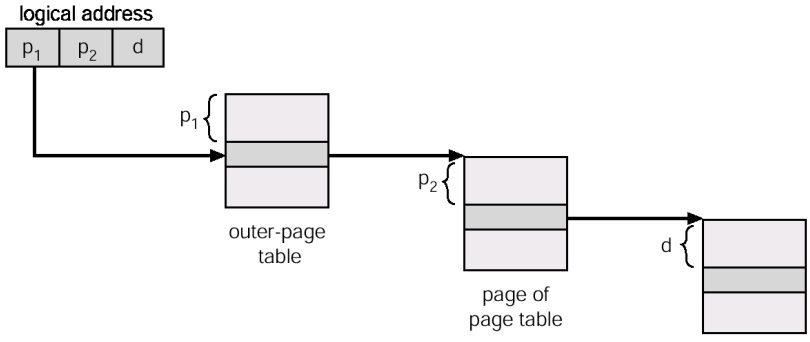


P1 índice na tabela de páginas principal

P2 índice na tabela de páginas secundária

Tranformação de endereços

- Usando tabela de páginas com dois níveis

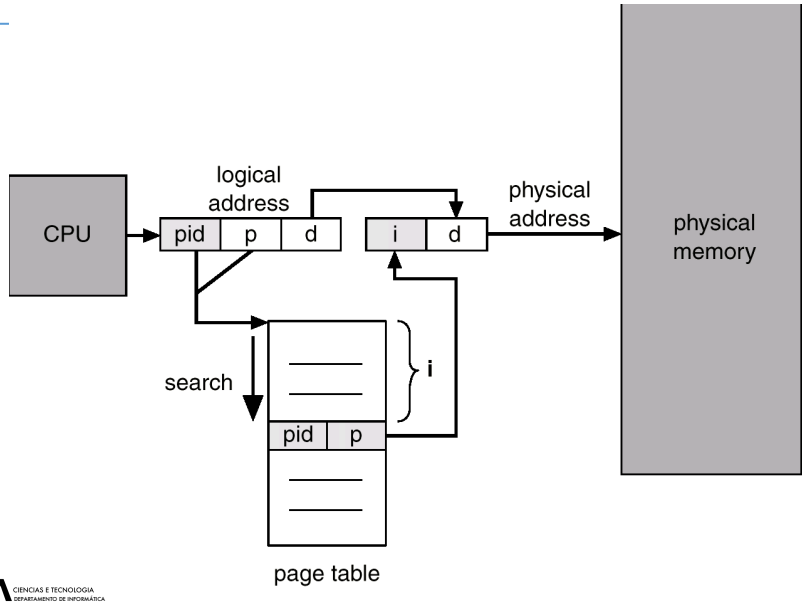


O TLB continua a conter pares (nº página virtual, nº da página física). Quando não está no TLB, obriga a dois acessos a memória para a página (3 para resolver o endereço)

Tabela de páginas invertida

- Uma entrada para cada "frame" (página real) na RAM. O índice é o nº da *frame*
- Cada entrada contém um par (*ID* do processo que "possui" a *frame*, nº página virtual desse processo)
- Diminui muito o espaço ocupado mas complica muito o processo de transformação do EV
  - Difícil de implementar no hardware. Para ser eficiente, exigiria a tabela numa memória associativa, ou uma tabela de dispersão (hash)
- Dificulta a partilha de páginas entre processos
- *Um tabela de frames é uma forma do SO gerir frames livres vs ocupadas*

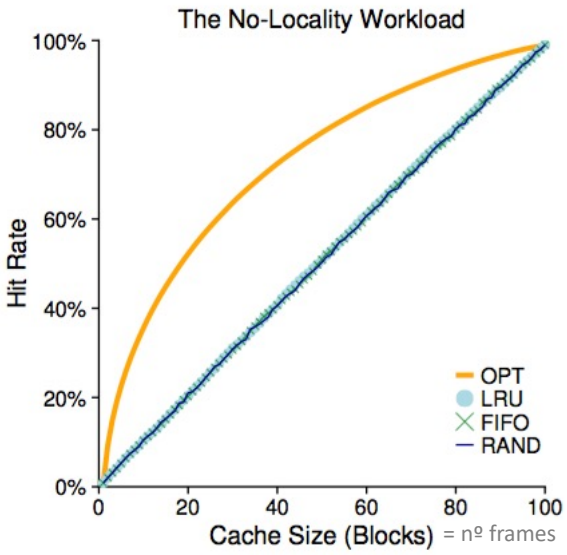
Tabela de páginas invertida



Substituição de páginas

- As páginas deixam de estar em memória quando:
  - Deixam de ser usadas (todos os processos que a usavam já não a usam) → a frame é dada como livre
  - Há falta de memória: já está toda ocupada ou se atingiu um nível limite de memória livre (*watermark*) → pode ter de libertar frame(s)
- Políticas (algoritmos) para escolher a página a retirar de memória?
  - **Problema semelhante ao das caches de memória**
  - **Objectivo: Escolher a que minimiza os *page-faults*!** Mas não conhecemos o futuro, senão escolhíamos a "usada mais tarde"
  - Escolher com base em heurísticas/tendências, exemplos: *FIFO*, *Least-Frequently-Used*, ***Least-Recently-Used*** (tendo em conta os princípios de localidade / *working-set*)

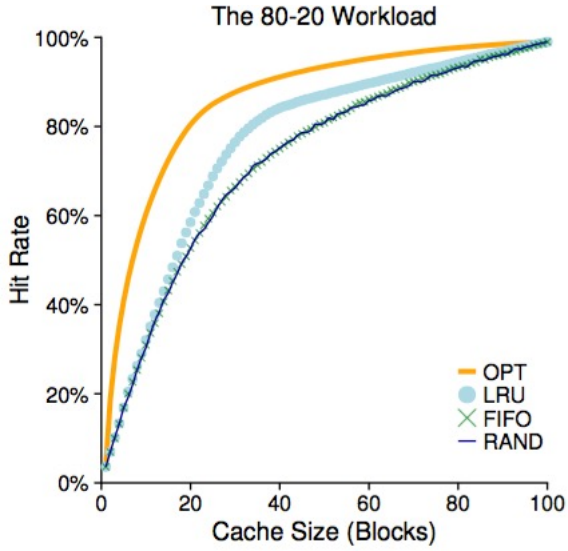
Avaliação por simulação de *workloads*



10000 acessos a 100 páginas

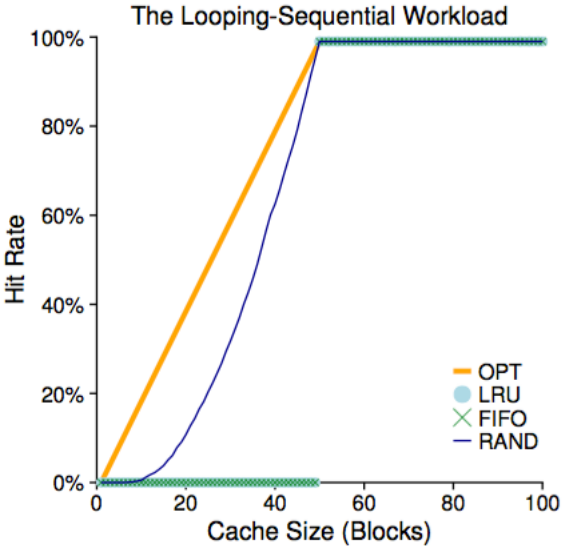
= nº frames

Avaliação por simulação de *workloads*



17

Avaliação por simulação de *workloads*

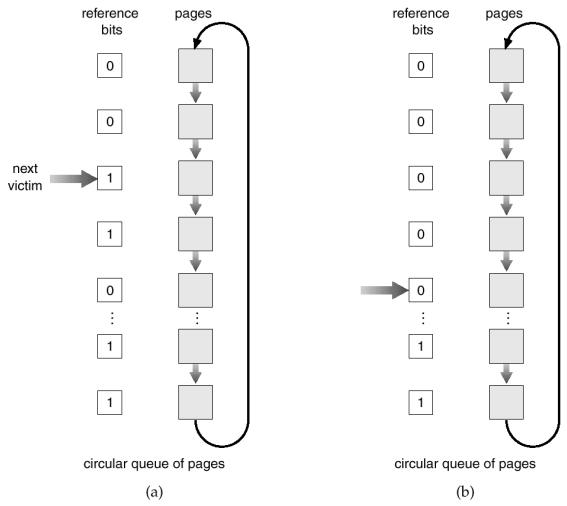


10000 acessos sequenciais a 50 páginas

18

Pseudo-LRU

- LRU é difícil/ineficiente de implementar.
- Aproximação pseudo-LRU de 1 bit: **Clock**
  - O SO mantém um **índice** de nº de frame (clock hand)
  - Cada página tem um bit de "acesso" que começa a 0
- A cada acesso a uma página, passa a 1 o bit indicando o acesso; se escrita, passa a 1 o bit "dirty" (hardware)
- Quando de um page-fault usa o índice para percorrer as frames à procura de uma livre ou com bit de acesso a 0. Se não há livre:
  - Todas as frames acedidas vão tendo o bit passado para 0 (dá uma segunda hipótese). No fim volta ao início
  - A primeira frame não acedida é a vítima
  - Se dirty tem de ser escrita (swapped out)
- O índice é deixado a seguir à vítima

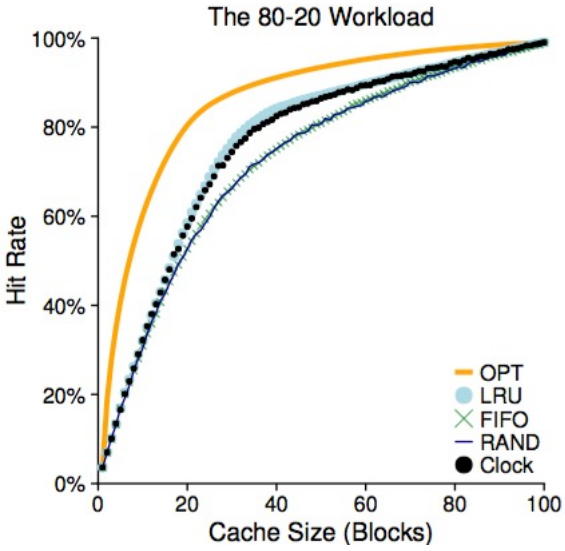


Clock ou Second Chance

- Clock ou Second Chance (segunda hipótese):

```
static int C;
victim = -1
do{
   if frames[C].RefBit == 1
         frames[C].RefBit = 0
   else
     victim = C
  C = (C+1) % N_PAGS   // next frame
} while ( victim == -1)
return  victim;
```


Avaliação por simulação de *workloads*



Variante: tentar reduzir swap outs

- Despejar uma página escrita (*dirty*) vai obrigar a swap-out, o que é mais demorado e usa mais disco de swap
- Variante: ter preferência pelas não escritas
  - Primeiro só escolhe se frame não acedida e não escrita, mas memoriza a primeira não acedida e escrita
  - Se não encontrar nenhuma não escrita, usa a memorizada