

Sistemas Operacionais

Quinta Lista de Exercícios – Solução

Norton Trevisan Roman

30 de novembro de 2012

1. **4 KB:** o endereço 20.000 estará na página $20.000/4.096 = 4$. A página 4 começa no endereço $4 \times 4KB = 16.384$. Então o deslocamento será de $20.000 - 16.384 = 3.616$. De um modo mais simples, temos que

$$\begin{array}{r|l} 20.000 & 4.096 \\ \hline 3.616 & 4 \end{array}$$

Da mesma forma, o endereço 32.768 estará na página $32.768/4.096 = 8$, deslocamento 0. O endereço 60.000 na página $60.000/4096 = 14$, que inicia em 57.344 \rightarrow deslocamento $60.000 - 57.344 = 2.656$

8 KB: 20.000 em $20.000/8.192 = 2$, deslocamento $20.000 - 16.384 = 3.616$; 32.768 em $32.768/8.192 = 4$, deslocamento 0; e 60.000 em $60.000/8.192 = 7$, deslocamento $60.000 - 57.344 = 2.656$

2. (a) 20 está na página 0k-4k \rightarrow moldura 8-12k $\rightarrow 8 \times 1.024 + (20 - 0) = 8.212$
(b) 4100 está na página 4k-8k \rightarrow moldura 4k-8k $\rightarrow 4100$
(c) 8300 está na página 8k-12k \rightarrow moldura 24k-28k $\rightarrow 24 \times 1.024 + (8.300 - 8 \times 1.024) = 24.684$
3. Suponha a TLB inicialmente vazia.

- (a) Como a página tem 4KB, se o passo nos levar, em cada iteração, para além dessa janela, então haverá uma ausência (page fault) sempre, indefinidamente (já que não se volta a páginas já carregadas). Assim, a cada incremento de 4.096B no endereço, haverá uma ausência na TLB. Para o valor de M, no entanto, temos que saber o tamanho de um inteiro, já que cada elemento do arranjo nos move um inteiro na memória. Como queremos, para a falha ocorrer, que o endereço seja superior a 4KB (4.096), então temos que $M \geq 4096/sizeof(int)$. Nesse caso, N somente comandará quantas ausências haverá.

- (b) Para garantir uma ausência na TLB a cada acesso a um elemento de X, M deveria ser pelo menos $4.096/sizeof(int)$. Contudo, se N for pequeno, novas repetições do laço vão encontrar a página na TLB (deixada lá por execuções anteriores).

Como queremos uma ausência a cada execução do laço, isso significa que nenhuma página acessada pode estar na TLB. Ou seja, N deve ser grande o suficiente para que toda página trazida à TLB seja substituída pelo menos uma vez (evitando, assim, que uma re-execução do laço encontre as páginas iniciais na TLB).

Isso ocorre quando todas as 64 entradas da TLB são carregadas. Nesse caso, a TLB está cheia e, para carregar a 65ª entrada, teremos que remover a primeira (isso supondo um mecanismo de substituição do tipo FIFO). Assim, no momento em que a 65ª entrada foi carregada, uma reexecução do laço não encontrará mais $x[0]$ na TLB.

Como cada entrada corresponde a uma página, e cada página contém 4KB, isso ocorrerá após o 262.144º byte ($64 \times 4.096 = 256KB$), ou seja, precisamos de 256KB para utilizar toda a TLB. Como queremos uma ausência na TLB a cada passo, isso ocorrerá quando $x > 256KB$. Nesse caso, $N > 256KB/sizeof(int)$.

4. O pior caso seria rodar todos os n processos, cada um usando todo seu endereçamento virtual v . Assim, cada processo ocuparia nv B, dos quais r estariam em RAM, restando $nv - r$ para o disco. Isso é pouco realista, pois raramente o número máximo de processos será atingido, e menos ainda todos eles usarão seu espaço de endereçamento virtual inteiro.
5. Em k instruções teremos gasto um tempo $k \times 10(ns) + n(ns)$. A média por instrução será de $\frac{10k+n}{k} = 10 + \frac{n}{k}$.
6. Com páginas de $8KB$, o deslocamento ocupa 13 bits (pois $2^{13} = 8K$). Temos então $32 - 13 = 19$ bits para endereçamento das páginas $\rightarrow 2^{19} = 512K$ páginas $\rightarrow 512K$ entradas na tabela. Como cada entrada tem o tamanho de uma palavra (32 b), temos 512K palavras transferidas somente para a tabela. A uma taxa de 1 palavra a cada 100 ns, teremos um total de $512K * 100 = 52428800ns \approx 52,43ms$. Como o quantum do processo é de 100ms, restam $100 - 52,43 = 47,57ms$ para o processo. Assim, 52,43% do processamento é gasto com a tabela.
7. (a) Páginas de $4K$ geram deslocamento de 12 bits ($2^{12} = 4K$). Restam então $48 - 12 = 36$ bits para indexar as páginas, levando a $2^{36} = 64G$ entradas.
(b) A página de instruções ficará sempre na TLB (100% hit). Páginas de dados terão 100% hit até que o programa termine de acessar seus dados. Com $4KB$, cada página guarda 1024 inteiros de 4B (long int) \rightarrow a cada 1024 referência aos dados haverá um miss (e 1 referência extra à memória). Isso se repetirá até o fim do arranjo.
8. (a) A tabela multinível reduz o número de páginas que precisam estar na memória. Para programas que acessam muito um número pequeno de páginas, somente precisamos da página de primeiro nível e de poucas páginas nos outros níveis.
(b) Páginas de $16KB$ levam a um deslocamento de 14b ($16K = 2^{14}$). Com endereços de 38b sobram $38 - 14 = 24b$ para endereçamento das tabelas. Como cada uma tem $4 \times 8 = 32b$, podemos alocar 12b para cada uma que há espaço de sobra nelas.
9. (a) Com $4GB$ de endereçamento virtual e páginas de $16KB$, temos $4GB \div 16KB = 256K$ páginas (alternativamente, podemos tratar em bits: $16KB = 2^{14}$ e $4GB = 2^{32}$, então $2^{32-14} = 2^{18} = 256K$).
Para as molduras, devemos usar a memória física ($1GB$), e $1GB \div 16KB = 64K$ molduras.
(b) Se há 256K páginas, há 256K entradas na tabela de nível único.
(c) Como há $4GB$ de endereços virtuais, precisaremos de 32 bits para todos ($2^{32} = 4G$). Para os reais, bastam 30 ($2^{30} = 1GB$)
(d) Como temos 64K molduras, precisaremos de 16 bits para endereçá-las ($2^{16} = 64K$). Somando os bits R e M teremos 18 bits por entrada (ou seja, 2,25B, o que nos leva a 3B). Com 256K entradas, teremos $256K \times 3 = 768KB$
(e) Um programa de $8MB$, com páginas de $16KB$, usaria $8MB \div 16KB = 512$ páginas. Com 8 bits para a tabela secundária, cada tabela consegue endereçar $2^8 = 256$ páginas. Assim, como precisamos endereçar 512 páginas, bastam 2 tabelas de nível 2 (além, é claro, da tabela de nível 1), para esse processo.
10. Com 9b para a tabela de nível 1 e 11 para a de nível 2, sobraram $32 - 9 - 11 = 12b$ para o deslocamento dentro da página $\rightarrow 2^{12} = 4KB$.
Com 32b de endereçamento, temos um espaço de $4GB$. Como as páginas têm $4KB$, teremos $4GB \div 4KB = 1M$ páginas.
11. O número de páginas dependerá somente de d , pois haverá sempre $2^a \times 2^b \times 2^c$ dessas páginas, não importando como os bits são distribuídos entre a , b e c .
12. Páginas de $4KB$ usam 12 bits ($2^{12} = 4K$). Sobram então $32 - 12 = 20$ bits para o endereçamento das páginas. Com páginas de nível único, teremos um total de $2^{20} = 1M$ entradas, independente de quantas são efetivamente usadas.
Com dois níveis, precisaremos da tabela de nível 1, e de outras 2 (página mais alta e mais baixa), num total de 3. Como cada página endereça $2^{10} = 1K$, esse arranjo terá $3 \times 1K = 3K$ entradas.

13. O tempo médio para hit e miss deve ser 2, ou seja

$$\frac{h \times 1 + m \times 5}{h + m} = 2 \Rightarrow h + 5m = 2h + 2m \Rightarrow h = 3m$$

Assim, a cada 4 acessos, 3 devem ser acertos e no máximo 1 uma falha na TLB \Rightarrow a taxa de hits deverá ser de, no mínimo, $\frac{3}{4} = 0,75$

14. Páginas de 8KB levam a um deslocamento de 13b ($2^{13} = 8K$). Sobram então $48 - 13 = 35b$ para o endereçamento das páginas $\rightarrow 2^{35} = 32G$ páginas (e entradas na tabela de páginas)
15. A tabela invertida mapeia a RAM. Para que haja um comprimento médio de 1, ela deve ter o mesmo número de páginas da RAM, ou seja, $\frac{256MB}{8KB} = 32K$. Queremos, no entanto, que o comprimento médio seja < 1 , ou seja, o tamanho terá que ser $> 32K$. Contudo, como seu tamanho é potência de 2, o próximo tamanho após 32K é $2 \times 32K = 64K$, que garante um tamanho médio de 0,5.

16. **FIFO**: De início, haverá falta para as 4 primeiras:

0 1 7 2

1 7 2 3 – falta (3), 2, 7 e 1 estão lá

7 2 3 0 – falta (0), 3 está lá

6 faltas.

LRU: De início, falta para as 4 primeiras (em ordem de tempo de acesso):

0 1 7 2

1 7 2 3 – falta (3)

1 7 3 2 – acesso à 2

1 3 2 7 – acesso à 7

3 2 7 1 – acesso à 1

2 7 1 0 – falta (0)

7 1 0 3 – falta (3)

7 faltas

17. $D \rightarrow$ a primeira com $R = 0$

18. Aging: Cada página recebe um contador. A cada tique, o contador é deslocado à direita, e o bit R correspondente adicionado à esquerda

Contador	0111	1011	1010	1101	0010
00000000	00000000	10000000	11000000	11100000	01110000
00000000	10000000	01000000	00100000	10010000	01001000
00000000	10000000	11000000	11100000	01110000	10111000
00000000	10000000	11000000	01100000	10110000	01011000

Contador	1010	1100	0001
01110000	10111000	11011100	01101110
01001000	00100100	10010010	01001001
10111000	11011100	01101110	00110111
01011000	00101100	00010110	10001011

19. A página mais antiga com $R = 0$ é a marcada com 1620. Sua idade é $2204 - 1620 = 584 > 400$, portanto esta será a removida
20. Como $2204 - 1213 = 991$, a página marcada com 1213 será removida, se $\tau = 400$. Se $\tau = 1000$, a página não será removida, pois está dentro do working set.
21. (a) 2, pois $(R, M) = (0, 0)$

- (b) 3, é a mais antiga
 - (c) 1, é a menos recentemente usada
 - (d) 2, é a mais antiga, com R=0
22. O fragmento B acessará $64 \times 64 = 4.096$ palavras consecutivas na memória. Como o programa ocupa uma página, elas começarão no início da seguinte. Com 128 palavras em cada moldura, teremos $4.096/128 = 32$ page faults no processo.
- No caso do fragmento A, o laço pula de 64 em 64 palavras. Como cada moldura armazena 128, isso ocasionará um total de $128/64 = 2$ acessos por página somente (ou seja, serão lidas somente 2 palavras por página). Num universo de 4.096 palavras, isso leva a $4.096/2 = 2.048$ page faults.
23. Com 65.536 e páginas de 4.096 temos $65.536/4.096 = 16$ páginas. O segmento de texto consome $32.768/4.096 = 8$ páginas, o de dados $16386/4.096 = 4,0005 = 5$ páginas, e a pilha $15.870/4.096 = 3,87 = 4$ páginas, num total de 17. Assim, não caberá na RAM.
- Já com páginas de 512 temos $65.536/512 = 128$ páginas. O segmento de texto consome $32.768/512 = 64$ páginas, o de dados $16386/512 = 32,004 = 33$ páginas, e a pilha $15.870/512 = 30,996 = 31 = 4$ páginas, num total de 128 \Rightarrow caberá na RAM.
24. Se o SO permitir compartilhamento de página (ex: texto do programa), sim.
25. Como cada falta leva $2.000\mu s$, 15.000 faltas tomarão 30s ($30.000.000\mu s$). Assim, o programa gastará 30s tratando tão somente de faltas de página e 30s processando instruções. Ao dobrarmos a memória, o número de faltas deve cair pela metade, gastando $30/2 = 15s$. Assim, o programa levará 30s (instruções) + 15s (faltas) = 45s.
26. A instrução poderia estar parte em uma página e parte em outra, causando 2 possíveis faltas somente para processar a instrução: uma para achar a primeira parte da instrução, e outra para achar a segunda parte. A palavra de memória poderia também estar parte numa página e parte em outra, causando outras 2 faltas \rightarrow total de 4. Se, contudo, o sistema exige que as palavras estejam alinhadas na memória (que não é o caso do Pentium), então serão 2 faltas para a instrução e 1 para a palavra de dados (total de 3).
27. Com 10b, cada moldura/página conterá 1024B (1KB)
- (a) $(14,3) \Rightarrow 14 \times 1024 + 3 = 14.339$
 - (b) Protection fault: não se pode escrever lá
 - (c) Page fault: o endereço buscado está em disco
 - (d) Protection fault: salto é usado para desvio de fluxo em programas \rightarrow irá desviar para uma instrução que será executada. A moldura, contudo, não tem permissão para execução