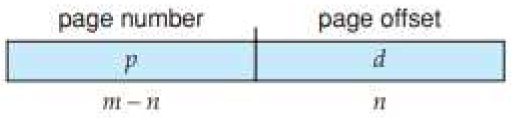
2013 - 2014 Thema 2.1 ITSD

BSS Solutions Week 4

Chapter 8

Exercise 1.1 Page sizes
Why are page sizes always powers of 2?

Answer 1.1, see p 364 and solution 8.3
The page size (like the frame size) is defined by the hardware. The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy. If the size of the logical address space is 2^m, and a page size is 2^n addressing units (bytes or words), then the high-order m - n bits of a logical address designate the page number, and the n low-order bits designate the page offset. Thus, the logical address is as follows:



where p is an index into the page table and d is the displacement within the page.

Exercise 1.2 Paging
On a system with paging, a process cannot access memory that it does not own; why? How could the operating system allow access to other memory? Why should it or should it not?

Answer 1.2, see solution 8.14
An address on a paging system is a logical page number and an offset. The physical page is found by searching a table based on the logical page number to produce a physical page number. Because the operating system controls the contents of this table, it can limit a process to accessing only those physical pages allocated to the process. There is no way for a process to refer to a page it does not own be cause the page will not be in the page table.
To allow such access, an operating system simply needs to allow entries for non-process memory to be added to the process's page table. This is useful when two or more processes need to exchange data - they just read and write to the same physical addresses (which may be at varying logical addresses). This makes for very efficient interprocess communication.

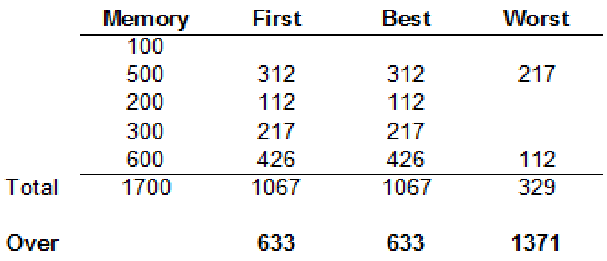
Exercise 1.3 Memory Reference
Consider a paging system with the page table stored in memory.
a) If a memory reference takes 160 nanoseconds, how long does a paged memory reference take?
b) If we add associative registers, and 75 percent of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that finding a page-table entry in the associative registers takes zero time if the entry is there .)

Answer 1.3, see solution 8.20

a) 320 nanoseconds: 160 nanoseconds to access the page table and 160 nanoseconds to access the word in memory.
b) Effective access time = 0.75 × (160 nanosecond s) + 0.25 × (320 nanosecond s)
120 + 80 = 200 nanoseconds.

Exercise 1.4 Memory Partitions
Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB,and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 312 KB, 217 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

Answer 1.4, see solution 8.11 Alternative A

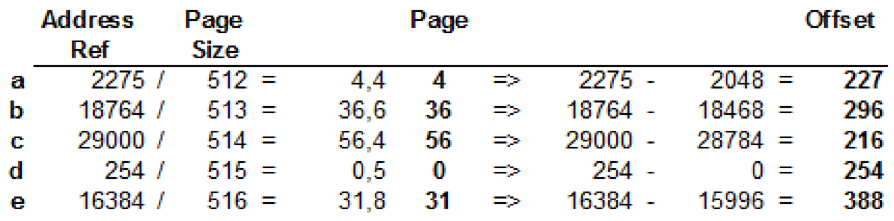


	Memory	First	Best	Worst
	100			
	500	312	312	217
	200	112	112	
	300	217	217	
	600	426	426	112
Total	1700	1067	1067	329
Over		**633**	**633**	**1371**

In this example, first-fir and best-fit turns out to be equally best.

Exercise 1.5 Page Numbers
Assuming a 1-KB page size, what are the page numbers and offsets for the following address reference s (provided as decimal numbers)?
a) 2275
b) 18764
c) 29000
d) 254
e) 16384

Answer 1.5, see solution 8.17 Alternative A



	Address Ref		Page Size			Page						Offset
a	2275	/	512	=	4,4	**4**	=>	2275	-	2048	=	**227**
b	18764	/	513	=	36,6	**36**	=>	18764	-	18468	=	**296**
c	29000	/	514	=	56,4	**56**	=>	29000	-	28784	=	**216**
d	254	/	515	=	0,5	**0**	=>	254	-	0	=	**254**
e	16384	/	516	=	31,8	**31**	=>	16384	-	15996	=	**388**

Chapter 9

Exercise 2.1 Page Reference String
Consider the following page reference string:

1, 3, 2, 4, 2, 1, 5, 3, 2, 1, 6, 3, 7, 6, 3, 2, 1, 7, 3, 6.

How many page faults would occur for the following replacement algorithms, assuming four, and six frames?
Remember that all frames are initially empty, so your first unique pages will cost one fault each.

- FIFO replacement
- Optimal replacement
- LRU replacement

Example Table

Page Reference String

Frames	1	3	2	4	2	1	5	3	2	1	6	3	7	6	3	2	1	7	3	6	Page Faults
1																					
2																					
3																					
4																					
Fault																					= 0

Frames	1	3	2	4	2	1	5	3	2	1	6	3	7	6	3	2	1	7	3	6	
1																					
2																					
3																					
4																					
5																					
6																					
Fault																					= 0

Answer 2.1, see solution 9.8 Alternative A

Page Reference String | Page Faults

FIFO

Frames	1	3	2	4	2	1	5	3	2	1	6	3	7	6	3	2	1	7	3	6
1	1						5			1	6	3	7			2	1	7	3	6
2		3					3			5	1	6	3			7	2	1	7	3
3			2				2			2	5	1	6			3	7	2	1	7
4				4			4			4	2	5	1			6	3	7	2	1
Fault	1	1	1	1			1			1	1	1	1			1	1	1	1	1

= 14

FIFO

Frames	1	3	2	4	2	1	5	3	2	1	6	3	7	6	3	2	1	7	3	6
1	1												7	6						
2		3											1	7						
3			2										3	1						
4				4									2	3						
5							5						4	2						
6											6		5	4						
Fault	1	1	1	1			1				1		1	1						

= 8

OPT

Frames	1	3	2	4	2	1	5	3	2	1	6	3	7	6	3	2	1	7	3	6
1	1						1				1		1			1				
2		3					3				3		3			3				
3			2				2				2		7			7				
4				4			5				6		6			2				
Fault	1	1	1	1			1				1		1			1				1

= 9

OPT

Frames	1	3	2	4	2	1	5	3	2	1	6	3	7	6	3	2	1	7	3	6
1	1												1							
2		3											3							
3			2										2							
4				4									7							
5							5						5							
6											6		6							
Fault	1	1	1	1			1				1		1							

= 7

LRU

Frames	1	3	2	4	2	1	5	3	2	1	6	3	7	6	3	2	1	7	3	6
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
4				4	4	4	5	5	5	5	6	6	7	6	6	6	6	7	7	6
Fault	1	1	1	1			1				1		1	1				1		1

= 10

LRU

Frames	1	3	2	4	2	1	5	3	2	1	6	3	7	6	3	2	1	7	3	6
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
2		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	
3			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	
4				4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	
5							5	5	5	5	5	5	5	5	5	5	5	5	5	
6											6	6	7	6	6	6	6	7	7	
Fault	1	1	1	1							1		1	1				1		1

= 9

Exercise 2.2 Copy-on-write
What is the copy-on-write feature and under what circumstances is it beneficial t o use this feature? What is the hardware support required to implement this feature ?

Answer 2.2, see solution 9.3
When two processes are accessing the same set of program values (for instance, the code segment of the source binary), then it is useful to map the corresponding pages into the virtual address spaces of the two programs in a write-protected manner. When a write does indeed take place, then a copy must be made to allow the two programs to individually access the different copies without interfering with each other. The hardware support required to implement is simply the following: on each memory access, the page table needs to be consulted to check whether the page is write protected. If it is indeed write protected, a trap would occur and the operating system could resolve the issue.

Exercise 2.3 Hit ratio of the TLB
Explain the term hit ratio.

Answer 2.3, see p 437, see solution 9.42
The hit ratio for the TLB refers to the percentage of virtual address translations that are resolved in the TLB rather than the page table. Clearly, the hit ratio is related to the number of entries in the TLB , and the way to increase the hit ratio is by increasing the number of entries in the TLB . This, however, does not come cheaply, as the associative memory used to construct the TLB is both expensive and power hungry.

Exercise 2.4 TLB Reach
Explain the term TLB reach. And explain three ways of how to increase the TLB reach.

Answer 2.4, see p 437, see solution 9.43
The TLB reach refers to the amount of memory accessible from the TLB and is simply the number of entries multiplied by the page size. Ideally, the working set for a process is stored in the TLB . If it is not, the process will spend a considerable amount of time resolving memory references in the page table rather than the TLB.
Three ways to increase the TLB reach

1. If we double the number of entries in the TLB ,we double the TLB reach. However, for some memory-intensive applications, this may still prove insufficient for storing the working set.
2. Another approach for increasing the TLB reach is to either increase the size of the page or provide multiple page sizes. If we increase the page size - say, from 8 KB to 32 KB - we quadruple the TLB reach. However, this may lead to an increase in fragmentation for some applications that do not require such a large page size as 32 KB. Alternatively, an operating system may provide several different page sizes.
3. Providing support for multiple page sizes requires the operating system - not hardware - to manage the TLB . For example, one of the fields in a TLB entry must indicate the size of the page frame corresponding to the TLB entry. Managing the TLB in software and not hardware comes at a cost in performance.