BSS Practicum Week 3 Exercises

Andre Nanninga
23-may-2014

Chapter 8

> *1.1 Why are page sizes always powers of 2?*

The translation between pages and frames is done by taking the head of a memory address and replacing it with a frame address.

With a page size of 128 bytes the every bit beyond 7 bits is considered the head. So the address `11010 1001` can simply be translated by taking the head: `11` and finding the matching frame to this page number, for example `01`. The resulting physical address would then be `01010 1001`.

When a page size is not a power of 2 this would not be possible.

> *1.3 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?*

320 nanoseconds. We must first access memory for the page table and frame number (160 nanoseconds) and then access the desired byte in memory (160 nanoseconds).

> *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 pagetable entry in the associative registers takes zero time if the entry is here.)*

The access time when a reference is found is 160 nanoseconds, 0 nanoseconds to find the entry in the TLB and 160 nanoseconds to access the memory. The access time when no reference is found is 320 nanoseconds as explained above.

> *1.4 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?*

	Memory in KB	**First-Fit**	**Best-Fit**	**Worst-Fit**
	100			
	500	*P1* (312 KB)	*P1* (312 KB)	*P2* (217 KB)
	200	*P3* (112 KB)	*P3* (112 KB)	
	300	*P2* (217 KB)	*P2* (217 KB)	
	600	*P4* (426 KB)	*P4* (426 KB)	*P3* (112 KB)
Total	**1700**	**1067**	**1067**	**329**

Both the first-fit and best-fit provide the same solution and efficiency while the worst-fit algorithm is unable to even assign each process some memory.

> *1.5 Assuming a 1-KB page size, what are the page numbers and offsets for the following address reference s (provided as decimal numbers)?*

A 1 KB page size means page sizes of 10 bits, converting the address from decimal to binary we can take the last 10 bits as the page offset and the preceeding bits as the page number.

```
2275 (dec)  = 1000 1110 0011      (bin)
page number = 10                  = 2
page offset =   00 1110 0011      = 227
```

```
18764 (dec) = 0100 1001 0100 1100 (bin)
page number = 0100 10             = 18
page offset =        01 0100 1100 = 332
```

```
29000 (dec) = 0111 0001 0100 1000 (bin)
page number = 0111 00             = 28
page offset =        01 0100 1000 = 328
```

```
254 (dec)   = 0000 1111 1110      (bin)
page number = 00                  = 0
page offset =   00 1111 1110      = 254
```

```
16384 (dec) = 0100 0000 0000 0000 (bin)
page number = 0100 00             = 16
```

```
page offset =        00 0000 0000 = 0
```

Chapter 9

> *2.1 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*

FIFO replacement

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

12 Faults

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		7				7		7	
2		3					3				3		3				1		1	
3			2				2				2		2				2		3	
4				4			4				4		4				4		4	

5							5				5		5				5		5	
6											6		6				6		6	
Fault	*1*	*1*	*1*	*1*			*1*				*1*		*1*				*1*		*1*	

9 Faults

Optimal replacement

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

8 Faults

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 Faults

LRU replacement

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

12 Faults

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																				

> *2.2 What is the copy-on-write feature and under what circumstances is it beneficial to use this feature? What is the hardware support required to implement this feature?*

Copy-on-write is a feature implemented by an operating system to help preserve memory. Normally when a process forks itself a copy of it's memory is made so that both the parent as the child have different memory blocks assigned to them. With the copy-on-write feature the child is initially assigned the same memory block as the parent. This helps preserve memory. Only when either the parent of the child tries to write to these memory blocks is a copy made and are both the parent and child assigned their own blocks.

No special hardware is needed but most operating systems do reserve some memory blocks for when such a copy-on-write happens.

> *2.3 Explain the term hit ratio.*

The term *hit ratio* refers to the percentage of logical address to physical address translated which are cached in the TLB. For example, a 80% hit ratio means that 80% of the memory block translations are cached in the TLB, the other 20% must first be looked up in the page table.

> *9.43 Explain the term TLB reach. And explain three ways of how to increase the TLB reach.*

The term *TLB reach* refers to amount of memory accessible by the TLB and is simply the numbers of entries multiplied by the page size.

One way to increase the reach of the TLB is to simply increase the number of entries in the TLB. Another method is to increase the page size but this may lead to inefficient use of memory. The third way is to use multiple page sizes, this way we can make efficient use of memory while still increasing the TLB. The downside however is that the OS itself needs to manage the TLB for this method to work.