Cole Vikupitz

CIS 415

16 May 2017

Assignment 3

Textbook Questions

1. OSC 7.16: *In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods (months). Resources break or are replaced, new processes come and go, and new resources are purchased and added to the system. If deadlock is controlled by the banker's algorithm, which of the following changes can be made safely (without introducing the possibility of a deadlock), and under what circumstance?*

a. *Increase Available (new resources added)*

By adding more resources, a process may not have to wait for resources held by other processes, so this change will bring no risk of deadlock.

b. *Decrease Available (resources are permanently removed from the system)*

By taking resources away, we risk the possibility of having a process that is unable to complete its job, because the maximum amount of resources it needs is now greater than the number of resources in the system. Therefore, this change will bring a risk of a deadlock.

c. *Increase Max for one process (the process needs or wants more resources than allowed)*

This change will bring a risk of a deadlock. This is because a process may now need more resources than what is available, therefore risking that process never completing.

d. *Decrease Max for one process (the process decides it does not need that many resources)*

Assuming no process requesting more resources than what is available existed before, this should not bring a risk of deadlock, since the process should still not need as many resources as there are available at one time.

e. *Increase the number of processes*

We may add another process with no risk of deadlock, only if the process's maximum claim of resources is less than or equal to the system's available resources. If not, the algorithm should raise an error or flag.

f. *Decrease the number of processes*

Assuming no processes were at risk of a deadlock before, this change should bring no risk of deadlock. With less processes, there should be less competition over resources.

2. OSC 8.13: *Compare the memory organization schemes of contiguous memory allocation, pure segmentation, and pure paging with respect to the following issues:*

a. *External fragmentation*

Both contiguous memory allocation and segmentation are affected by external fragmentation and the need for compaction since both methods rely on allocating memory blocks of varying size. With contiguous memory allocation, we are reserving memory blocks to contain the whole process. With segmentation, we reserve multiple memory blocks for the program's segments, or logical units, which can be many different sizes. When the processes are complete, the blocks are freed and bigger processes that are ready to run are unable to since the old memory blocks are not large enough. Paging is unaffected by this since it creates equal sized memory blocks (frames and pages) no matter how big the program is.

b. *Internal fragmentation*

Since contiguous memory and segmentation are affected by external fragmentation, they are not affected by internal fragmentation. However, paging is affected by this since some programs may require some number of frames plus a fraction of one. Paging will only allocate a whole number of frames for a program.

c. *Ability to share code across processes*

Since contiguous memory allocation reserves an entire memory block for a process, no other processes have access to its data, so this method does not allow for sharing code across processes. Segmentation allows for code sharing given the use of protection bits. Segments that share a segment level and have sharing permissions are allowed to share their data between each other. Paging also allows for code sharing, the same way segmentation allows it.

3. OSC 8.25: *Consider a paging system with the page table stored in memory.*

a. *If a memory reference takes 50 nanoseconds, how long does a paged memory reference take?*

We need to access the memory for the page table, which will take 50 nanoseconds, followed by accessing the byte we want in memory, taking another 50 nanoseconds. Therefore, a paged memory reference will take **100 nanoseconds**.

b. *If we add TLBs, and 75% of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds if the entry is present.)*

The effective memory reference time is found by calculating $(2 - \alpha + (\alpha * \varepsilon / \tau)) * \tau$, where α is the hit ratio (75%), ε is the associative lookup (2 nanoseconds), and τ is the memory cycle (50 nanoseconds).

$T_{eff} = (2 - 0.75 + (0.75 * 2) / 50) * 50 = (1.25 + (1.5 / 50)) * 50 = (1.25 + 0.03) * 50$

$= 1.28 * 50 =$ **64 nanoseconds**

4. OSC 9.14: *Assume that a program has just referenced an address in virtual memory. Describe a scenario in which each of the following can occur. (If no such scenario can occur, explain why.)*

a. *TLB miss with no page fault*

The page is loaded into memory, but is not currently in the TLB.

b. *TLB miss and page fault*

The page is not in memory, so it needs to be loaded into memory, and also added to the TLB.

c. *TLB hit and no page fault*

The page was loaded into memory and was also in the TLB for quick access.

d. *TLB hit and page fault*

This cannot occur, since the TLB is a cache of recent virtual memory translations. We need to bring the page into memory before it can exist in the TLB.

5. OSC 9.21: *Consider the following page reference string:*

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1

Assume demand paging with 3 frames, how many page faults would occur for the following replacement algorithms?

a. *LRU replacement*

7	**2**	**3**	**1**	**2**	**5**	**3**	**4**	**6**	**7**	**7**	**1**	**0**	**5**	**4**	**6**	**2**	**3**	**0**	**1**
7	7	7	1	-	1	3	3	3	7	-	7	7	5	5	5	2	2	2	1
	2	2	2	-	2	2	4	4	4	-	1	1	1	4	4	4	3	3	3
		3	3	-	5	5	5	6	6	-	6	0	0	0	6	6	6	0	0

Total of **18 page faults**.

b. *FIFO replacement*

7	**2**	**3**	**1**	**2**	**5**	**3**	**4**	**6**	**7**	**7**	**1**	**0**	**5**	**4**	**6**	**2**	**3**	**0**	**1**
7	7	7	1	-	1	-	1	6	6	-	6	0	0	0	6	6	6	0	0
	2	2	2	-	5	-	5	5	7	-	7	7	5	5	5	2	2	2	1
		3	3	-	3	-	4	4	4	-	1	1	1	4	4	4	3	3	3

Total of **17 page faults**.

c. *Optimal replacement*

7	**2**	**3**	**1**	**2**	**5**	**3**	**4**	**6**	**7**	**7**	**1**	**0**	**5**	**4**	**6**	**2**	**3**	**0**	**1**
7	7	7	1	-	1	-	1	1	1	-	-	1	-	1	1	1	1	-	-
	2	2	2	-	5	-	5	5	5	-	-	5	-	4	6	2	3	-	-
		3	3	-	3	-	4	6	7	-	-	0	-	0	0	0	0	-	-

Total of **13 page faults**.