HW7: Deadlocks and Memory

Misha Ward

Deadlock:

Q7.11

Consider the traffic deadlock depicted in Figure 7.10.

a. *Show that the four necessary conditions for deadlock hold in this example.*

   A deadlock can definitely occur with this example. Cars can be considered the work or process while the streets are actually the resources the cars use.

   A deadlock needs four things: Hold and Wait, no preemption, circular wait, and mutual exclusion…

   Hold and Wait: in this example, each road has cars waiting for a different road.

   No Preemption: for each road, the road cannot be used by other cars until the cars from that road can go somewhere else.

   Circular Wait: for circular wait, each road of cars is waiting for other resources. For example, for roads r1, r2, r3, r4, r1 is waiting for r2, r2 for r3, r3 for r4, and r4 for r1. This circle prevents real progress from being made.

   Mutual Exclusion: Currently, the roads are big enough for only one way traffic preventing other cars from using the resource until those cars are cleared from the road.

b. *State a simple rule for avoiding deadlocks in this system.*

   The simple rule for avoiding this scenario is that cars can't continue to hold a road or the cross street which would allow for other cars to use the interception. Another way would to expand the roads themselves to allow multiple car lanes.

Q7.12

Assume a multithreaded application uses only reader-writer locks for synchronization. Applying the four necessary conditions for deadlock, is deadlock still possible.

   Yes, it is possible. The synchronization locks make it possible to have a hold and wait as a thread could hold one reader-writer lock waiting for another thread to process. Additionally, mutual exclusion is possible as reader-writers cannot be shared if there is a thread writing. Preemption is possible since you cannot take a lock away and circular waits happen due to a wait for the threads pending the opening of the lock and the thread within the lock.

Main Memory:

Q8.11

Given six memory partitions process for 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)? Rank the algorithms in terms of how efficiently they use memory.

First-fit

115KB into 300KB partition, 500KB into 600 partition, 358 into 750KB partition, 200KB into 350KB partition, 375KB into 392KB (750KB - 358KB) partition.

Final: (185KB, 100KB, 150KB, 200KB, 17KB, 125KB)

Best-fit

115KB into 125KB partition, 500KB into 600KB partition, 358KB into 750KB partition, 200KB into 200KB partition, and 375KB into 392KB (750KB - 358KB).

Final: (300KB, 100KB, 350KB, 0KB, 17KB, 10KB)

Worst-fit

115KB into 750KB, 500KB into 635KB, 358KB into 600KB, 200KB into 350KB, and 375KB must wait (no extra storage).

Based on this example, the worst one is worst-fit as it does not allow all the processes to use the memory. Best fit is very efficiency in terms of space-time but not necessarily runtime since its O(n) when it has to cycle through all the memory spaces to determine if its the best fit. If you want a speedy algorithm and you know this will be the actuals you will see in the system, then it might be best to do first-fit since it has an O(1) constant time runtime.

Q8.13

Compare the memory organization schemes of contiguous memory allocation, pure segmentation, and pure paging with respect to the following issues:

HW7: Deadlocks and Memory

Misha Ward

External Fragmentation:

With external fragmentation, contiguous allocation with partition that have fixed-sizes do not suffer but contiguous allocation with variable sized partitions do. Segmentation for external fragmentation does suffer but pure paging does not since partitions and pages have fixed sizes.

Internal Fragmentation:

Internal Fragmentation does not cause segmentation and partitions of variable sizes to suffer since a segment or partition is the exact size as it needs to be. Additionally, internal fragmentation could cause contiguous allocation with fixed sized partitions/paging to suffer.

Ability to share code across processes:

Unfortunately, for contiguous allocation does not have support for code sharing. However, segmentation does as long as the segments of a process keeps the data separate. This would then allow for code to be shared between processes. Finally, paging also allows processes to share code but at the page level granularity.

Q8.23

Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.

Formula: Logical address space size = # of pages * page size

a) How many bits are required in the logical address?

12 + 8 = 20 bits

256 × 4096= 2^8 * 2^12= 2^20

b) How many bits are required in the physical address?

12 + 6 = 18 bits

64 * 4039 = 2^6 * 2^12 = 2^18

Q8.28

Consider the following segment table:

HW7: Deadlocks and Memory

Misha Ward

a) 0,430 = 219 + 430 = 649
b) 1,10 = 2300 + 10 = 2310
c) 2,500 = 90 + 500, illegal reference, trap started
d) 3,400 = 1327 + 400 = 1727
e) 4,112 = 1952 + 112, illegal reference, trap started

Virtual Memory:

Q9.19

Assume that we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty frame is available or if the replaced page is not modified and 20 milliseconds if the replaced page is modified. Memory-access time is 100 nanoseconds.

Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?

Effective access time (200) = ((1 - p) * 100) + (p * (100 + ((1 - .70) * 8ms) + .(7 * 20ms))

200 = 100 - 100p + 100p + (.3 * 8ms)p + (.7*20 ms)p

200 = 100 + (2.4ms + 14 ms)p

200 = 100 + 16.4ms p

200 = 100 + 16.4ms p

100 = 16.4ms

p = 100/ 16.4ms

Q9.21

Consider the following page reference string:

....

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

- *LRU replacement*
- *FIFO replacement*
- *Optimal replacement*

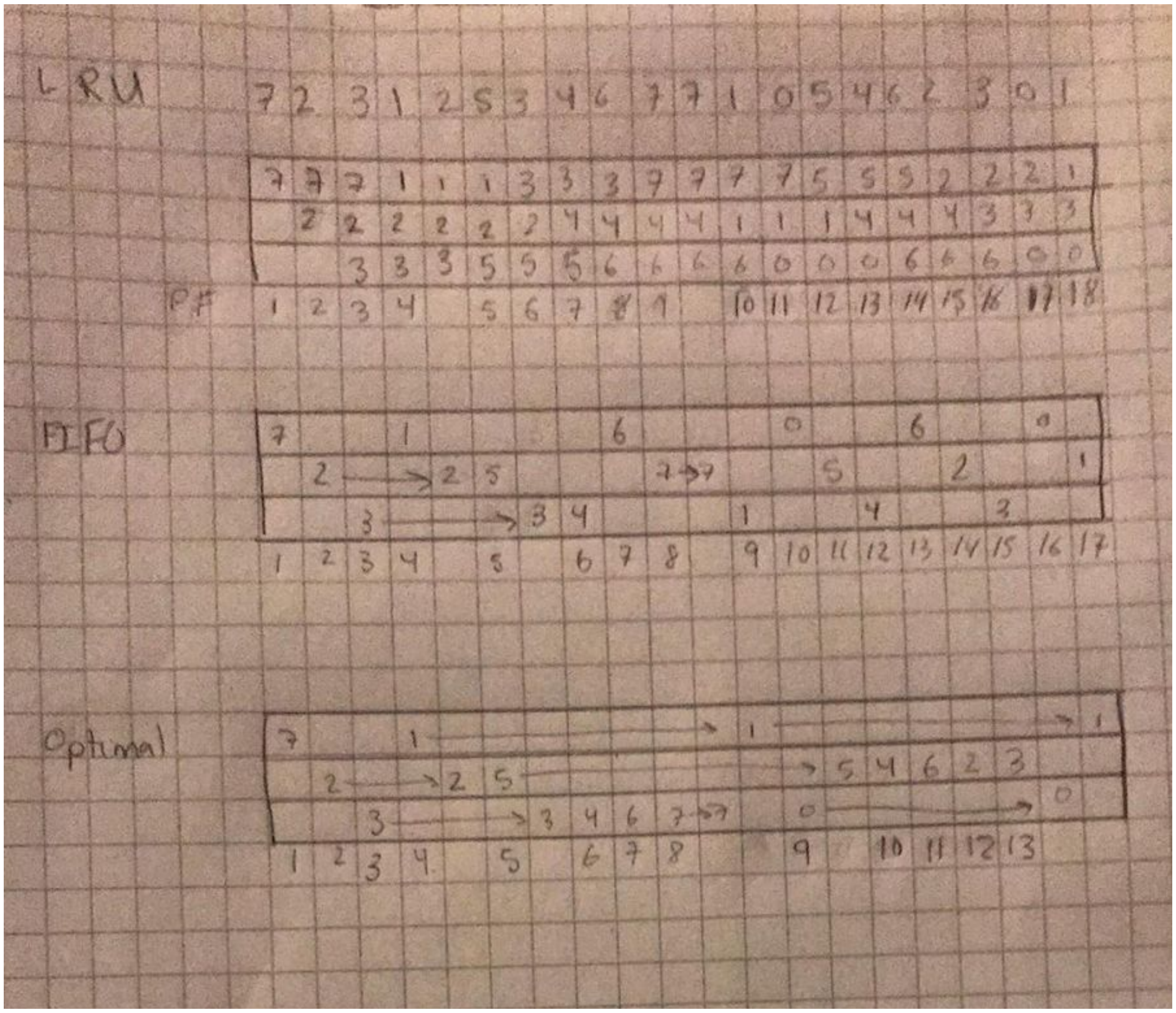
HW7: Deadlocks and Memory

Misha Ward

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

# of Frames	LRU	FIFO	Optimal
3	18	17	13

Work below:



LRU

	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	3	3	3	7	7	7	7	5	5	5	2	2	2	1
		2	2	2	2	2	2	4	4	4	4	1	1	1	4	4	4	3	3	3
			3	3	3	5	5	5	6	6	6	6	0	0	0	6	6	6	0	0
PF	1	2	3	4		5	6	7	8	9		10	11	12	13	14	15	16	17	18

FIFO

	7			1					6				0			6			0	
		2 →			→ 2	5				7 →	7			5			2			1
			3 →				→ 3	4				1			4			3		
	1	2	3	4		5		6	7	8		9	10	11	12	13	14	15	16	17

Optimal

	7			1 →							→	1 →							→	1
		2 →			→ 2	5 →							→	5	4	6	2	3		
			3 →				→ 3	4	6	7 →	7		0 →						0	
	1	2	3	4		5		6	7	8			9		10	11	12	13		

Q9.24

Discuss situations in which the least frequently used (LFU) page replacement algorithm generates fewer page faults than the least recently used (LRU) page replacement algorithm. Also discuss under what circumstances the opposite holds.

HW7: Deadlocks and Memory

Misha Ward

A situation when the LFU page replacement algorithm would generate fewer page faults includes a system where there could be 5 pages in memory with a sequence of memory accesses being 1 1 1 2 3 4 5 6 1, the sixth page would actually result in another number besides 1 from being used but another number like 2 being replaced with 6 allowing the system to keep 1 and not resulting in a page fault. However, if using the LRU algorithm, a sequence with the same amount of pages would be 1 2 3 4 5 6 2 as the six would replace 1 but two would not result in a page fault.

Q9.30

A page-replacement algorithm should minimize the number of page faults. We can achieve this by distributing heavily used pages evenly over all of memory, rather than having them complete for a small number of page frames. We an associate with page frame a counter of the number of pages associated with that frame. Then, to replace a page, we can search for the page frame with the smallest counter.

a) *Define a page-replacement algorithm using this basic idea. Specifically address these problems:*
   i) *What is the initial value of the counters?*
   ii) *When are counters increased?*
   iii) *When are counters decreased?*
   iv) *How is the page to be replaced*
b) *How many page faults occur for your algorithm for the following reference string with four page frames?*
c) *What is the minimum number of page faults for an optimal page strategy for the reference string in part b with four page frames?*

a)

i) initial counter should be 0…

ii) counters should be increased when a new page is linked to a frame.

iii) the counter is decreased when the page is not linked to the frame anymore.

iv) The page is replaced when selected as the frame with the smallest counter and FIFO queue should be used if there are any ties or issues.

b)

Couldn't finish the entire problem, was still working on the calculations :(

c)

Couldn't finish the entire problem, was still working on the calculations :(