CS 346 – Review for Test #2 (Scheduling, deadlock, memory and disk management)

1. Consider a system with 4 types of resources: R1 (3 units), R2 (2 units), R3 (3 units), R4 (2 units). A non-preemptive resource allocation policy is used. At any given instant, a request is not entertained if it cannot be completely satisfied. A process unable to obtain a resource will sleep until its request can be granted. For example, if a process requests an R3 and an R4 and only an R4 is available, then the process is not given either resource and it goes to sleep. A sleeping process cannot do any computation. It will automatically awaken and acquire the resources when they are all available. Three processes, P1, P2 and P3, request resources as follows:

Process P1:	Process P2:	Process P3:
request 2 units of R2	request 2 units of R3	request 1 unit of R4
1 second of computation	2 seconds of computation	2 seconds of computation
request 1 unit of R3	request 1 unit of R4	request 2 units of R1
2 seconds of computation	2 seconds of computation	3 seconds of computation
request 2 units of R1	request 1 unit of R1	*release* 2 units of R1
2 seconds of computation	2 seconds of computation	2 seconds of computation
release 1 unit of R2	*release* 1 unit of R3	request 1 unit of R2
release 1 unit of R1	2 seconds of computation	1 second of computation
2 seconds of computation		request 1 unit of R3
release 1 unit of R3		1 second of computation
1 second of computation		
request 2 units of R4		
2 seconds of computation		

Assume that:

- Processes can do their computations in parallel on different CPUs.
- A request and release of a resource take an insignificant amount of time.
- When a job is finished, it releases all its resources.
- All processes begin at time 0.

a. Create a table itemizing what occurs to each process and each resource at each point in time. In other words, state what happens at time = 0 seconds, time = 1 second, etc. Continue the table until all jobs have completed. Mention whether resource requests are granted and if a job goes to sleep.

time	P1	P2	P3	R1 (3)	R2 (2)	R3 (3)	R4 (2)
0	request 2 R2	request 2 R3	request 1 R4		both given to P1	2 given to P2	1 given to P3
	compute	compute	compute				
1	request 1 R3	compute	compute			last given to P1	
	compute	compute	compute				
2	compute	request 1 R4	request 2 R1	2 given to P3			last given to P2
	compute	compute	compute				
3	request 2 R1	compute	compute				
	SLEEP	compute	compute				
4	SLEEP	request 1 R1	compute	last given to P2			
	SLEEP	compute	compute				
5	gains 2 R1	compute	release 2 R1	2 transfer P3->P1			
	compute	compute	compute				
6	compute	release 1 R3	compute			1 now available	
	compute	compute	compute				
7	release R1,R2	compute	request 1 R2	1 now available	1 transfer P1->P3		
	compute	compute	compute				
8	compute	done release 1,3,4	request 1 R3	2 now available		1 transfer P2->P3	1 now available
	compute		compute				
9	release R3		done release 2,3,4		1 now available	3 now available	2 now available
	compute						
10	request 2 R4						both given to P1
	compute						
11	compute						
	compute						
12	done release 1,2,4,4			3 now available	2 now available		2 now available

b. Will the processes all be able to finish without any deadlock? If there is deadlock, when does it occur and which processes are involved?

Refer to the table. There is no deadlock.

c. Do any processes have to sleep? If so, which one(s) and when?

Refer to the table. P1 sleeps during time 3 and 4.

2. Assume that a certain system has a total of 12 DVD drives. There exist four processes P1, P2, P3 and P4 requiring a maximum of 6, 5, 4, and 7 DVD drives respectively. At a given time, there is 1 drive allocated to P1, 5 to P2, 2 to P3 and 4 to P4. Is this a safe state? Why or why not? If it is safe, what is the safe sequence?

Total resources available *12*

Process	*Has*	*Needs*	*Future needs*
1	*1*	*6*	*5*
2	*5*	*5*	*0*
3	*2*	*4*	*2*
4	*4*	*7*	*3*

Initially, how many resources are unallocated? *0*

The only process that we can guarantee future needs is process 2.
Let it finish.
At that point, it will disappear, which effectively means its needs become zero.

Total resources available *12*

Process	*Has*	*Needs*	*Future needs*
1	*1*	*6*	*5*
2	*0*	*0*	*0*
3	*2*	*4*	*2*
4	*4*	*7*	*3*

How many resources are unallocated? *5*

Any other process can be given its maximum future needs since 5,2,3 are all <= 5.
Let's do process 1. Let it finish, so it will no longer need anything.

Total resources available: *12*

Process	*Has*	*Needs*	*Future needs*
1	*0*	*0*	*0*
2	*0*	*0*	*0*
3	*2*	*4*	*2*
4	*4*	*7*	*3*

How many resources are unallocated? *6*

Again, it doesn't matter whether we continue with process 3 or 4.
Let 3 finish, and certainly we will have enough resources to let 4 finish.

The safe sequence is 2,1,3,4 (others exist, but we must start with 2).
We are in a safe state since a safe sequence exists.

3. Suppose an operating system uses the non-preemptive Shortest Job Next scheduling algorithm. Consider the arrival and execution times for the following four processes.

Process #	Execution time	Request/arrival time
1	20	0
2	25	15
3	15	30
4	10	45

a. Construct a Gantt chart showing what interval of time each process executes and is in its ready state. Also indicate the system load during each interval of time.

Gantt chart: X = execute; R = ready

time interval	*Job 1*	*Job 2*	*Job 3*	*Job 4*	*# jobs ready or running*
0--5	*X*				*1*
5--10	*X*				*1*
10--15	*X*				*1*
15--20	*X*	*R*			*2*
20--25		*X*			*1*
25--30		*X*			*1*
30--35		*X*	*R*		*2*
35--40		*X*	*R*		*2*
40--45		*X*	*R*		*2*
45--50			*R*	*X*	*2*
50--55			*R*	*X*	*2*
55--60			*X*		*1*
60--65			*X*		*1*
65--70			*X*		*1*

b. What is the turnaround time of each process?

Turnaround time of each job:

1	*20*
2	*30*
3	*40*
4	*10*

c. What is the average system load between time = 0 and the time when the last process has finished executing?

It is the average of the numbers in the last column of part A, approximately 1.43.

d. What is the maximum system load? During what interval(s) of time is this maximum realized?

The maximum load is 2, during 15-20 and 30-55.

4. Suppose we wish to run two periodic tasks, T1 and T2, in a hard real-time system. The periods of the tasks are 9 for T1 and 12 for T2. The system uses the rate monotonic scheduling algorithm. Compute the maximum CPU utilization for the following five scenarios: when the execution time of task T1 is 1, 2, 3, 4 and 5.

Rate-monotonic scheduling																			
Job periods 9 and 12.																			
Let's examine execution times for the first job as 1,2,3,4,5.																			
First, schedule T1. Notice that it may start its period twice during one period of T2.																			
Try to maximize the execution of T2 in each case.																			
execution time of T1:		1				2				3				4				5	
		T1	T2			T1	T2			T1	T2			T1	T2			T1	T2
	1	X			1	X			1	X			1	X			1	X	
	2		X		2	X			2	X			2	X			2	X	
	3		X		3		X		3	X			3	X			3	X	
	4		X		4		X		4		X		4	X			4	X	
	5		X		5		X		5		X		5		X		5	X	
	6		X		6		X		6		X		6		X		6		X
	7		X		7		X		7		X		7		X		7		X
	8		X		8		X		8		X		8		X		8		X
	9		X		9		X		9		X		9		X		9		X
	10	X			10	X			10	X			10	X			10	X	
	11		X		11	X			11	X			11	X			11	X	
	12		X		12		X		12	X			12	X			12	X	
	13				13				13				13	X			13	X	
	14				14				14				14				14	X	
	15				15				15				15				15		
	16				16				16				16				16		
	17				17				17				17				17		
	18				18				18				18				18		
max exec of T2:		10					8				6				5				4
Let's calculate the utilization of each case.																			
exec of T1		1					2				3				4				5
period of T1		9					9				9				9				9
exec of T2		10					8				6				5				4
period of T2		12					12				12				12				12
U		0.94					0.89				0.83				0.86				0.89

5. Which type of memory or disk allocation is particularly susceptible to external fragmentation? How does it occur?

Contiguous; After we assign space to many files/processes, the free space remaining between them may become too small to accommodate something large.

6. Briefly explain the rationale of RAID bit striping.

If we have 8 or more disks, then let disk i store the i-th bit of every byte. Then, when we fetch data, access all 8 disks at the same time.

7. What is the difference between a page fault and a segmentation fault?

A page fault means you are trying to access something from the memory system (as in a fetch/load or a store), but the data is not in RAM currently. It needs to be brought in from disk. A segmentation fault means you are trying to access memory where you shouldn't be. The logical address you are accessing does not belong to your process or is, for example, code rather than data.

8. What is a working set? What is its purpose? What information is needed to compute the working set?

The working set is the collection of all logical pages that a process has used recently. The purpose is to help determine how many pages of physical memory should be allocated to a process. To compute the working set, we need to know how far back in time (i.e. number of memory references in the past) we should look.

9. Suppose main memory consists of 3 frames. Consider the following reference string of logical page numbers. 6, 1, 5, 6, 2, 1, 8, 1, 5. Determine the contents of RAM after each memory access in the reference string, and indicate which accesses are page faults for the following page replacement algorithms:
   a. Second chance FIFO

After inserting:	*6*	*1*	*5*	*6*	*2*	*1*	*8*	*1*	*5*
Contents	*6*	*6*	*6*	*6*	*2 (1)*	*2 (1)*	*2 (1)*	*2 (1)*	*5 (1)*
		1	*1*	*1*	*1 (0)*	*1 (1)*	*1 (0)*	*1 (1)*	*1 (0)*
			5	*5*	*5 (0)*	*5 (0)*	*8 (1)*	*8 (1)*	*8 (0)*
	PF	*PF*	*PF*		*PF*		*PF*		*PF*

   b. LRU

After inserting:	*6*	*1*	*5*	*6*	*2*	*1*	*8*	*1*	*5*
Contents	*6*	*6*	*6*	*6*	*6*	*6*	*8*	*8*	*8*
		1	*1*	*1*	*2*	*2*	*2*	*2*	*5*
			5	*5*	*5*	*1*	*1*	*1*	*1*
	PF	*PF*	*PF*		*PF*	*PF*	*PF*		*PF*
					1 is LRU	*5 is LRU*	*6 is LRU*		*2 is LRU*

c. clairvoyant

After inserting:	*6*	*1*	*5*	*6*	*2*	*1*	*8*	*1*	*5*
Contents	*6*	*6*	*6*	*6*	*2*	*2*	*8*	*8*	*8*
		1	*1*	*1*	*1*	*1*	*1*	*1*	*1*
			5	*5*	*5*	*5*	*5*	*5*	*5*
	PF	*PF*	*PF*		*PF*		*PF*		

10. In disk scheduling, the initial state of a disk is the list of track numbers that need to be visited, the current track location of the read/write head, and the direction (ascending or descending) of the read/write head.
    a. Give an example of an initial disk state for which the elevator algorithm has better performance than shortest seek next. Justify your answer.

    The head is at track 22 going up. The track requests are 10, 20, 30.
    Elevator algorithm visits in this order: go from 22 to 30, back down to 20 and then 10. Total distance traveled = |22 – 30| + |30 – 20| + |20 – 10| = 8 + 10 + 10 = 28
    SSN visits in this order: go from 22 to 20, then 30, then 10. Total distance traveled is |22 – 20| + |20 – 30| + |30 – 10| = 2 + 10 + 20 = 32.
    28 < 32, so elevator wins!

    b. Give an example of an initial disk state for which shortest seek next has better performance than the elevator algorithm. Justify your answer.

    The head is at track 15 going up. The track requests are 10, 11, 30, 31.
    Elevator algorithm visits in this order: go from 15 to 30, then 31. Turn around and go to 11, then 10. Total distance traveled = |15 – 30| + |30 – 31| + |31 – 11| + |11 – 10| = 15 + 1 + 20 + 1 = 37.
    SSN visits in this order: from 15 to 11 then 10 then 30 then finally 31. Total distance traveled = |15 – 11| + |11 – 10| + |10 – 30| + |30 – 31| = 4 + 1 + 20 + 1 = 26.
    26 < 37, so SSN wins!

11. Suppose a paging memory system has a RAM size of 128 MB and a page size of 8 KB. Assume that we wish to support a logical address space of 4 GB.
    a. For a logical address, how many bits are in its page offset?

    Since the page size is 2^{13} bytes, the page offset has 13 bits.

b. For a physical address, how many bits are in its page offset?

Both the logical and physical page offsets are the same, so this is also 13 bits.

c. How many bits are allocated to a logical page number?

4 GB → The logical address has a total of 32 bits. 32 – 13 = 19 bits for the logical page number.

d. How many bits are allocated to a physical page number?

128 MB → The physical address has 27 bits. 27 – 13 = 14 bits for the physical page number.

e. What is the total number of logical pages?

We allocate 19 bits for the logical page number, so there must be 2^{19} logical pages (~ ½ million).

f. What is the total number of physical pages (frames)?

Since we allocate 14 bits for the physical page number, there are 2^{14} physical pages (~ 16000).

g. How much space is needed to store the page table?

Each of the 2^{19} entries in the page table must identify a physical page number. The physical page number can be accommodated in 2 bytes, so we need 2^{20} bytes or 1 MB of space.

h. Suppose the integer variable x has virtual (i.e. logical) address 0x12345678. What is the range of hexadecimal address numbers that are in the same virtual page as x?

Recall that the page offset has 13 bits. That means the rightmost 13 bits of a logical address can range from all 0s to all 1s for all the addresses on the same logical page. In other words, a logical address has this form: xxxx xxxx xxxx xxxx xxxy yyyy yyyy yyyy, where the x bits give the page number and the y bits give the page offset. The binary representation of 0x12345678 is 0001 0010 0011 0100 0101 0110 0111 1000. All we need to do is keep the first 19 bits unchanged and let the last 13 bits vary. Thus the low and high addresses are:
0001 0010 0011 0100 0100 0000 0000 0000 = 0x12344000
0001 0010 0011 0100 0101 1111 1111 1111 = 0x12345fff