جامعة بيرزيت

BIRZEIT UNIVERSITY

Electrical and Computer Engineering Department

ENCS339 Operating Systems 1st Semester 2018/2019

Midterm Exam Instructor: Dr. Adnan H. Yahya Time: 90min

Q	ABET	Max	Earned
Q1	e	18	
Q2	e	15	
Q3	a	16	
Q4	c	20	
Q5	c	18	
Q6		20	
∑		107	

Student Name:__________________________ Student Number:___________

Please answer all questions using the exam sheets ONLY.

Please show all steps of your solutions. Max grade is:107.

Question 1 (18%) A computer system has only 32GB of **physical memory (RAM)**. The system has a 16KB **page size** and 48-bit logical address space. CPU generated addresses are 6 bytes each[yes: not a power of 2!].

(a) 2% Indicate on the diagram below which of the bits of the **logical address** of 48 bits are used for page number (**p**) and for offset (**d**). Most significant (MSB) is bit #0 and least significant (LSB) is bit #47
16KB=2**14 Bytes, thus 14 bits [34:47] are used for offset (displacement) and

0	10	20	30	34 47
				++++Displacement++

(b) 2% How many **frames** are there in the RAM?

RAM is 32GB, each frame is 16KB, # of Frames= 32GB/16KB=2MFrames [addressable using 21 bits]

(c). 2% Ignoring page table overhead and OS needs, how many **pages** can a process have (max) to be runnable in **contiguous** memory allocation mode?
32GB=2MPages (2 mega pages)

(d). 2% How many **bits** are **minimally** needed for frame number of this computer in page map tables (PMTs)?
21 bits to address the 2MFrames,

(e) 2% Given a 4GB Process what is the size of the Page Map Table (PMT) in **bytes** and **pages** if the PMT is flat (one level)?
Flat means the table has 4GB/16KB= 1/4MPages= 256Kpages. Each page needs 21 bits or 4bytes for addresses of frames for a total of 256x4K=1MB. 1MB =1MB/16KB=220/2**14=2**6=64 pages.**

(f) 2% Given the 4GB Process: how many levels are needed for the PMT using multi-level paging of PMT, if needed?
First level has 16KB/4Bytes=4KPages=4K*16KB=64MB.
Second level has 4K*4KPages=16M*16KB=256GB.
So we have only 2 levels.

(g)3% With a **two level** paging of PMT, find the maximum size (address space, in bytes) that a job **can** have?
256GM, as shown earlier.

(h) 3% How many levels of page tables would be required to map a full 48 bit virtual address space (top level: one page max)? Explain.
2 levels gave 256GB or 238B, 3 Levels will give (2**38) x 4K=(2**38) x (2**14)=2**52Bytes; So we need 3 levels.**
Another way: Each page has 4K entries. Needs 14 bits. So 14bits for displacement, 12bits for first level, 12 for second for a total of 14+12+12=38bits. The last 10 bits are for the third level! Note that size of such job with these 3 levels (last level has only 1K entries out of 4K) is 248B= 256TB**

Question 2 (15%) Consider a computer system involving 5 processes (P1, P2, P3, P4, P5) and 4 different types of resources (R1,R2,R3,R4). The current state of the processes and resources is reflected in the tables below.

Currently Available Resources			
R1	**R2**	**R3**	**R4**
1	**4**	**2**	**0**

	Current Allocation				Max Need				Still Needs			
Process	**R1**	**R2**	**R3**	**R4**	**R1**	**R2**	**R3**	**R4**	**R1**	**R2**	**R3**	**R4**
P1	0	1	1	2	0	3	1	2	**0**	**2**	**0**	**0**
P2	1	0	0	0	1	7	5	0	**0**	**7**	**5**	**0**
P3	1	3	5	4	2	3	5	6	**1**	**0**	**0**	**2**
P4	0	6	3	2	0	6	5	2	**0**	**0**	**2**	**0**
P5	0	0	1	4	0	6	5	6	**0**	**6**	**4**	**2**

(a) 5% Use Banker's algorithm to check if this system is currently deadlocked, or can any process become deadlocked if it continues working from the current state? Why or why not? If not deadlocked, give an execution order

Deadlocked □YES □ **NO**

P1 → [1,5,3,2]P3→[2,8,8,6]P2→[3,8,8,6]P4→[3,14,11,8]P5→[3,14,12,12] **Order: smallest index first**

P1 → [1,5,3,2]P4→P2→P3→P5; P1 → [1,5,3,2]P4→P2→P3→P5; ..

P4 → [1,10,5,2]P1→[1,11,6,4]P2→[2,11,6,4]P3→[3,14,11,8]P5→[3,14,12,12] **Order: P4 THEN smallest index first More exist**

If Not deadlocked: Execution Order is (just add indices): P1→ P3→ P2→ P4→ P5

2% Fill the following table:

Total Resources in the System			
R1	**R2**	**R3**	**R4**
3	**14**	**12**	**12**

(b)4% If a request from process P1 asks for the resource vector (0, 2, 0, 1).
Can the request be immediately granted? Why or why not? If yes, show an execution order. Explain your answer.

Request Can be Granted: □**YES** □ **NO**

Exceeds max of resource R4

If granted, Execution Order is (just add indices): P___→ P___→ P____→ P___→ P___

(c)4% If instead of (b), process P2 asks for the resource vector (0, 3, 2, 0), can the request be immediately granted? Why or why not? If yes, show an execution order. Explain your answer.

If (0, 3, 2, 0) is granted, Available becomes: [1,1,0,0], Still needs P2=[1,4,3,0]; Available is less than still needs for all. None can start. None can finish.

Request Can be Granted: □**YES** □ **NO If granted Available= [|]]]]]]]]]]]]]]]**

P1 → [1,5,3,2]P3→[2,8,8,6]P2→[3,8,8,6]P4→[3,14,11,8]P5→[3,14,12,12] **Order: smallest index first**

No process can finish.

If granted, Execution Order is (just add indices): P___→ P___→ P____→ P___→ P___

Question 3 (16%: 4% each) Consider a dynamic (contiguous) partitioning system in which the (free) memory consists of the following list of **holes** (free partitions), sorted by increasing memory address (all sizes are in Megabytes):

4 → 12 → 18 → 8 → 9 → 13 → 21 → 19

↑ Hole List Start Pointer (HLPTR)

Suppose a new process Pa requiring 11 MB arrives, followed by a process Pb needing 9MB of memory. Show the list of holes **after both of** these processes are placed in memory for each of the following algorithms (start with the original list of holes for each algorithm). Assume that **the hole List Start Pointer** is moved to the closest hole to the allocated (or to the newly created after each allocation): from left to right and **circular**.

i) First Fit-5%:

4 → 1 → 18 → 8 → 9 → 13 → 21 → 19

4 → 1 → 9 → 8 → 9 → 13 → 21 → 19

ii) Worst Fit -5%:

4 → 12 → 18 → 8 → 9 → 13 → 10 → 19

4 → 12 → 18 → 8 → 9 → 13 → 10 → 10

iii) Best Fit-:

4 → **1** → 18 → 8 → 9 → 13 → 21 → 19

4 → **1** → 18 → 8 → X → 13 → 21 → 19

iv) Best Fit Plus 3- meaning best fit but each process gets exactly (size +3) hole:

4 → 12 → **4** → 8 → 9 → 13 → 21 → 19

4 → X → **4** → 8 → 9 → 13 → 21 → 19

Question 4 (20%, 5% each) Consider the following process arrival, CPU burst (in milli-seconds) and explicit **priorities** of the processes A, B, C and D. Assume that **5** represents highest (preferred) priority and 1 lowest.

Draw the Gantt charts for and compute the turnaround and wait times and fill the table entries.
F: Finish Time, TA: TurnAround Time, W: Wait Tim

Process	Arrival time	CPU burst time	Priority	Priority/P			FCFS			SJF			SRTF		
				F	TA	W	F	TA	W	F	TA	W	F	TA	W
A	3	10	1	34	31	21	34	31	21	20	17	7	20	17	7
B	14	10	1	44	30	20	44	30	20	30	16	6	30	16	6
C	0	10	5	10	10	0	10	10	0	10	10	0	10	10	0
D	2	15	5	24	22	7	24	22	7	44	42	27	44	42	27
Avge				X	23.25	12	X	23.25	12	X	21.25	10	X	21.25	10

(a) Priority/preemptive:

0

Time	910 24 34 44
Process	CCCCCC DDDDDDDDD AAAAAAAAABBBBBBBBBBB

(b) FCFS (First Come First Served).

0

Time	910 24 34 44
Process	CCCCCC DDDDDDDDD AAAAAAAAABBBBBBBBBBB

(c) SJF (Shortest Job First).

0

Time	910 20 30 44
Process	CCCCCC AAAAAAAAABBBBBBBBB DDDDDDDDD CCCCCC BBBBBBBBB AAAAAAAAA DDDDDDDDD

(d) SRTF (Shortest Remaining Time First).

0

Time	910 20 30 44	
Process	CCCCCC AAAAAAAAABBBBBBBBB DDDDDDDDD CCCCCC BBBBBBBBB AAAAAAAAA DDDDDDDDD	

Question 5 (18%) The producer-consumer problem is a common example of cooperating processes. A **producer process** produces information that is consumed by a **consumer process**. Here, the producer process and the consumer process communicate using a **bounded buffer** implemented in shared memory.

Producer Process	**Consumer Process**

Memory region shared by both processes:

```
#define BUFFER_SIZE 10

typedef struct {
   . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Producer Process

```
item nextProduced;

while (1) {
   /* produce an item in nextProduced
*/
    while (((in + 1) % BUFFERSIZE) ==
out)
       ;  /* do nothing */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
}
```

Consumer Process

```
item nextConsumed;

while (1) {
    while (in == out)
       ;  /* do nothing */
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    /* consume the item in
nextConsumed */
}
```

a. (5%)Assume that the **Consumer Process** happens to be the first to run. Assume that the Consumer Process is allowed to run for a long time. **Select what happens. Explain your answer.**

1- The Consumer will be busy waiting **2- The buffer is full which produces an exception (fault).**

3- Buffer will be filled due to the long time **4- Control will be passed immediately to Producer process.**

in=out=0 and nothing can happen except busy waiting (2% for explanation)

b. (5%) Assume that the Consumer Process eventually is swapped out (or the very long time quantum is finished), and the **Producer Process** gets its chance to run. Assume that the Producer Process is allowed to run for a long time, (enough time to fill the buffer). **Select what happens. Explain your answer.**

1- The producer process will be busy waiting **2- Control will passed immediately to Producer process.**

3-Buffer will be filled due to the long time then process goes to busy waiting.

4-The buffer will overflow and an exception (interrupt) will be generated.

After the buffer is full. (in+1) % BUFFERSIZE=out and nothing can happen except busy waiting (2% for explanation)

c. (4%)For this part of the problem, assume we have re-started both processes, so they are just ready to start, with the shared memory variable having their values as initialized in the code. Describe one very fortunate **(optimistic)** sequence of executions which allows the processes to keep doing useful work. Your answer might take the form: Producer process runs until _once__ then Consumer process runs once until _OR Producer process runs until _buffer is full then Consumer process runs until Buffer is empty (or any alternating arrangement: add 2 remove 2, add 3 remove 3 and so on).

d. (4%)Is this program in need of improvement? If so, Suggest at least one way to improve performance. YES, E.g. remove busy waiting for example by sleep awake,

Question 6 (20%) True or false, add a line (only one) of explanation as to WHY. Fill the table (-3% if not), (-3% max for no explanations)

1. **□True □False:** An operating system is a program that acts as an intermediary between the user of a computer and the computer hardware Users cannot access hardware except through OS
2. **□True □False:** A user-level process cannot modify its own page table entries OS task
3. **□True □False:** Context switch time on modern hardware is small enough to be ignored entirely when designing a CPU scheduler. All is relative: context switch time needs to compare to time quantum
4. **□True □False:** Races happen in processes when the final result is affected by execution order.
5. **□True □False:** In a multiprocessor system with enough CPUs (cores) a process gets assigned to a given processor (core) permanently to avoid context switches. No relation. Context switch even 1 core
6. **□True □False:** Paging avoids the problem of external fragmentation of memory in a multi-programming environment but has internal fragmentation. Any frame can be used, so no external fragmentation. Can have 1 Byte or Full
7. **□True □False:** A process can move form a **ready** state to the **waiting** state, say if a device it needs becomes available. Through Running State
8. **□True □False: I**n a symmetric multiprocessor, threads cannot always be run on any processor. Symmetric means equal power/capability.
9. **□True □False:** An atomic operation is a machine instruction or a sequence of instructions that must be executed to completion without interruption. Finished in full in one go.
10. **□True □False:** Shortest Job First and Priority scheduling algorithms can lead to starvation? Short jobs keep coming all the time preventing longer jobs from being scheduled.
11. **□True □False:** Two processes reading from the same physical address access the same contents. One way of sharing
12. **□True □False:** A SJF scheduler may preempt a previously running longer job. Only after finishing previously running jobs we invoke the SJF scheduler.
13. **□True □False:** If all jobs have identical run lengths, a RR scheduler (with a time-slice much shorter than the jobs' run lengths) provides better average turnaround time than FIFO. Take much longer to finish, all jobs.
14. **□True □False:** The longer the time slice, the more RR scheduler looks like a FIFO scheduler. Most jobs will finish within one time quantum, if not all: FCFS.
15. **□True □False:** If a physical address is 32 bits and each page is 4KB, the top (Most Significant) 18 bits exactly designate the physical page number. 4KB needs 12 bits and leaves 20 not 18 for page#
16. **□True □False:** Paging approaches suffer from internal fragmentation, which decreases as the size of a page decreases. Any frame can be used, so no external fragmentation. Can have 1 Byte or Full
17. **□True □False:** Threads that are part of the same process share the same stack. Different stacks/scratchpads for different threads.
18. **□True □False:** With kernel-level threads, multiple threads from the same process can be scheduled on multiple CPUs simultaneously. Of course: parallelism is a good product of threading.
19. **□True □False:** With producer/consumer relationships and a finite-sized circular shared buffer, producing threads must wait until there is an empty element of the buffer. No writing to a full buffer: basic synchronization premise.
20. **□True □False:** A thread can hold only one lock at a time. As many.
Keys are for resources. A thred can have many locks.

Q	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F	□T □F