Noemi Lemonnier
#40001085

COMP 346 Theory Assignment 3

Question 1

I.

a) Relocatable programs are programs that can be accessed dynamically. Having a dynamic partitioning will allow programs to be loaded or accessed by sections and not all at once can do this.
b) A program is considered relocatable if it can be stored and accessed by jumping locations in memory via relocation registers, ranging from base to limit register. In a modern system, a relocatable program is one that cannot assume its location beforehand, which means it has to be relocatable.
c) For the OS management context having some processes that are relocatable can be very helpful! The OS management context will save time and memory space by only loading sections of processes needed in memory.

II.

Small page sizes	***Large page sizes***
- more frequent external fragmentation - slower access time - more context switches	- less external fragmentation - quicker access time - larger internal fragmentation

III.

Paging is more efficient than segmentation. It has a higher speed and it uses less memory than segmentation. Also, segmentation requires a programmer while paging is not visible to him/her. It is used to access large linear address space without requiring more physical memory. Segmentation also require that the programmer must be aware of memory limit.

IV.

Segmentation lets programs and data to be divided up into segments, which helps sharing and protection. Also, it offers security of individual segments of a program. Paging does not help with sharing like segmentation does.

Question 2

a)

Critical section for wait()	***Critical section for signal()***

if(sem.value <0){
    save_state(current);
    State[current] = Blocked;
    Enqueue(current, sem.queue);
    current = select_from_ready_queue();
    State[current] = Running;
    restore_state(current);
}
	if(sem.value <= 0){
	    k = Dequeue(Sem.queue);
	    State[k] = Ready;
	    Enqueue(k, ReadyQueue);
	}

Noemi Lemonnier
#40001085

b)
If sem.value is bigger than 0 and it goes through wait(). sIt won't execute the if statement, enable interrupts and go signal(). In signal it disable interrupts and sem.value becomes x+1 and it execute the if statement and then enable interrupts. This will simply signal without being able to execute the wait because sem.value will never be below 0. This show that many processes will be signalled and none will have to wait which can lead to a problem of consistency because none state will be saved.
c)
No because disabled/enabled interrupts are used to make the operation atomic and they will switch off interrupts on the hardware level compared to synchronized that is on a software level.

Question 3
EAT = hit rate * hit time + miss rate * miss penalty
EAT = 80% * 1ms + 20%*(1+0.2ms)
EAT = 0.8 + 0.24
EAT= 1.04 ms

Question 4
a)

	0	**1**	**2**	**0**	**1**	**2**	**0**	**1**	**2**	**3**	**6**	**7**	**6**	**7**	**0**	**1**	**2**	**3**	**4**
F1	0	0	0	0	0	0	0	0	0	3	3	3	3	3	0	0	0	3	3
F2		1	1	1	1	1	1	1	1	1	6	6	6	6	6	1	1	1	4
F3			2	2	2	2	2	2	2	2	2	7	7	7	7	7	2	2	2

▨ swap
■ vault
There is 11 vaults.
b)

	0	**1**	**2**	**0**	**1**	**2**	**0**	**1**	**2**	**3**	**6**	**7**	**6**	**7**	**0**	**1**	**2**	**3**	**4**
F1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
F2		1	1	1	1	1	1	1	1	1	1	7	7	7	7	7	2	2	2
F3			2	2	2	2	2	2	2	3	6	6	6	6	6	6	6	3	4

▤ swap & vault
There is 10 vaults.

Noemi Lemonnier
#40001085

c)

	0	1	2	0	1	2	0	1	2	3	6	7	6	7	0	1	2	3	4
F1	0	0	0	0	0	0	0	0	0	3	3	3	3		0	0	0	3	3
F2		1	1	1	1	1	1	1	1	1	6	6	6	6	6	1	1	1	4
F3			2	2	2	2	2	2	2	2	2	7	7	7	7	7	2	2	2
ws	0	1	2						{0,1,2}										
ws										{1,2,3}									
ws											{2,3,6}								
ws													{3,6,7}						
ws														{6,7}					
ws															{6,7,0}				
ws																{7,0,1}			
ws																	{0,1,2}		
ws																		{1,2,3}	
ws																			{2,3,4}

▨ vault

There is 11 vaults.

Question 5

a)

Having a page table instead of a mapped-memory for the CPU will have an advantage in access time: the CPU memory will be faster to access.

b)

Having a page table instead of a mapped-memory for the CPU will have an disadvantage in terms of CPU memory: it takes a little of memory to stored the page table, it is actually quite a big amount of memory used by it.

Question 6

i)

A global replacement algorithm is allowed to choose any page in memory while a local page replacement algorithm can only choose pages within the same process/processes sharing a memory partition.

ii)

A global replacement algorithm will lack of scalability while a local page replacement algorithm has scalability and principle of locality.

Question 7

i)

If $T_{PF} > T_{FS}$, then pages faults occurrence < to service a page fault → best multiprogramming

ii)

If $T_{PF} < T_{FS}$, then pages faults occurrence > to service a page fault → terrible multiprogramming

Noemi Lemonnier
#40001085

iii)
If $T_{PF} = T_{FS}$, then system will perform a little bit better even if page faults occurrence is still stable.

Question 8
Advantage: This system will not require to wait for I/O when it is done.
Disadvantage: This can lead to starvation or even deadlock depending on the poor quality of the code.

Question 9
a)
Pre-emptive scheduling is when a process switches from running state to ready state or from waiting state to ready state. Non pre-emptive scheduling is when a process terminates or switches from running to waiting for state this kind of CPU scheduling.
Strict non pre-emptive scheduling will likely not be used in a computer system because it does not allow priority, context switch and it will likely consume all CPU cycle without caring about waiting jobs.
b)
If the quantum size is too small it will require context switching more often. Although the down side is that this will slow down the system significantly.
If the quantum size is larger, it will increase the response time by doing First In First Out scheduling. Although the down side is that this will decrease the degree of multiprogramming.

Question 10
The multilevel feedback queue-scheduling algorithm, in contrast, allows a process to move between queues.
This algorithm will allow higher priority queues to run for a longer time and lower priority queue to run for less time on the CPU. If a process takes too much time, it will be lowered its priority to allow other processes that did not run yet, this will prevent starvation. Although, this priority hierarchy does not allow fairness.

Question 11 (can refer to attach .xls file)
a)

1. ***FCFS scheduling***

P0	P1	P2	P3	P4
20	35	56	63	75

2. ***Non-preemptive SJF scheduling***

P3	P4	P1	P0	P2
7	19	34	54	75

3. ***Non-preemptive priority scheduling***

P1	P4	P0	P2	P3
15	27	47	68	75

Noemi Lemonnier
#40001085

4. ***Pure Round-Robin scheduling with the quantum = 3***

process	space	3*?	what's left
P0	20	18	2
P1	15	15	0
P2	21	21	0
P3	7	6	1
P4	12	12	0

P0	P1	P2	P3	P4	P0	P1	P2	P3
3	6	9	12	15	18	21	24	27
P4	P0	P1	P2	P4	P0	P1	P2	P4
30	33	36	39	42	45	48	51	54
P0	P1	P2	P0	P2	P0	P2	P3	
57	60	63	66	69	71	74	75	

b)

	W(P0)	W(P1)	W(P2)	W(P3)	W(P4)
FCFS	0	20	35	56	63
SJF	34	19	54	0	7
PRIORITY	27	0	47	68	15
ROBIN	51	45	53	68	42

W(P3) is 68 because it is 7 I could not do 3*3 so I did 3*2 + 1 unit at the end

c)

	R(P0)	R(P1)	R(P2)	R(P3)	R(P4)
FCFS	0	20	35	56	63
SJF	34	19	54	0	7
PRIORITY	27	0	47	68	15
ROBIN	0	3	6	9	12

d)

	TT(P0)	TT(P1)	TT(P2)	TT(P3)	TT(P4)
FCFS	20	35	56	63	75
SJF	54	34	75	7	19
PRIORITY	47	15	68	75	27
ROBIN	71	60	74	75	54

TT(P3) is 75 as it is the last process