Mohammad Ali Zahir

ID: 40077619

COMP 346 Theory Assignment 3

1.

i.

a) A relocatable program is a program that can be executed without any changes, and from any address from the memory.

b) If the data has a relative address, then it is relocatable. However, if it an absolute address (cannot be changed), then it would not be relocatable.

c) If the programs were not relocatable, this would mean that many programs would be running in the same memory location. This would cause the execution of the programs to be slow. Then, they would find an open memory address and be executed there.

ii.

Small sizes pages means that there will be less wasted memory versus the large page sizes. The waste of memory in the large page size may cause internal fragmentation. There will be more page faults for smaller page size compared to larger page sizes.

iii.

There will be no external fragmentation. It also allows for easy swapping between disk and memory because both frames and pages are of the same size.

iv.

There is no internal fragmentation. In terms of the paging mechanism, there will be less memory space occupied in the segmentation mechanism.

2.

a) Critical section for wait():

```
if (sem.value < 0) {
    save_state (current) ; // current process
    State[current] = Blocked; //A gets blocked
    Enqueue(current, sem.queue);
    current = select_from_ready_queue();
    State[current] = Running;
    restore_state (current); //B starts running
}
```

Critical section for signal():

```
if (sem.value <= 0){
  k = Dequeue(sem.queue);
  State[k] = Ready;
  Enqueue (k, ReadyQueue);
 }
```

b) Let sem.value = 1. If we go through the wait method, the if statement is not executed and it automatically goes to enable interrupts. After this, it would go to the signal() method, where it would first disable interrupts and then change the value of sem.value to 2. This will then skip the if statement and then enable interrupt. This will just signal without execute the wait since there is nowhere in the code where the value of the sem is reset to 0.

c) No it will not be executed. We need to be able to switch interrupts on a hardware level, which is used to make the operation atomic. Hence, if B is disabled, we would not have synchronisation and it will not be atomic.

3.

We know that our page fault rate is 0.02

Page fault service time = 20 milliseconds → 20 000 μs

TLB access time = 0.2 μs

Memory access time = 1 μs

Page table access time = 1 μs (same time as the memory access, since they both reside in the memory)

We have to first find out the effective memory access time but only for the page hit:

- Hit time: memory access time + TLB access time = 1.2 μs
- Miss time: memory access time + TLB access time + Page table access time = 2.2 μs

$$Effective\ memory\ access\ time\ = (hit\ rate\ *\ hit\ time)\ +\ (miss\ rate\ *\ miss\ time)$$

$$Effective\ memory\ access\ time\ = (0.8\ *\ 1.2)\ +\ (0.2\ *2.2)$$

$$Effective\ memory\ access\ time\ = 1.4\ \mu s$$

Now we have to the effective for the page fault with the value that we found above being the new hit time

$$Effective\ memory\ access\ time\ = (hit\ rate\ *\ hit\ time)\ +\ (miss\ rate\ *\ miss\ time)$$

$$Effective\ memory\ access\ time\ =\ (0.98 * 1.4)\ +\ (0.02\ *\ 20\,000)$$

$$Effective\ memory\ access\ time\ =\ (hit\ rate\ *\ hit\ time)\ +\ (miss\ rate\ *\ miss\ time)$$

$$Effective\ memory\ access\ time\ =\ 401.37\mu s$$

Hence, the effective memory access time for the system is $401.37\mu s$

4. Let R={0, 1, 2, 0, 1, 2, 0, 1, 2, 3, 6, 7, 6, 7, 0, 1, 2, 3, 4}

a) LRU algorithm (least recently used)

	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

We will highlight where the page faults occurred on the table, hence based on this algorithm we would have 11 page faults.

b) Belady Optimal Algorithm (which will be used the in the most future)

	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

We will highlight where the page faults occurred on the table, hence based on this algorithm we would have 10 page faults.

c) Working set algorithm with Δ=3 (Δ represents window size)

	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
Δ	{0)	{0,1}	{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}	{1,2,3}	{2,3,6}	{3.6,7}	{3.6,7}	{0,6,7}	{0.6,7}	{0.1,7}	{0.1,2}	{3.1,2}	{3.4,2}

We will highlight where the page faults occurred on the table, hence based on this algorithm we would have 11 page faults.

5.

a) The big advantage of this system would be the the difference in access time, which is that the CPU memory can be accessed faster.

b) The disadvantage of this system would be the memory efficiency, which is that you would use more memory to use the page table than the amount it would take to store it.

6.

i) The global replacement algorithm allows us to choose any page that we would want in the memory, contrary to the local page replacement algorithm which only allows to choose pages in the memory which are shared through the same processes.

ii) The main disadvantage for the global replacement algorithm would be the lack of scalability. This can result in poor system performance, which would duplicate processes making your memory even slower.

7.

i) $T_{PF} > T_{FS}$: We would have less page faults than service page faults, which would lead to the best multiprogramming result.

ii) $T_{PF} < T_{FS}$: We would have more page faults than service page faults, which would lead to the worst multiprogramming result.

iii) $T_{PF} = T_{FS}$: We would have a relatively stable system based on the number of faults being relatively close to each other.

8.

Advantages: Would not need for the File I/O to manually close the file (using the .close() method)

Disadvantages: Since it does not go through File I/O, we are susceptible to privacy leaks as well deadlock since it will always wait for the ending (.close()) state.

9.

a) Pre-emptive scheduling is a form of scheduling which is done when the process switches from running state to a ready state, or even more specifically more waiting to ready state. Non-pre-emptive scheduling would be when the process switches from running to a waiting state. By using strict non-pre-emptive scheduling we would be canceling priority and context-switching, hence this is not a viable option.

b) A low quantum size would mean that more context switching would happen, which will slow the system down significantly thus leading to poor overall system performance. A high quantum size would mean less context switching would happen, but response time would still increase due to the fact it will use FIFO scheduling, which is a bad way to do multiprogramming.

10.

The first advantage that we have for such a queue is that we can allow process to move between queues. Another advantage is that the more higher a priority queue priority is, it allows for more time on the CPU. The more process a time takes, it will lower the priority for such

process, which will allow the other processes to run on the CPU as well. Because of this, it will prevent starvation on our program, but allows eliminates fairness because low priority process might run at the same as high priority ones.

11.

Service times will be written in the cell

a) FCFS scheduling: First come first serve

P0 (20)	P1 (35)	P2 (56)	P3(63)	P4(75)

b) Non-pre-emptive SJF scheduling: Shortest Job First

P3(7)	P4(19)	P1(34)	P0{54)	P2(75)

c) Non-preemptive priority scheduling : lower number means higher priority

P1(15)	P4 (27)	P0 (47)	P2 (68)	P3 (75)

d) Pure Round-Robin scheduling with the quantum = 3: Find highest possible multiple of 3

Process	**Service Time**	**Highest multiple of 3**	**Remainder**
P0	20	18	2
P1	15	15	0
P2	21	21	0

P3	7	6	1
P4	12	12	0

P0 (3)	P1 (6)	P2 (9)	P3 (12)	P4 (15)	P0 (18)	P1 (21)	P2 (24)	P3 (27)

After the first wave, the used service time for each process updated to:

P0: 6, 14 remaining
P1: 6, 9 remaining
P2: 6, 15 remaining
P3: 6, 1 remaining
P4: 3, 9 remaining

P4 (30)	P0 (33)	P1 (36)	P2(39)	P4 (42)	P0 (45)	P1 (48)	P2 (51)	P4 (54)

After the second wave, the used service time for each process updated to:

P0: 12, 8 remaining
P1: 12, 3 remaining
P2: 12 ,9 remaining
P3: 6, 1 remaining
P4: 9, 0 remaining

So far the process P4 has been used to it's maximum time

P0 (57)	P1 (60)	P2 (63)	P0(66)	P2 (69)	P0 (71)	P3 (72)	P2 (75)

After the third wave, the used service time for each process updated to:

P0: 20, 0 remaining
P1: 15, 0 remaining

P2: 21 ,0 remaining
P3: 7, 0 remaining
P4: 9, 0 remaining

All the processes have reached their maximum time

b)

Waiting Time/ Algorithms	P0	P1	P2	P3	P4
FCFS	0	20	35	56	63
SJF	34	19	54	0	7
Priority Scheduling	27	0	47	68	15
Round-Robin	69	57	72	71	51

c)

Response Time/ Algorithms	P0	P1	P2	P3	P4
FCFS	0	20	35	56	63
SJF	34	19	54	0	7
Priority Scheduling	27	68	47	75	15
Round-Robin	0	3	6	9	12

d)

Turn-Around Time/ Algorithms	P0	P1	P2	P3	P4
FCFS	20	35	56	63	75
SJF	54	34	75	7	19

Priority Scheduling	47	15	68	75	27
Round-Robin	71	60	75	72	54