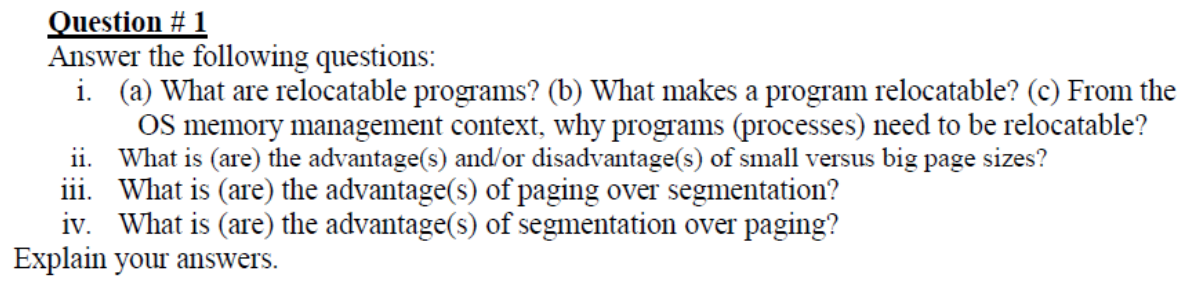
COMP 346 – Fall 2020
Assignment 3

Name: Hualin Bai

ID: 40053833



Question # 1

Answer the following questions:

i. (a) What are relocatable programs? (b) What makes a program relocatable? (c) From the OS memory management context, why programs (processes) need to be relocatable?

ii. What is (are) the advantage(s) and/or disadvantage(s) of small versus big page sizes?

iii. What is (are) the advantage(s) of paging over segmentation?

iv. What is (are) the advantage(s) of segmentation over paging?

Explain your answers.

Question 1:

I.

(a) Relocatable programs are programs that can be accessed dynamically (programs that can be accessed or loaded into memory in parts instead of being loaded the entire process all at once).

(b) A program is relocatable if it can be stored and accessed by jumping locations in memory via relocation registers, ranging from base to limit register.

(c) Programs need to be relocatable in order to save memory space. When programs are relocatable, they don't need to be entirely loaded into memory at once, only the parts of the process that need to be used will be loaded into memory.

ii.

Small page size:

Advantage: reduces internal fragmentation, better with locality of reference

Disadvantage: bigger page table, more page fault, overhead in reading/writing of pages

Big page size:

Advantage: smaller page table, less page faults, less overhead in reading/writing of pages

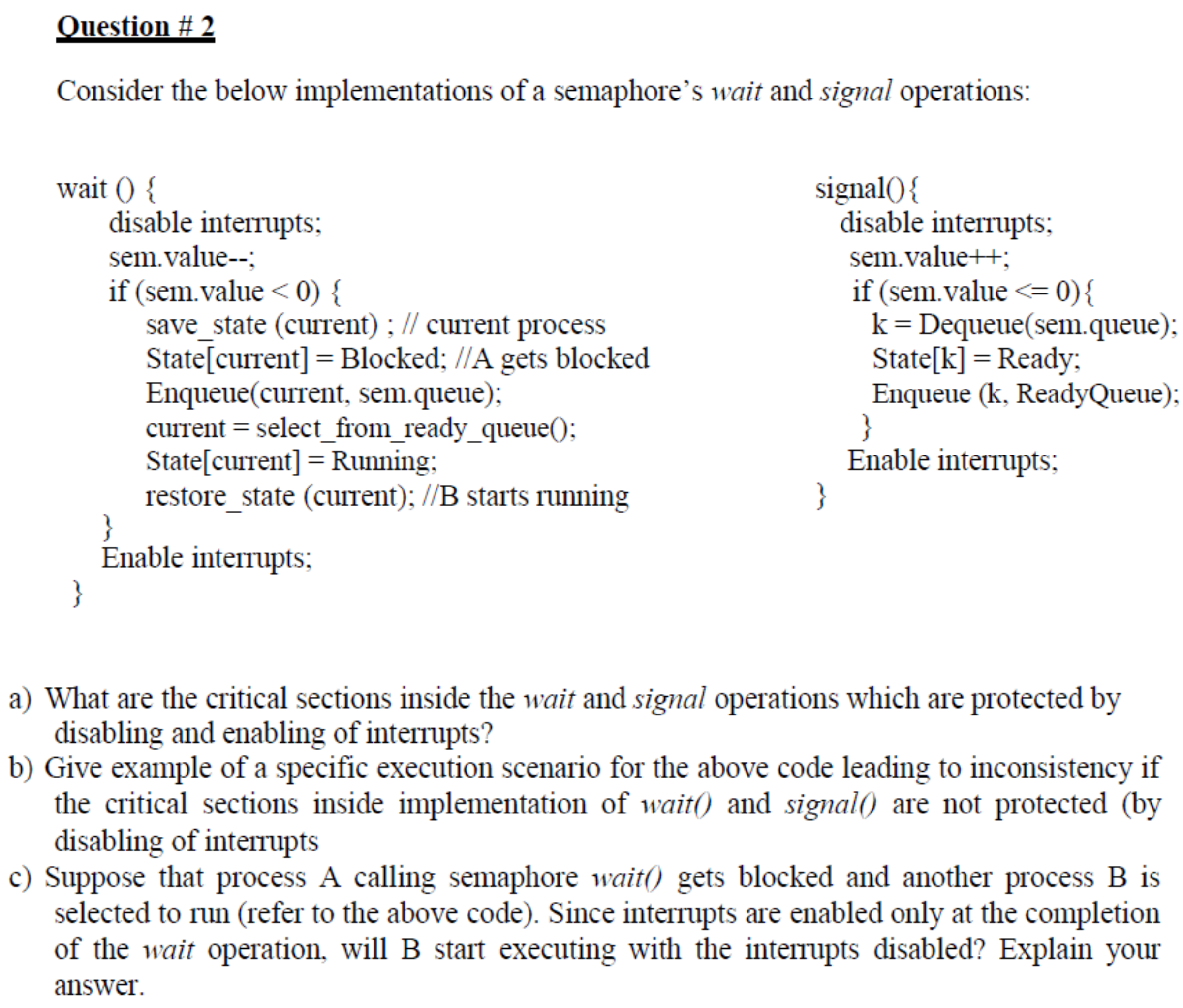
Disadvantage: more internal fragmentation, worse locality of reference.

iii.

Paging increases speed and saves on memory whereas segmentation does not necessarily save on memory or speed. Paging is not visible to a programmer. On the other hand, segmentation requires a programmer to be aware of the segments in his implementation.

iv.

Advantages of segmentation over paging are: there will be no internal fragmentation; the memory space occupied by segment table is less than page table in paging mechanism. Segmentation offers security of individual segments of a program.



Question # 2

Consider the below implementations of a semaphore's *wait* and *signal* operations:

```
wait () {
      disable interrupts;
      sem.value--;
      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
     }
     Enable interrupts;
  }
```

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

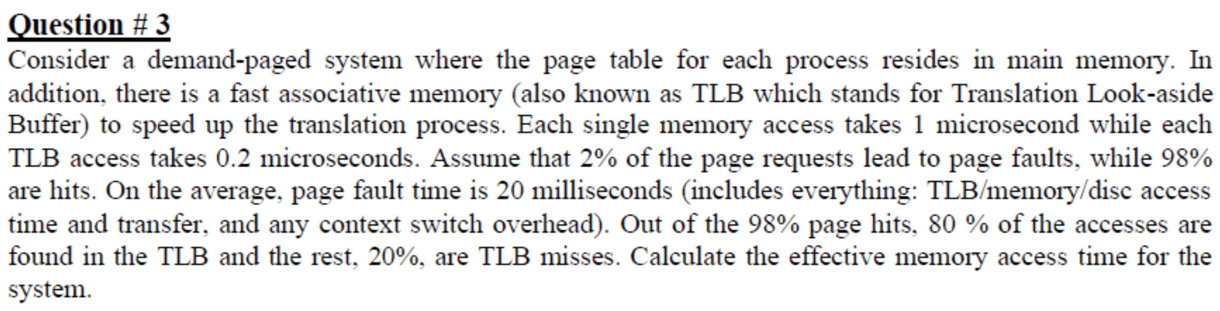
a) What are the critical sections inside the *wait* and *signal* operations which are protected by disabling and enabling of interrupts?
b) Give example of a specific execution scenario for the above code leading to inconsistency if the critical sections inside implementation of *wait()* and *signal()* are not protected (by disabling of interrupts
c) Suppose that process A calling semaphore *wait()* gets blocked and another process B is selected to run (refer to the above code). Since interrupts are enabled only at the completion of the *wait* operation, will B start executing with the interrupts disabled? Explain your answer.

Question 2:

(a) sem.value++, sem.value, sem.queue, and current process are all critical sections.

(b) If it is interrupted when the semaphore is saving the state of current process, any other program that uses the current process information will be in inconsistency.

(c) No, B will not able to execute, this will cause deadlock, since interrupts are disabled and it will be enable when A finish the executing, but B blocked A, B cannot execute until interrupts are enable interrupts.



Question # 3

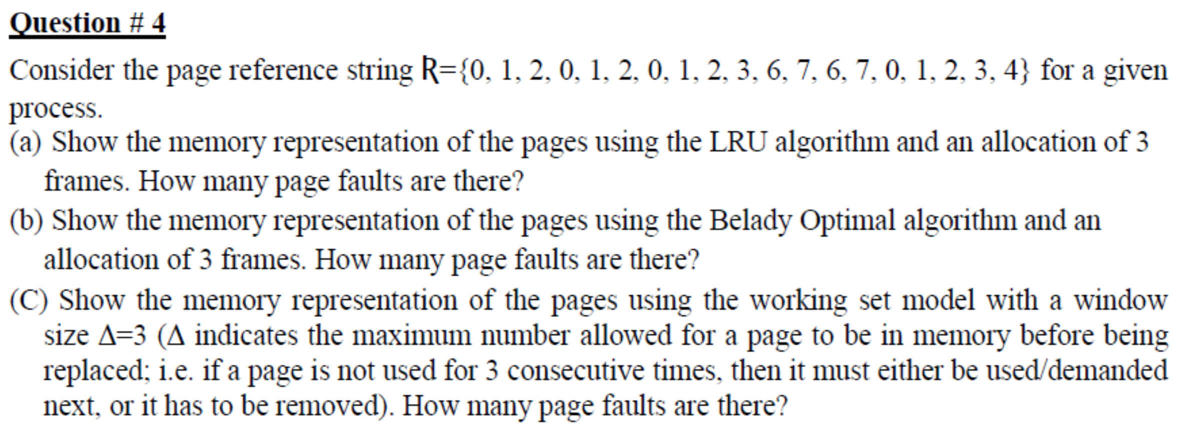
Consider a demand-paged system where the page table for each process resides in main memory. In addition, there is a fast associative memory (also known as TLB which stands for Translation Look-aside Buffer) to speed up the translation process. Each single memory access takes 1 microsecond while each TLB access takes 0.2 microseconds. Assume that 2% of the page requests lead to page faults, while 98% are hits. On the average, page fault time is 20 milliseconds (includes everything: TLB/memory/disc access time and transfer, and any context switch overhead). Out of the 98% page hits, 80 % of the accesses are found in the TLB and the rest, 20%, are TLB misses. Calculate the effective memory access time for the system.

Question 3:

Effective Memory Access Time=TLB hit ratio*(TLB access time+ Main memory access time)+(1-TLB hit ratio)*(TLB access time+ 2* main memory access time)

TLB hit ratio=(TLB hits)/(number of queries into TLB)=80/98

Effective Memory Access Time=(80/98)*(0.2+1)+(1-80/90)*(0.2+2*1)=1.384 microseconds



Question # 4

Consider the page reference string R={0, 1, 2, 0, 1, 2, 0, 1, 2, 3, 6, 7, 6, 7, 0, 1, 2, 3, 4} for a given process.

(a) Show the memory representation of the pages using the LRU algorithm and an allocation of 3 frames. How many page faults are there?

(b) Show the memory representation of the pages using the Belady Optimal algorithm and an allocation of 3 frames. How many page faults are there?

(C) Show the memory representation of the pages using the working set model with a window size Δ=3 (Δ indicates the maximum number allowed for a page to be in memory before being replaced; i.e. if a page is not used for 3 consecutive times, then it must either be used/demanded next, or it has to be removed). How many page faults are there?

Question 4:

(a) Using LRU algorithm:

0	1	2	0	1	2	0	1	2
0	0	0						
	1	1						
		2						
F	F	F						

3	6	7	6	7	0	1	2	3	4
3	3	3			0	0	0	3	3
1	6	6			6	1	1	1	4
2	2	7			7	7	2	2	2
F	F	F			F	F	F	F	F

Total page fault is 11

(b) Using Belady Optimal algorithm:

0	1	2	0	1	2	0	1	2
0	0	0						
	1	1						
		2						
F	F	F						

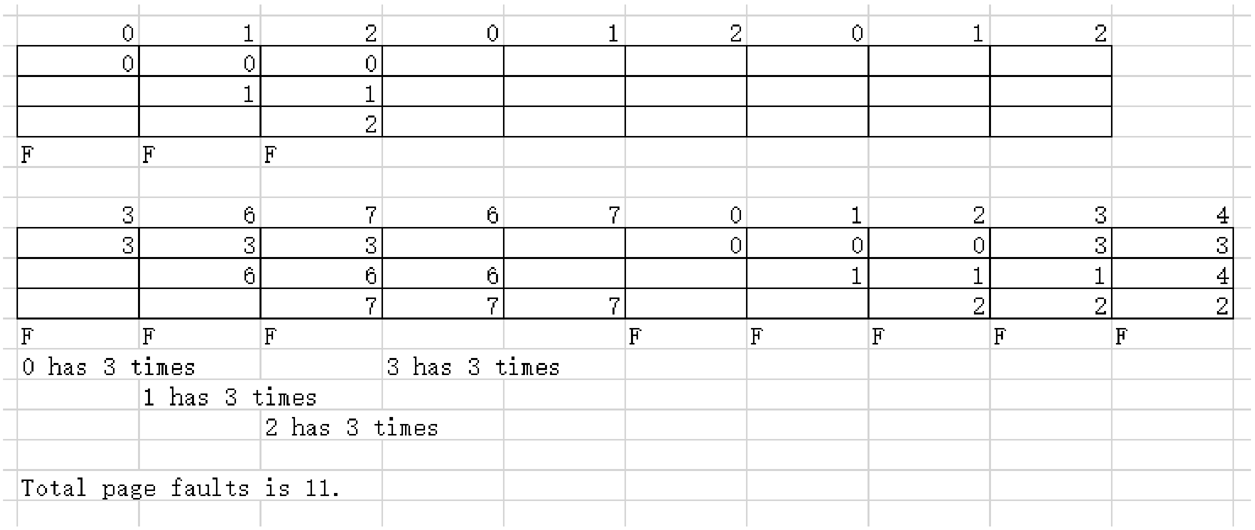
3	6	7	6	7	0	1	2	3	4
0	0	0				1	1	1	4
1	1	7				7	2	2	2
3	6	6				6	6	3	3
F	F	F				F	F	F	F

Total page fault is 10

(C)

Working-set model is based on the assumption of locality. This model uses a parameter, Δ, to define the working-set window. The idea is to examine the most recent Δ page references. The set of pages in the most recent Δ page reference is the working set. If a page is in active use, it will be in the working set. If it is no longer being used, it will drop from the working set Δ time units after its last reference. Thus, the working set is an approximation of the program's locality. The accuracy of the working set depends on the selection of Δ. If Δ is too small, it will not encompass the entire locality; if Δ is too large, it may overlap several localities. In the extreme, if Δ is infinite, the working set is the set of pages touched during the process execution.

Total page fault number is 11.



0	1	2	0	1	2	0	1	2	
0	0	0							
	1	1							
		2							
F	F	F							
3	6	7	6	7	0	1	2	3	4
3	3	3			0	0	0	3	3
	6	6	6			1	1	1	4
		7	7	7			2	2	2
F	F	F			F	F	F	F	F
0 has 3 times			3 has 3 times						
	1 has 3 times								
		2 has 3 times							
Total page faults is 11.									

Question # 5

Consider a system that would implement the page table on the CPU if feasible.
(a) Give an advantage of this strategy.
(b) Give a disadvantage of this strategy.

Question 5:

(a) Advantage:

The page table helps in reducing the external fragmentation and also it is simple to implement and assume the efficiency memory management technique. The swapping becomes easy when the size of the page and frame are equal.

(b) Disadvantage:

It leads to reducing the memory space needed for storing the page tables.

It also takes a long lookup time and is difficult for sharing memory implementation.

Question # 6

Explain (i) an advantage and (ii) a disadvantage that a global page replacement algorithm has over a local page replacement algorithm.

Question 6:

Advantage:

Global replacement allows a process to select a replacement frame from the set of all frames, even if that frame is currently allocated to some other process; that is, one process can take a frame from another. Local replacement requires that each process select from only its own set of allocated frames.

Disadvantage:

Global replacement algorithm is that a process cannot control its own page-fault rate. The set of pages in memory for a process depends not only on the paging behavior of that process but also on the paging behavior of other processes. Therefore, the same process may perform quite differently because of totally external circumstances.

Question # 7

Consider a system that adjusts the degree of multiprogramming by monitoring the mean time between page faults (i.e. T_{pf}) and the mean time to service a page fault (i.e. T_{fs}). Describe the

performance of the paging system in terms of the degree of multiprogramming when (i) T_{pf} is greater than T_{fs}, (ii) T_{pf} is less than T_{fs} and (iii) T_{pf} is equal to T_{fs}.

Question 7:

Degree of multiprogramming is the number of processes in memory, if the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

(1) If $T_{pf} > T_{fs}$, means that mean time between page faults is greater than the mean time to service a page fault, which means page fault is served another page fault will occur, it will increase the degree of multiprogramming.

(2) If $T_{pf} < T_{fs}$, means that mean time page fault is less than the mean time to service a page fault, it will decrease the degree of multiprogramming.

(3) If $T_{pf} = T_{fs}$, means both of them are equal, the degree of multiprogramming will be stable. It will perform slightly better than if it were less but page faults will still occur fairly often.

Question # 8

Some systems automatically open a file when it is referenced for the first time and close the file when the job terminates. Discuss the advantages and disadvantages of this scheme as compared to the more traditional one, where the user has to open and close the file explicitly.

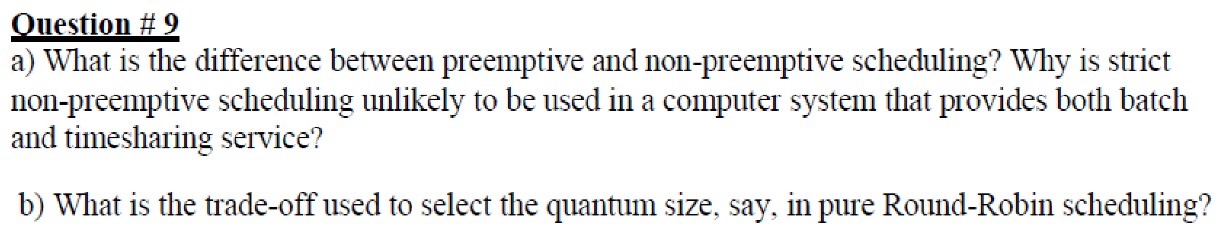
Question 8:

Advantage:

Automatically open/close a file means that I/O operations will be minimized.

Disadvantage:

It requires additional expenses and more overhead compared to explicit opening and closing is required. And deadlock or starvation may happen.



Question # 9

a) What is the difference between preemptive and non-preemptive scheduling? Why is strict non-preemptive scheduling unlikely to be used in a computer system that provides both batch and timesharing service?

b) What is the trade-off used to select the quantum size, say, in pure Round-Robin scheduling?

Question 9:

(a)

Preemptive scheduling allows a process to yield the CPU to another process.

Non-preemptive does not allow this, a process will remain in the CPU until it is finished.

Strict non-preemptive is unlikely in a batch and timesharing system, because non-preemptive scheduling does not allow for priority and context switching before a process terminates. In an environment that has batch and timesharing where process need to be swapped out of the CPU often this type of scheduling would be very slow.

(b)

A quantum size that is too small in pure Round-Robin Scheduling will be very slow because it will context-switch very often. A large quantum size will result in FCFS scheduling, thus increased response time, and lowering the degree of multiprogramming.

The performance of the Round-Robin scheduling algorithm depends heavily on the size of the time quantum. If the time quantum is extremely large, the RR policy is the same as the FCFS policy. In contrast, if the time quantum is extremely small, the RR approach can result in a large number of context switches. Thus, we want the time quantum to be large with respect to the context switch time. A rule of thumb is that 80 percent of the CPU bursts should be shorter than the time quantum.

Question # 10

What advantage is there in having different values of the scheduling quantum on different levels of a multilevel feedback queuing system? Your answer should consider all aspects such as fairness, starvation, efficiency, etc.

Question 10:

Different quantum lengths on different queue levels attempt to shorten execution of short jobs, make it more fair for longer jobs and reducing frequency of context switching at the same time.

Processes with no need for frequent servicing can be in a queue with a larger quantum, requiring fewer context switches to complete the processing, making more efficient use of the computer. However, for some interactive processes, processes need more frequent servicing which can be in a queue with a small-time quantum. It can reduce the starvation.

Question # 11

Consider the following set of prioritized processes, where a smaller priority value represents a higher priority.

Process	Service Time	Priority
0	20	3
1	15	1
2	21	3
3	7	5
4	12	2

Assume that all processes arrived at the same time, however they are inserted in the ready list in the order indicated in the above table.

a) Draw Gantt charts for the execution scenarios assuming:
- FCFS scheduling
- Non-preemptive SJF scheduling
- Non-preemptive priority scheduling
- Pure Round-Robin scheduling with the quantum = 3

b) What is the waiting time of each process in each case?
c) What is the response time of each process in each case?
d) What is the turn-around time of each process in each case?

Question 11:

Waiting time=Turnaround time – Burst time

Response time= Time at which the process gets the CPU for the first-Arrival time.

Turnaround time=Burst time + waiting time

Turnaround time=Exit time-arrival time

1. FCFS Scheduling

	P0	P1	P2	P3	P4
0	20	35	56	63	75

Process	Burst Time	Priority	Finish Time	Response Time	Waiting Time	Turnaround Time
P0	20	3	20	0	0	20
P1	15	1	35	20	20	35
P2	21	3	56	35	35	56
P3	7	5	63	56	56	63
P4	12	2	75	63	63	75

2. Non-preemptive SJF scheduling

	P3	P4	P1	P0	P2
0	7	19	34	54	75

Process	Burst Time	Priority	Finish Time	Response Time	Waiting Time	Turnaround Time
P0	20	3	54	34	34	54
P1	15	1	34	19	19	34
P2	21	3	75	54	54	75
P3	7	5	7	0	0	7
P4	12	2	19	7	7	19

3. Non-preemptive priority scheduling

	P1	P4	P0	P2	P3
0	15	27	47	68	75

Process	Burst Time	Priority	Finish Time	Response Time	Waiting Time	Turnaround Time
P0	20	3	47	27	27	47
P1	15	1	15	0	0	15
P2	21	3	68	47	47	68
P3	7	5	75	68	68	75
P4	12	2	27	15	15	27

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

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26				
	P0	P1	P2	P3	P4	P0	P1	P2	P3	P4	P0	P1	P2	P3	P4	P0	P1	P2	P3	P4	P0	P1	P2	P3	P4	P0	P1	P2	P0	P2
0	3	6	9	12	15	18	21	24	27	30	33	36	39	40	43	46	49	52		55	58	61	64			67		70	72	75

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

Process	Burst Time	Priority	Finish Time	Response Time	Waiting Time	Turnaround Time
P0	20	3	72	0	52	72
P1	15	1	61	3	46	61
P2	21	3	75	6	54	75
P3	7	5	40	9	33	40
P4	12	2	55	12	43	55