ICS 143A: Principles of Operating Systems
Homework #2 (150 pts)

Last/First name:
Student ID #:

Question 1: Scheduling

[10pts] a) Which of the following scheduling algorithms could result in starvation? Assume that a process burst time can be infinite. Briefly explain your answer for each of the algorithm.

1. First-Come First-Served (FCFS)
2. Shortest Job First (SJF) (Non-preemptive)
3. Shortest Remaining Time First (SRTF) (Preemptive)
4. Round Robin (RR)
5. Priority (Preemptive)

All except RR could lead to starvation.
(1) FCFS is non preemptive, so it can lead to starvation if one of the processes runs forever
(2) SJF can lead to starvation if there are many small jobs continuously arriving. Longer jobs may never execute. Same explanation of FCFS could be used here since it's non-preemptive
(3) SRTF can lead to starvation if there are many small jobs continuously arriving. Longer jobs may never execute.
(5) Priority can lead to starvation if there are many high priority jobs. Lower priority jobs may never execute.

Every algorithm mentioned must come with an explanation. 1pt for the correct answer and 1 pt for the correct explanation.

[75pts] b) Consider the set of processes:

Process ID	Arrival Time	Burst Time	Priority
P1	0	60	3
P2	3	30	2
P3	4	40	1
P4	9	10	4

Draw the GANTT chart for the following scheduling algorithms.

- First-Come First-Served (FCFS)
- Shortest Job First (SJF) (Non-preemptive)
- Shortest Remaining Time First (SRTF)
- Round Robin (RR) (Time Quantum = 10)
- Priority (Preemptive)

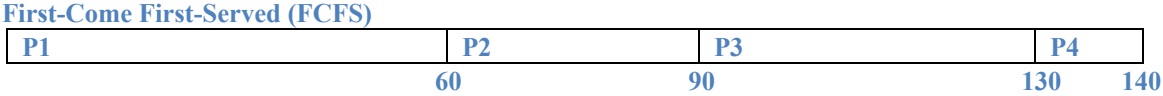
And **complete the following table** with the average waiting time and turnaround time of each algorithm:

Scheduling Algorithm	Average waiting time	Average turnaround time
First Come First Served (FCFS)	66	101
Shortest Job First (SJF) (Non-preemptive)	53.5	88.5
Shortest Remaining Time First (Preemptive)	32.2	67.2
Round Robin (RR) (Time Quantum = 10)	58.5	93.5
Priority (Preemptive)	57.7	92.7

When **Priority** is being used, a smaller priority value means higher execution priority . For tie-breaking cases, the process with the earlier arrival time should execute.

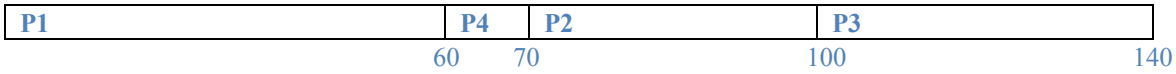
-10pts for each gantt chart. 5pts if the gantt has minor mistakes
-5pts for each correct row in the table (2.5 pts for each answer). If the value in the table is wrong but calculations seem to be correct (and the gantt charts are correct), 2pts for each answer.
-Showing the calculation of average turnaround time and waiting time is optional, but no partial credit is granted if they are missing

First-Come First-Served (FCFS)



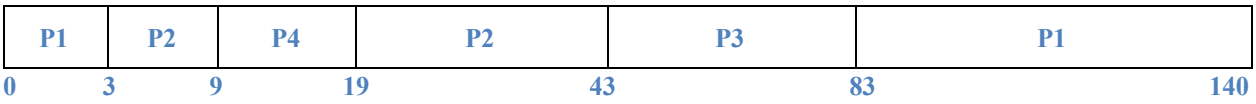
Average waiting time: (0 + (60 - 3) + (90 - 4) + (130 - 9)) /4 = (0 + 57 + 86 + 121) / 4 = 66
Average turnaround time: ((60 – 0) + (90 - 3) + (130 - 4) + (140 - 9)) / 4 = (60 + 87 + 126 + 131) / 4 = 101

Shortest Job First (SJF) (Non-preemptive)



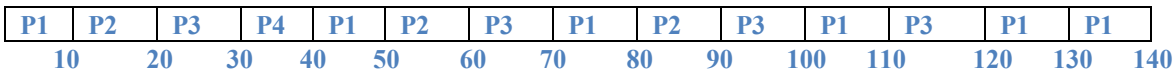
Average waiting time: (0 + (60 – 9) + (70 - 3) + (100 - 4)) / 4 = (0 + 51 + 67 + 96) / 4 = 53.5
Average turnaround time: ((60 - 0) + (70 - 9) + (100 - 3) + (140 - 4)) / 4 = (60 + 61 + 97 + 136) / 4 = 88.5

Shortest Remaining Time First (SRTF) (Preemptive)



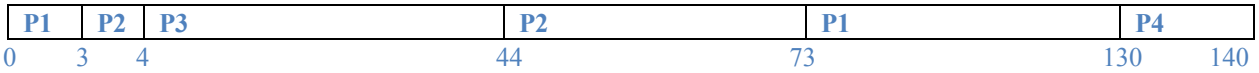
Average waiting time: (0+(83-3)) + (0+(19-9)) + (43-4) + 0 = (80+10+39+0) / 4 = 32.24
Average turnaround time: (140-0) + (43-3) + (83-4) + (19-9) = 67.25

Round Robin (RR) (Time Quantum = 10)



P1 waiting time: 0 + (40 - 10) + (70 - 50) + (100 - 80) + (120 - 110) = 30 + 20 + 20 + 10 = 80
P2 waiting time: (10 - 3) + (50 - 20) + (80 - 60) = 7 + 30 + 20 = 57
P3 waiting time: (20 - 4) + (60 - 30) + (90 - 70) + (110 - 100) = 16 + 30 + 20 + 10 = 76
P4 waiting time: (30 - 9) = 21
Average waiting time: (80 + 57 +76 + 21) / 4 = 58.5
Average turnaround time: ((140 - 0) + (90 - 3) + (120 - 4) + (40 - 9)) / 4 = (140 + 87 + 116 + 31) / 4 = 93.5

Priority (Preemptive)



Average waiting time: ((0 + (70-3))+
(0 + (44-4))+
0+
(130-9)) / 4 = (70 + 40 + 0 + 121) / 4 = 57.75
Average turnaround time: ((130 – 0) + (73 - 3) + (44 - 4) + (140 - 9)) / 4 = 92.75

Question 2: Critical Sections [40pts]
Consider two processes **P0** and **P1** executing concurrently on a single cpu. P0 and P1 are executing the following algorithm that provides 2-process solution to the critical section problem:

flag[0] = false;
flag[1] = false;

P0:	**P1:**
0: **while** (true) {	0: **while** (true) {
1: flag[0] = true;	1: flag[1] = true;
2: **while** (flag[1]) **{**	2: **while** (flag[0]) **{**
3: flag[0] = false;	3: flag[1] = false;
4: **while** (flag[1]) **{**	4: **while** (flag[0]) **{**
5: no-op;	5: no-op;
6: }	6: }
7: flag[0] = true;	7: flag[1] = true;
8: }	8: }
9: *critical section*	9: *critical section*
10: flag[0] = false;	10: flag[1] = false;
11: remainder section	11: remainder section
12: }	12: }

Specify which of the following requirements are satisfied or not by this algorithm. If the requirement **is satisfied,** explain why. If the requirement **is not** satisfied, show an execution sequence of P1 and P2 which illustrates the requirements' violation.

1. Mutual Exclusion
2. Progress
3. Bounded Wait

Assume P0 and P1 may context switch at any point in time.

Mutual exclusion is satisfied with this algorithm. Whereas, progress and bounded waiting are not satisfied by this algorithm.

Mutual exclusion:
-P0 or P1 only enters the critical section if the check in line 2 fails
-Check can only fail if the other process is executing the remainder section or the inner-loop. Notice the process always sets its own flag to true before executing line 2

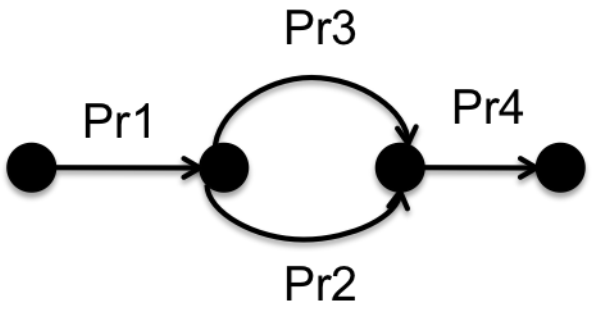
Progress:
Both processes may get stuck checking the flags if their execution is interleaved

P0	**P1**
sets flag[0] to true at line 1	
	sets flag[1] to true at line 1
Pass line 2 check	
	Pass line 2 check
sets flag[0] to false at line 3	
	sets flag[1] to false at line 3
Pass line 4 check	
	Pass line 4 check
sets flag[0] to true at line 7	
	sets flag[1] to true at line 7
Pass line 2 check again ...	
	Pass line 2 check again ...

Mutual exclusion satisfied + explanation**: 5pts if mentioned its satisfied + 11pts for explanation (no partials)**
Progress not satisfied + example**: 5pts if mentioned it's not satisfied + 13 pts for example (no partials)**
Bounded waiting: they should mention that there's no reason to talk about bounded waiting if progress is not satisfied. **6 pts if they mention this. Any other attempts will get 0.**

Question 3: Semaphores

In an operating system processes can run concurrently. Sometimes we need to impose a specific order in execution of a set of processes. We represent the execution order for a set of processes using a process execution diagram. Consider the following process execution diagram. The diagram indicates that **Pr1** must terminate before **Pr2**, **Pr3** and **Pr4** start execution. It also indicates that **Pr4** should start after **Pr2** and **Pr3** terminate and **Pr2** and **Pr3** can run concurrently.



The process execution diagram can also be represented using **Serial** and **Parallel** notation.
The above execution diagram is represented as **Serial(P1, Parallel(P2, P3) , P4)**.

We can use semaphores in order to enforce the execution order. Semaphores have two operations as explained below.

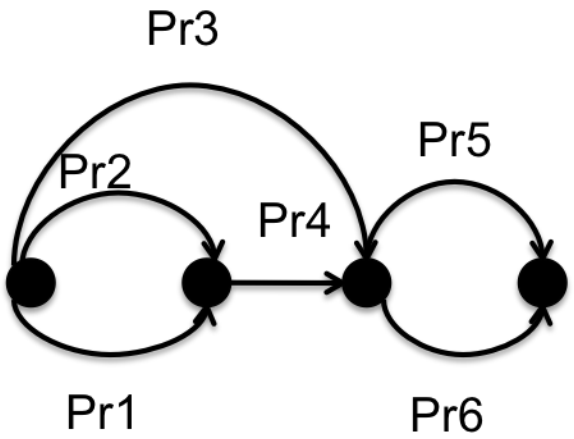
- **P** (or wait) is used to acquire a resource. It waits for semaphore to become positive, then decrements it by 1.
- **V** (or signal) is used to release a resource It increments the semaphore by 1, waking up a blocked processes, if any.

We can use the following semaphores to enforce the execution order above:

s1=0; s2=0; s3=0;
Pr1: body; V(s1); V(s1);
Pr2: P(s1); body; V(s2);
Pr3: P(s1); body; V(s3);
Pr4: P(s2); P(s3); body;

This assume that the semaphores **s1**, **s2**, and **s3** are created with an initial value of **0** before processes **Pr1**, **Pr2**, **Pr3**, and **Pr4** execute.

Based on these definitions explanation, answer the questions about the following process execution graph.



[5pts] a) Represent the process execution diagram using the **Serial** and **Parallel** notation.
Serial(Parallel(Serial(Parallel(Pr1, Pr2), Pr4), Pr3),Parallel(Pr5, Pr6))

5pts if correct. No partial credit

[20pts] b) Use the semaphores **s1,s2,s3,s4** to enforce the execution order according to the process execution diagram. Assume the following initialization values for **s1,s2,s3,s4:**

s1=0; s2=0; s3=0; s4=0;

Pr1: body; V(s1);	2pt
Pr2: body; V(s2);	2pt
Pr3: body; V(s3)V(s3);	4pt
Pr4: P(s1); P(s2); body; V(s4) V(s4);	4pt
Pr5: P(s3); P(s4); body;	4pt
Pr6: P(s3); P(s4); body;	4pt

2pts for the first two processes, 4pts for each of the last four processes (20pts overall). Note the semaphores may be used in a different order