## Homework Assignment #5

## **Question 1: Scheduling**

a) Which of the following scheduling algorithms could result in starvation? For those algorithms that could result in starvation, describe a situation in which starvation is likely to occur?

- 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

b) Consider the set of process (smaller priority number implies higher priority; i.e. 1 – highest priority):

Process ID	Arrival Time	Burst Time	Priority
P1	0	70	3
P2	10	50	2
P3	30	20	1
P4	50	20	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) (Preemptive)
- Round Robin (RR) (Time Quantum = 10, Round from P1 to PN if PN arrives)
- Priority (Non-preemptive)

c) For the processes listed above, complete the following table:

Scheduling Algorithm	Average waiting	Average
	time	turnaround time
First Come First Served (FCFS)		
Shortest Job First (SJF) (Non-		
preemptive)		
Shortest Job First (SJF)		
(preemptive)		
Round Robin (RR) (Time Quantum		
= 10)		
Priority (Non-preemptive)		

d). Briefly reason why the average waiting time of preemptive SJF is guaranteed to be no larger than that of non-preemptive. (No need to prove)

e). Consider a set of 10 processes with identical burst times and similar arrival times scheduled using a round robin algorithm. Let the overhead of context switch be 2 milliseconds. To reduce the relative context switch overhead, we wish to bound the total time for context switch to be one-fifth of the total waiting time. Develop an equation to calculate the minimum value of quantum q for this condition to be satisfied (Hints: Think carefully about the components of waiting time).

## **Question 2: Test-and-set, Critical Section**

Consider the following code segment covered in the lecture:

var j: 0...n-1 key:Boolean, lock:= false; 0: repeat { waiting[i] = true; key:=true; 1: while (waiting[i] and key) do key:=Test-and-2: Set(lock); waiting[i]:=false; 3: 4: critical section 5: j:=j + 1 mod n while ((j <> i) and (not waiting[j])) do j := j + 16: mod n; 7: if j = i then lock:=false; 8: else waiting[j] = false; 9: remain section 10: } until false

(a).The above program that we saw in the lecture can achieve both bounded waiting and mutual exclusion with test-and-set. Assume line 3 of this code above is deleted. Determine if the algorithm still works as desired after that line is deleted.

Consider the following algorithm that provides a solution to the 2 process critical section problem.

flag[0] = false;			
flag[1] = false;			
P0:	P1:		
0: <b>while</b> (true) {	0: <b>while</b> (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: }		

b). Specify which of the following requirements are satisfied or not by this algorithm. Explain why or why not.

- 1. Mutual Exclusion
- 2. Progress
- 3. Bounded Waiting

## **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.



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 the blocked processes, if any.

We can use the following semaphores to enforce the execution order.

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;

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 this explanation, answer the questions about the following process execution graph.



a) Use semaphores to enforce execution order according to the process execution diagram.