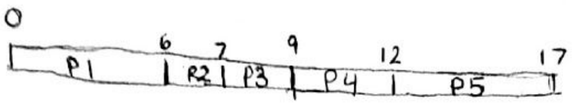
<u>Assignment 2</u>

1. Implement the counting semaphores using binary semaphores. Pseudo-code for the implementation is given in the attached notes on BB: 10xcountingSemUsingBinarySem.pdf. Use the bounded-buffer producer-consumer problem as uploaded on BB (first code at https://jlmedina123.wordpress.com/2014/04/08/255/). Show the output of this code for BOTH the incorrect and correct implementations. Also, attach your codes (two versions: for incorrect and correct solutions) on BB.
    a. **Separate File**
2. Prove/disprove that SJF is optimal in terms of average turnaround times of the processes.
    a. **SJF is optimal in terms of average turnaround times of the processes. Given a set of processes, moving short jobs before a longer one decreases the waiting time of the shorter job more than it increases the waiting time of the longer job. This causes the average turnaround times of the processes to be lower.**
3. For what types of workloads does SJF deliver the same turnaround times as FIFO?
    a. **SJF has the same turnaround times as FIFO when all the jobs are the same size or linearly increase in size as they come in.**
4. For what types of workloads and quantum lengths does SJF deliver the same response times as RR?
    a. **SJF has the same response times as RR when the scheduling quantum of RR is bigger than the biggest job and the jobs arrive in linearly increasing size.**
5. What happens to response time with SJF as job lengths increase?
    a. **The average response time increases.**
6. What happens to response time with RR as quantum lengths increase? Explain in words and write an equation that gives the worst-case response time, given N jobs.
    a. **Response time increases because the time a process has to wait for its next available time slot is proportional to the number of different processes and the maximum time they can execute. On a system with *n* processes and a quantum size of *Q*, a process will have to wait *Q(n-1)* in the worst case.**
7. "Preemptive schedulers are always less efficient than non-preemptive schedulers." Explain how this statement can be true if you define efficiency one way and how it is false if you define efficiency another way.
    a. **If you define efficiency in terms of response time, how long it takes the system to respond to a given input, then preemptive schedulers are always more efficient because they can quickly switch to the thread/process that needs to process the input. If you define efficiency as a ratio between the amount of CPU time spent running userspace programs vs OS services then preemptive schedulers are always less efficient because they switch between processes more often. This causes the scheduler to be used more often and more context switches. This time is purely overhead.**
8. What is the priority inversion problem? Does this problem occur if round-robin scheduling is used instead of priority scheduling? Discuss
    a. **The priority inversion problem is where a higher priority process is preempted by a lower priority process which effectively inverts the relative priorities of the processes. No, the problem will not occur with round robin scheduling because processes can only use a certain amount of time before it is forced to wait for the next process to go.**
9. Does Peterson's solution to the mutual exclusion problem work when process scheduling is preemptive? How about when it is non-preemptive?
    a. **It works when the scheduling is preemptive, which is the case it was designed for. It might fail if the scheduling is non-preemptive. One case is where 'turn' is 0 but the process 1 runs first. This causes it to loop forever and never release the CPU to other processes.**
10. Consider the following set of processes, with the length of the CPU burst time given in milliseconds:
    a. Process | Burst-Time | Priority

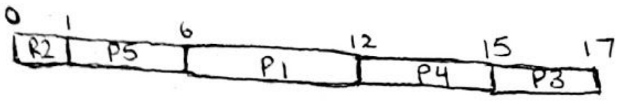
b. P1 | 6 | 3
c. P2 | 1 | 1
d. P3 | 2 | 5
e. P4 | 3 | 4
f. P5 | 5 | 2
g. The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.
    i. Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.
        1. **On image**
    ii. What is the turnaround time of each process for each of the scheduling algorithms in part (a)?
        1. **On image**
    iii. What is the waiting time of each process for each of the scheduling algorithms in part (a)?
        1. **On image**

10.

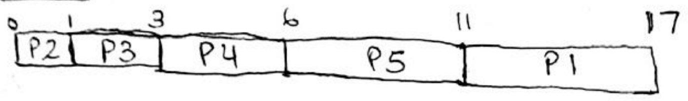
a) FCFS:



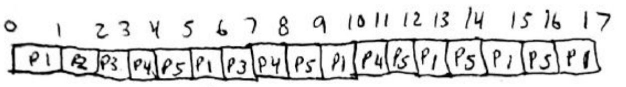
Priority:



SJF:



RR:



b) Turnaround Time:

	FCFS	Priority	SJF	RR
P1	6	12	17	17
P2	7	1	1	2
P3	9	17	3	7
P4	12	15	6	11
P5	17	6	11	16
Avg	10.2	10.2	7.6	10.6

c. Waiting Time

	FCFS	Priority	SJF	RR
P1	0	6	11	11
P2	6	0	0	1
P3	7	15	1	5
P4	9	12	3	8
P5	12	1	6	11
Avg.	6.8	4.2	6.8	7.2

iv. Which of the schedules in part a results in the minimal average waiting time (over all processes)?
    1. **SJF has the minimal average waiting time.**

11. The aging algorithm with a = 1/2 is being used to predict run times. The previous four runs, from oldest to most recent, are 40, 20, 40, and 15 msec. What is the prediction of the next time?
    a. **Using all previous four runs, the prediction is: (((40 + 20) / 2 + 40) / 2 + 15) / 2 = *25 ms*.**
12. Explain what a multi-level feedback scheduler is and why it approximates SRTF. True or False (also give an explanation for your choice): If a user knows the length of a CPU time-slice and can determine precisely how long his process has been running, then he can cheat a multi-level feedback scheduler.
    a. **A multi-level feedback scheduler stores processes into different "levels." Processes that exceed the allotted time slice are automatically moved down into a lower "level" while processes for I/O requests or block will be moved to a higher "level." It approximates SRTF because processes that tend to depend a lot on I/O reside typically in higher levels and have priority over processes that are just computational.**
    b. **True. Right before the user's time slice expires, they can execute a dummy I/O request that makes the process appear to be "I/O Bound." This allows the user to keep the task in the upper tier of the multi-level feedback scheduler which means it gets more CPU time.**
13. Consider a system consisting of processes P1, P2, ..., Pn, each of which has a unique priority number. Write the pseudocode of a monitor that allocates three identical line printers to these processes, using the priority numbers for deciding the order of allocation. Start with the following and populate the two functions request_printer() and release_printer():

```
monitor printers {
int num_avail = 3;
int waiting_processes[MAX PROCS];
int num_waiting;
condition c;

void request_printer(int proc_number) {
 if(num_avail > 0) {
     num_avail--;
     return;
 }

 waiting_processes[num_waiting++] = proc_number;
 sort(wait_processes);
 while(num_avail == 0 && waiting_processes[0] != proc_number) {
     c.wait();
     waiting_processes[0] = waiting_processes[--num_waiting];
     sort(wait_processes);
     num_avail--;
 }
}
void release_printer() {
 num_avail++;
 c.broadcast();
 }
}
```