Scheduling Problems

1. Assume you have the following jobs to execute with one processor

<table>
<thead>
<tr>
<th>Job</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

The jobs are assumed to have arrived in the order 1,2,3,4,5

a) give a Gantt chart illustrating the execution of these jobs using FCFS, RR (quantum = 1), SJF, and a non preemptive priority scheduling algorithm

b) What is the turnaround time of each job for each of the above scheduling algorithms?

c) What is the waiting time of each job for each of the above scheduling algorithms?

d) What is the schedule with the minimal average waiting time (over all jobs)?

2. One gate at the northwest parking ramp can handle on the average six cars per minute. There are 2 gates available. The division of traffic and parking wants to guarantee that the expected length of the line of cars waiting to enter is less than or equal to three.

- What is the highest allowable average arrival rate, \( \alpha \), to use only one gate?
- What is the highest allowable average arrival rate for both gates to be operational and have the expected average queue length less than or equal to three?

3. A system processes jobs which require a service time close to a given figure. Once in a while, however, a much smaller type of job needs to service, but this happens very rarely. It is therefore decided not to apply RR policy because it would cause unnecessary job switching overhead.

Instead, the following scheme is proposed: any new “standard” job is appended at the end of the queue as usual; if such an exceptional job arrives (can be recognized at the point of arrival), it is inserted into the queue at the rear of all small jobs already present, but preceding all standard jobs in the queue. The server always takes the first job from the queue.

Design the job queue as a linked list and write the programs for inserting a job. Would an array implementation be suitable?

4. Assuming that some other parts of the OS create processes allocate memory and so on, write the code for the scheduler to
- decide which job to run next
- take care of the movement of the processes between queues and lists

Assume that there is a hardware variable CLOCK that keeps the current date and time of day measured in microseconds since midnight (hardware updates CLOCK). For the queues you can use any data structure you with (array, linked list, combination of these, ...). We assume that when a process is created it gets a number to be used as its identification (0,1, ...). Via this number we have the access to the information about the process (like whether it needs to be placed in the high priority queue, runtime gathered so far, and so on). We assume further that there are n possible resources guarded by semaphores sem[i] (i = 1, ... , n) and that a process asks for a resource by executing P(sem[i]) and releases the resource by executing V(sem[i]). In case the process dies abruptly, V is executed in behalf of it by the logout process. The resources maybe different, but from the viewpoint of the scheduler, the only difference is whether this semaphore refers to input/output operations or something else. For simplicity we assume that the processes do their input/output operations by referring to the corresponding semaphore (P followed by V). to simplify the wait credit calculation replace “1-minute load average” by “number of processes” in the queues 0 to 6 at that point of time”.

So: write the following processes:

1) inserting a new process in the appropriate queue (0,1, or 6); this task is called by the JOB MANAGER after creating a new process,
2) deciding which process to run next
3) movement of the processes up and down the queues
4) P process for taking care of the P operation
5) V process for taking care of the V operation

The tasks 2 and 3 are called by P and V. they can also be called after the hardware interrupts (like “device done”, CLOCK has reached another so many milliseconds, and the like). Actually the caller is the interrupt service handler, which

1) causes the state of the process involved to change to ready
2) sets the scheduler variable change to contain the number of the process thus affected (otherwise this variable should be zero, so the scheduler can tell one call from another)

In writing the code you can freely combine the aforementioned tasks into any number of processes that call each other. Note, however, that the scheduler rarely returns to the calling program (in case of P/V), but more commonly selects another process to be run and jumps to wherever that process was stopped and hardware continues execution from that point on.