COMP 2432 Operating Systems

Tutorial 7

1. CPU Scheduling.

Draw Gantt charts for the following set of processes using different scheduling algorithms: (*a*) **FCFS**, (*b*) **SJF**, (*c*) **SRT**, (*d*) **Priority** (Unix/Linux convention), (*e*) **Priority** (Windows convention), (*f*) **Priority with preemption** (Unix/Linux convention), (*g*) **Priority with preemption** (Windows convention), (*h*) **RR** with quantum 5, (*i*) **RR** with quantum 4, (*j*) **RR** with quantum 3, and (*k*) **RR** with quantum 2.

Let us assume the ***tie-breaking rule*** when multiple processes are eligible: the process with ***smaller ID*** will go first. Compute the waiting time and turnaround time for each process. What are the average values for waiting time and turnaround time? How many context switching *decisions* are made in each case?

Process	*Burst Time*	*Arrival Time*	*Priority*
P_1	9	0	4
P_2	5	1	7
P_3	2	2	2
P_4	3	3	7
P_5	6	4	4

2. More CPU Scheduling.

Draw Gantt charts for the following set of processes using different scheduling algorithms: (*a*) **FCFS**, (*b*) **SJF**, (*c*) **SRT**, (*d*) **Priority** (Unix/Linux convention), (*e*) **Priority** (Windows convention), (*f*) **Priority with preemption** (Unix/Linux convention), and (*g*) **Priority with preemption** (Windows convention), (*h*) **RR** with quantum 5, (*i*) **RR** with quantum 4, (*j*) **RR** with quantum 3, and (*k*) **RR** with quantum 2. Based on the same tie-breaking rule as in **Question 1**, compute the waiting time and turnaround time for each process. What are the average values for waiting time and turnaround time? How many context switching *decisions* are made in each case?

Process	*Burst Time*	*Arrival Time*	*Priority*
P_1	17	0	4
P_2	10	1	7
P_3	5	2	2
P_4	4	3	7
P_5	13	4	4

What are the response times and average response time if each process echoes a message *just before the middle* of its execution (e.g., after P_1 executes for almost 8.5 time units, and P_2 for almost 5 time units). Assume that it takes negligible time for the I/O, without incurring any context switching cost. Do you have any *observation*?

3. Multi-level Scheduling.

Assume that a *multi-level feedback queue* is adopted to schedule the following processes for execution. Processes first enter the *high priority queue* based on **RR** with quantum 2. A process that cannot complete its execution after a service time of 4 in this high priority queue will be demoted to the *medium priority queue* based on **RR** with quantum 3. A process that cannot complete its execution after a service time of 6 in this medium priority queue will be demoted to the *low priority queue* based on **FCFS**.

Assume that **Fixed Priority scheduling** is adopted. In other words, processes in the lower priority queue will ***not*** get executed until all processes in the higher priority queues have completed their execution. Draw a Gantt chart for the process execution schedule. Compute the process waiting and turnaround time.

Now, assume that **Time Slicing scheduling** is adopted with a ratio of 6:3:1 in allocated time slices, in the format of 12 time units, 6 time units and 2 time units respectively on the three queues. In other words, after the first queue uses up 12 time units for its processes, the CPU will be given to the second queue regardless of whether there are still some processes in the first queue. When 6 time units are used up in the second queue, the CPU is given to the third queue. After the third queue uses up its time units, the CPU is given back to the first queue for 12 time units. Draw a Gantt chart for the process execution schedule. Compute the process waiting and turnaround time.

Process	*Burst Time*	*Arrival Time*
P_1	7	0
P_2	12	1
P_3	13	2
P_4	9	3
P_5	8	4