COMP 2432 Operating Systems
Written Assignment Submission to BlackBoard
Deadline: Sunday, March 31, 2024

1. CPU Scheduling.

Draw Gantt charts for the following set of processes using different scheduling algorithms: (*a*) **SRT**, (*b*) **Priority with preemption** (Linux convention), (*c*) **Priority with preemption** (Windows convention), (*d*) **RR** with quantum $Q = 3$, and (*e*) **RR** with quantum $Q = 2$. Let us assume the standard tie-breaking rules when multiple processes are eligible, i.e., the process with smaller ID will go first. *Compute* the *waiting time* and *turnaround time* for each process under the algorithms (*a*) to (*e*).

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

(*f*) Response time measures the time taken for the first response of a process. However, response time does not always provide a good measure for the responsiveness of interactive processes since it only measures the time taken for the first response. Waiting time is a useful measure for system performance too, but normally we just measure the total waiting time for a process by adding up all the waiting periods. The first waiting period is normally more important, as it measure the response time at the beginning of a process. As execution continues, subsequent waiting periods are not as much as a concern. Here, let us take a weight on each waiting period and add them up, instead of simply adding them up in our normal total waiting time computation. In summary, assume that a weighting factor w is adopted. The first waiting period will take a weight of 1, the second period a weight of w, the third period a weight of w^2, the fourth period a weight of w^3, and so on. The resulting waiting time would be the *total weighted waiting time*. If $w = 1$, this is the same as the total waiting time, and if $w < 1$, this will be smaller. *Compute* this *total weighted waiting time* for the processes for $w = 0.9$ and $w = 0.8$. *Compute* also the total weighted waiting time for the extreme case of **RR** with quantum $Q = 1$. Note that for non-preemptive algorithms, total weighted waiting time is the same as total waiting time as there is only one waiting period. State briefly your observations.

2. Multi-level Scheduling.

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 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 queue will be demoted to the *low priority queue* based on **FCFS**.

(*a*) Assume that **Fixed Priority scheduling** is adopted. *Draw* a Gantt chart for the process execution schedule. *Compute* the process *waiting* and *turnaround time*.

(*b*) Assume that **Time Slicing scheduling** is adopted with a ratio of 5:3:2 in allocated time slices, in the format of 10 time units, 6 time units and 4 time units respectively on the three queues. *Draw* a Gantt chart for the process execution schedule. *Compute* the process *waiting* and *turnaround time*.

Process	*Burst Time*	*Arrival Time*	*Priority*
P_1	15	0	4
P_2	7	1	3
P_3	11	2	1
P_4	5	4	2
P_5	12	6	5

3. Contiguous Memory Allocation.

A memory system currently has three holes of size 210K, 291K, and 254K and the total memory is 1000K. You are given 12 requests: 84, 90, 97, 81, 79, 20, 77, 64, 56, 68, 22, and 38K, arriving in that order. *Indicate* how the requests are satisfied with (*a*) **first-fit** algorithm, (*b*) **best-fit** algorithm, (*c*) **worst-fit** algorithm, and (*d*) an ***optimal*** algorithm that makes a decision *after* seeing all requests using an oracle. There are *multiple possible answers* for the optimal algorithm. You are to give *two* possible answers to (*d*). What are the *utilizations* for the four algorithms?

(*e*) It is found that there are some small off-by-one errors in managing the holes and keeping track of the sizes. The total size is found to increase from 755K to 756K, since the size of one of the three holes has been under-reported by 1K. Since it is unknown which hole has been under-reported for its size, you would try three different cases (*i*) to (*iii*) to see whether an improvement to the optimal algorithm can be made: (*i*) 211K, 291K, 254K, (*ii*) 210K, 292K, 254K, (*iii*) 210K, 291K, 255K. *Give* the request satisfaction scenario if utilization can be improved in some of the three. Otherwise, please state that no improvement is possible.

(*f*) You are given some tasks of size *x*K, *y*K and *z*K respectively and there are at most **9** tasks each. *Determine* the *best* that you could achieve if you are to fill up a hole of size *S*K, for the following values of *x*, *y*, *z* and *S*, by minimizing the unused space left, if any. *Show* how many tasks of size *x*, how many of size *y* and how many of size *z* are used in each case. (*g*) Now if we relax the requirement so that there are no upper limit on the number of tasks for each type, *determine* the *minimal* unused space left and the *task mix*. (*h*) If we *tighten* the requirement so that there are still at most 9 tasks of each size, but we also require that each type of tasks must be used *at least once*, *determine* the *minimal* unused space left and the *task mix*.

x	36	26	29
y	49	57	39
z	59	62	49
S	800	775	570

4. Segmentation.

Consider the segment table for process P_1 containing the following.

P_1

Segment	*Base*	*Length / Limit*
0	3011	135
1	1901	234
2	5678	543
3	2432	304
4	4434	787
5	1011	345
6	3901	135

Suppose that **segment 3** of P_1 is a shared segment with **segment 6** of P_2, and P_2 has 7 segments of size 55, 604, 103, 212, 72, 352 and 304 respectively, starting from segment 0 to segment 6. Note that **segment 3** of P_1 has the same size as **segment 6** of P_2, since it is shared. Assume that the computer has a memory of 8KB allocated for users, with physical address from 0 to 8191. Those bytes at the lowest end of the memory as well as those at the highest end of the memory, from 0 to 999, and from 6789 to the end, are reserved by the operating system. Segments for P_2 are to be allocated from the free memory in the order of 0, 1, 2, 3, 4, 5, and 6, if necessary, using (*a*) **first-fit** algorithm, (*b*) **best-fit** algorithm, (*c*) **worst-fit** algorithm. *Show* the *three segment tables* for P_2 for segments allocated under the three algorithms. Note that it is useful to draw diagrams showing the memory allocation to the used segments for clarity.

(*d*) *Translate* the following logical addresses for P_1 and P_2 by *filling* in the table below.

Allocation algorithm for P_2		**FF**	**BF**	**WF**
Logical address	Physical address for P_1	Physical address for P_2		
(0, 44)				
(1, 231)				
(2, 82)				
(3, 199)				
(4, 56)				
(5, 304)				
(6, 135)				