UNIVERSITY OF BIRMINGHAM

M.Sc. CS/AI & CS
Computer Systems
Process Scheduling, Concurrency and Threads [Solutions]

Exercise #1: Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed (in milliseconds). In answering the questions, use non-preemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

a) What is the average turnaround time for these processes with the FCFS scheduling algorithm?

For the given set of processes, here is the FCFS schedule:

P1	P1	P1	P1	P1	P1	P1	P1	P2	P2	P2	P2	P3
0	1	2	3	4	5	6	7	8	9	10	11	12

Turnaround Time for P1 = 8 – 0 = 8 ms
Turnaround Time for P2 = 12 – 0.4 = 11.6 ms
Turnaround Time for P3 = 13 – 1 = 12 ms
Average Turnaround Time = (8 + 11.6 + 12) / 3 = 10.53 ms

b) What is the average turnaround time for these processes with the SJF scheduling algorithm?

For the given set of processes, here is the SJF schedule:

P1	P1	P1	P1	P1	P1	P1	P1	P3	P2	P2	P2	P2
0	1	2	3	4	5	6	7	8	9	10	11	12

Turnaround Time for P1 = 8 – 0 = 8 ms
Turnaround Time for P2 = 13 – 0.4 = 12.6 ms
Turnaround Time for P3 = 9 – 1 = 8 ms
Average Turnaround Time = (8 + 12.6 + 8) / 3 = 9.53 ms

c) The SJF algorithm is supposed to improve performance, but notice that we chose to run process P1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P1 and P2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge scheduling.

For the given set of processes, here is the SJF (with future-knowledge) schedule:

-	P3	P2	P2	P2	P2	P1	P1	P1	P1	P1	P1	P1	P1
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Turnaround Time for P1 = 14 – 0 = 14 ms
Turnaround Time for P2 = 6 – 0.4 = 5.6 ms
Turnaround Time for P3 = 2 – 1 = 1 ms
Average Turnaround Time = (14 + 5.6 + 1) / 3 = 6.87 ms

Exercise #2: The traditional UNIX, the scheduler enforces an inverse relationship between priority numbers and priorities: the higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

$$\text{Priority} = (\text{recent CPU usage} / 2) + \text{base}$$

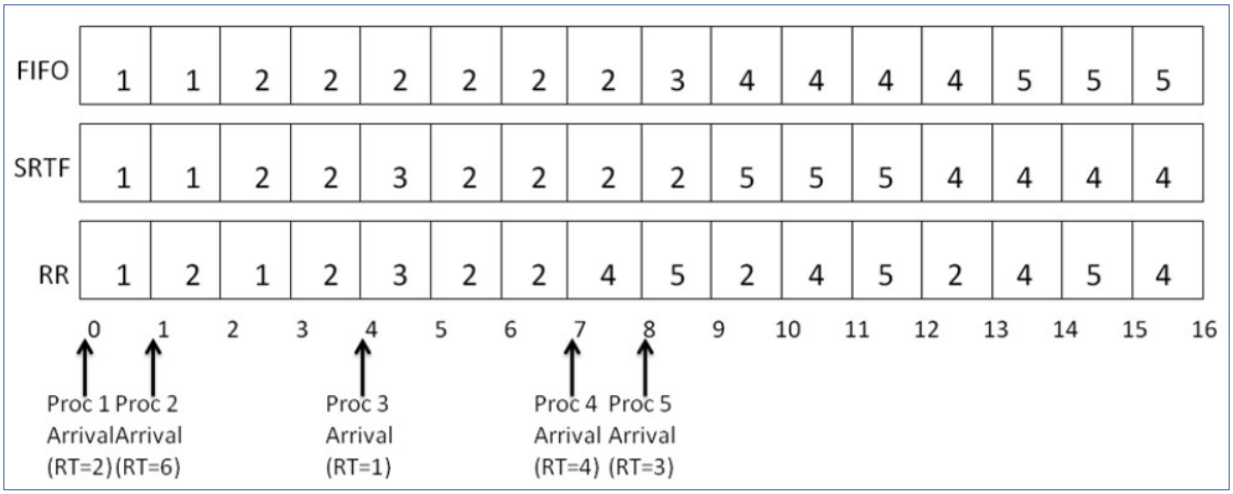
where base = 60 and recent CPU usage refers to a value indicating how often a process has used the CPU since priorities were last recalculated. Assume that recent CPU usage for process P_1 is 40, for process P_2 is 18, and for process P_3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

The priorities assigned to the processes are 80, 69, and 65 respectively. The scheduler lowers the relative priority of CPU-bound processes.

Exercise #3: Consider the following table of processes and their associated arrival and running times.

Process ID	Arrival Time (AT)	Running Time (RT)
1	0	2
2	1	6
3	4	1
4	7	4
5	8	3

(a) Show the scheduling order for these processes under 3 different policies: First Come First Serve (FCFS), Shortest-Remaining-Time-First (SRTF), Round-Robin (RR) with a quantum size = 1, by filling in the Gantt chart with ID of the process currently running in each time quantum. Assume that context switch overhead is 0 and that new RR processes are added to the head of the queue and new FCFS processes are added to the tail of the queue.



b) Compute the response time for each process in each schedule above & fill in the following table:

Scheduler	Process 1	Process 2	Process 3	Process 4	Process 5
FIFO	0 – 0 = 0	2 – 1 = 1	8 – 4 = 4	9 – 7 = 2	13 – 8 = 5
SRTF	0 – 0 = 0	2 – 1 = 1	4 – 4 = 0	12 – 7 = 5	9 – 8 = 1
RR	0 – 0 = 0	1 – 1 = 0	4 – 4 = 0	7 – 7 = 0	8 – 8 = 0

Exercise #4: A supermarket has sensors on each entrance that detect when a customer enters or leaves. These sensors send a signal to a central computer whenever a customer enters or leaves. This computer, amongst other things, keeps a running count of how many people are in the shop at any one time. This is used to control a traffic light system that only allows customers to enter the store when the number of customers is below a safe limit. Last week, the store was empty - but the system refused to let anyone in. What went wrong? How would you fix it?

The sensors at the entrances send signals to the central computer system when a customer enters or leaves the store. On receiving a signal, the computer either increments or decrements (depending on the type of signal) the count of how many customers are inside the supermarket. Now, this increment / decrement operation, as it is given in the question, has not been implemented properly i.e. the counter is a shared resource and the increment / decrement operations do not take a lock before updating it, which means that we can have race conditions. It is quite possible that the value of the counter will be incorrect after a number of these operations, causing the system to malfunction i.e. the store is empty and the system thinks that there are customer inside the shop.

The obvious solution will be to implement a locking mechanism e.g. mutual exclusion before updating the value of counter.

Exercise #5: Give three example applications where using threads will be more efficient than using processes.

a) A Web server that services each request in a separate thread.
b) A parallelized application such as matrix multiplication where different parts of the matrix may be worked on in parallel.
c) An interactive GUI program such as a debugger where a thread is used to monitor user input, another thread represents the running application, and a third thread monitors performance.

Exercise #6: Suppose that two long-running processes, P1 and P2, are running in a system. Neither program performs any system calls that might cause it to block, and there are no other processes in the system. P1 has 3 threads and P2 has 2 threads. The system may use either kernel or user threads.

a) What percentage of CPU time will P1 get if the threads are kernel threads? Briefly Explain.
b) What percentage of CPU time will P1 get if the threads are user threads? Briefly Explain.

a) If the threads are kernel threads, they are independently scheduled and each of the five threads will get a share of the CPU. Thus, the 3 threads of P1 will get 3/5 of the CPU time. That is, P1 will get 60% of the CPU.

b) If the threads are user threads, the threads of each process map to one kernel thread, so each process will get a share of the CPU. The kernel is unaware that P1 has three threads. Thus, P1 will get 50% of the CPU.

Exercise #7: Your friend Jane needs to run a simulation for her thesis and her adviser wants her to run it for a fixed problem size. Jane can make 90% of the program parallel, with 10% of it being sequential.

a) What is the maximum speedup Jane can expect on 10 processors?
b) What would be the maximum speedup on an infinite number of processors?

a) Speedup = 1 / ((1 – f) + f / p) = 1 / ((1 – 0.9) + 0.9 / 10) = 1 / (0.1 + 0.09) = 1 / 0.19 = 5.26x	b) Max Speedup = 1 / ((1 – f) + f / ∞) = 1 / ((1 – 0.9) + 0.9/ ∞) = 1 / (0.1 + 0) = 1 / 0.1 = 10x