OPERATING SYSTEM: CSET209

SRTF: ALSO FIND THE CPU IDLE TIME.

P. No.	AT	I/O	CPU TIME	I/O	CT	TAT	WT
1	0	4	14	2			
2	0	8	28	4			
3	0	12	42	6			

PRIORITY SCHEDULING

P. No.	AT	Priority	CPU	I/O	CPU	CT	TAT	WT
1	0	2	1	5	3			
2	2	3 (low)	3	3	1			
3	3	1 (high)	2	3	1			

SRTF AND FCFS SCHEDULING

P. No.	AT	CPU	I/O	CPU	I/O	CPU	I/O	CPU	I/O	CT	TAT	WT
1	0	10	90	10	90	10	90	10	90			
2	0	10	90	10	90	10	90	10	90			

OUTLINE

- Multilevel Queue Scheduling
- Multilevel Feedback Queue
- Advanced Scheduling Algorithm: Highest response ration next, Lottery scheduling

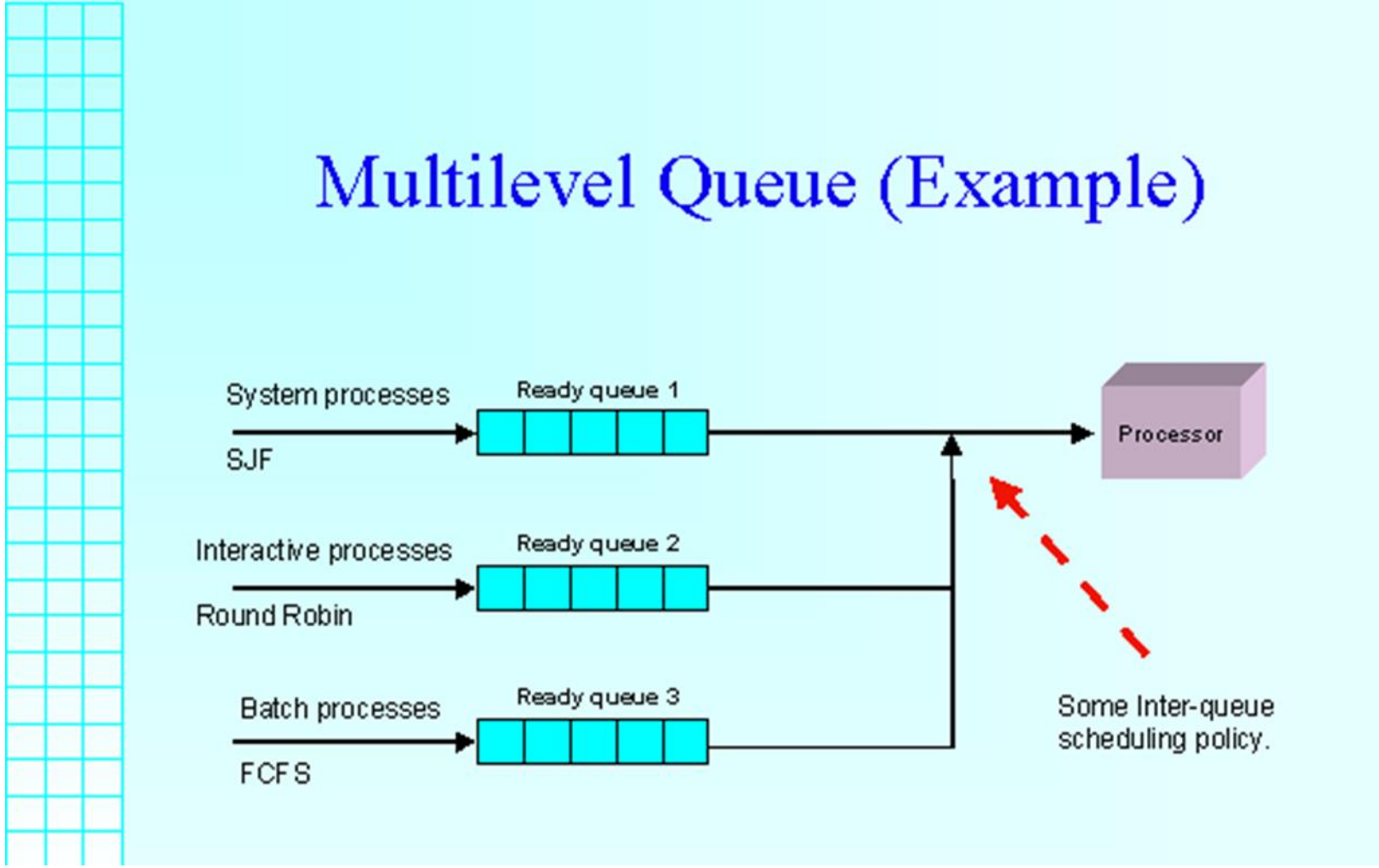
QUESTIONS TO ASK YOURSELF !

- Q1. Which of the scheduling algorithms seen so far is the best?
- Q2. How we can combine different algorithms together to arrive at a better algorithm?

MULTILEVEL QUEUE SCHEDULING

- Algo is suitable for processes that can be categorized into groups.
- Here, ready queue is partitioned into multiple queues. For e.g.,
  - Foreground (Interactive)
  - Background (Batch)
- Each queue follows its own scheduling algorithm

MULTILEVEL QUEUE SCHEDULING

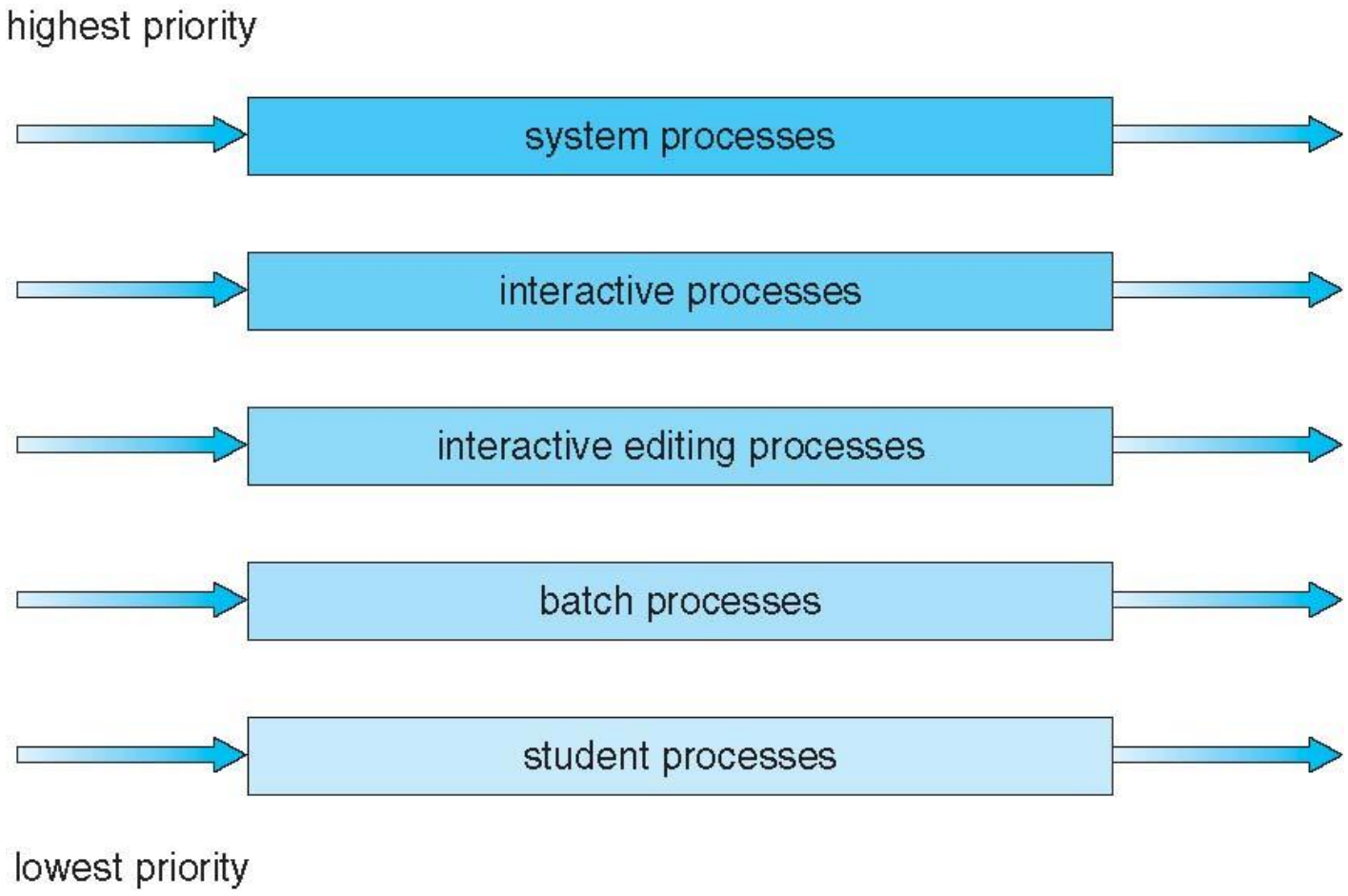


IMG SRC: https://slideplayer.com/slide/14919762/

MULTILEVEL QUEUE SCHEDULING

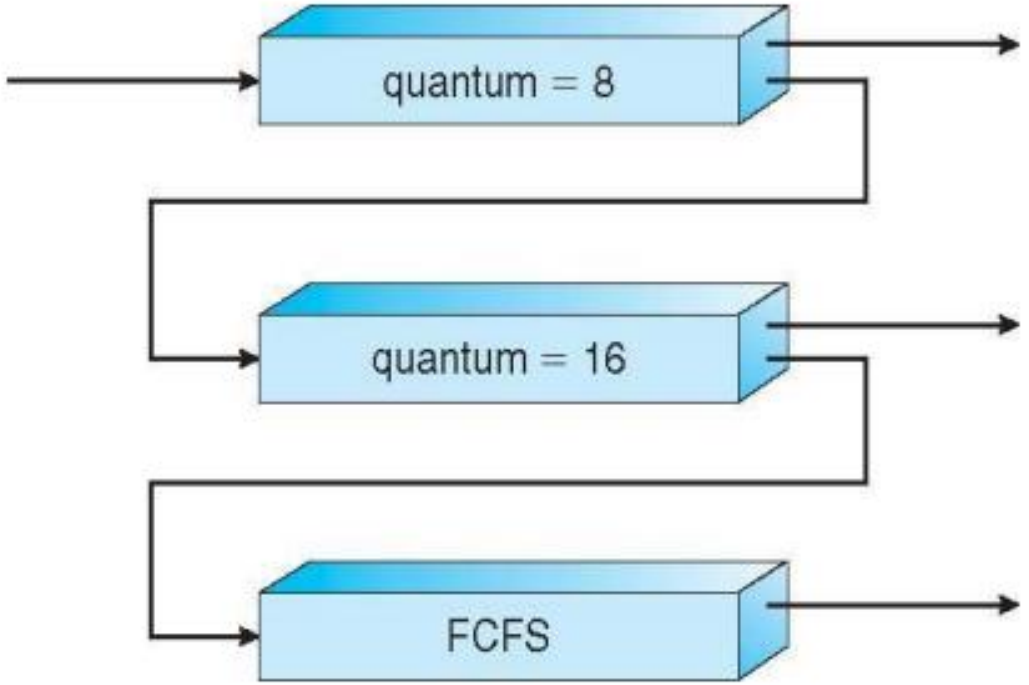
- Inter-queue scheduling is required, commonly implemented as fixed-priority preemptive scheduling.
- For example:
  - fixed priority scheduling (lead to starvation)
  - Time slice for queues: 80% to interactive and 20% to batch

MULTILEVEL QUEUE SCHEDULING



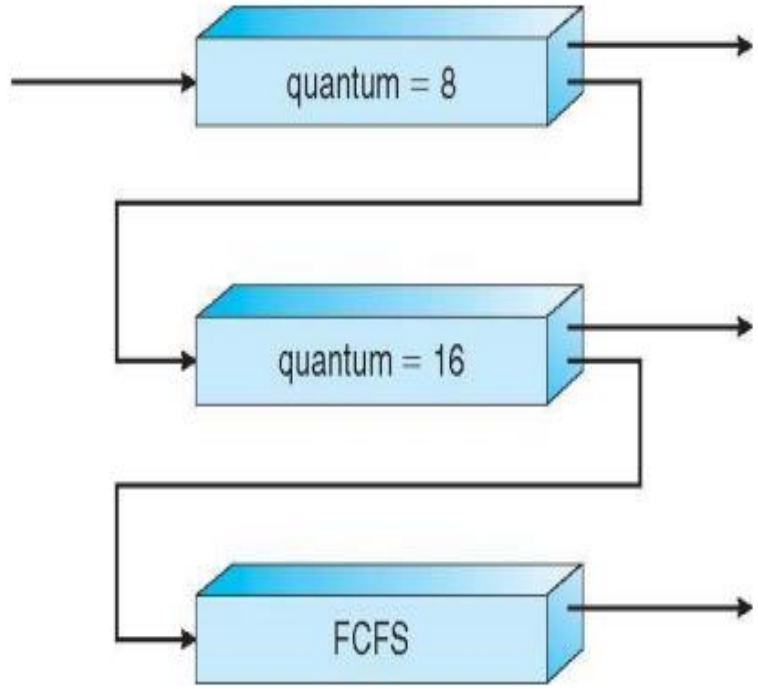
MULTILEVEL FEEDBACK QUEUE SCHEDULING

- In MLQ, processes do not move between queues because of predefined priorities
  - Advantage: Low scheduling overhead
  - Disadvantage: Inflexible i.e. a process is permanently assigned to a particular queue.
- Multilevel Feedback Scheduling allows a process to move between queues.
  - If a process uses too much CPU time then it gets moved to a lower-priority queue.
  - If a process waits for too long then it gets moved to a higher-priority queue. This kind of aging prevents starvation.



MULTILEVEL FEEDBACK QUEUE : EXAMPLE

- **Three queues**:
  - Q0 – RR with time quantum 8ms
  - Q1 – RR time quantum 16ms
  - Q2 – FCFS
- The scheduler first executes all processes in queue 0.
- Only when queue 0 is empty will it execute processes in queue 1.
- Processes in queue 2 will be executed only if queues 0 and 1 are empty.
- A process that arrives for queue 1 will preempt a process in queue 2.
- A process in queue 1 will in turn be preempted by a process arriving for queue 0.

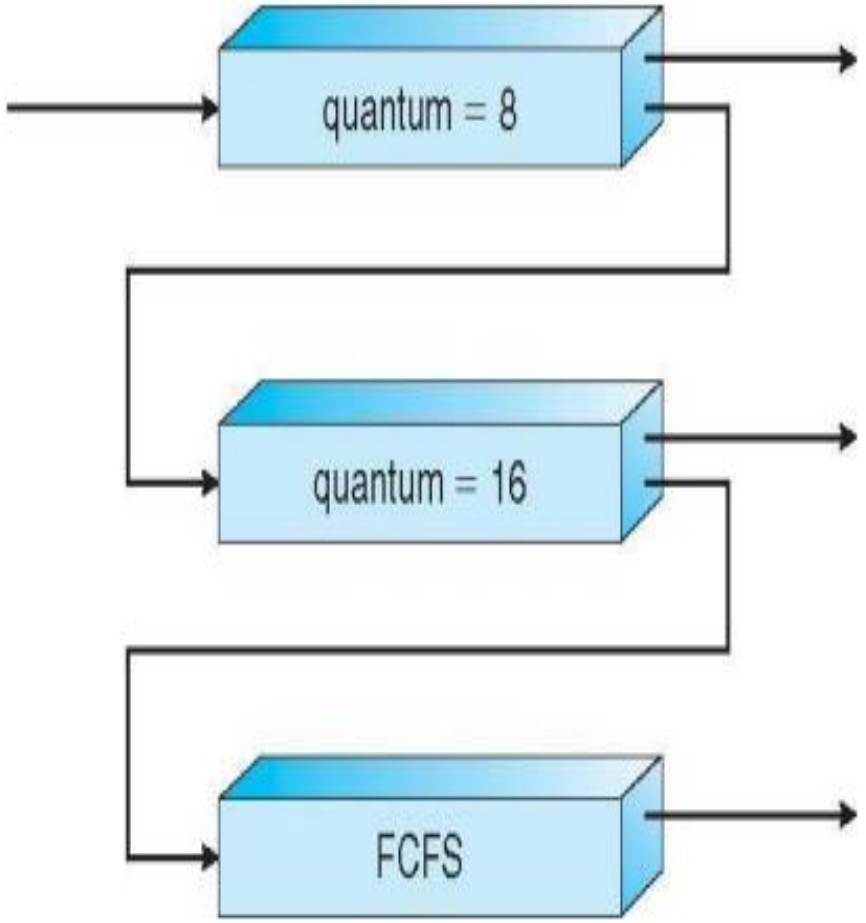


- **Aim**: Preference given to short lived processes

MLFQ scheme leaves I/O-bound and interactive processes in the higher-priority queues OR lower-priority queues?

higher-priority queues

An entering process is put in queue 0. A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1. If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2. Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty. To prevent starvation, a process that waits too long in a lower-priority queue may gradually be moved to a higher-priority queue.

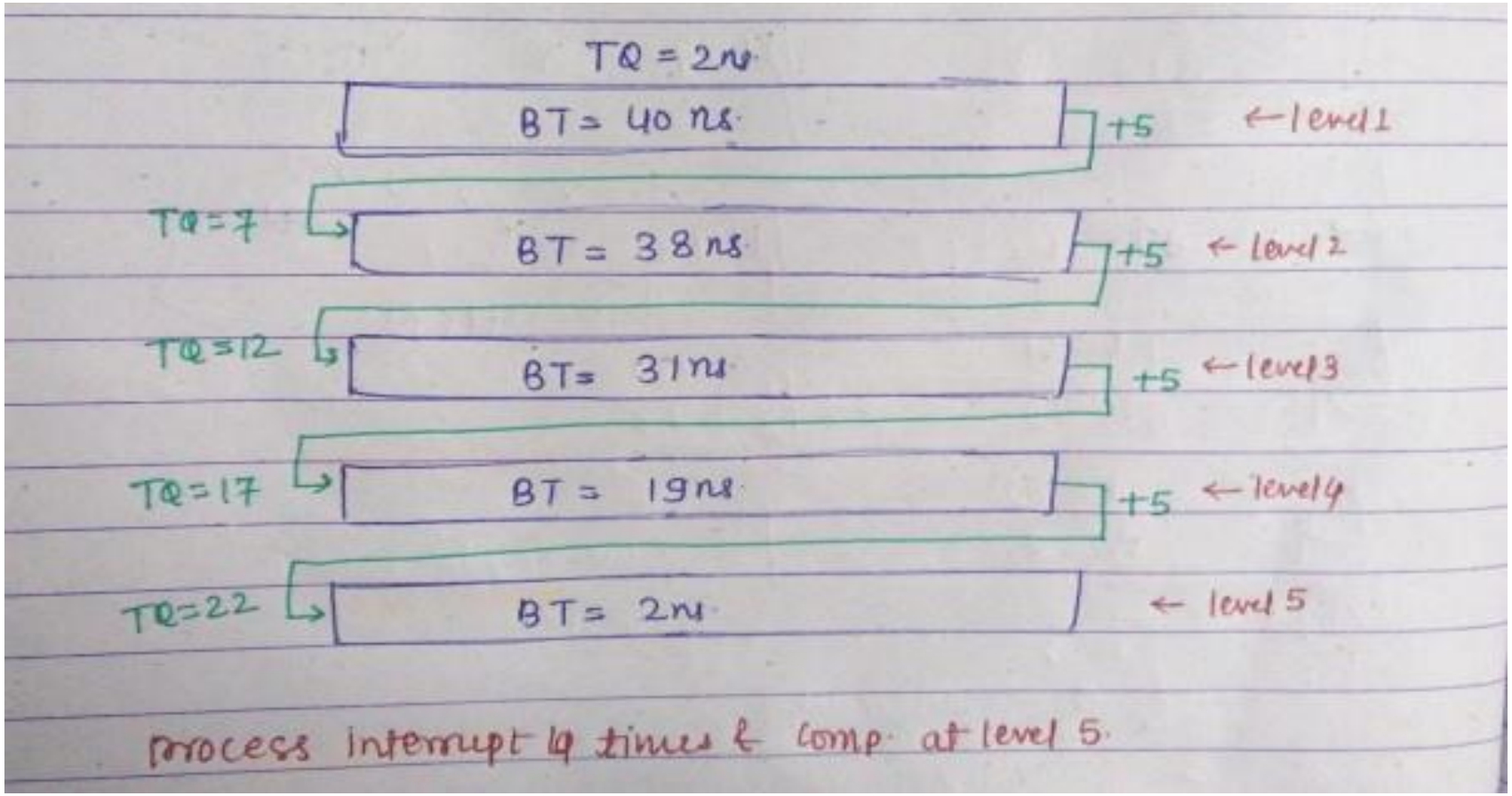


MULTILEVEL FEEDBACK QUEUE SCHEDULING

- Multilevel Feedback Queue Scheduler is defined by following parameters:
  - Number of queues
  - Scheduling algo for each queue
  - Method to determine when to upgrade a process
  - Method to determine when to demote a process
  - Method to determine which queue a process will enter when that process needs service

Question: Consider a system which has a CPU bound process, which require the burst time of 40 seconds. The multilevel Feed Back Queue scheduling algorithm is used and the queue time quantum '2' seconds and in each level it is incremented by '5' seconds. Then how many times the process will be interrupted and on which queue the process will terminate the execution?

Question: Consider a system which has a CPU bound process, which require the burst time of 40 seconds. The multilevel Feed Back Queue scheduling algorithm is used and the queue time quantum '2' seconds and in each level it is incremented by '5' seconds. Then how many times the process will be interrupted and on which queue the process will terminate the execution?



HIGHEST RESPONSE RATIO NEXT (HRRN)

- **Non-preemptive scheduling algorithm:** Once a process is selected for execution will run until its completion
- Modification of **Shortest Job Next (SJN)** in order to reduce the problem of starvation.
- In the HRRN scheduling algorithm, the CPU is assigned to the next process that has the **highest response ratio** and not to the process having less burst time.
- **Response Ratio = (W+S)/S**

  Where, **W** = Waiting Time, **S** = Burst Time

HIGHEST RESPONSE RATIO NEXT (HRRN)

- **Steps:**
  - Calculate the waiting time for all the processes. Waiting time simply means the sum of the time spent waiting in the ready queue by processes.
  - Processes get scheduled each time for execution in order to find the response ratio for each available process.
  - Then after the process having the highest response ratio is executed first by the processor.
  - In a case, if two processes have the same response ratio then the tie is broken using the FCFS scheduling algorithm.

HRRN: EXAMPLE

- At t = 0, no process is available in the ready queue,

  therefore, CPU is idle for t = 0 to 1
- At t = 1, only P1 is available, therefore it is executed till completion
- At t = 4, only P2 has arrived, so P2 is executed till completion
- At t = 10, there is P3, P4, and P5 in the ready queue, so in order to

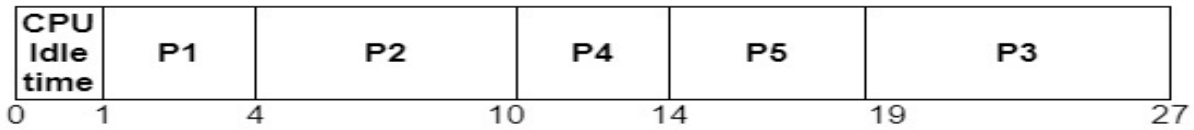
  schedule the next process, we need to compute the response ratio:
  - P3 = ((10-5)+8) / 8 = 1.625
  - P4 = ((10-7) +4) / 4 = 1.75
  - P5 = ((10-8) +5) / 5 = 1.4

  Since P4 has the highest response ratio, so P4 is executed next.
- At t = 14, compute the response ratio of P3 and P5: P3 = ((14-5) +8) / 8 = 2.125

  P5 = ((14-8) +5) / 5 = 2.2

  So P5 is executed next followed by P3

Process	Burst Time	Arrival Time
P1	3	1
P2	6	3
P3	8	5
P4	4	7
P5	5	8



HIGHEST RESPONSE RATIO NEXT (HRRN)

- **Advantages**
  - Gives better performance than the shortest job first Scheduling.
  - There is a reduction in waiting time for longer jobs and also it encourages shorter jobs.
  - Throughput increases.
- **Disadvantages**
  - Practical implementation of HRRN scheduling is difficult because we cannot know the burst time of every process in advance.
  - There may occur overhead on the processor.

Find average TAT and average waiting time using HRRN algorithm.

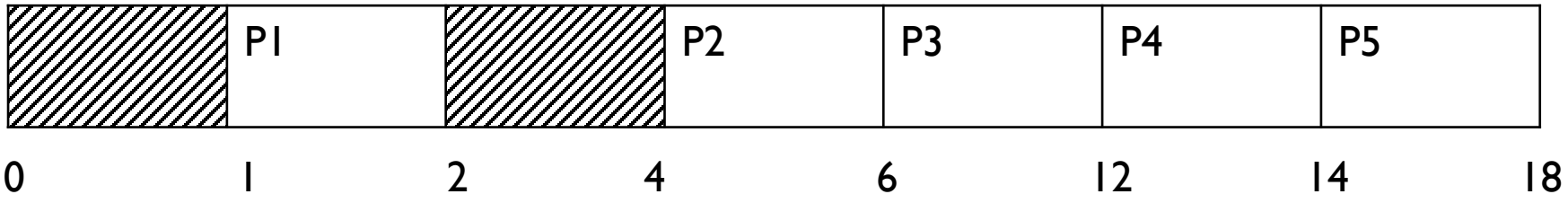
P.No.	AT	BT	CT	TAT	WT
1	0	3			
2	2	6			
3	4	4			
4	6	5			
5	8	2			

Find average TAT and average waiting time using HRRN algorithm.

P.No.	AT	BT	CT	TAT	WT
1	1	1			
2	4	2			
3	5	6			
4	6	2			
5	7	4			

Find average TAT and average waiting time using HRRN algorithm.

P.No.	AT	BT	CT	TAT	WT
1	1	1			
2	4	2			
3	5	6			
4	6	2			
5	7	4			



LOTTERY SCHEDULING

Lottery Scheduling is a type of process scheduling, somewhat different from other Scheduling. Processes are scheduled in a random manner. Lottery scheduling can be preemptive or non-preemptive. It also solves the problem of starvation. Giving each process at least one lottery ticket guarantees that it has a non-zero probability of being selected at each scheduling operation. In this scheduling every process has some tickets and the scheduler picks a random ticket and process having that ticket is the winner and it is executed for a time slice and then another ticket is picked by the scheduler. These tickets represent the share of processes. A process having a higher number of tickets give it more chance to get chosen for execution.

LOTTERY SCHEDULING

Example – If we have two processes A and B having 60 and 40 tickets respectively out of total of 100 tickets. CPU share of A is 60% and that of B is 40%. These shares are calculated probabilistically and not deterministically.

Explanation:

We have two processes A and B. A has 60 tickets (ticket number 1 to 60) and B has 40 tickets (ticket no. 61 to 100).

Scheduler picks a random number from 1 to 100. If the picked no. is from 1 to 60 then A is executed otherwise B is executed.

An example of 10 tickets picked by the Scheduler may look like this follows:

Ticket number - 73 82 23 45 32 87 49 39 12 09.

Resulting Schedule - B B A A A B A A A A.

A is executed 7 times and B is executed 3 times. As you can see that A takes 70% of the CPU and B takes 30% which is not the same as what we need as we need A to have 60% of the CPU and B should have 40% of the CPU. This happens because shares are calculated probabilistically but in a long run(i.e when no. of tickets picked is more than 100 or 1000) we can achieve a share percentage of approx. 60 and 40 for A and B respectively.

TICKET DISTRIBUTION IN LOTTERY SCHEDULING

- **Static Distribution** − The number of tickets assigned to each process in this method is fixed and does not change over time. For instance, a process with a higher priority may be assigned 100 tickets. On the other hand, a process with a lower priority may be assigned only 10 tickets. This method is simple to implement, but it may not result in the most efficient or equitable resource distribution.

- **Dynamic Distribution** − In this method, the total amount of tickets allocated to each process may fluctuate over *time according* to the system's behavior. For example, if a strong-priority process is taking up materials and starving other operations, its ticket count may be minimized to give other procedures an increased likelihood of being selected. Although this technique has a greater computation overhead, it may result in more effective and equal resource allocation.

- **Weighted Distribution** − In this method, the total amount of tickets designated to each process is determined by factors other than its priority. Other factors, which include the quantity of processing power it has already consumed, also play a role. Despite having the same priority, an operation that consumed a lot of processing power may be allocated a lesser number of tickets than a procedure that has employed very little CPU time. This approach can be difficult to put in place, but it can help prevent processes from monopolizing resources.

SELECTION OF A WINNING TICKET: RANDOMNESS FOR FAIRNESS

- Once the tickets have been allocated, choosing a winning ticket comes in the lottery schedule. It is a lottery-style random draw process, where everyone has an equal chance to win the prize. The operating system uses a random number generator to ensure a fair and unbiased selection process.
- The random drawing of tickets is a defining characteristic in the scheduled allocation process for lottery ticket selection and lends an element of chance to it. This randomness is important to prevent processes with consistently higher ticket counts from taking over the CPU. Instead, each process has an equal chance of being chosen for execution--no matter how many tickets it holds. That lends fairness to resource distribution.

PROCESS EXECUTION AND TICKET REDISTRIBUTION: BALANCING FAIRNESS AND EFFICIENCY

- When a winning ticket is selected, the process that corresponds to it gains access to execute on the CPU. After that, the implementation goes as in any other scheduling algorithm: The chosen process runs until it either gives up the CPU voluntarily or uses its allotted time quantum.
- The lottery scheduling algorithm may involve the redistribution of tickets after a process is completed. There are several possible reasons: a change in process priority, changes to resource requirements or the completion of this execution phase. The operating system re-determines the tickets allocated to each process and adjusts accordingly to the system's current state.

ALGORITHM STEPS

- Ticket Allocation Algorithm
- Random Ticket Selection Algorithm
- Execution and Ticket Redistribution

ADVANTAGES OF LOTTERY SCHEDULING

- Fairness: Lottery process scheduling is designed to be fair because every process has a chance of being selected. This ensures that no process is starved of CPU time, and it also helps to prevent monopolization of the CPU by a single process.
- Flexibility: The lottery process scheduling algorithm is highly flexible and can be easily adapted to handle various scheduling requirements, such as real-time scheduling, multi-processor scheduling, and so on.
- Simple implementation: The implementation of the lottery process scheduling algorithm is relatively simple compared to other scheduling algorithms, making it easier to develop and maintain.
- Combining with other scheduling algorithms: Lottery scheduling can be combined with other scheduling algorithms such as round-robin or priority scheduling to achieve specific scheduling goals.

DISADVANTAGES OF LOTTERY SCHEDULING

- Overhead: The lottery process scheduling algorithm requires additional overhead because it must generate lottery tickets for each process and randomly select a winner. This overhead can be significant in systems with many processes.
- Complexity: Although the implementation of the lottery process scheduling algorithm is relatively simple, the algorithm itself can be quite complex, especially when compared to other scheduling algorithms such as round-robin or FIFO.
- Security: Lottery process scheduling is not a secure algorithm because it is based on randomness, which can be manipulated. An attacker could potentially gain an unfair advantage by manipulating the generation of lottery tickets, leading to security vulnerabilities.

THANK YOU
?