Round Robin (RR) Scheduling is a widely used CPU scheduling algorithm in operating systems. It assigns a fixed time quantum (or time slice) to each process in a cyclic manner. When a process exceeds its time quantum, it is preempted and placed back in the queue, allowing the next process to execute. This process continues until all processes are completed.

**Approach Using a Queue:**

To simulate Round Robin scheduling using a **queue**, we can model processes and their bursts (execution time needed) as follows:

1. **Enqueue processes** in the queue with their remaining burst times.
2. **Process execution**: For each process, we:
   * Let it execute for a fixed time quantum (e.g., TQ).
   * If a process has more time remaining than the quantum, it is preempted and re-enqueued with the remaining burst time.
   * If the process completes (i.e., its remaining burst time becomes 0), it is removed from the queue.
3. The queue ensures the processes are executed in a cyclic manner.

**Code Implementation**

import java.util.LinkedList;

import java.util.Queue;

class Process {

int processID;

int burstTime; // The remaining time for the process to execute

Process(int processID, int burstTime) {

this.processID = processID;

this.burstTime = burstTime;

}

}

public class RoundRobinScheduling {

// Simulate Round Robin Scheduling

public static void roundRobin(Queue<Process> processes, int timeQuantum) {

Queue<Process> queue = new LinkedList<>(processes);

// Process each task in the queue

while (!queue.isEmpty()) {

Process currentProcess = queue.poll();

System.out.println("Processing: P" + currentProcess.processID);

if (currentProcess.burstTime > timeQuantum) {

// Process gets preempted, reduce the burst time and re-enqueue

currentProcess.burstTime -= timeQuantum;

queue.add(currentProcess); // Re-enqueue with remaining burst time

System.out.println("P" + currentProcess.processID + " preempted. Remaining time: " + currentProcess.burstTime);

} else {

// Process finishes execution

System.out.println("P" + currentProcess.processID + " completed. Time taken: " + currentProcess.burstTime);

}

}

}

public static void main(String[] args) {

// Create a list of processes with their burst times

Queue<Process> processes = new LinkedList<>();

processes.add(new Process(1, 12)); // Process 1 with burst time 12

processes.add(new Process(2, 6)); // Process 2 with burst time 6

processes.add(new Process(3, 8)); // Process 3 with burst time 8

int timeQuantum = 4; // Time quantum of 4 units

roundRobin(processes, timeQuantum);

}

}

**Explanation:**

1. **Process Class**:
   * Each process has an ID and a burstTime, which represents how much time the process needs to finish its execution.
2. **roundRobin Method**:
   * The method accepts a queue of processes and a timeQuantum.
   * We use a LinkedList to simulate the queue and dequeue processes one by one.
   * If a process requires more time than the time quantum, we reduce its burst time by the quantum and re-enqueue it.
   * If a process completes (i.e., its remaining burst time is less than or equal to the time quantum), we simply print that the process is completed.
3. **Main Method**:
   * We define a set of processes with their burst times and a time quantum.
   * We then call the roundRobin method to simulate the scheduling.

**Example Output:**

Given processes with burst times [12, 6, 8] and a time quantum of 4, the output will look like:

Processing: P1

P1 preempted. Remaining time: 8

Processing: P2

P2 completed. Time taken: 6

Processing: P3

P3 preempted. Remaining time: 4

Processing: P1

P1 preempted. Remaining time: 4

Processing: P3

P3 completed. Time taken: 4

Processing: P1

P1 completed. Time taken: 4

**Explanation of the Output:**

* **P1** starts with a burst time of 12. After running for 4 units, its remaining time is reduced to 8, and it is re-enqueued.
* **P2** runs for 6 units, which is less than the time quantum, so it finishes execution in one cycle.
* **P3** starts with a burst time of 8. After running for 4 units, its remaining time becomes 4, and it is re-enqueued.
* The process repeats until all processes finish execution.

**Time and Space Complexity:**

* **Time Complexity**: The time complexity of the Round Robin scheduling simulation is O(n \* T), where n is the number of processes and T is the average time quantum or the burst time of each process. In the worst case, each process can be preempted multiple times before it finishes.
* **Space Complexity**: The space complexity is O(n) due to the space used by the queue to hold all the processes.