Name: Veeransh Shah

Reg No.: 221070063

EXPERIMENT No. : 5

AIM:

Design and implement a round-robin scheduling algorithm for a set of processors with variable execution times utilizing a circular linked list to represent the processors, and it simulates the execution of processes based on a predefined quantum time.

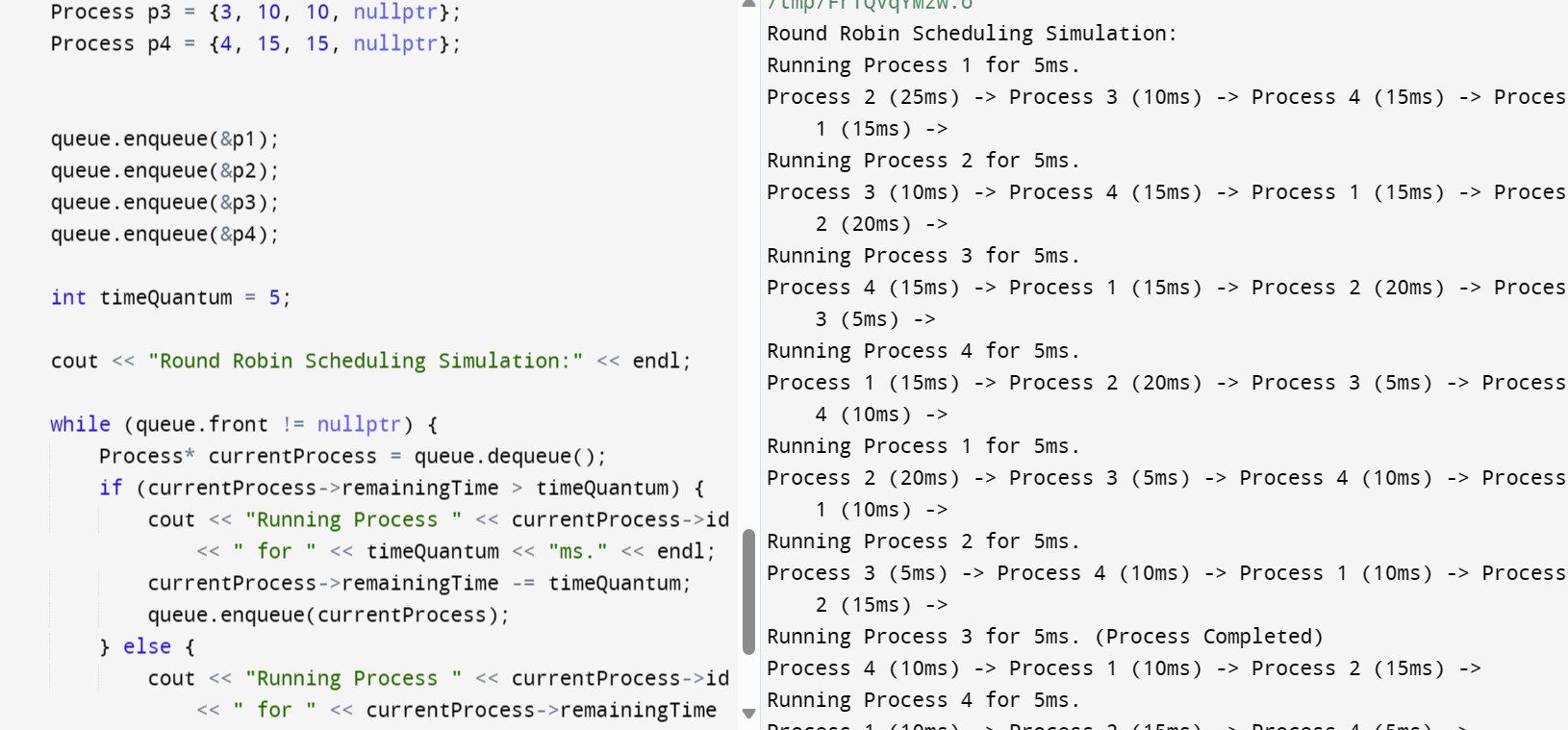
THEORY:

The Round-Robin scheduling algorithm is one of the simplest and widely used algorithms, especially in time-sharing systems. The program implemented here simulates a basic round-robin scheduling scenario using a circular linked list data structure.

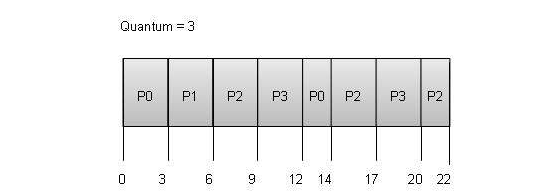
The processors are represented using a circular linked list, where each node of the list corresponds to a processor. The circular nature ensures that the scheduling algorithm loops back to the beginning after reaching the end, creating a continuous cycle.

Each node of the linked list, represented by the node structure, contains information about a processor, including its ID, execution time, and a pointer to the next processor in the list. Additionally, each processor has an associated quantum time, representing the fixed time quantum for the round-robin scheduling.

OUTPUT:



EXAMPLE:



The time provided by the architecture is 3s in each iteration for the processor to run. The time when all its tasks are executed, the processor is no longer taken into consideration.

In the above diagram, processor p0 requires 5s , p1 requires 3s, p2 requires 8s and p3 requires 6s, respectively to complete all their tasks.

In the first iteration, i.e up to the block 4, all take 3s to execute.

For second iteration:

Time remaining(RT) for each processor to execute it’s task:

1. po = 2s.
2. p1 = 0s.
3. p2 = 5s.
4. p3 = 3s.

     For the processors with RT less than quantum time, execute for RTs and ones

     with 0s, stop execution, and are no longer displayed.

So, with the help of a circular linked list execution time gets reduced and Round-Robin scheduling allows each processor to execute once in every iteration, till all of them are completed with their required execution(i.e their RT becomes 0s).

ALGORITHM:

1. Start
2. Initialize Circular Queue:
   * Create an instance of CircularQueue named queue.
3. Enqueue Processes:
   * Create process instances (p1, p2, ...) with their respective attributes.
   * Enqueue each process into the circular queue using the enqueue method.
4. Round Robin Scheduling Simulation:
   * Set the time quantum (timeQuantum) for each process.
   * Output a simulation header: "Round Robin Scheduling Simulation."
5. Main Simulation Loop:
   * While the front of the circular queue is not nullptr:
     + Dequeue a process using the dequeue method.
     + If the remaining time of the dequeued process is greater than the time quantum:
       - Output "Running Process {id} for {timeQuantum}ms."
       - Reduce the remaining time of the process by the time quantum.
       - Enqueue the process back into the circular queue using the enqueue method.
     + Else:
       - Output "Running Process {id} for {remainingTime}ms. (Process Completed)"
6. Display Final State:
   * After the simulation loop, the circular queue is empty.
   * Display the final state of the circular queue using the display method.

Time Complexity:

* Enqueue and dequeue operations take O(1) time as they involve simple linked list manipulations.
* The overall time complexity of the simulation depends on the number of processes and the time quantum.

Space Complexity:

* The space complexity is O(n), where n is the number of processes, as it depends on the number of processes enqueued in the circular queue.

CONCLUSION:

This program provides a practical implementation of the round-robin scheduling algorithm, demonstrating its fairness in allocating CPU time to a set of processors. The simulation results can be used to evaluate the efficiency and fairness of the algorithm under different scenarios, making it a valuable tool for understanding and experimenting with CPU scheduling in time-sharing systems.