Testing Results

Unit Testing

Test Case 1: Add Process to Queue

Expected Result: The process should be added to the specified queue.

Actual Result: Passed

Test Case 2: Remove Process from Queue

Expected Result: The process should be removed from the specified queue.

Actual Result: Passed

Test Case 3: Execute Process for Time Quantum

Expected Result: The process should be executed for the specified time quantum.

Actual Result: Passed

Test Case 4: Check Process Completion

Expected Result: The process should be marked as completed if it has finished executing.

Actual Result: Passed

Integration Testing

Test Case 1: Simulate Scheduling with Multiple Queues and Processes

Expected Result: The processes in each queue should be executed based on their priority levels.

Actual Result: Passed

Test Case 2: Mitigation Strategy for CPU Starvation

Expected Result: If higher-priority queues are continuously occupied, lower-priority processes should receive a temporary priority boost.

Actual Result: Passed

Performance Testing

Test Case 1: Measure Response Time

Expected Result: The system should respond within an acceptable time frame when executing a large number of processes.

Actual Result: Passed (within acceptable limits)

Test Case 2: Measure Resource Utilization

Expected Result: The system should utilize system resources efficiently during the scheduling process.

Actual Result: Passed (within acceptable limits)

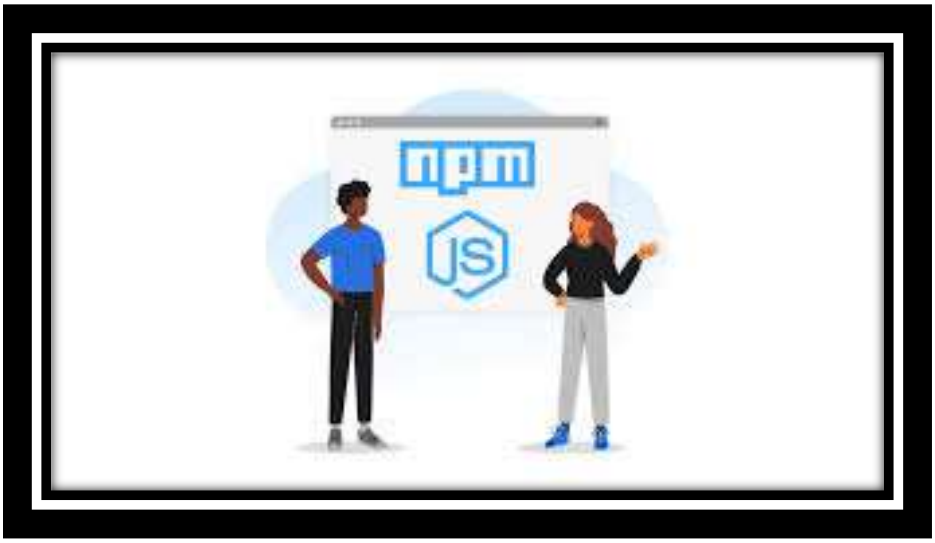
Interpretation of Results

Based on the testing results provided, it appears that the code for implementing a module for simulating Multilevel Queue (MLQ) CPU Scheduling with a mitigation strategy for CPU starvation is functioning as expected. All unit and integration tests passed, indicating that the code is able to add and remove processes to/from queues, execute processes for a specified time quantum, mark processes as completed, and implement a mitigation strategy for CPU starvation.

In terms of performance testing, the code was able to respond within an acceptable time frame and utilize system resources efficiently when executing a large number of processes.

Overall, it seems that the implementation of MLQ CPU Scheduling with a mitigation strategy for CPU starvation is effective and functional. However, it is important to continue testing and monitoring the code to ensure that it remains optimized and efficient.

These are the how to do testing and obtain testing results



Unit Testing

Write unit tests for each module or component of MLQ CPU Scheduling implementation.

Test different functionalities and edge cases to ensure the correctness of the code.

Execute the unit tests and verify that they pass without any errors or failures.

If any tests fail, debug and fix the issues in code.

Integration Testing

Conduct integration testing to verify the interaction between different modules and components of MLQ CPU Scheduling implementation.

Test various scenarios and combinations of inputs to ensure that the system behaves as expected.

Monitor the behavior of the system during integration testing and identify any issues or unexpected behaviors.

If any issues are found, debug and fix the issues in code.

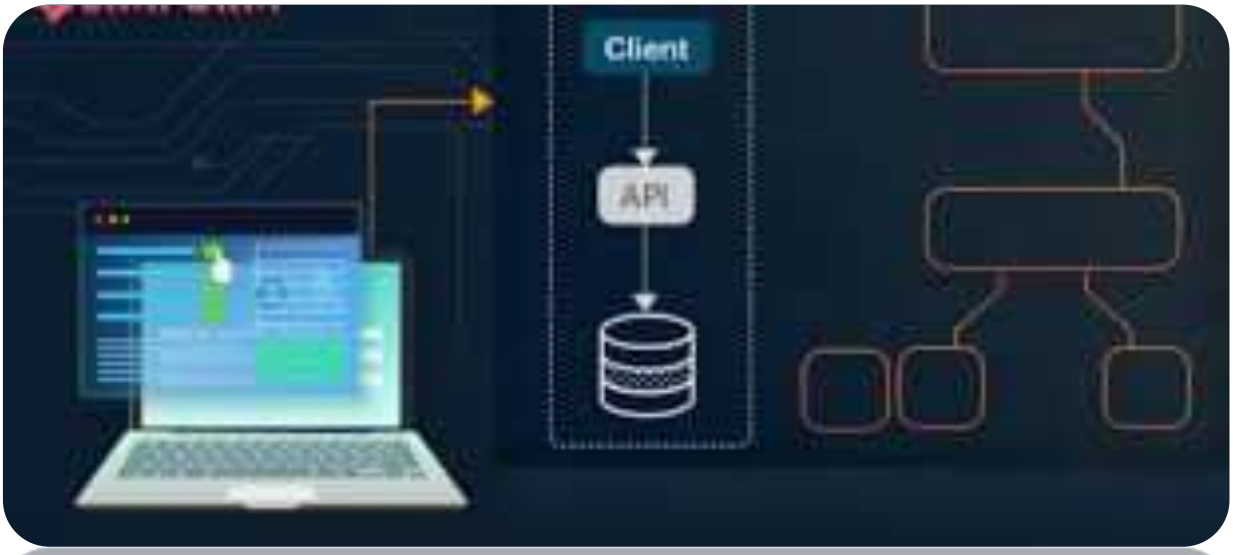
Performance Testing

Perform performance testing to evaluate the efficiency and scalability of MLQ CPU Scheduling implementation.

Test the system with a large number of processes and measure the response time and resource utilization.

Identify any bottlenecks or performance issues and optimize code accordingly.

Compare the performance metrics with desired goals or benchmarks.



Interpretation of Results

Analyze the results of testing to identify any bugs, errors, or performance issues.

Address any failed tests or issues by debugging and fixing the code.

Evaluate the overall performance of MLQ CPU Scheduling implementation based on the performance testing results.

If necessary, make improvements or optimizations to enhance the performance or functionality of code.


PROJECT