Mohammad Almasri (40248819)

Mamadou Diao Kaba (27070179)

Report

Process Class

Models an individual process in the system, encapsulating process attributes such as ID, arrival time, and execution time. It extends Thread, allowing each process to be executed concurrently.

Attributes: Include identifiers and timings (process ID, arrival time, and execution time) and flags for process status (whether it has started or executed at least once).

Methods:

- **execute(int executeTime, int currentTime):** Simulates the execution of the process for a specified quantum of time, updating its remaining execution time and printing relevant events (start, pause, resume, finish).
- **run():** Invokes the execute method, adhering to the Thread class contract, to simulate the process's execution lifecycle.

Scheduler Class

Orchestrates the scheduling and execution of multiple processes based on a hybrid approach that blends round-robin and shortest job first (SJF) principles.

Attributes: Track turnaround and waiting times, maintain a clock to simulate the passage of time, and hold a list of processes to be scheduled.

Methods:

- **run():** Implements the scheduling logic, dynamically choosing between round-robin and SJF based on process execution history. It calculates the execution quantum for each process, schedules the process execution, updates the clock, and manages process queues.
- **shortestJobFirst(ArrayList<Process> processes):** Identifies the next process to execute based on the shortest remaining execution time, applicable after each process has executed at least once.

ProcessScheduling Class

Serves as the entry point to the simulation, managing input/output operations and orchestrating the execution of the simulation.

Methods:

- **readProcesses(String inputFilePath):** Reads process data from an input file, instantiating and returning a list of Process objects.
- **runScheduler(Scheduler scheduler, ArrayList<Process> processes):** Starts the scheduler thread, thereby kicking off the simulation according to the scheduling logic defined in the Scheduler class.
- **main(String[] args):** Sets up the simulation environment, including process list initialization and output redirection, then starts the scheduler and handles the output.

Flow of the Program

1. Initialization: The ProcessScheduling.main method begins by reading process information from "input.txt", creating Process instances for each line in the file.
2. Scheduling Setup: A Scheduler instance is initialized with the list of processes. The scheduler manages the execution order of these processes, applying a hybrid scheduling algorithm that adapts between round-robin and SJF based on each process's execution history.
3. Execution: The scheduler thread is started, which in turn manages the execution of process threads according to the scheduling policy. The scheduler keeps track of the simulation's global time (clock) and dynamically decides the quantum for process execution.
4. Output Handling: Throughout the simulation, process start, pause, resume, and completion events are logged. The scheduler calculates waiting times and turnaround times, which are output at the simulation's conclusion.
5. Result Compilation: After all processes have been executed, the scheduler prints the waiting times for each process. The entire simulation's output is redirected to "output.txt", capturing the sequence of process executions and their respective waiting times.

Conclusion

Using threads to simulate processes and a scheduler provided a practical insight into concurrent programming and the complexities of CPU scheduling. The implementation accurately models the behavior of processes waiting for execution, being scheduled based on dynamic quantum calculations, and executing concurrently within a simulated CPU environment.

Advantages

- **Dynamic Quantum Calculation**: Unlike traditional round-robin scheduling, which uses a fixed quantum, this implementation adjusts the quantum dynamically based on the remaining execution time of processes, leading to potentially more efficient use of CPU time and reduced waiting times for shorter jobs.
- **Hybrid Scheduling Approach**: The combination of Round Robin for the initial execution and SJF for subsequent executions ensures fairness and efficiency, allowing short jobs to complete quickly while still providing CPU time to longer jobs.

Drawbacks

- **Complexity**: The dynamic quantum calculation and the hybrid scheduling policy introduce additional complexity compared to pure round-robin scheduling, potentially making the scheduler harder to implement and debug.
- **Potential for Starvation**: While the implementation aims to prevent starvation by ensuring each process is executed at least once with Round Robin scheduling, processes with significantly longer execution times might still experience delays, especially if many short jobs arrive continuously.

Comparison to Pure Round Robin

- **Efficiency**: The hybrid approach is potentially more efficient than pure round robin, as it can reduce the total waiting time by prioritizing shorter jobs after every process has been executed at least once.
- **Fairness**: Pure round-robin is inherently fair due to its cyclic nature but can lead to inefficient CPU usage with varying process execution times. The implemented approach tries to balance fairness with efficiency but may slightly favor shorter processes after the initial execution cycle.

In conclusion, this simulation project provides valuable insights into the challenges and strategies of process scheduling. The use of threading to simulate the concurrent execution of processes and a scheduler adds realism to the simulation, offering a hands-on experience with the complexities of scheduling algorithms in operating systems.