Computer Operating Systems

BLG 312E

Project 1 Report

Kemal Tahir Bıcılıoğlu
bicilioglu21@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 26.03.2025

1. Job Selection and Scheduling

Scheduler selects the next job based on the following criterias given in the homework description:

- **Priority:** Jobs with a lower numeric priority (higher priority) are chosen first.
- **Arrival Time:** Only jobs that have arrived (i.e., `arrival_time` $\leq$ simulation counter) are considered and is a candidate for the current job. If the priority of 2 jobs have equal priority, the one with arrived first is chosen
- **Remaining Time:** When 2 jobs have equal priority and arrival time, the one with the least remaining execution time is chosen.

To be able to prevent a job to use the time slices again and again the job is marked as last executed job after it is executed, the scheduler tracks the index of the last executed job. If the current candidate is the same as the last scheduled job and there is an alternative eligible job, the alternative is selected. This mechanism ensures that a job that was preempted does not immediately resume unless it is the only option.

2. Logging Implementation

The logging module writes all key events (forking, executing, preemption via SIGSTOP, resuming via SIGCONT, termination via SIGTERM, waiting for a job when no current eligible job) to the log file (`logs/scheduler.log`).

- Timestamps are generated from a simulation time counter (`g_current_time`) and formatted as HH:MM:SS.
- Each log entry records the event type, job name, and process ID.
- If any signal delivery fails (e.g., via `kill`), an error message is logged using the same logger function.

3. Error Handling

Each call to the `kill` function is checked for success:

- **SIGTERM vs. SIGKILL:**
  SIGTERM is used for termination of a job, allowing the process to clean up before exiting. SIGKILL is used to force an immediate shutdown, but in this implementation SIGTERM is preferred.
- If a signal fails to send (i.e., `kill` returns a negative value), an error log is recorded with the error description.

This ensures that any issues with signal delivery are properly documented for debugging purposes.

4. Answers to Evaluation Questions

4.1. Scheduling Fairness Analysis

The scheduler ensures fairness by considering each job’s arrival time, priority, and remaining execution time, and by avoiding immediate execution of a preempted job when alternatives exist.

Additional Strategies:

- *Aging:* Gradually increasing the priority of waiting jobs to prevent starvation.
- *Multi-Level Feedback Queues:* Dynamically adjusting priorities based on process behavior, which can enhance fairness between CPU-bound and I/O-bound processes.

4.2. Edge Cases and Failure Scenarios

Potential issues include:

- **Non-Responsive Processes:** A job might not respond to signals (e.g., SIGTERM). Our implementation logs an error if signal delivery fails.
- **Starvation:** Lower-priority jobs might never get scheduled if higher-priority jobs continuously occupy the CPU. Incorporating aging techniques can help alleviate this issue.
- **Deadlock:** Although not directly applicable in our simplified scheduler, more complex systems might encounter deadlock, requiring additional handling mechanisms.

Our implementation addresses these issues by:

- Logging any failures in signal execution, allowing for troubleshooting and potential fallback mechanisms (e.g., escalating from SIGTERM to SIGKILL if needed).
- Implementing a candidate selection logic that always chooses an alternative eligible job when available, mitigating the risk of starvation.