Computer Operating Systems - Homework Assignment

BLG312E - Spring 2025

Deadline: 26.03.2025 23.59

Academic Integrity and Plagiarism Warning

All work submitted must be your own. Plagiarism in copying code from other students is strictly prohibited and will result in severe academic penalties, including a zero for the assignment and potential disciplinary action. Making use of online sources or AI-generated content without proper understanding and modification will be treated likewise.
If you have any questions or require clarifications, please get in touch with me: **Mehmet Selahaddin Şentop (Email: sentop22@itu.edu.tr)**.

1 Overview

In this assignment, you will implement a preemptive priority-based process scheduler using `fork()`, `exec()`, and process control mechanisms such as `SIGSTOP` and `SIGCONT`. Your scheduler will:

- Implement a variant of Round Robin scheduling with priority.
- Consider arrival time to determine when a job enters the system.
- Assign jobs to priority queues based on their priority values.
- Preempt a process when its time quantum expires using `SIGSTOP`.
- Resume execution of paused jobs using `SIGCONT`.
- Run inside a Docker container.
- Record a screen capture of the execution to demonstrate correctness.

2 Implementation Details

2.1 Process Scheduling Algorithm

Your scheduler must follow these rules:

1. There will be a time quantum (defined in `jobs.txt`).
2. Jobs will have arrival time, priority, and execution time.
3. When a process starts, it runs until either:
   - Its execution completes within the time quantum, or
   - The time quantum expires, in which case it is preempted and paused with `SIGSTOP`.
4. If a process is preempted, the scheduler selects the next highest priority job (lower value means higher priority). It must be a job other than the one just preempted as long as there is another job. If there is no other, the same job will be selected.
5. If multiple jobs have the same priority, the earliest arriving job runs first. If they arrived at the same time, the one with less remaining execution time will run first. If they have the same remaining execution time, the one that was listed earlier in the input file will run first.
6. When a previously preempted process is scheduled again, it resumes with `SIGCONT`.

2.2 Process Creation and Execution

The scheduler will read a list of jobs from a file (`jobs.txt`) with the format:

```
<job_name> <arrival_time> <priority_level> <execution_time>
```

Additionally, a time slice value must be provided at the beginning of the file, indicating the time quantum for the Round Robin scheduler. (You may assume that the job name will be provided as a single token.)

Example input file (`jobs.txt`):

```
TimeSlice 3
jobA 0 1 6
jobB 2 2 9
jobC 4 1 4
```

This input file must be provided externally as `jobs.txt`.

The scheduler will `fork()` a child process for each job and use `exec()` to execute a separate program for each job.

3 Logging Requirements

You must create a log file named `scheduler.log` that records all process state changes. The log must follow the format:

```
[TIMESTAMP] [INFO] Forking new process for jobA
[TIMESTAMP] [INFO] Executing jobA (PID: XXXX) using exec
[TIMESTAMP] [INFO] JobA ran for 3 seconds. Time slice expired - Sending SIGSTOP
[TIMESTAMP] [INFO] Forking new process for jobB
[TIMESTAMP] [INFO] Executing jobB (PID: YYYY) using exec
[TIMESTAMP] [INFO] JobB ran for 3 seconds. Time slice expired - Sending SIGSTOP
[TIMESTAMP] [INFO] Resuming jobA (PID: XXXX) - SIGCONT
[TIMESTAMP] [INFO] JobA completed execution. Terminating (PID: XXXX)
[TIMESTAMP] [INFO] Forking new process for jobC
[TIMESTAMP] [INFO] Executing jobC (PID: ZZZZ) using exec
[TIMESTAMP] [INFO] JobC ran for 3 seconds. Time slice expired - Sending SIGSTOP
...
```

Each log entry must include:

- A timestamp (as given in the example output file).
- The event type (e.g., `INFO` or `ERROR`).
- The process action (forking, executing, pausing, resuming, jobX completed/ran for, etc.) and its PID.
- Information at the end if a signal like `SIGSTOP` or `SIGCONT` is sent.

4 Written Questions

As part of the evaluation, you must submit a separate file named `answers.pdf` answering the following questions:

- **Scheduling Fairness Analysis:** How does your scheduler ensure fairness among processes? What scheduling strategies could be used to improve fairness?
- **Edge Cases and Failure Scenarios:** What are some possible failure cases in your scheduler (e.g., a process not responding, starvation, deadlock)? How does your implementation handle these cases?

5 Evaluation Criteria

Your submission will be graded as follows:

Criteria	Points
Correct implementation of `fork()` and `exec()`	15
Proper use of `SIGSTOP`, `SIGCONT`	10
Multi-level scheduling logic	10
Logging execution details	10
Well-documented code	10
Screen recording of execution (showing correct behavior)	25
Scheduling fairness analysis	10
Explanation of edge cases and failures	10

6 Submission Instructions

Submit a single zip file named after your student number such as `150220000.zip`. The zip file must include

- Your well-documented source code along with a makefile
- A README file explaining how to run your program
- A report file `report.pdf` discussing your design decisions and findings, along with your answers for written questions in section 4
- The log file `scheduler.log`
- A **screen recording** of the execution in the command line interface. This recording will be reviewed to ensure correctness and adherence to the requirements.