ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 312E

Operating Systems Homework-1 Report

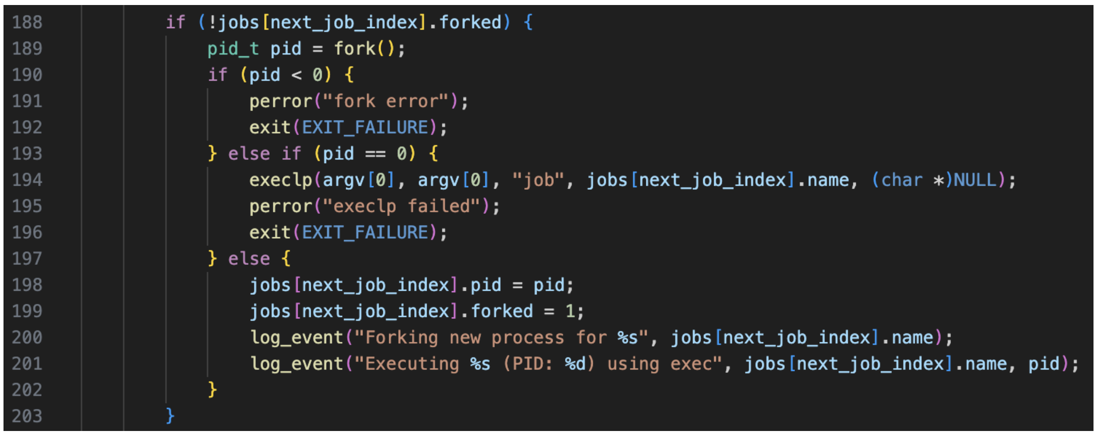
CRN	:	22610
LECTURER	:	Kemal Bıçakçı

820220330 - Ece Nil Kombak

SPRING 2025

1 Design Decisions and Observations

1.1 Code Overview



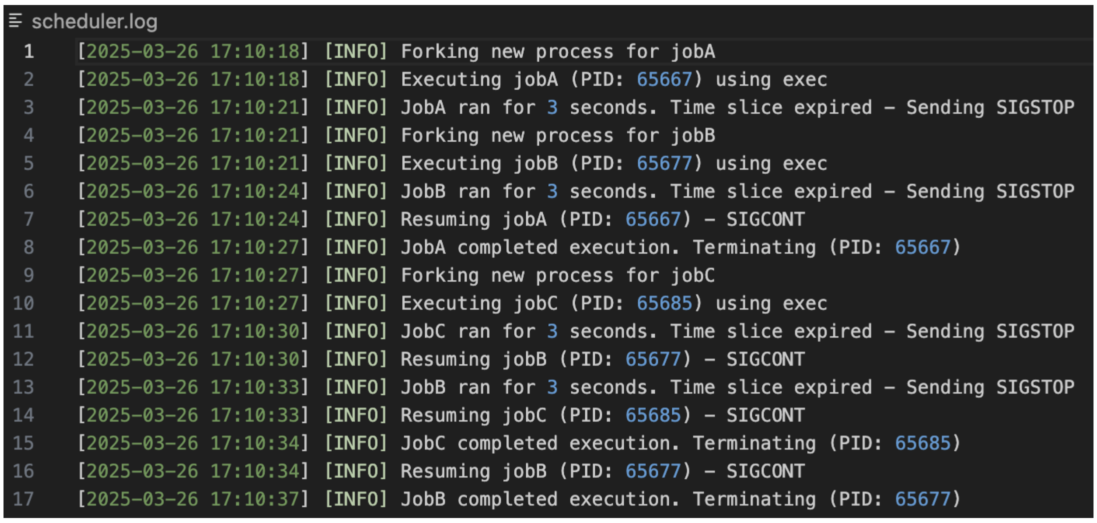
```c
        if (!jobs[next_job_index].forked) {
            pid_t pid = fork();
            if (pid < 0) {
                perror("fork error");
                exit(EXIT_FAILURE);
            } else if (pid == 0) {
                execlp(argv[0], argv[0], "job", jobs[next_job_index].name, (char *)NULL);
                perror("execlp failed");
                exit(EXIT_FAILURE);
            } else {
                jobs[next_job_index].pid = pid;
                jobs[next_job_index].forked = 1;
                log_event("Forking new process for %s", jobs[next_job_index].name);
                log_event("Executing %s (PID: %d) using exec", jobs[next_job_index].name, pid);
            }
        }
```

Figure 1: Use of fork() and exec()

The code is made up of a single executable that works in two modes as a scheduler and as a job, which makes it easier to deploy. When you run it without extra arguments, it reads a list of jobs from a file and uses a priority-based round-robin approach to schedule them, using fork() to create new processes and execlp() to run each job. The scheduler uses SIGSTOP to pause a job when its time slice is over and SIGCONT to resume it later, ensuring that every job gets a fair turn.
It selects the next job based on parameters like arrival time, priority, remaining execution time, and the order in the file, and it logs every important event such as forking, executing, pausing, resuming, and terminating with a timestamp in a log file. The code also checks for errors like failed fork() or exec() calls making sure that even unresponsive or slow jobs do not block others, thus handling potential edge cases and ensuring fairness in scheduling.

Running the code (video is also included in the zip file):https://youtu.be/mZXBOzBCjDY

1.2 Observations



```
scheduler.log
[2025-03-26 17:10:18] [INFO] Forking new process for jobA
[2025-03-26 17:10:18] [INFO] Executing jobA (PID: 65667) using exec
[2025-03-26 17:10:21] [INFO] JobA ran for 3 seconds. Time slice expired - Sending SIGSTOP
[2025-03-26 17:10:21] [INFO] Forking new process for jobB
[2025-03-26 17:10:21] [INFO] Executing jobB (PID: 65677) using exec
[2025-03-26 17:10:24] [INFO] JobB ran for 3 seconds. Time slice expired - Sending SIGSTOP
[2025-03-26 17:10:24] [INFO] Resuming jobA (PID: 65667) - SIGCONT
[2025-03-26 17:10:27] [INFO] JobA completed execution. Terminating (PID: 65667)
[2025-03-26 17:10:27] [INFO] Forking new process for jobC
[2025-03-26 17:10:27] [INFO] Executing jobC (PID: 65685) using exec
[2025-03-26 17:10:30] [INFO] JobC ran for 3 seconds. Time slice expired - Sending SIGSTOP
[2025-03-26 17:10:30] [INFO] Resuming jobB (PID: 65677) - SIGCONT
[2025-03-26 17:10:33] [INFO] JobB ran for 3 seconds. Time slice expired - Sending SIGSTOP
[2025-03-26 17:10:33] [INFO] Resuming jobC (PID: 65685) - SIGCONT
[2025-03-26 17:10:34] [INFO] JobC completed execution. Terminating (PID: 65685)
[2025-03-26 17:10:34] [INFO] Resuming jobB (PID: 65677) - SIGCONT
[2025-03-26 17:10:37] [INFO] JobB completed execution. Terminating (PID: 65677)
```

Figure 2: scheduler.log results

The scheduler manages jobs using a Round Robin algorithm with priority. It starts each job, stops it when the time slice is over and then resumes it later, ensuring every job gets a fair chance. This approach is good because it prevents one job from taking all the CPU time but it can also lead to many context switches if the time slice is too short. Overall, the scheduler balances fairness and responsiveness.

2 Written Questions

Scheduling Fairness Analysis

The scheduler makes sure that each job gets a fair turn by selecting the next job based on its priority, arrival time, remaining time, and the order in which it appears in the file. This approach prevents one job from using all the CPU time and helps prevent low-priority jobs from being ignored. We can improve fairness by increasing the priority of jobs that have been waiting for a long time.

Edge Cases and Failure Scenarios

Some problems that might happen are a process not responding, a job never being scheduled, or signals being handled incorrectly. The code fixes this by stopping a job when its time slice is over with SIGSTOP and then restarting it with SIGCONT so that no job runs forever. More error checking could be added to for reliability.