Simple Scheduler

Ishaan Agrawal and Aniket Gupta

January 11, 2024

1 GitHub Repository

GitHub Repository Link.

2 How to Run

1. Go to the appropriate directory (OS/Scheduler/)
2. Compile the shell using

```
gcc simple-scheduler.c
```

3. Start the scheduler using

```
./a.out <NCPU> <TSLICE>
```

4. To delete the executables

```
make clean
```

3 Implementation

3.1 Sample Files

Included are sample C files, p1.c, p2.c, p3.c, p4.c, p5.c as well as the Makefile, for testing purposes.

3.2 Program Overview

The scheduler program is designed to manage processes within a shell environment. It utilizes fork() to create child processes and employs a round-robin algorithm to allocate CPU time efficiently.

3.3 Process Abstraction

The scheduler employs a process abstraction, represented by a struct, to encapsulate relevant information about each process. This includes attributes like process ID, priority, execution status, wait time, and execution time.

3.4 Ready Queue Implementation

A ready queue is established using a priority heap data structure. This queue organizes processes based on priority and arrival time, ensuring that higher-priority or earlier-arriving processes are given precedence.

3.5 Time Slicing Mechanism

The scheduler employs a time-slicing approach, where each process is allocated a fixed time quantum for execution. A timer is used to enforce the time-slice duration, and upon expiration, a signal (SIGALARM) is sent to the scheduler. Once SIGALARM signal is sent, the scheduling algorithm is invoked.

3.6 Scheduling Algorithm

Within the timer signal handler, the scheduler executes its scheduling algorithm. This algorithm involves traversing the queue to identify terminated processes using the waitpid(pid, NULL, WNOHANG) system call. Terminated processes are removed from the queue and stored in a separate array.

3.7 Managing Running Processes

The scheduler also handles running processes by temporarily halting their execution, removing them from the queue, and subsequently resuming their execution.

3.8 Concurrency Control

To maximize system resources, the scheduler utilizes the SIGCONT signal to simultaneously restart a specified number of processes (NCPU) after the completion of a time slice.

3.9 User Termination Handling

The program accommodates user-initiated termination via the Ctrl-C signal (SIGINT). When received, the scheduler gracefully captures the signal, allowing it to print information about the terminated processes and proceed with normal execution.

3.10 Error Handling and Resilience

The program incorporates robust error handling mechanisms, including checks for system call failures and invalid inputs, to ensure stable and reliable operation.