OS Term Project #1

Prof. Eun-Seok Ryu

Project 1: **Simple scheduling**

Programming assignment #1 due by Nov. 20. 2024 (11:59pm) KST

1. Back ground: Round-Robin Scheduling

Round Robin (RR) scheduling algorithm is designed specifically for time-sharing systems. It is a preemptive version of first-come, first-served scheduling. Processes are dispatched in a first-in-first-out sequence, but each process can run for only a limited amount of time. This time interval is known as a time-slice or **quantum**. It is similar to FIFO scheduling, but preemption added to switches between processes.

2. Basic requirement for CPU scheduling

-Parent process :

Create 10 child processes from a parent process. Parent process schedules child processes according to the **round-robin scheduling policy**. Assume your own scheduling parameters: e.g. **time quantum**, and timer tick interval. Parent process periodically receives **ALARM signal by registering timer event**. Students may want to refer to **setitimer system call**. The ALARM signal serves as periodic timer interrupt (or time tick). The parent process maintains **run-queue and wait-queue**. Run-queue holds child processes that are ready state. Wait-queue holds child processes that are not in ready state. **The parent process performs scheduling** of its child processes: The parent process **accounts for the remaining time quantum of all the child processes**. The parent process gives time slice to the child process by **sending IPC message through msgq**. Students may want to refer to **msgget, msgsnd, msgrcv system calls**. Please note that there is **IPC_NOWAIT flag**. The parent process accounts for the waiting time of all the child processes.

-Child process :

A child process simulates the execution of a user process. Workload consists of **infinite loop of dynamic CPU-burst and I/O-burst.** The execution begins with two parameters: (cpu_burst, io_burst). Each value is **randomly generated**. When a user process receives the time slice from OS, the user process makes progress. To simulate this, the child process makes progress when it is in CPU-burst phase. Besides, the parent process sends **IPC message** to the currently running child process. **When the child process takes IPC message from msgq, it decreases CPU-burst value.**

3. Optional requirement for I/O involvement

-The child process :

Children makes I/O requests after CPU-burst. To simulate this, child accounts for the remaining CPU-burst. **If CPU-burst reaches to zero, the child sends IPC message to the parent process with the next I/O-burst time**.

-Parent process :

The parent process receives IPC message from a child process, it checks whether the child begins I/O-burst. Then, the scheduler takes the child process out of the run-queue, and moves the child process to the wait-queue. (so that the child cannot get scheduled until it finishes I/O) The parent process should remember I/O-burst value of the child process. Whenever time tick occurs, the parent process decreases the I/O-burst value. (for all the processes in the wait-queue) When the remaining I/O-burst value of a process reaches to zero, the parent process puts the child process back to the run-queue. (so that it can be scheduled after I/O completion) The scheduling is triggered by several events, for example: the expiry of time quantum (of a process), process makes I/O request (completing CPU-burst).

4. The output of the program : hard-copy report

Print out the scheduling operations in the **following format**: (at time t, process pid gets CPU time, remaining CPU-burst) run-queue dump, wait-queue dump. Print out all the operations to the following file: schedule_dump.txt. Students would like to refer to the following C-library function and system call: sprintf, open, write, close. All the processes should run at least for 1min. Print out the scheduling operations during (0 ~ 10,000) time ticks. (Note: C/C++ programming language is recommended, but is not limited to C/C++.)

Any question: TA: Jaeyeol Choi (jaychoi@skku.edu)