Report

b07902016 林義閔

Design

I stimulate the CPU scheduling by assigning priorities among child processes via the syscall `sched_setscheduler` (1 for processes that are not running, 99 for the process that is running).

The main process (scheduler) is running on CPU 0 (with highest priority) and all child processes are running on CPU 1.

For every time unit, the scheduler do the things as follow:

1. Push processes that are currently ready to the ready queue (processes are sorted by their ready time at first).
2. The scheduler behaves depending on the policy:
    1. FIFO:
        1. If there is no process running and ready queue (a FIFO queue) is not empty, pick the first ready process to run (got to 2.)
        2. If there is a process running, substract the executing time by 1.
    2. RR:
        1. If there is no process running and the ready queue (linked list) is not empty, pick a process to run (set the time quantum to 500 and got to 3.)
        2. If there is a job running and time quantum = 0, switch to next job if the ready queue is not empty (set the time quantum to 500 and got to 3.)
        3. If there is a process running, substract the executing time and time quantum by 1.
    3. SJF:
        1. If there is no process running and ready queue (a priority queue) is not empty, pick the shortest process to run (got to 2.)
        2. If there is a process running, substract the executing time by 1.
    4. PSJF:
        1. If there is no process running and ready queue (a priority queue) is not empty, pick the shortest process to run (got to 3.)
        2. If there is a process running and the shortest process in the ready queue is shorter than the running process, switch to the shortest process in the ready queue (go to 3.)
        3. If there is a process running, substract the executing time by 1.
3. If the running process is finished (executing time = 0), do some terminating routine (call `wait` and print time information) and prepare for next process to run. If all processes are finished, exit the scheduler process.

Kernel Version

`4.14.25`

Comparison

The order that scheduler runs all processes is identical to the theoretical order for all test cases. But for running time, there exists difference between the results and the theoretical results. There are possible reasons:

1. Context-switching is a pure overhead, expecially for user-space scheduler, it has to maintain some data structure (for ready queue) and make system calls to change the sheduling of the kernel (sometimes it even has to perform I/O operations). All of the above will directly increase the time needed for context switching, and thus cause the timing difference between the scheduler and child processes.
2. Since both scheduler and child processes are running in user space, we cannot control whether they are kicked out of CPU and thus causing the timing difference between the scheduler and child processes.