Operating System Project 1 Report

B04902012 劉瀚聲
B04902032 馬揚格
B04902040 王郁婷
B04902080 徐琮賀

1 Design

1.1 Main Structure

For each process, its attributes (ready time, execution time, start time and process id) are stored in a structure `processData`. A structure `processList` is constructed to maintain a list of `processData`, while processes in it are sorted by ready time.

The scheduler process `S` itself is limited to run on CPU 0 with lowest nice value -20 at the beginning. Once a child process `P` is forked, `P` will limit itself to run on CPU 1, and its nice value is determined by scheduling principle. After finishing setting these property, `P` then executes `./child`, a process that will run million empty iterations for n times, with n passed through `argv[1]`. To make `P` able to print it's own name, it's name is passed through `argv[2]`.

To schedule, `S` idles a process `P1` and awake another process `P2` by setting nice value of `P1` to 19 and setting nice value of `P2` to -20. Child processes won't compete with `S` for CPU resources because they are affined to different CPU.

1.2 FIFO

We construct two pointers `st` and `ed` pointing to processes in `processList`, both pointing to the first process at the beginning. `st` is maintained to point to the executing process (if exists), and `ed` is maintained to point to the first unforked process.

`S` checks if the process pointed by `ed` is ready every time unit. Once the child process is ready , `S` forks it, and `ed` moves right. `S` waits non-blockingly for the process pointed by `st` every time unit. Once the child process terminates, `st` moves right.

A process is awaken if:

(1) It is pointed by `st` and has been forked.

(2) It is forked and `st` and `ed` are pointing to the same process.

Actually, the processes between `st+1` and `ed-1` forms the ready queue.

1.3 Round Robin

We construct two pointers `st` and `ed` pointing to processes in `processList`, both pointing to the first process at the beginning. `st` is maintained to point to the executing process (if exists), and `ed` is maintained to point to the first unforked process, as in FIFO.

A counter `cnt` is recorded, starting at zero, and increases after each time unit. Once `cnt` reaches 500, `cnt` is set back to zero, `S` idles the currently executing process, and let `st` point to the next unfinished process and awake it. However, if the currently executing process terminates before `cnt` reaches 500, then `cnt` is also back to zero, and let `st` point the next unfinished process and awake it.

1.4 SJF and PSJF

We construct a pointer `ed` just like before. Also, an integer `st` stores the number of terminated process. Once a process terminates, `st` increases by 1.

Once we need to decide which process to awake, we choose the forked but unfinished process with shortest estimated remain execution time. The remain execution time of a process `P` is estimated by substracting the real executing time from the original declared executing time.

In SJF, we need to decide which process to awake when the currently executing process terminates, while in PSJF, we also need to decide whenever a new child process is forked.

2 Theoretical Result

We represent the result by listing n lines, while n is the number of child processes, one line for each process. A line includes the name, the starting time (units) and finishing time of each process. We briefly choose only the last testcase for each scheduling principle.

`FIFO_3.txt`:
P1 0 7999
P2 8000 12999
P3 13000 15999
P4 16000 16999
P5 17000 17999
P6 18000 18999
P6 19000 22999

`RR_3.txt`:
P3 4200 18199
P1 1200 20199
P2 2700 20699
P6 7200 28199
P5 6700 30199
P4 6200 31199

`SJF_3.txt`:
P1 100 3099
P4 3100 3109
P5 3110 3119
P6 3120 7119

P7 7120 11119

P2 11120 16119

P3 16120 23119

P8 23120 32119

`PSJF_3.txt`:

P2 500 999

P3 1000 1499

P4 1500 1999

P1 0 3499