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 nice value -15 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.

Start time of each process in each round `.startTime` is recorded. Once `t-p.startTime` reaches 500, while t is current time, `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 Result Comparison

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. The left column is the expected result, while the right column is the real result. Each result is represented by 3 string, name of processes, starting execution time, and finishing execution time. The time unit of expected results is the same as input, while the time unit of real results is nanosecond.

FIFO_3.txt:

```
P1 0 7999         | P1 1493065028.196616382 1493065044.205663952
P2 8000 12999     | P2 1493065028.669497706 1493065054.619804524
P3 13000 15999    | P3 1493065028.797498116 1493065061.119574122
P4 16000 16999    | P4 1493065028.998274853 1493065063.525177652
P5 17000 17999    | P5 1493065029.189938795 1493065065.956369779
P6 18000 18999    | P6 1493065029.285423430 1493065068.458171224
P6 19000 22999    | P6 1493065029.397933143 1493065076.564132067
```

RR_3.txt:

```
P3 4200 18199     | P3 1493065117.721537651 1493065162.698014685
P1 1200 20199     | P2 1493065115.421440165 1493065166.369149774
P2 2700 20699     | P1 1493065113.116603589 1493065170.6872949
P6 7200 28199     | P6 1493065121.953934253 1493065190.427482526
P5 6700 30199     | P5 1493065120.777512321 1493065193.764699366
P4 6200 31199     | P4 1493065120.13402939 1493065195.120458270
```

SJF_3.txt:

```
P1 100 3099       | P1 1493065299.596460952 1493065305.383947051
P4 3100 3109      | P4 1493065300.1547981 1493065305.400244388
```

```
P5 3110 3119        | P5 1493065300.73337982 1493065305.429877079
P6 3120 7119        | P6 1493065300.5338934 1493065313.97858228
P7 7120 11119       | P7 1493065300.213499687 1493065320.736832319
P2 11120 16119      | P2 1493065300.21832565 1493065330.299326131
P3 16120 23119      | P3 1493065300.69470871 1493065343.641767236
P8 23120 32119      | P8 1493065300.405467985 1493065360.742870254
```

PSJF_3.txt:

```
P2 500 999          | P2 1493065433.524447760 1493065434.506524721
P3 1000 1499        | P3 1493065434.508243367 1493065435.489155267
P4 1500 1999        | P4 1493065435.491027503 1493065436.469196007
P1 0 3499           | P1 1493065432.580471591 1493065439.738848908
```

We can see from the results shown above that execpt for RR scheduling, the other three scheduling have the same results of the finish sequence of the processes in our output and the theoretical result we computed.

The reason why the difference might exist is that in RR scheduling, the parent process switches the child processes each time the counter used to count the time units passed in the parent process exceeds the time quantum(500 in this case), while not knowing whether the same number of time units has passed in the child process or not. Consequently, we can not predict precisely when a process will end and thus not able to determine the finish sequence of the processes in the real RR scheduling.

3 Contribution Distribution

劉瀚聲：	Report: creating, calculating expected results
	Code: main function and structure, child process
馬揚格：	Code: implementing system call, FIFO scheduling
王郁婷：	Report: calculating expected results
	Code: RR scheduling
徐琮賀：	Code: SJF and PSJF scheduling