曾彬輝

B05902099

2020 OS Project 1 Report

1. Design

The Linux scheduler provides a FIFO policy, processes under which has precedence over normal processes. Naturally, our scheduler and its children run under this policy, so other processes on the computer can't interfere with our processes. Processes have different priorities under FIFO policy, and those with a lower priority can't run if other processes with higher priorities are ready to run. We exploit this property to perform a context switch. Our main scheduler runs the process with the highest priority first and also makes sure there's always only one child (assuming the scheduler has at least one child) as the second highest priority, and the other children are in the lowest priority. When the scheduler blocks, the child with the highest priority gets the chance to run.

Basically, I tried to implement SCHED_FIFO for FIFO, and SCHED_FIFO + SCHED_IDLE+SCHED_OTHER for RR, SJF and PSJF.

Design Ex: FIFO

Sort the ones needed to be executed and scheduled: according to their starting time, enter ready state. Start to calculate the time, and whenever it requires fork, fork it. Until all sub-programs over, the program ends.

System Call

There is only one system call, which is my_time.c

It is designed as follows:

```
asmlinkage int sys_my_time(unsigned long *s_sec, unsigned long *s_nsec,
                           unsigned long *e_sec,unsigned long *e_nsec,
                            int BE, int *pid)
```

BE is an integer to determine whether it is the beginning (1) or ending (0). When it is the beginning, record the starting time, else, record the end time.

`getnstimeofday()` is used to record the kernel time. We can use it to get the precision in nanosecond.

Running Result

I have tried to implement the scheduling algorithms introduced in class, and I will show the results I have in the demo video.

2. Kernel Version

The kernel version I am using is the same as the ones TA provided for HW1, which is 4.14.25.

3. Experiment Result

FIFO_1.txt

```
FIFO
5
P1 0 500
P2 0 500
P3 0 500
P4 0 500
P5 0 500
```

Result:

```
P1 13088
P2 13089
P3 13090
P4 13091
P5 13092

[Project1] 13088 1556267696.005955744 1556267697.447623058
[Project1] 13089 1556267697.448563280 1556267698.887764870
[Project1] 13090 1556267698.888653958 1556267700.375917190
[Project1] 13091 1556267700.377058738 1556267701.881777997
[Project1] 13092 1556267701.882504746 1556267703.341983999
```

FIFO_1 (total error ~ 0.41630197 secs)

The actual runtime, somehow shorter than the theoretical runtime.

Due to time limit, I only tried to analyze FIFO_1.txt

Conclusion

In conclusion, I found out that basically we can have 3 priorities:

- Running process: Highest priority
- Next ready/available process: 2nd Highest priority
- Other processes: Lowest priority