Operating System 2020 Fall

Project #1

b06902059 資工三 謝宜儒

Kernel Version

- Linux 4.14.25 : https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.14.25.tar.xz

Design

main.c

- Read the inputs (scheduling policy, process name, ready time and execution time).
- Start scheduling according to the given scheduling policy.

priority.c

- Implement two functions to raise or reduce the priority of a process
  - `raise_priority()` : use `sched_setscheduler` to raise the priority of a process, we use `SCHED_FIFO` as policy and the priority is given as a parameter.
  - `reduce_priority()` : use `sched_setscheduler` to reduce the priority of a process, we use `SCHED_FIFO` as a policy and the priority is set to 20 (a relatively low priority in our design).

process.h

- Like the following, define a structure `process` that stores the name, the ready time, the execution time, and the pid of a process.

```
struct process{
  char name[40];
  int ready_time;
  int exec_time;
  pid_t pid;
}
```

process.c

- `unit_time()`: a function to define one unit of execution time.
- Implement a function `assign_proc_to_cpu()`to assign a process to a particular CPU.

    - Use `sched_setaffinity()` to set processor affinity.
- Implement a function `exec_process()` to execute(fork) a process.
    - First, fork a child process.
    - In the child process, we record the start time and run `unit_time()` for several times (according to the process's execution time). Then, record the end time and print the start time and the end time to dmesg using `syscall()`. Finally, print process name and its pid and `exit(0)`.
    - In the parent process, we immediately reduce the priority of the child process using `reduce_priority()` to simulate the "ready" state of a process. Besides, using `assign_proc_to_cpu()`, we move the child process to a CPU different from the one that the scheduler runs. By doing so, we make sure that there is no preemption between the scheduler and the forked processes.

schedule.c

- First, we assign the scheduler to a particular CPU. This will later avoid preemption between the scheduler and the forked processes.
- Then, start scheduling:

  In a `while(1)` loop, we repeatedly do the following:
    - Step1 : Check if any running process has finished its execution.
        - If all the processes have finished their execution, break from the while loop.
    - Step2 : Check if any process is ready.
    - Step3 : Determine next process to be executed in the next unit of time.
    - Step4 : Check if there is any need for context switch (end the running process and start the next process).
        - Context switch is done by raising the priority of the to-be-executed process and simultaneously reducing the priority of the running process
    - Step5 : Run a unit time (`unit_time()`).
        - Also, add 1 to the variable `elapsed_time`.
- For Step3 in the above, we determine the next process to be executed in the next unit of time based on the scheduling policy we use.
    - FIFO (First In First Out)
        - If there is a running process p_i, then the next process will still be p_i; if not, then choose the process with the earliest ready time.
    - RR (Round Robin)
        - If there is a running process p_i, check if it has used up the time quantum. If the time quantum is not used up, the the next process will still be p_i. If it is used up, then the scheduler will search for the next process (ready and not finished yet).
        - If not, choose the next process in the ready queue.
    - SJF (Shortest Job First)
        - If there is a running process p_i, then the next process will still be p_i; if not, then choose the process with the shortest execution time.
    - PSJF (Preemptive Shortest Job First)
        - Choose the process with the shortest remaining time as the next process.

Comparison (Theoretical Result vs Pratical Result)

In the following, we denote the execution time as : (end time - start time)

We choose 2 testing data from each scheduling policy.

For pratical results, the unit of execution time is 1 second. For theoretical results, the unit of execution time is the unit time in the input files.

FIFO_1.txt

- Input

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

- Output

```
[Project1] 24180 1587558861.967880353 1587558862.729725502
[Project1] 24181 1587558862.730009100 1587558863.468883411
[Project1] 24182 1587558863.469069362 1587558864.207853938
[Project1] 24183 1587558864.208044528 1587558864.938805432
[Project1] 24184 1587558864.939135541 1587558865.721264123
```

	P1	**P2**	**P3**	**P4**	**P5**
execution time (practical)	0.7619	0.7388	0.7388	0.7308	0.7821
execution time (theoretical)	500	500	500	500	500

In this case, all processes should execute for a similar time span, i.e. 500 unit time. The first and the last process has relatively longer execution time than the other processes, which is against the theoretical result.

FIFO_2.txt

- Input

```
FIFO
4
P1 0 80000
P2 100 5000
P3 200 1000
P4 300 1000
```

- Output

```
[Project1] 24297 1587559284.664612123 1587559404.452109664
[Project1] 24298 1587559404.452321476 1587559411.887039864
[Project1] 24299 1587559411.887235810 1587559413.377834886
[Project1] 24300 1587559413.378023589 1587559414.867238165
```

	P1	P2	P3	P4
execution time (pratical)	119.7875	7.4347	1.4906	1.4892
execution time (theoretical)	80000	5000	1000	1000

In this case, the execution time of P1 should be 16 times as long as the one of P2. However, the pratical execution time of P1 is 119.7875 seconds, and the execution time of P2 is 7.4347 seconds, which is a little different from the theroretical result.

The execution time of P2 should be 5 times as long as the ones of P3 and P4. Observing the pratical results, they are much more similar with the theoretical result.

As a result, I infer that the longer execution time, the larger the error between pratical and theoretical result may be.

RR_3.txt

- Input

```
RR
6
P1 1200 5000
P2 2400 4000
P3 3600 3000
P4 4800 7000
P5 5200 6000
P6 5800 5000
```

- Output

```
[Project1] 2558 1586968816.387176188 1586968837.789551765
[Project1] 2556 1586968811.522355474 1586968840.812489968
[Project1] 2557 1586968813.956473475 1586968841.499935292
[Project1] 2561 1586968821.19661374 1586968852.697406869
[Project1] 2560 1586968820.245810367 1586968855.594832515
[Project1] 2559 1586968819.474868103 1586968856.928757901
```

	P1	P2	P3	P4	P5	P6
execution time (pratical)	29.2901	27.5435	21.4024	37.4539	35.3490	31.5008
execution time (theoretical)	19000	18000	14000	25000	23500	21000
theoretical/pratical	648.6833	653.5117	654.1322	667.4872	664.7995	666.6497

We can infer from the table above that the P4, P5 and P6 in pratical run faster than the P1, P2, and P3. It seems that processes with longer theoretical execution time will run faster in pratical.

RR_4.txt

- Input

```
RR
7
P1 0 8000
P2 200 5000
P3 300 3000
P4 400 1000
P5 500 1000
P6 500 1000
P7 600 4000
```

- Output

```
[Project1] 2674 1586969862.786953647 1586969868.985683996
[Project1] 2675 1586969863.576557322 1586969869.764674282
[Project1] 2676 1586969864.355139287 1586969870.543990502
[Project1] 2673 1586969862.15074687 1586969882.717831090
[Project1] 2677 1586969865.129044668 1586969888.136150984
[Project1] 2672 1586969861.244788574 1586969890.856739602
[Project1] 2671 1586969860.441708571 1586969895.167725751
```

	P1	P2	P3	P4	P5	P6	P7
execution time (practical)	34.7260	29.6120	20.5671	6.1987	6.1881	6.1888	23.0071
execution time (theoretical)	23000	19500	13500	4000	4000	4000	15000
theoretical/pratical	662.3279	658.5168	656.3881	645.2965	646.4019	646.3288	651.9726

The processes (P1, P2, P3, P7) with longer theoretical execution time tends to run faster in practical, which is a trend similar to the result of RR_3.txt.

SJF_1.txt

- Input

```
SJF
4
P1 0 7000
P2 0 2000
P3 100 1000
P4 200 4000
```

- Output

```
[Project1] 2928 1586971619.926980071 1586971623.26888338
[Project1] 2929 1586971623.58778734 1586971625.073113343
[Project1] 2930 1586971625.084054835 1586971631.272837087
[Project1] 2927 1586971631.28270778 1586971641.763874460
```

	P1	P2	P3	P4
execution time (pratical)	10.4811	3.3419	1.4861	6.1888
execution time (theoretical)	7000	2000	1000	4000
theoretical/pratical	667.8688	598.4619	672.9022	646.4124

As in the table, the value of theoretical/pratical varies greatly, especially in P2.

SJF_5.txt

- Input

```
SJF
4
P1 0 2000
P2 500 500
P3 1000 500
P4 1500 500
```

- Output

```
[Project1] 3130 1586972996.874638215 1586972999.936685044
[Project1] 3131 1586972999.936963734 1586973000.719077004
[Project1] 3132 1586973000.719245295 1586973001.482961753
[Project1] 3133 1586973001.483128276 1586973002.243242349
```

	P1	P2	P3	P4
execution time (pratical)	3.0620	0.7821	0.7637	0.7601
execution time (theoretical)	2000	500	500	500
theoretical/pratical	653.167	639.3044	654.7073	657.8081

As in the table, the value of theoretical/pratical is similar, meaning that the pratical result is close to that of the theoretical result. As for P2, it seems to run a little bit longer in pratical.

PSJF_3.txt

- Input

```
PSJF
4
P1 0 2000
P2 500 500
P3 1000 500
P4 1500 500
```

- Output

```
[Project1] 3314 1586974227.451414999 1586974228.221074497
[Project1] 3315 1586974228.225105556 1586974228.983605363
[Project1] 3316 1586974228.983930144 1586974229.744266496
[Project1] 3313 1586974226.653996871 1586974232.22801102
```

	P1	**P2**	**P3**	**P4**
execution time (pratical)	5.5741	0.7696	0.7585	0.7603
execution time (theoretical)	3500	500	500	500
theoretical/pratical	627.9040	649.6881	659.1957	657.6351

As in the table, the value of theoretical/pratical is similar, meaning that the pratical result is close to that of the theoretical result. As for P1, it seems to run a little bit longer in pratical. This may due to context switchtes that P1 suffers.

PSJF_5.txt

- Input

```
PSJF
5
P1 100 100
P2 100 4000
P3 200 200
P4 200 4000
P5 200 7000
```

- Output

```
[Project1] 3396 1586974609.269074084 1586974609.424117889
[Project1] 3398 1586974609.426825195 1586974609.731382439
[Project1] 3397 1586974609.424322239 1586974615.912395983
[Project1] 3399 1586974615.912572691 1586974622.95975939
[Project1] 3400 1586974622.96148607 1586974632.763010833
```

	P1	**P2**	**P3**	**P4**	**P5**
execution time (pratical)	0.1551	6.4889	0.3045	7.0472	9.8016

execution time (theoretical)	100	4000	200	4000	7000
theoretical / pratical	644.7453	616.4373	656.8144	567.6013	714.169

The value of theoretical/pratical varies greatly, especially in P4 and P5.

Conclusion

There are many possible reasons for the error between pratical and theoretical result:

1. Since we run the code on a virtual machine, there is potential unstability.
2. The way we choose the next process is to iteratively check each process and pick one from the ready queue, so the number of processes may affect the scheduling time.
3. There may be other processes on the machine, so the processes we create may suffer from context switch overhead.
4. The timer of the scheduler will probably not be sychronized as the one on the forked processes, hence resulting in the errors.
5. Even if runnig same number of loops (say 1000000), different process may have different execution time.