OS project 1

b07902047 羅啓帆

April 2020

1. Design

- Priorities
  - scheduler: 2
  - running task: 3
  - waiting task: 1
- Helper functions (in `utils.h`)
  - `one_time_unit()`: wait one time unit and do nothing
  - `set_priority(Task *task, int priority)`: set the priority of `*task`
  - `create(Task *task)`: create a child process for `*task`
  - `execute(Task *task, int duration)`: execute `*task` for `duration` time units
  - `clear(Task *task)`: `wait` the child process related to `*task` to prevent zombies process
- Workflow

  Program `main` will first setup the environment (share memory, cpu mask, priority and scheduler), then read the input data. Then, base on the input, it will first determine which schedule scheme it should use and then execute it. Each schedule scheme will calculate when to run each process and how long should it run. Finally, after calculation, it will run the command one by one, using the helper functions mentioned above.
- Details
  - `main.c`
    - use `sched_setaffinity` to run on one cpu only
    - use `sched_setscheduler` to set the scheduler type (`SCHED_FIFO`)
    - use `sched_setparam` to set process priority
  - `fifo.c`
    - sort task by their arrive time
    - create task (fork child process)
    - check if other tasks will arrive when executing this task
      - if yes, then the execution of the task must be interrupted to create process for other tasks
  - `sjf.c`
    - sort all task by their arrive time
    - when choosing new task to execute, sort **arrived task** by their total running time and choose the smallest
    - then, use similar methods described in `fifo.c`
  - `psjf.c`
    - similar to `sjf.c`, the only difference is that we need to consider the **currently running** task when determining the next running task
  - `rr.c`
    - maintain a queue with two pointers

2. Kernel version & Testing platform

- Kernel version: 4.15.4
- Testing platform: Ubuntu 16.04LTS on 8-Core Intel Core i9 @ 2.3 GHz

3. Comparison between theoretical and experimental results

- One time unit: 0.003418456 sec (tested with TIME_MEASUREMENT.txt, 5000 units takes 17.09228 sec)
- Results

deviation(%)	1	2	3	4	5
FIFO	0.18059	1.2897	0.61584	2.8392	0.80962
SJF	1.5483	0.68046	1.6274	0.76532	0.35877
PSJF	1.3616	0.51578	1.7098	1.1181	1.6579
RR	3.0114	2.0954	1.6217	0.93182	1.1933

real(s)	1	2	3	4	5
FIFO	8.53071	293.57000	79.10869	11.24964	79.26104
SJF	48.59939	52.65827	111.24032	37.89080	12.00752
PSJF	86.62507	37.40907	12.16917	48.39348	53.16948
RR	8.80350	31.41078	104.21679	79.35713	79.56275

theory(s)	1	2	3	4	5
FIFO	8.54614	297.40567	78.62449	10.93906	78.62449
SJF	47.85838	52.30238	109.45896	37.60302	11.96460
PSJF	85.46140	37.60302	11.96460	47.85838	52.30238
RR	8.54614	30.76610	102.55368	78.62449	78.62449

- Analysis
  - Almost all *real time* are close to but slightly larger than *theory time*
- Reason for deviations
  - Other process might use the CPU when we're testing, causing the real time larger then theory time
  - Time spent on context switches is not counted in theory time, but is counted in real time
  - The speed of CPU may be different between TIME_MEASUREMENT and other input files

References

[1] Linux Documentation: `https://linux.die.net`

[2] Linux Kernel 4.x: `https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/`

[3] Add Custom System Call: Adding a Hello World System Call to Linux Kernel