B07902112
陳功阜

Project 1

CSIE 3310
April 29, 2020

Design

- All tests are done on Ubuntu 16.04 server running linux kernel version 4.14.25.
- For all four policies, we have (1) a main process running on CPU 0 with `maxPriority`, reading the input, spawning and scheduling the child processes, and (2) the child processes all running on CPU 1.
- The main process sorts the input of the child processes by their ready time in ascending order, then by their input orders.
- The main process spawns all child processes at a time. The child processes sets the CPU affinity, and writes to the parent via a pipe. The main process starts scheduling the child processes if all child processes have written to the parent.
- After the child process writes to the parent, the child process reads from the parent via another pipe, causing it to block. The main process can then start execution of the child process by writing to the pipe.
- After the child process is started, it executes the unit-of-time loop n times, where n is the specified execution time.
- If there is currently no processes to schedule, the main process may wait by executing the unit-of-time loop, or by calling `waitpid` on a given child process.

The *policy* used by `sched_setscheduler` and the *priority* assigned to each child depends on the input policy:

FIFO

- The chosen *policy* is `SCHED_FIFO`.
- All child processes are assigned `maxPriority` since it doesn't matter.
- The main process waits to start the next process in two ways:
  - If no processes were spawned yet, the main process executes the unit-of-time loop until the 0-th process is ready and then starts it.
  - Otherwise, the main process calls `waitpid` on the previously started process, then starts the current process as soon as `waitpid` returns.
- The main process then starts the next child process in order.

RR

- The chosen *policy* is `SCHED_RR`.
- Whenever processes are ready, they are added to back of the queue.
- A child process will be assigned `maxPrority` if it is asked to execute, and assigned `maxPriority` − 1 if it is asked to preempt.
- The main process waits to start the next process in three ways:
  - If no processes were spawned yet, the main process executes the unit-of-time loop until the 0-th process is ready and then starts it.
  - Otherwise, if the previously started process has execution time more than 500 time units, the main process executes the unit-of-time loop 500 times, preempts the previously started child process, subtract its execution time by 500 and adds it to the end of the queue.
  - Finally, if the previously started process has execution time less than or equal to 500 time units, the main process calls `waitpid` on the previously started process.
- The main process will then choose the next process by removing it from the front of the queue.

SJF

- The chosen *policy* is `SCHED_FIFO`.
- All child processes are assigned `maxPriority` since it doesn't matter.
- The main process waits to start the next process in the same ways as FIFO.
- The main process will consider processes that are ready but not yet started, and the one with the minimum execution time will be chosen to start.

PSJF

- The chosen *policy* is `SCHED_RR`.
- A child process will be assigned *priority* the same ways as RR.
- The main process waits to start the next process in two ways:
  - If no process were spawned yet or if new processes will be ready before the previously started process finishes execution, the main process will execute the unit-of-time loop until the new processes are ready.

  - Otherwise, the main process calls `waitpid` on the previously started process.

- The main process will consider processes that are ready but not yet finished, and the one with the minimum execution time will be chosen to start.
- If the chosen process differs from the previously started process, the previously started process will be preempted and the newly chosen process will be started.

Results

Comparison

- The start/finish below is calculated by dividing the time by the unit-of-time acquired from a separate run of TIME_MEASUREMENT.txt then rounding to the nearest integer for convenience (the conversion script `convert` is in the project folder).
- Note that the start/finish time is relative to the starting time of the first child process since the starting time of the main process isn't printed and thus can't be calculated.
- The difference is calculated as sum of the absolute difference between the actual and theoretical results.
- We will show an example for each policy. The converted results of all tests can be seen by running `./convert`, and the theoretical result can be acquired by running `./theory test/TEST_NAME.txt`.

FIFO

Process	Start	Start (theoretical)	Finish	Finish (theoretical)	Difference
P1	0	0	515	500	15
P2	515	500	1016	1000	31
P3	1016	1000	1551	1500	67
P4	1551	1500	2053	2000	104
P5	2053	2000	2554	2500	107

Table 1: Results of FIFO_1.txt

RR

Process	Start	Start (theoretical)	Finish	Finish (theoretical)	Difference
P1	0	0	18499	18500	1
P2	1497	1500	19020	19000	23
P3	2991	3000	16973	17000	36
P4	4986	5000	30088	30000	102
P5	5485	5500	29056	29000	71
P6	6981	7000	27029	27000	48

Table 2: Results of RR_3.txt

SJF

Process	Start	Start (theoretical)	Finish	Finish (theoretical)	Difference
P1	0	0	3007	3000	7
P2	3007	3000	4011	4000	18
P3	4011	4000	8023	8000	34
P4	9025	9000	11025	11000	50
P5	8023	8000	9025	9000	48

Table 3: Results of SJF_4.txt

PSJF

Process	Start	Start (theoretical)	Finish	Finish (theoretical)	Difference
P1	0	0	4004	4000	4
P2	1011	1000	2010	2000	21
P3	4004	4000	11009	11000	13
P4	5015	5000	7014	7000	29
P5	7015	7000	8015	8000	30

Table 4: Results of PSJF_2.txt

Conclusion

The results are all correct in terms of the order the processes start/finish, but there are still some minor difference between the actual and theoretical result. We can look at the various variables that may cause the difference shown:

My Scheduler

One may guess that the scheduler is the biggest factor, but if that is the case, then the more priority assignment and management the scheduler has to do, the more difference there should be. Therefore, one would expect policies like RR and PSJF to have a more significant difference, but it is in fact FIFO and SJF that has a greater difference, suggesting that the scheduler might not be the bottleneck.

Virtual Machine

The tests were done on a VirtualBox, which means that the guest OS running in the virtual machine is just another process on the host OS. Therefore, the guest OS can be context switched and it would delay the execution of all processes in the guest OS. Furthermore, because of load balancing, the virtual machine is constantly being switched between different CPUs, which might yield a performance penalty as well.

Cache Behavior

This can be combined with the previous point, since the virtual machine is constantly being switched between different CPUs, there may be cache misses during these switches. Also, some of the test results were faster than the theoretical results, suggesting that memory access patterns and cache behavior may be at play here.