Assignment 2

Operating Systems CS3305B

Albirawi, Zaid
250626065

March 6th, 2015

1. **Nice Values – Part 1**: On the assignment web page you will find a program called *cpuTimeWaste.c*. By default the nice value assigned to *cpuTimeWaste.c* is 0. You will study the effect of the *nice* command by running instances of *cpuTimeWaste.c* with different *nice* values. You can use the nice command at the command line to assign a nice value e.g.,

   nice –n 5 cpuTimeWaste

   You should use these values of nice values 0, 5, 10, 15, 19. You can use the *top* command to see the impact on the percentage of CPU used. Is there a wide variation of CPU utilization as the nice value increases? Provide an explanation for the behaviour seen.

   When running the *cpuTimeWaste.c* instances separately.

Nice Value	0	5	10	15	19
CPU usage in %	99.5	99.4	99.7	99.3	99.0

   No, there is no wide variation of CPU utilization as the nice value increases because the processes are not being challenged for processing power. No big processes are running at the moment, therefore, *cpuTimeWaste.c* will have mostly have the processor to itself.

   When running all the *cpuTimeWaste.c* instances at the same time.

Nice Value	0	5	10	15	19
CPU usage in %	67.2	21.9	7.0	2.3	1.0

   Yes, as the nice value increases, the CPU usage percentage decreases. This is because the processes with lower nice value numbers take precedence over the processes with higher nice values. Hence get more time with the CPU and utilize it more.

2. **Nice Values – Part 2:** Assume that you start two instances of *cpuTimeWaste.c* with the default nice value of 0. Both processes should have similar values for the percentage of CPU used. Assume that one of the processes will have the default nice value.

- What should the nice value be of the other process to ensure that the process with the default nice value receives approximately 75% of the CPU?

  $1.25^x = \frac{0.75}{0.25}$

  $1.25^x = 3$

  $log\,1.25^x = log\,3$

  $x * log\,1.25 = log\,3$

  $x = \frac{log\,3}{log\,1.25}$

  $x \approx 5$

  Nice value should approximately be 5.

- What should the nice value be of the other process to ensure that the process with the default nice value receives approximately 90% of the CPU?

  $1.25^x = \frac{0.9}{0.1}$

  $1.25^x = 9$

  $log\,1.25^x = log\,9$

  $x * log\,1.25 = log\,9$

  $x = \frac{log\,9}{log\,1.25}$

  $x \approx 10$

  Nice value should approximately be 10.

3. **Impact of Scheduling Policies on Interactive Programs – Part 1:** You will find on the assignment web page a program (*cpuTimeWasteFIFO.c*) that modifies *cpuTimeWaste.c* such that it changes the default scheduling class. Instead of SCHED_OTHER (which indicates CFS) it uses SCHED_FIFO. You will see that the code uses the sched_setscheduler system call. Now try running an interactive program, e.g. token.c from assignment 1, and/or several Linux commands. Please describe your observations and explain the behaviour you observed.

   The whole shell became very laggy. The real time processes don't precedence over the CFS processes anymore, therefore, the RT processes will now have to wait in line with the CFS processes. This is because both RT and CFS processes are running on the same scheduling policy, **F**irst **I**n **F**irst **O**ut.

4. **Impact of Scheduling Policies on Interactive Programs – Part 2:** Modify the code in token.c (call it tokenRR.c) so that it uses the sched_setscheduler system call to set the scheduling policy to SCHED_RR. Start *cpuTimeWasteFIFO.c* and tokenRR.c. Use different orders i.e., start cpuTimeWasteFifo.c before tokenRR.c and vice-versa. Please describe your observations and explain the behaviour you observed.

   Same behaviour as the previous question, the shell became very laggy. However, now the token process does not print until the *cpuTimeWasteFIFO.c* terminates. No matter what the order was, the *cpuTimeWasteFIFO* processes took precendce over the *tokenRR* process. This is because the *cpuTimeWasteFIFO* runs the first in first out policy which means that the *cpuTimeWasteFIFO* will run until its done. On the other hand, the *tokenRR* process runs the round robin policy which means that the process will run until its time slice is expired. Therefore, if we start the *tokenRR* processes before the *cpuTimeWasteFIFO* process the result will be identical to running *cpuTimeWasteFIFO* before *tokenRR* after the first time slice expires. Hence, the FIFO policy has precedence over the RR policy.

5. **Impact of Scheduling Policies for CPU-Bound Policies:** Please find on the assignment web page the program *pi.c*. This program computes the value of pi using a monte carlo method. This method requires as input the number of iterations. A higher number of iterations results in higher accuracy.

   You are to modify this program such that it also takes as input the number of child processes to be created. Each child process is to run the code for calculating pi. Assuming that you use a loop with index i then if i is even then SCHED_OTHER is to be used otherwise SCHED_FIFO is to be used. When a child has finished calculating pi then it should print out the elapsed time, which should be calculated as the time between the start of the pi calculation to the end of the calculation. A simple program that calculates the elapsed time (elapsedTime.c) is on the assignment web page.

   You should investigate the behaviour of the program you write for different values of the number of iterations and the number of child processes. What can you conclude from the results?

   This confirms what we already discussed above, the processes that have the schedule policy "SCHED_FIFO" will run faster than the the processes that have the schedule policy "SCHED_OTHER". This is because SCHED_FIFO processes have precedence over the SCHED_OTHER processes.

6. Conclusion

   - Provide an example of a type of application for which each scheduling class is well suited.

     SCHED_OTHER: Video editing applications.
     SCHED_RR: Background applications.
     SCHED_FIFO: Interactive applications.

   - Provide an example of a type of application for which each scheduling class is not well suited.

     SCHED_OTHER: Interactive applications.
     SCHED_RR: Interactive applications.
     SCHED_FIFO: Video editing applications.