Gurpreet Singh

CS3305 Assingment 2

1.

Nice value	CPU usage
-5	67%
0	22%
5	7.3%
10	2.0%
15	0.7%
19	0.7%

The nice command sets the priority of the process. The above table shows us that processes with a lower nice value will have higher priority on the CPU core and get more processing time.

2.

In order to ensure that the process with the default nice value recieves approximately 10% of the CPU, the new process’s nice value should be -10

In order to ensure that the process with the default nice value recieves approximately 95% of the CPU, the new process’s nice value should be 15

3.

Elapsed time for minimal function call is: 1123

Elapsed time for a system call is: 3692

My compare.c program uses the diff method shown in measuretime.c to first measure the time difference of running minimalFunctionCall(), an empty function with no parameters, and then measure the time difference of running getpid(). The results are displayed above illistrating that the minimal function call is faster than the getpid system call

4.

Processor #1:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Processor #2:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Processor #3:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Processor #4:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Processor #5:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Processor #6:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Processor #7:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Processor #8:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Processor #9:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Processor #10:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Processor #11:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Processor #12:
vendor_id : GenuineIntel

model name : Intel(R) Core(TM) i7 CPU 980 @ 3.33GHz

Version: Linux version 4.4.1-2-ARCH (builduser@foutrelis) (gcc version 5.3.0 (GCC)) #1 SMP PREEMPT Wed Feb 3 13:12:33 UTC 2016

5. This program (observer.c) creates a fork and uses the parent to monitor the child process. Every second the /proc/[pid]/stat file is read and scanned for the 14th and 15th stat value which returns time spent in user mode and kernel mode. I use a signal handler to check if the process has been killed and terminate the parent when that happens.

6.

Experiments:

Running two cpuTimeWaste and comparison1 with RR/FIFO/OTHER and viewed results with top.

I realized there was no difference with other processes and discovered that Real Time Scheduling processes are only effected by their scheduling policy when interacting with each other.

Running multiple RR/FIFO/OTHER processes on the same core and viewing results with top

In this experiment I could accurately make conclusions about all the different scheduling policies.

Scheduling Policy	Conclusion
SCHED_RR	When multiple processes were run with the RR scheduling policy, I noticed that each process is sharing an equal time using the CPU core's processing power. When 3 processes were run many times each process would be using 33% of the CPU.Therefore this experiement eccurately showed RoundRobin can be preempted by a time quantum.
SCHED_FIFO	When multiple processes were run with the FIFO scheduling policy, I noticed that each process was waiting at 0% usage until the other processes in the queue we're finished until it ran. Therefore this experiment accurately showed FIFO cant be preempted unless another process of higher priority is present in the execution queue.
SCHED_OTHER	When multiple processes were run with the OTHER (default) scheduling policy I noticed similar behaviour to the RR scheduling policy except OTHER was more reactive to other tasks running on the system. When run with cpuTimeWaste (without a real time scheduling policy) I realized it was given an equal priority and the cpu time was being equally shared.

7.

Experiements : Same as Question 6

In all scheduling policies I noticed the same behaviour as reported in question 6 but very little to none CPU usage because the function was IO intensive so the scheduling policy did not have any effect on how the processes were ordered of processing.