Chris Sterling

PA 3

The Linux Scheduler: A Benchmarked Solution

Abstract

When testing 27 different combinations of scheduling policies, process types, and policy scales, the SCHED_OTHER policy was the only policy that successfully scaled up to the HIGH level, however it was a fairly greedy and resource intensive scheduler. The I/O Bound processes seemed to run much faster than the CPU Bound/Mixed Processes.

Introduction

This benchmarking program was designed to test 27 different dimensions of optimization. There are three types of schedulers (SCHED_OTHER, SCHED_FIFO, SCHED_RR), three types of process types (CPU Bound, I/O Bound, Mixed), and three policy scales (Low, Medium, and High). If you look at all of these combinations in separate cases, then it is clear that depending on the application and the priority level, the best scheduling policy is variable.

Method

I based my CPU-bound code off of the pi-sched.c file that was originally supplied to us. I used a for-loop to spawn multiple forked processes.

I based my I/O-bound code off of the rw.c file. So that I didn't have to deal with concurrency issues and get accurate readings, I pull information out of "/dev/urandom" and into multiple rwinput-# files before the rw program starts. That way random information is in a static file and isn't a part of the time calculation. The program pulls information out of a the junk file and then writes information out to a new junk file.

I based my Mixed code off of mainly the pi-sched.c file. I calculate pi the appropriate number of iterations, and then write it out to a file. Once again, so I don't have to deal with concurrency issues, I write out the results of the pi calculation to pilog-# different files.

I used the "testscript" file as a wrapper program to call each program multiple times while varying the arguments for the number of simultaneous processes and the type of scheduling process. All of my results were output to a *.CSV file for easy analysis later.

Results & Analysis

The SCHED_FIFO scheduler type works better in a I/O BOUND process type. The CPU and mixed process type perform almost exactly the same. The SCHED_FIFO scheduler obviously doesn't scale

though, because every single time the program ran with 100 concurrent processes, the program would crash.

The SCHED_RR scheduler also doesn't scale well, as is apparent when 100 processes are running concurrently and the program crashes. This scheduler runs about a second (on average) faster than the SCHED_FIFO scheduler in all cases. This is a little more processor intensive than the FIFO scheduler.

The SCHED_OTHER scheduler was the only scheduler that I was able to get to scale. It did take a large amount of time to complete, but it did actually complete when run with multiple processes.

Overall, the best time seemed to come from the SCHED_OTHER scheduler, then the second best time came from the SCHED_RR scheduler, then SCHED_FIFO scheduler came last in time.

The best CPU performance on the other hand came from the SCHED_FIFO protocol, then the SCHED_RR scheduler, then the SCHED_OTHER protocol.

Conclusion

While the SCHED_OTHER scheduler ran faster timewise than the other scheduling policies, it was the most greedy protocol. It took up more clock cycles in general than the other two protocols. It is obvious at this point why the SCHED_OTHER policy is the default policy. It will always work. It may not be the optimal solution, but it completes the program and gets us to the finish line.