Lab 3: Real-time Scheduling

Welcome to the 3rd lab of the VE473 Advanced Embedded System course! Today we have two goals. First, we will write programs that run under the SCHED_RR (and advanced topic SCHED_FIFO) real-time scheduler. Next, we will use the kernel tracer (which the lecturer showed in class) to examine how this program runs.

Project Description and Exercise

First, we'll need a program to run as a real-time workload. We can generate a computation-intensive task through a while (or for) loop. It increments a loop index from zero to five hundred million (500,000,000) and (2) the body of the loop performs a simple one-line arithmetic calculation involving addition, multiplication, and assignment. This will provide a CPU bound task that's still computationally intensive but is guaranteed to finish. Your program should read in one input parameter to indicate on which core the program should run.

NOTE: Modern compilers are smart enough to optimize your loop away. Make sure to compile without any optimizations when you build your executable file for this exercise. With compiled with no optimization (e.g., simply running gcc -o program_name program_name.c), on the Raspberry Pi, the instructor's program runs for a few seconds. When compiled with -O1 it runs for much shorter. For any higher optimization level (e.g., -O2 or above) the program runs in several milliseconds. Compile your new program on your Raspberry Pi, without any optimizations, and then use the time command to verify that your program runs for some number of seconds.

Second, Now, use the trace-cmd command to record sched_switch events during an execution of your program on core 0. Recall that the syntax for this looks like the following example: "sudo trace-cmd record -e sched_switch ./program_name 0". Where 0 is the input parameter to the program. Note: Optionally, you can try this on other cores as well, to see to what extent running your program on different cores changes the set of processes that may preempt your program, but for the answers in this studio please use core 0. Make a copy of the trace.dat trace file that command generated (in the directory where you ran it), so you can inspect it later - the next time we run trace-cmd it will overwrite the current local trace.dat file. Then use Kernelshark to inspect the trace, and zoom in on specific portions of the trace (e.g., on CPU core 0 where you pinned the program) to examine how your program executes.

Third, Please run the following commands:

man 2 sched_setscheduler

and

man 2 sched_get_priority_min

and

man 2 sched_get_priority_max

to see documentation for the data structure and functions needed for this exercise. The documentation also helps you enhance your knowledge learned from class.

Modify your workload program so that it takes a second command line argument (in addition to the first argument, which specifies on which core the program should run) that gives the real-time priority with which it should run (valid values for that argument are in the range from the value returned by sched_get inclusive). Also modify your program so that it uses the sched_param data structure and the sched_setscheduler() function to run under the SCHED_RR scheduling class with the real-time priority that was passed in the second command line argument. Be sure your program checks that valid command line parameters were passed (in particular, that the passed priority is in the range from the value returned by sched_get_priority_min(SCHED_RR) to value returned by sched_get_priority_max(SCHED_RR) inclusive), and also checks the return value from calling the sched_setscheduler() function, and that it prints out an appropriate error message (and returns a negative value) in the event of failure.

Compile your program (without optimization) and run it both as root (using sudo) and not as root with different priority values ranging from 1 to 99. Find the largest number for which sched_setscheduler() succeeds when run as root, and the largest number for which sched_setscheduler() succeeds when run not as root.

Fourth, run your program on core 0 under trace-cmd with a real-time priority of 1, to generate a new trace, and inspect the resulting trace in Kernelshark.

Fifth, in kernelshark, filter your real-time trace to only show events from the processor that your program used. To do so, go to the Filter menu and select list CPUs, and then uncheck all but that CPU and then click on the Apply button. Do that for each of the CPUs.

Sixth, use the command ps -e -o cmd,rtprio to get a list of all processes on the system and their real-time priorities. A dash in the priority column means that this process does not have a real-time priority.

Seventh, run your program again on core 0 under trace-cmd (as root) with a real-time priority of 99. As the answer to this exercise,

Submission to each exercise:

0. List the names of the people who worked together on this lab.
1. Please show the output from running your new program (compiled without any optimizations) within the time command.
2. Please name three processes that interefered with the execution of your program on that core, and explain briefly how you know they did that based on the trace you examined.
3. Find the largest number for which sched_setscheduler() succeeds when run as root, and the largest number for which sched_setscheduler() succeeds when run not as root, and report those values as the answer to this exercise.
4. Please state whether any processes preempted your program (and if so which ones), and whether there are any meaningful differences in how these interruptions appear in this trace (kernelshark trace.dat), versus in your original (non-real-time) trace (kernelshark trace_original.dat).
5. Please state how many sched_switch events were recorded on CPU core 0 where your program ran, and compare that number to the number of sched_switch events were recorded on each of the other CPU cores.
6. Please state (1) what range of real-time priorities you see being used, (2) what processes have real-time priorities, and (3) speculate why they may need to be run with real-time priority.
7. Please state (1) how many sched_switch events occur on your program's processor, (2) whether your program is ever preempted, and if so (3) when and where is it preempted?

Advanced Topic (not included in the lab grade):

1. Repeat step 3 to 7 with SCHED_FIFO.
2. Generate a program with multiple threads. Assign each thread a scheduler and a priority. Then repeat step 3 to 7.