CMPS111 Spring 2016: Assignment 2: Scheduling

Goals

The primary goal for this project is to modify the FreeBSD scheduler to use lottery scheduling rather than the current scheduler.

This project will also teach you how to experiment with operating system kernels, and to do work in such a way that might crash a computer. You'll get experience with modifying a kernel, and (at some point) will probably end up with an OS that doesn't work, so you'll learn how to manage multiple kernels, at least one of which works.

Basics

The goal of this assignment is to get everyone up to speed on modifying FreeBSD and to gain some familiarity with scheduling. In this assignment you are to implement lottery scheduling in FreeBSD. A lottery scheduler assigns each process some number of tickets, then randomly draws a ticket among those allocated to readyprocesses to decide which process to run. That process is allowed to run for a set time quantum, after which it is interrupted by a timer interrupt and the process is repeated. The number of tickets assigned to each process determines the likelihood that it'll run at each scheduling decision, and thus (over the long term) the relative amount of time that it gets to run. Processes that are more likely to get chosen each time will get chosen more often, and thus will get more CPU time.

In addition, you'll need to modify the `nice()` system call to increase or decrease the number of tickets assigned to the current process.

Before you start

Before you start, please choose a team captain and set up his/her repository so it can be shared. Make sure that, as you go along, everyone pulls the most recent changes made by others, especially if you're modifying related parts of the repository.

If you have any questions about workflow, please discuss them in section or office hours the first week so that they can be worked out.

Details

In this project, you'll modify the scheduler for FreeBSD. This should mostly involve modifying code in`sys/kern/sched_ule.c`, though you may need to modify an include file or two. You'll also need to modify the`nice()` system call to increase or decrease the number of tickets allocated to a process. Note that both lottery scheduling and the modified `nice()` process must *only* be used for user processes—those whose (effective) user IDs are non-zero (non-`root`). This is a good way to ensure that you don't end up with deadlock or other problems.

As discussed in class, we *strongly* recommend that you read Section 4.4 from the optional text (*Design and Implementation of the FreeBSD Operating Systems*). This section explains how the current scheduler works, and describes which routines are called when. This information will be invaluable in figuring out which routines need to be modified to implement lottery scheduling. In particular, focus on the routines that are called both at context switch time and less frequently to place processes in the appropriate run queues.

Lottery Scheduling

The current FreeBSD scheduler uses 64 run queues for each of three types of processes. Some of these run queues only have system processes, and the rest are used for user processes. You're going to add exactly three queues for non-`root` user processes: one for interactive processes, one for timeshare processes, and one for idle processes. You can use the existing mechanism for deciding which processes go where, but once you've done that, you should place the processes in the appropriate (single) queue and use tickets to select the one to run. This means that only `root` processes will be placed into the (standard) 64 runqueues. For each level (interactive, timeshare, idle), run processes in the standard runqueues first; if there are none, use the lottery scheduler to choose a process from the lottery queue associated with that level. Only if there are no ready processes in either the lottery or standard queues at a level should you proceed to the next level. This isn't any different than how the scheduler works currently; the difference (that you have to implement) is that user processes are scheduled in a different way.

By default, each process gets 2000 tickets when it's created. This number can be increased or decreased using the `nice()` system call. A process may not have more than 100,000 tickets, and may not go below 1 ticket (so use a `uint32_t` to store the number of tickets). The `nice()` system call should enforce these limits; you may either return an error if they're exceeded, or you can simply set the number of tickets to the minimum or maximum, as appropriate.

Each time the scheduler is called (for a context switch), it uses the current mechanism to select a scheduler queue. If the selected queue is either the interactive or timeshare queue (and there are no root processes in the standard queues at that level), it picks a number from 0–*T*-1, where *T* is the total number of tickets in the queue being run. *T* should **not** be calculated during the context switch, but rather should be tracked as processes are added or removed from the queue—calculating *T* each time would be too slow. For the random number, use a 64-bit integer chosen from a pool of random numbers calculated during the infrequent scheduler calls—the random number generator shouldn't run at context switch time. The 64-bit integer can then be taken modulo *T*, yielding *r*. Go down the list of processes in the queue, adding the number of tickets each one has, until you reach a number larger than *r*, and that's the process to run.

NOTE: if you have a newer system that supports the `rdrand()` CPU instruction for generating random numbers, you *may* use that at context switch time if you wish, though you're not required to do so. If you do use it, however, make sure that `rdrand()` is available on all of your group members' systems, and that you document its use in a `README` file in the `asgn2` directory.

Building the Kernel

Rather than write up our own guide on how to build a FreeBSD kernel, we'll just point you at the guide from the FreeBSD web site. You don't need to worry about taking a hardware inventory, as long as you don't remove any drivers from the kernel you build (and there's no reason you should do this). Focus on Sections 9.4–9.6, which explain how to build the kernel and how to keep a copy of the stock kernel in case something goes wrong. Of course, we're happy to help you with building a kernel in lab section and/or office hours. A couple of suggestions will help:

- Try building a kernel with no changes *first*. Create your own config file, build the kernel, and boot from it. If you can't do this, it's likely you won't be able to boot from a kernel after you've made changes.
- Make sure all of your changes are committed *before* you reboot into your kernel. It's unlikely that bugs will kill the file system, but it *can* happen. Commit anything you care about using `git`, and ***push your changes to the server before rebooting***. The OS ate my code isn't a valid excuse for not getting the assignment done.

Deliverables

As usual, you'll commit your code using `git`, and submit your assignment by turning in a commit ID using eCommons (only the team captain turns in the commit ID). Please remember to turn in your written evaluation of your contribution and that of your teammates to the group project.

Your design document and any other supporting materials should go in the `asgn2` directory, which is already created for you in the `asgn2` branch. If you don't see the `asgn2` directory, you're probably in the wrong branch—run `git checkout asgn2` to get into the correct branch.

One final thing: you need to write up a few paragraphs describing what you contributed to the group effort, and how you'd rate the other members of your group. This (one) plain-text (ASCII) file should be submitted via eCommons along with your commit ID. The goal here is for us to understand how each person contributed to the group effort. We don't expect everyone to have done the same thing, but we expect that everyone will contribute to the project.

Hints

- ***START EARLY!*** Meet with your group ASAP to discuss your plan and write up your design document. design, and check it over with the course staff.
- Experiment! You're running in an emulated system—you *can't* crash the whole computer (and if you can, let us know...).
- Write up your design document first, after looking over the existing code in FreeBSD, and run through it to see what happens under different conditions. Our experience has shown that groups that do this have a *much*easier time getting things to work.
- Consider debugging with a non-random version of your random number generator—one that follows a predictable sequence of values returned—so that you know what to expect at each context switch.
- Test your scheduler. To do this, you might want to write several programs that consume CPU time and occasionally print out values, typically identifying both current process progress and process ID (example :**`P1-0032`** for process 1, iteration 32). Keep in mind that a smart compiler will optimize away an empty loop, so you might want to use something like this program for your long-running programs.

This project doesn't require a lot of coding (typically no more than several hundred lines of code), but does require that you understand FreeBSD and how to use basic system calls. You're encouraged to go to the class discussion section or talk with the course staff during office hours to get help if you need it.

IMPORTANT: As with all of the projects this quarter, the key to success is starting early. You can always take a break if you finish early, but it's impossible to complete a 20 hour project in the remaining 12 hours before it's due....

Project groups

You've been assigned to a project group with 2–3 other students in the class. You *must* turn in the same project, for which you'll all receive the same grade. Only the team captain turns in the commit ID via eCommons.

Everyone in the group needs to turn in his or her own brief writeup on the contributions from each person in the group. This is done via eCommons so the writeup is ***private***—it will *not* be shared with others in the group. We want your honest opinion on who contributed what, and we understand that it may be difficult to do this if others in the group can see what you wrote, so this is the *only* part of the assignment that isn't done as a group. While we expect to give everyone in the group the same grade, We'll take this feedback into account when we determine project grades.