Assignment 2: Scheduling in FreeBSD

Due February 6 before midnight
(Note: this is a group assignment. Make sure to signup for a group in https://docs.google.com/spreadsheets/d/1yDjXmUEw0LQxg__hqWD21m-DG-ocQxU5nKWRIGLtP0k/edit?usp=sharing)

Goals

The primary goal for this project is to modify the FreeBSD scheduler and implement a new scheduler called the Lottery 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.

What you will do in this assignment:

- **Study how process scheduling is implemented in the FreeBSD kernel.** The source code of the kernel is in /usr/src in your FreeBSD installation. The point where you should start is this file **sys/kern/sched_ule.c.** To learn more about scheduling read Section 4.4 from the optional text (*Design and Implementation of the FreeBSD Operating Systems*)
- **Learn how to recompile the kernel.** guide from the FreeBSD web site.
- **Implement Lottery Scheduling instead of the current scheduling algorithm.** Look for lottery scheduling in your textbook to learn more about it **(section 2.4 page 163)**.
- Implement benchmarks to test your lottery scheduler. More details in the rest of this document.
- Commit your code and other files to your gitlab repository. Your gitlab repository should include an asgn2 folder with the following inside it:
    - All the files that you changed in the FreeBSD kernel.
    - All the source files you used to benchmark the new scheduler.
    - Makefile.
    - README file with instructions on how to recompile the kernel and reproduce the benchmarks.
    - Design documentation and report of benchmark (could be a single file).
- Submit the commit id to Canvas with the emails of all the team members (only the team captain needs to do this.)

Grading out of 100 pts (these might change):

- Setup: 5 pts.
- Design document and benchmark report: 20 pts.
- Benchmark code and being able to reproduce the benchmark: 25 pts

- Running the kernel with the lottery scheduler and observing a correct output from the statistics printed by the kernel (we will give details about what you should print from inside the kernel): 50 pts
- **Note: If the kernel does not compile, you might get as low as 20 pts or lower out of 100 regardless of what you implemented.**

Before you start

Before you start, please choose a team captain and set up their 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. The size of the FreeBSD kernel source is larger than what is allowed in GitLab. For this reason, you have to only add/commit/push the files that you change. Do not add/commit/push binary files. You can push as much of the source code as you want as long as you do not exceed the GitLab size limit. 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.

Scheduling

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 will work with scheduling in FreeBSD. A scheduler decides 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 scheduler decides which process to run next.

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. Note that the scheduling you implement 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 your project. 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.

The FreeBSD ULE Scheduler

The current FreeBSD scheduler uses 64 run queues, some of which only have system processes, and the rest of which are used for user processes. The current FreeBSD scheduler selects a run queue based on the processes priority and each run queue runs as a FIFO queue. If the kernel wants to pick a process to run next, it takes one from the non-empty run queue with the highest priority. When the kernel wants to insert a process to the run queues, it inserts it to the run queue that corresponds to its priority.

Lottery Scheduling

You will modify the kernel (mostly in **sys/kern/sched_ule.c**) to implement a lottery scheduler **(see the textbook section 2.4 page 163)**. Rather than using the run queues, you will create a single queue for all user time-sharing processes. Your changes should affect the processes corresponding with user time-sharing processes only. (Refer to Table 4.2 in the optional textbook.) In Lottery scheduling, each process has a number of tickets that will be used to decide which process runs next. In your implementation set the number of tickets for a process to be its priority value that is determined by the "nice" value (see PRI_NICE in sched_ule.c) When the kernel wants to schedule a process to run next, it will pick a process according to the following probability:

Definition 1: The probability of picking process i = (the number of tickets of process i)/(the total number of tickets for all processes)

Here are the functionalities that your code needs to implement:

- A queue (call it *lottery queue*) for user time-sharing processes that will be used for lottery scheduling
- When the kernel attempts to add a process to a run queue for user time-sharing processes, you should intercept that call and place the process in your lottery queue.
- When the kernel wants to schedule a time-sharing process to run, you should intercept the call and pick a process from your lottery queue according to the probabilities as defined above in Definition 1.
- Whenever a process is added to or removed from the lottery queue you must print (using a kernel print that can be read using dmesg) a statement with statistics of the event: (1) the event type, add or remove, (2) the size of the queue, (3) the smallest number of tickets of a process in the lottery queue, (4) the largest number of tickets of a process in the lottery queue, (5) the total number of tickets of all processes, (6) the number of tickets of the process added to or removed from the lottery queue.

Benchmark

You will also have to write a benchmark to validate the lottery scheduler implementation. Your benchmark should show that the processes with higher nice values should be scheduled more often. What your benchmark should do is:

- Create multiple processes.
- Assign different nice values to the processes.
- Collect information about what processes are being scheduled (using the statistics produced from your lottery scheduler)
- Write the collected information in a file that shows relevant statistics. You can choose the type of statistics that will highlight how processes with higher nice values are scheduled more.
- You should include the results of the benchmark in your design document and benchmark report.
- Also, you should provide a script that allows the graders to recreate the benchmark.

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 8.4–8.6, 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.
- The kernel compilation can be a lengthy process. Find ways to compile the kernel faster (check piazza and ask the TAs/tutors)

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.

This project doesn't require a lot of coding, 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.

Project groups

You've been assigned to a project group with 2 or 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 Canvas.