Final Project - EECS 211

QEMU

Winter 2025

In this final project, you will modify a tick-based kernel to dynamically adjust the tick period under the kernel's control. Your goal is to modify the kernel so that the tick period will be dynamic. You should have a policy for how the tick period changes based on the behavior of the processes.

You are required to measure the difference in the performance and write a report about your adaptive tick interval algorithm.

1 Background

For this assignment, you will use the xv6 kernel (provided as a zip file named xv6-riscv-riscv.zip) for the RISC-V 64 bit architecture. This includes all of the core components of an operating system kernel while still being accessible to hack on.

2 xv6 Kernel

The xv6 kernel includes the various components you would expect to be in an operating system kernel. The actual scheduler is in `proc.c`. You should be able to see that the scheduler by default uses a round-robin scheduler.

The code for handling interrupts and exceptions is in `trap.c`. This is, for example, where the code that handles when a userspace process calls a syscall and the system traps to the kernel resides, as well as where interrupts are handled.

Userspace applications have a very standard set of syscalls, include `read`, `write`, `fork`, etc.

3 Setup

1. Extract the xv6 Kernel zip file.
2. Test

   ```
   cd xv6-riscv-riscv
   make qemu
   ```

   That will load the emulator with the kernel, and by default the kernel loads the shell process (`sh`). This is a basic shell, but should be reasonably familiar.

   You can use this basic shell to execute all the existing applications in the `user` folder.

   To exit QEMU you can enter `ctrl+a` and then `x` to exit QEMU.

4 How to modify the kernel to implement a dynamic tick interval policy:

To modify the scheduler to support dynamic tick interval policy you can directly edit the scheduler function inside proc.c and then re-run `make qemu`.

You can use the behavior of processes and CPU to set a time interval for the current process based on your chosen dynamic tick interval algorithm. You can find some useful variables defined in cpu and process structs (proc.h) such as the total number of running processes, the total number of sleeping processes, etc.

5 Deliverable

The objective of this project is to implement a dynamic tick interval policy. This should respond to the operation of processes in the system with the goal of reducing the number of ticks while preventing a process from monopolizing the CPU.

You should use the runtime program provided in `user` folder to test the performance of your dynamic tick interval policy compared with the original fixed interval algorithm. Use `forktest`, `usertests`, and `ls` programs (average of 10 times each), and report the performance comparison (execution time, total number of ticks, context switches, or etc.) in your report.