CS169: Implementing WeenixOS

Jackson Chaiken
Faculty Sponsor: Tom Doeppner (twd@cs.brown.edu)

Spring 2020

1 Introduction

My capstone project involved implementing 5 pieces of the Weenix operating system over the course of the Spring 2020 semester. The capstone project consisted of completing five sub-projects, each of which are a major piece of an operating system. These projects were Processes and Threads, Drivers, Virtual File System, System V File System, and Virtual Memory and were written entirely in the C programming language. Overall, this project offered hands on experience in low-level coding and debugging, adapting to an existing codebase, and learning how core parts of an operating system work by building them, supplementing what was covered throughout the course.

2 Projects

2.1 Processes and Threads

The Processes and Threads project involves writing the building blocks of an operating system: threads, processes, and synchronization primitives. This part of the project also involved building a simple, FIFO scheduler for the operating system to use when processes yield. This part of the assignment also involved implementing the waitpid system call, in which a process can wait on one, or any, of its child processes to terminate before returning. The most fulfilling part of this project was the context it gives to "user-land" tasks, as it increased my understanding of what goes into creating processes and threads of control in the kernel.

2.2 Drivers

The Drivers project consisted of building the teletype system for Weenix as well as completing the SATA driver the Weenix uses to write blocks to disk. This project also had students implement the /dev/null and /dev/zero memory devices, which are provided on UNIX operating systems as well. This project mainly served to acquaint myself with the software engineering principles that

go into designing an operating system for which different drivers can be easily written to add support for more hardware devices.

2.3 Virtual File System

The Virtual File System project builds an interface between Weenix and various file system implementations. In VFS, I wrote system calls related to files, such as open, close, read and write. At the virtual file system level, we also learned how to implement file locking and reference counting for management of open files.

2.4 System V File System

The S5FS project consisted of building our own file system to use with Weenix's virtual file system that we had previously implemented. The System V File System is a relatively simple file system, but introduces the concepts of inodes and disk blocks. In addition, this project exposes students the implementation details behind directories and file system organization.

2.5 Virtual Memory

Finally, the Virtual Memory project ties together all of the work previously done and after its completion Weenix becomes a full-fledged operating system, as previous to it we could not run a project in "user-land" and were strictly bound to the confines of the kernel. By implementing memory management for a process, which involves mapping addresses to physical chunks of memory, handling page faults when a mapping does not yet exist, and handling shared memory between processes.