CSCI 1690: WeenixOS

Fernando Cisneros
Faculty Sponsor: Tom Doeppner

May 14, 2023

I Abstract

My capstone project involved implementing WeenixOS, a simple Unix-based Operating System, throughout the 2023 Spring Semester as offered by CSCI 1690, a half-credit course intended to build off the material covered in CSCI 1670: Operating Systems. Implementing Weenix involved completing 5 interconnected projects to serve as the basis for a functional Operating System: Processes and Threads, Drivers, Virtual File System, System V File System, and Virtual Memory. This project, written in C, offered a chance to understand the intricacies of topics including the fundamentals of operating-system structure, processes and threads, managing storage, and basic file-system design. Atop the exploration of foundational operating systems principles building, Weenix has allowed for the opportunity to understand, build, and debug an existing large code-base.

Implementing this operating system does not require a student to build Weenix from the ground-up. Students are provided with an existing code base capable of handling aspects such as the operating system's boot sequence and basic kernel operations such as memory allocation to ensure that this project can be completed within a given semester. Furthermore, as a student enrolled in CSCI 1690 I received help from an assigned mentor from the course staff throughout the duration of this project.

II Assignments

Processes and Threads

The Processes and Threads assignment involved building the basic kernel-level code needed to handle threads, processes, and synchronization primitives that serve as the backbone for running user-level executables in later projects. Upon completion, my Weenix implementation was able to run several threads and processes concurrently in kernel mode, clean exiting threads and processes upon completion, and successfully exit the OS Kernel.

Throughout this project, I established a greater understanding of the lifecycle of threads and processes. Additionally, I found that I reinforced my understanding of existing system calls such as waitpid that were a necessary component in cleaning resources used by a process through an ancestor process such as a process's parent process or the init process.

Drivers

The Drivers project involved building device drivers for terminals, disks, and memory devices used within Weenix. This project was valuable in understanding how low-level drivers interact with high-level user-facing interfaces when implementing such components such as Weenix's tty devices–which consisted of a driver needed to interact with the system's keyboard and a line discipline used to format and buffer a user's input. Such work granted me a better understanding of how users would interact with files later on in Weenix's implementation (principally after implementing Weenix's Virtual File System). Working on Drivers also served as a useful introduction to memory devices (/dev/null and /dev/zero) and block devices used to perform I/O operations to disk.

Virtual File System

As previously mentioned, the Virtual File System project builds directly off of previous projects to build a Unix-style interface between the Weenix OS and its file systems. This project was especially tedious as I had to implement and manage various data structures that would allow multiple processes to access, edit, read, and discard existing files appropriately without losing reference to such files throughout a process' lifetime. Although some mechanisms such as mounting file systems were handled by pre-existing code, working on this project allowed me to begin building an understanding of how file systems cache data to manage expensive read/write operations and how to appropriately reference and dereference files when in use.

System V File System

The System V File System project requires students to implement a rudimentary Unix-based file. This project allowed me to learn the details behind directories and file system organization through the introduction of new structs such as inodes, which are abstractions that can be used to represent files in use and allowed me to work more in-depth with data blocks in which the contents of a given file are stored. Additionally, this project involved implementing high-level system calls for operating on block devices or files such as reading, writing, searching for existing files or directories, and creating new entries within our previously built file system. Such high-level system calls built upon previous work done in VFS and thus allowed me to build a better understanding of our existing Weenix architecture.

Virtual Memory

The last project, Virtual Memory, utilized my previously implemented device drivers file system to create a fully functional operating system. This project focused on enabling the Weenix kernel to manage multiple virtual address spaces of all processes, run user-level code, and execute system calls. In this project managing virtual address space for a process required correctly handling a process' "memory map" represented as a linked list of virtual memory areas that correspond to some memory object which provides pages of memory from files or disks to a given process when needed. Additionally, this project required the operating system to correctly manage shared memory objects when loading data into a process' memory upon request through the implementation of a page fault handler.