Student Min Seong Kang

Course: CSCI1690

Faculty: Tom Doeppner

Term: Sprint 2024

WeenixOS

1 Introduction

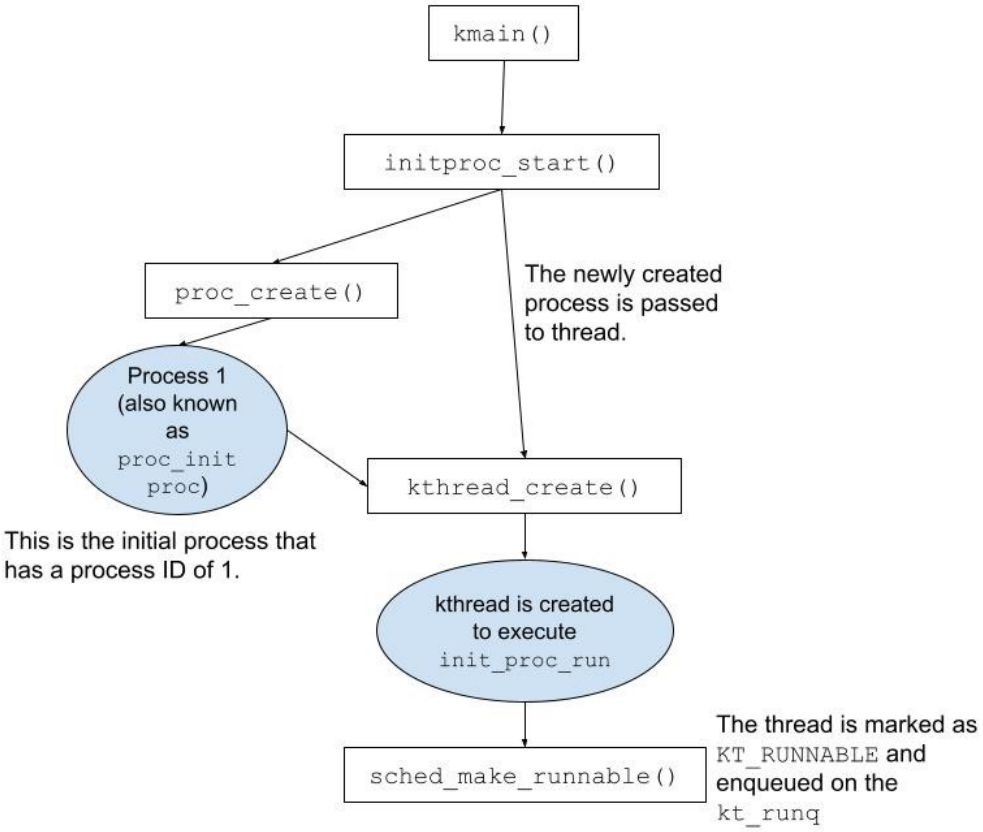
As a part of my capstone declaration, I have completed the Weenix Operating Systems project in the context of CSCI1690. Weenix is a Unix-like kernel that has all the components to serve as a standalone operating system. The core parts of Weenix are: Processes and Threads, Drivers, the Virtual File System, the System V File System, and Virtual Memory, implemented in C. The project has helped me to substantially develop crucial skills for software developers including low-level device-based thinking, parallel and concurrent programming, and test engineering for complex multiplex execution.

2 Implementation

2.1 Processes and Threads

This part is a fundamental block of the OS that manages processes, threads, and their scheduling. The mechanics of processes and threads in Weenix are quite straightforward – there is no preemption, and each process handles only a single thread at a time. The challenging aspect of the assignment is correctly handling inter-process relationships. Firstly, a parent process can wait for a child process. More im-

portantly, the former needs to perform cleanup for the latter. This design choice is due to a thread's inability to deallocate its own structures. Another significant portion of this OS component is correct state management, as illustrated in the diagram below:



2.2 Drivers

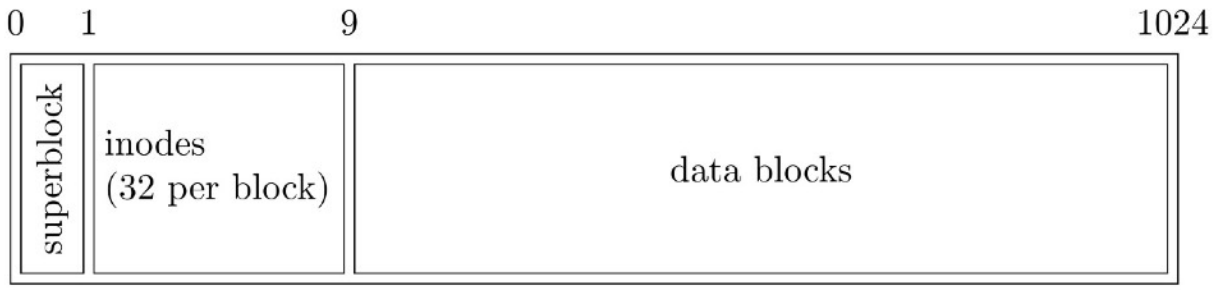
This section deals with the drivers in the OS needed to operate different devices within the kernel code. There are two types of devices: block devices and character devices. Character devices are designed to transmit or receive data one character at a time, while block devices handle fixed-size blocks of data. The main objective is to perform read/write operations on these devices, enabling higher-level applications to function correctly.

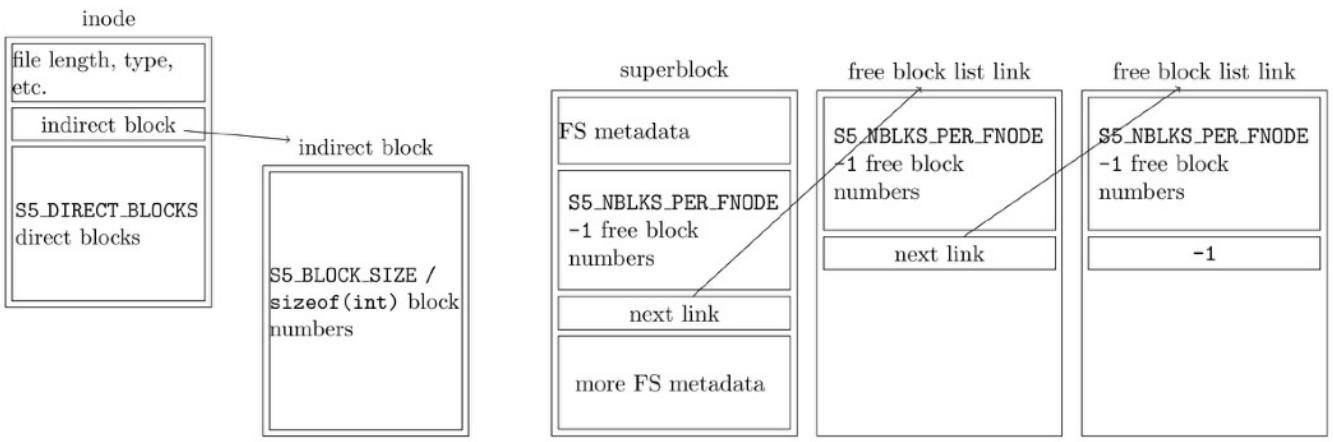
2.3 Virtual File Systems

The Virtual File System (VFS) is the backbone of file system management in Weenix. Serving as an interface between the OS and various file systems, VFS allows the addition of multiple file system types to the kernel, enabling access through the same UNIX-style syntax. In other words, VFS provides an abstraction layer to deliver polymorphic functionalities across different file systems.

2.4 System V File System

The System V File System (S5FS) is based on the original Unix file system. While it lacks some of the advanced features of modern file systems, it serves as a solid introduction to the core features of basic file systems. The structure of a disk device that uses S5FS is as follows:





Inodes represent files on the device and contain metadata, including the addresses of data blocks. A noteworthy feature is the `indirect_block` field of an inode object, which is a disk block that maps more data block numbers to other disk blocks. This layer of indirection enables the system to handle large files. Finally, Weenix uses a caching mechanism with memory objects that hold dirty pages until their reference count drops to zero.

2.5 Virtual Memory

The virtual address space for a process (also known as its “memory map”) is stored as a linked list of virtual memory areas (also referred to as “memory regions”), each of which correspond to some memory object which provides pages of memory to the process on demand. On top of managing virtual address space for each process and loading the data via page faults, Weenixs needs to support system calls that affect VM. The foremost challenge is to implement fork() procedure that clones the VM space for a child process. In order to fulfill the copy-on-write the implementation requires a more sophisticated memory object in the form of Shadow Objects.