Lab5

5140379041

Jiacheng Ma

This lab is to implement a simple file system for JOS. Firstly, the file system part should be implemented, include how to access or modify a block. This naïve file system use a special environment to access the disk, and other environment will use ipc to access the file. The file system environment reserve 3GB in its address space to map disk. When visiting to an address in the region, a page fault may occur, and in the handler the file system copies the block to the counterpart region in the address space, and then it can access it and transport it to other process. A spawn function is also provided to start other environments. It will fork a new environment firstly and then copy the program segments to the corresponding region and setup the trap frame of it. The procedure is like the fork function, but the source of the segments are from the disk.

Questions:

How long approximately did it take you to do this lab?

* Sorry, I forget it …
* However, the challenge I choose really takes me a long time

We redesigned the file system this year with the goal of making it more comprehensible in a week-long lab. Do you feel like you gained an understanding of how to build a file system? Feel free to suggest things we could improve.

* Actually, we’ve already get the essence of building a file system in CSE class, however this lab put the file system in a real operating system, which will bring some difficulty. The main difficulty lays on how to communicate with the hardware and maintain a cache of files. The cache is too simple, though.

*Challenge!* The block cache has no eviction policy. Once a block gets faulted in to it, it never gets removed and will remain in memory forevermore. Add eviction to the buffer cache. Using the PTE\_A "accessed" bits in the page tables, which the hardware sets on any access to a page, you can track approximate usage of disk blocks without the need to modify every place in the code that accesses the disk map region. Be careful with dirty blocks.

* Firstly, I set an array to store all in-memory blocks, when a block is mapped into the memory, it will be added in the array. If the array is full, an eviction operation will occur, in which all inactive blocks will be written into the disk and the pages will be released.
* While deciding which block is active, I reserve 2 bits (bit 10 and bit 11) in the page table to indicate how recent these pages are accessed. While finding the array is full, the program will scan the array and clear the access bit of the page, and mark bit 10 and bit 11 to one. For these non-accessed pages, bit 10 and bit 11 will decrease. When bit 10 and bit 11 both become 0, the page will be removed. The program will scan the array again and again until one page is swapped out.