Assignment Report

121090697

In this report, when citing a reference, such as the xv6-book, feel free to use footnotes[1].

Your report should follow the template with the following section structure.

No page limitation

1 Introduction [5']

1.
   - The file system of xv6 follows a hierarchical structure where directories contain files and subdirectories. It is based on inodes, which are data structures that store information about files. The file system maintains a file allocation table (FAT) to keep track of free and allocated disk blocks. The xv6 file system could benefit from improved performance optimizations. For instance, implementing caching mechanisms or employing better disk scheduling algorithms could enhance overall file system speed and efficiency.
   - About memory management, xv6 utilizes a simple paging system. It employs a two-level page table structure to map virtual addresses to physical memory. And about aspects that need further development, it might have the chance to support advanced memory management techniques such as demand paging or memory swapping.
2. In this assignment, I have managed to let xv6 support big files (from 268BSIZE BYTES to 65803BSIZE BYTES) by adding a double-indirect block. And the corresponding functions to truncate inodes. And I have added support to symbolic links.
3. I think my progress was quite smooth.

2 Implementation [3']

In this section, you should present your design. For each **TODO** task, explain your implementation in a separate subsection.

As Assignment 4 is clearly introduced in the instructions, you only need to clearly describe your design for the basic part. (Flowcharts are still welcomed)

For bonus tasks, you may provide more detailed descriptions.

2.1 Bigfile

Basic modifications on *file.h* and *fs.h* had been introduced in the instructions, I will only show the design in *fs.c*.

2.1.1 bmap()

After subtracting **NDIRECT** and calculating the index in indirect and double_indirect block (bn_1 and bn_2 respectively). The code would check if the indirect block is already allocated for the *inode*. If not, it allocates a new block and stores the address. Next, I read the indirect block into a buffer cache. The code then retrieves the array of block addresses stored in the indirect block and assigns it to *a*. Then I would check if the address for the desired block within the indirect block is already

[1] Reference: https://pdos.csail.mit.edu/6.828/2005/readings/pdp11-40.pdf

allocated. If not, the code allocates a new block and stores the address. It is then the same when dealing with the blocks in double-indirect. The buffer cache should be released, allocated block address should also be returned at the end of the function.

2.1.2 itrunc()

The implementation of this function is easy to think of. I used a double for_loop to release all the blocks and release the buffer cache.

2.2 Symlink

2.2.1 sys_symlink()

The function would first read arguments and check their validity. Then I use *nameiparent()* to resolve the parent directory of the given path and return a pointer to the corresponding structure. I would check if there is already a file or directory with the same name as the symbolic link in the parent directory. After checking all these validities, I would create a new inode to store the symbolic link. I take *path*, **T_SYMLINK** as parameters. I then call *writei()* to write the target path into the symbolic link's inode at offset 0. Finally, the function releases the resources, including the inode of the symbolic link and the parent directory, and ends the atomic operation on the file system.

2.2.2 sys_open()

The function uses a while_loop to continue reading the linked file of the symbolic link. If the linked file is also a symbolic link, it will continue to read the linked file until it might seem as a cycle or it read file finally.

3 Test [2']

Briefly discuss which part of your implementation helped you pass each test. (Several sentences for each subsection should suffice.)

3.1 bigfile

bmap() for writing 65803 blocks, and *itrunc()* for truncate inode.

3.2 symlink (if done)

sys_sysmlink() for creating symbolic link, and *sys_open()* for reading it.

4 Comments [0']

Maybe you can allow students to write reports using markdown.