Assignment4 Report

Name:余永琦 ID:120090761

Note: I have finished the bonus

Environment

I complete and test my program on the GPU cluster provided by the course.

OS version: CentOS Linux release 7.5.1804 (Core)

CUDA version: CUDA SDK 11.7.20220729

GPU information:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 515.65.01    Driver Version: 515.65.01    CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Quadro RTX 4000     Off  | 00000000:AF:00.0 Off |                  N/A |
| 34%   62C    P0    59W / 125W |    138MiB /  8192MiB |    100%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    142887      C   ...20090694/csc3150/HM3/main      123MiB |
+-----------------------------------------------------------------------------+
```

Execution steps

One way: “$ sbatch slurm.sh”
Another way: “$ sh slurm.sh”

Note: For the sbatch running, you can check the result on the file result.out. For the sh running, it will output the result in terminal.

How I designed my program?

In this assignment, we are asked to simulate a mechanism of file system management via GPU's memory.

Basically, the we divide the volume(disk storage) into three parts to implement the file system. One is for the volume control block(super block), in total 4096 bytes, to indicates whether the blocks is empty. One is for the file control block(FCB) to record all the details of a file, including file size, address, created time, modified time and a valid bit. The each FCB entries takes 32 bytes and in total 1024 FCB entries. The other one is for the content of the files. There are 1024 file in total and each file has maximum size of 1024KB, so in total the three part takes 1024*1024KB.

In the main task, I store the file information in FCB as following: the first 20 bytes 0-19 to store the file name, bytes 20-21 to store the created time, 22-23 to store the modified time, 26-27 to store the size(2 bytes is enough since max size is 2**13 bits so 2**16 bits is enough), bytes 28-30 to store the address of file (3 bytes is enough since the max address is 1024*1024KB which is 2**23 bits), bytes 31 to store the valid bit(actually it only takes one bit).

We also need to implement open, read, write, ls_s, ls_d, and rm operations in main task. In the **open** operation, we first check whether the file we want to open is existed. If so, we return the pointer to that FCB. Otherwise, if the mode is read, then the open operation raise error, if the mode is write, we find an empty FCB entry and create a file with 0 size. In the **read** operation, we begin from the start address of the file which is stored in the FCB and read the specified size to the output buffer. Notice that if the read size larger than the file size, it will raise error. In the **write** operation, if the file size is not 0, we need to first clean content of it by setting its super block to 0. Then we need to do a compaction. We compact all the file after the empty hole to fill the empty hole, so the file system is always compacted because we maintain it before and after write operation, also rm operation. Then, after the compaction, we find an empty block right after the non empty one, so the system will still be compacted. Then we write the content from the input buffer to the disk. In the **ls_d** and **ls_s** operation, we first get all the valid files. Then, in the ls_d operation, we sort

the file by the modified time and print them out. In the ls_s operation, we sort the file by size and print them out, if the sizes of files are the same, we sort them by created time. I used insertion sort to the files. In the **rm** operation, we need to set the valid bit of the file to 0 and set the corresponding super block to 0, and do a compaction in the end to keep the files compacted.

Bonus

In bonus, we need to implement tree-structured directories. To achieve the tree structure, we need to store additional pointers and information in the FCB entries. For the file FCB entries, we need to store a pointer that point to the next file/directory, and a pointer that point to its parent. For the directory FCB entries, we need another pointer that point to its first child. For each pointer, we use one more bit to indicate whether it is valid, that is whether it has parent/next/child.

Since the FCB entires contains more information compared to the one in the main task. So we reallocate the space in FCB so that we can store all the information. For the details, you can reference to **figure1**.

```
// The file FCB information contains:
// file address: 26-28 address from 0 - 2**23 so 23 bits is enough
// file size: 24-25 --> max file size is 2**10 bits, needs 10 bits, so 16 bits can handle
// time: 20-24 --> create: 20-21; modified: 22-23
// parent pointer: 11 bits
// next pointer: 11 bits
// valid bit: 1
// indentity bit to check if it's file or dir: 1
// in total 24 bits, 3 bytes, 29-31

// The directory FCB information contains:
// time: 20-24 --> create: 20-21; modified: 22-23
// parent pointer: 11 bits
// first child pointer: 11 bits
// next pointer: 11 bits
// a bit to check whether is in the current directory
// in total 36 bits, takes 5 bytes, 26 - 30
// valid bit and indentity bit in 31
```

Figure1:FCB information of file and directory

In bonus, we need to implement open, read, write, ls_d, ls_s, mkdir, rm, rm_rf, cd, cd_p and pwd operations. The **open**, **read**, **write, ls_d** and **ls_s** operations are very similar to the one in main task, except that we can not open directory and ls_d, ls_s only list the file in the current directory. In the **mkdir** operation, we find an empty FCB entry and set its valid bit and identity bit to 1, and store its name, size, times, pointers, also update the modified time of the current directory. In the **rm** operation, besides the work in the main task, we also need to update the pointers of

the file, and update the modified time of the current directory. In the **rm_rf** operation, we find the location of the directory, set its valid bit to 0, and also delete all the files and subdirectories in it, to delete the subdirectories, since we can not use recursion in the CUDA program, I use a for loop to iterate 3 times to implement it, because the tree structure has maximum deep 3. In the **cd** and **cd_p** operations, we just need to traverse the FCB entries, set the origin one current bit to 0, and set the target's current bit to 1. In the **pwd** operation, we just keep traverse the parent of the current directory until we reach the root, and then we concatenate their name to get the path.

In the initialization part, we need to initialize the root directory, so there are 1023 space left for us to create files or directories.

The above show how I design my program, in my code I define many for the implementation details, please go to my codes and read the comments.

Problems I met in this assignment

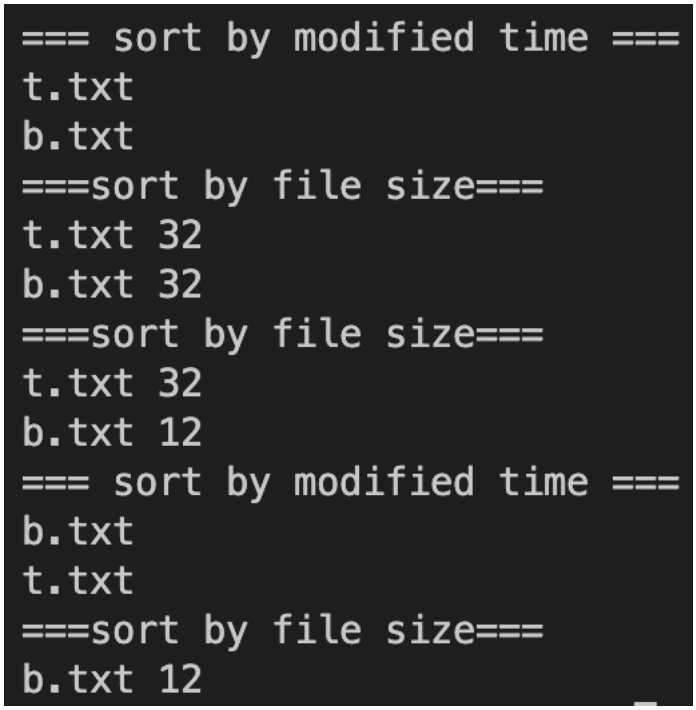
The main problem I met is that we can not use recursion in rm_rf operation in CUDA, so I use a for loop instead to solve this problem.

What did I learn?

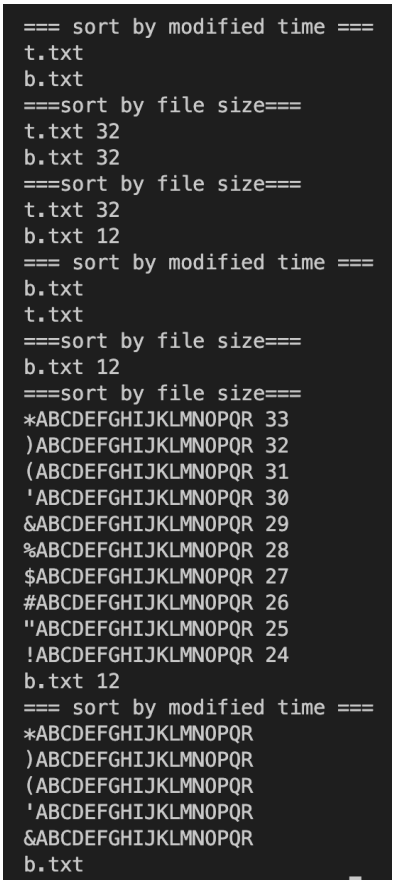
In this assignment, I learned how to design a file system. I learned how the basic file operations like open, read, write, rm, rm_rf, cd, cd_p, pwd works. Also, I learned to use volume control block to allocate the free block for the file system, and use file control block to keep track of all the files. Besides, I learned about the tree structure of file system and how to maintain it in the bonus part.

Screenshot

Testcase1



Testcase2



Testcase3 and testcase4 is too long to put in my report, I got the result exactly the same as the demo output!

Bonus