Report of project 4

119010384

SSE

Manual and abstract

In this project, I designed a file system for CUDA GPU. I wrote implementation of fs_open, fs_write, fs_read, fs_gsys(RM), fs_gsys(LS_D/LS_S) functions. These functions served as basic functions of a basic file system. As a result, the customer can create, access, modify files in my file system. To test my file system, first type **make** in terminal, then type **sbatch ./slurm.sh** to execute. You can add test cases in user program.

Design of program

In this project, I use FCB to store information of directories and use bitmap to store the whole information of volume. To illustrate my implementation of my program, I will firstly talk about my fcb entry design, then I will illustrate some self-created functions and fs_open, fs_write, fs_read, fs_gsys(RM), fs_gsys(LS_D/LS_S) in order.

FCB Entry Design:

[0:19] file name

[20:21] op

[22 23] block index

[24 25] size

[26 27] creation time

[28 29] modification time

[30] active bit

[31] allocated bit

__device__ uchar *fs_get_block_address(FileSystem *fs, u32 node_index);

This function returns the block index where the consecutive blocks starting from that block index are empty. The algorithm is that it scans every bit of super block, when meeting 0 bit, the variable increments and once count attains size specified, the block index will be returned. Once the 1 bit is scanned, the count resets 0.

__device__ void fs_set_block_entry(FileSystem* fs, u32 node_index, char* filename, u32 storage_index, u32 time, u32 createTime, u32 size, uchar use_bit) ;

This function updates the super block. the input is blkidx- blockindex and blknum – block number. The consecutive blocks starting from blkidx and the length is blknum can be reset as free 0 or in use 1 by calling this function.

Now I will illustrate functions fs_open, fs_write, fs_read, fs_gsys(RM), fs_gsys(LS_D/LS_S) in order.

__device__ void fs_read(FileSystem *fs, uchar *output, u32 size, u32 fp)

fs_read() functions inputs filename and outputs the fcb entry address. The fcb entry stores information of this file.

I iterate fcb entries to find the entry whose file name is the corresponding file name. when the entry found, return the fcb index.

If fcb entry is not found, it means that now there is that file in file system. then the file system will create fcb entry for this file. the file system will itearate all fcb entries to find free fcb entry. If the active bit of fcb entry is 0, free, then the fcb entry is free. Once the fcb entry is found and assigned. The file name and creation time will be recorded in that fcb entry.

__device__ void fs_read(FileSystem *fs, uchar *output, u32 size, u32 fp)

File system will find the fcb entry according to fcb address fp. Then use block index in fcb entry to get block address. The bytes starting from block address will be sent into buffer output.

__device__ u32 fs_write(FileSystem *fs, uchar* input, u32 size, u32 fp)

The file system finds fcb entry. Then there are two cases. The first case is that the file has been allocated memory. The other case is that the file hasn't been allocated memory. the allocated bit in fcb entry tells the case.

In the first case, the file system use block index to find the block address. If the old block size is larger than the input size, then just put all things in old blocks. If the old block size is smaller than the unput size, then use search_free_blk() function to find new blocks assigned to fcb entry.

In the second case, we allocate memory for this file. just use search_free_blk() to find block index for this file.

After memory allocation, the file system updates bitmap by calling update_super_block().

__device__ void fs_gsys(FileSystem *fs, int op)

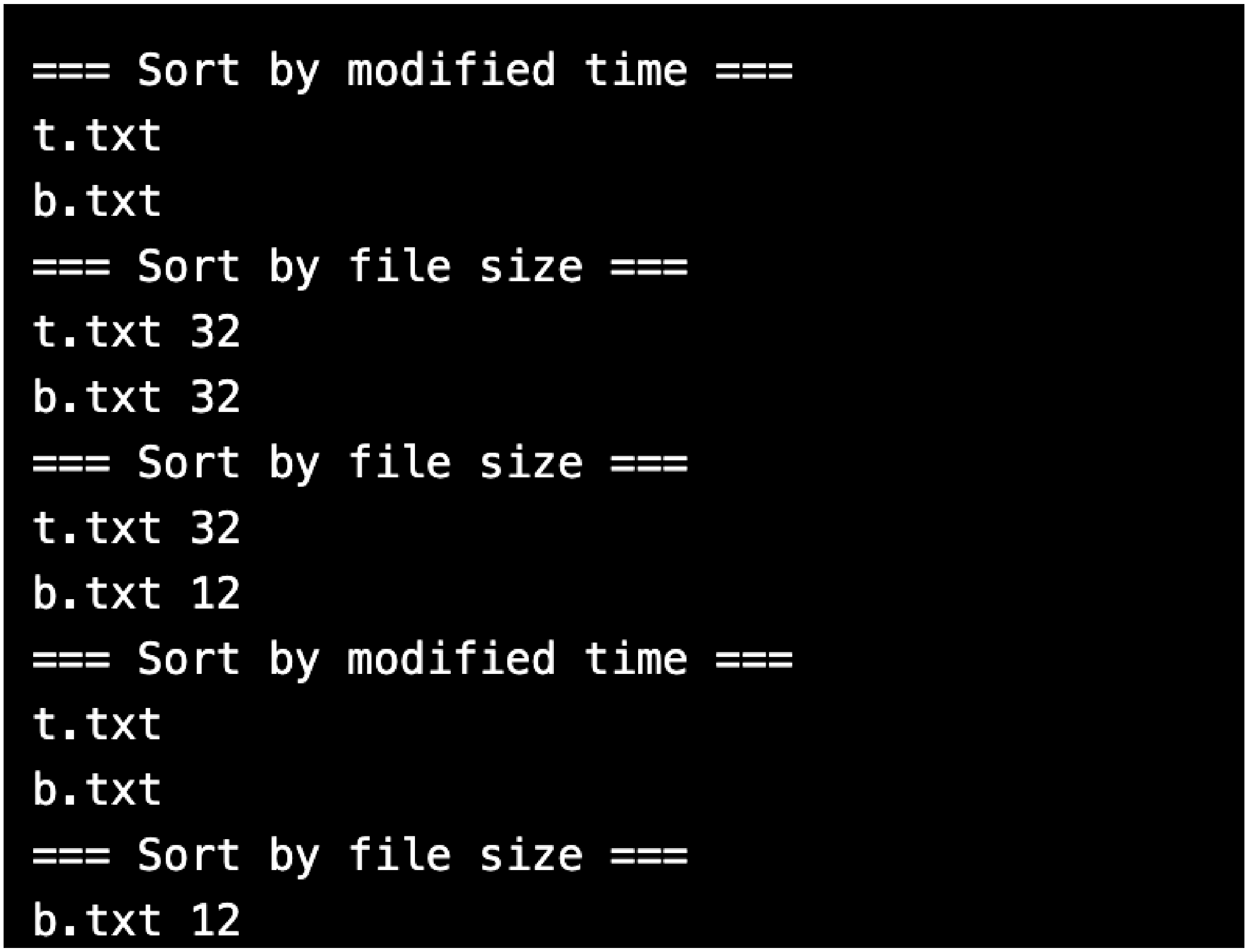
There are two options, list by modification time or list by file size. I create an array to store active fcb entries' fcb address. Then I will sort this array according to attributes modification time and file size. Each fcb address can access modification time and size. I use selection sort to sort the array. Then print the files.

__device__ void fs_gsys(FileSystem *fs, int op, char *s)

The file system will firstly find the fcb entry according to file name. Then delete this fcb entry and update bitmap by calling update_super_block().

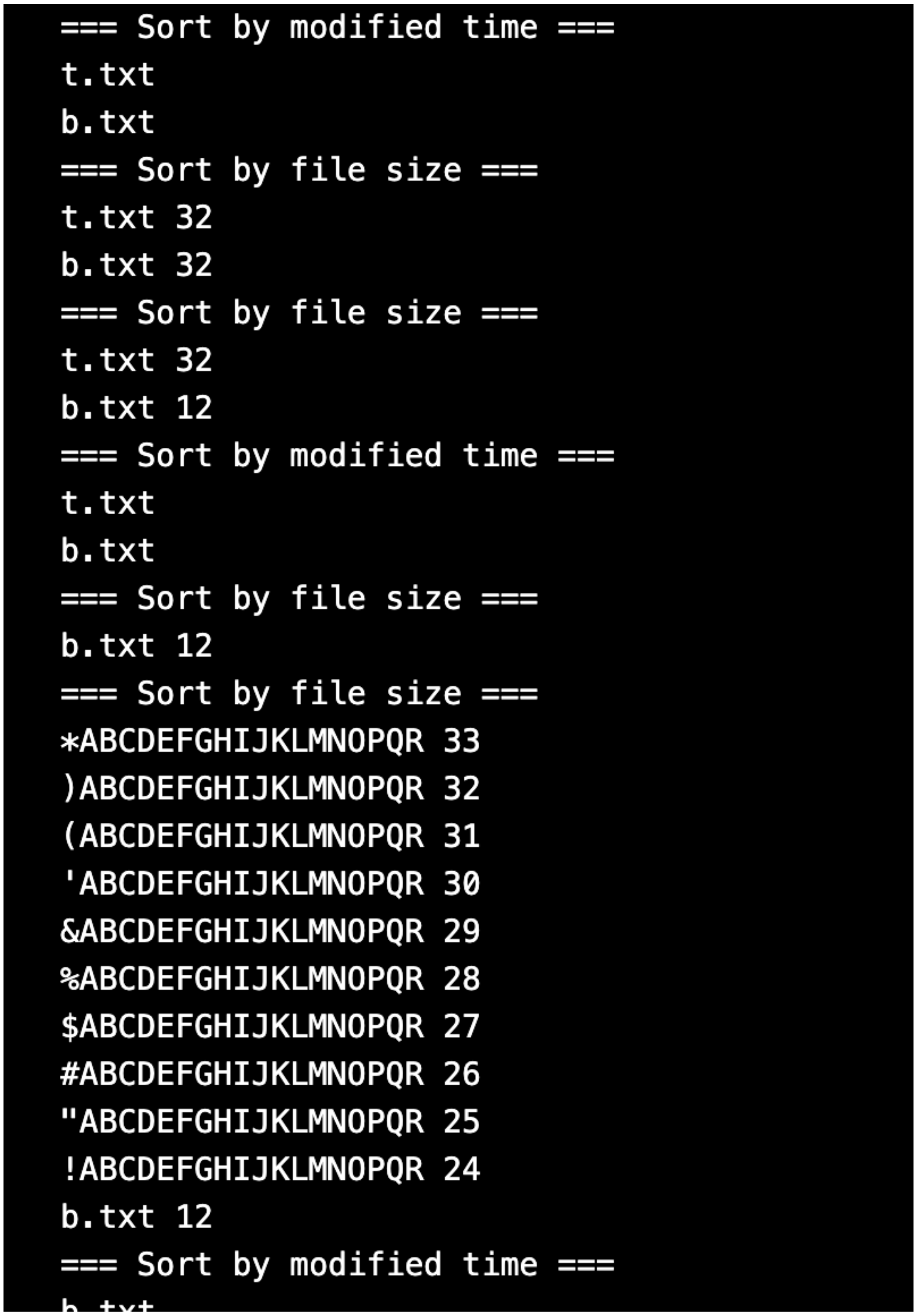
Output

Test 1:



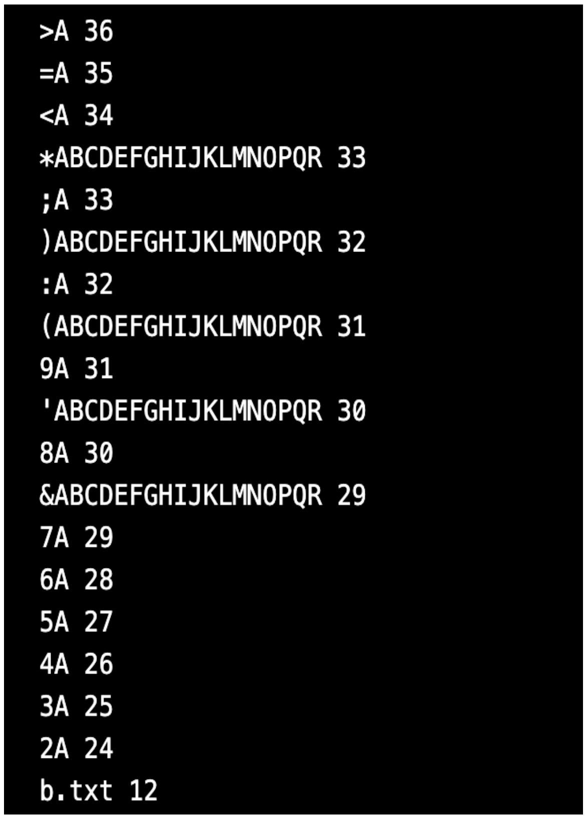
```
=== Sort by modified time ===
t.txt
b.txt
=== Sort by file size ===
t.txt 32
b.txt 32
=== Sort by file size ===
t.txt 32
b.txt 12
=== Sort by modified time ===
t.txt
b.txt
=== Sort by file size ===
b.txt 12
```

Test 2:



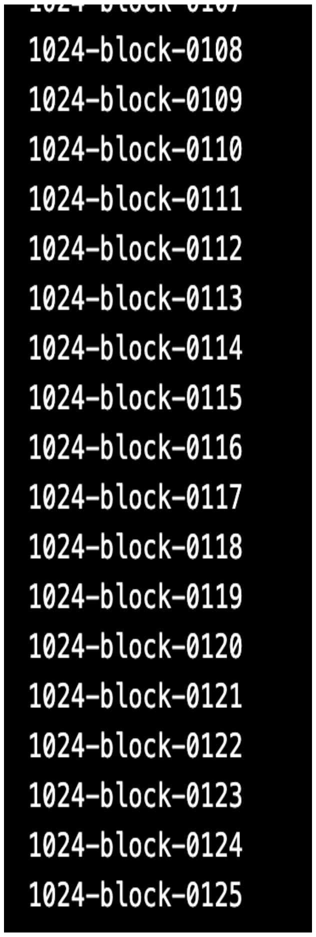
```
=== Sort by modified time ===
t.txt
b.txt
=== Sort by file size ===
t.txt 32
b.txt 32
=== Sort by file size ===
t.txt 32
b.txt 12
=== Sort by modified time ===
t.txt
b.txt
=== Sort by file size ===
b.txt 12
=== Sort by file size ===
*ABCDEFGHIJKLMNOPQR 33
)ABCDEFGHIJKLMNOPQR 32
(ABCDEFGHIJKLMNOPQR 31
'ABCDEFGHIJKLMNOPQR 30
&ABCDEFGHIJKLMNOPQR 29
%ABCDEFGHIJKLMNOPQR 28
$ABCDEFGHIJKLMNOPQR 27
#ABCDEFGHIJKLMNOPQR 26
"ABCDEFGHIJKLMNOPQR 25
!ABCDEFGHIJKLMNOPQR 24
b.txt 12
=== Sort by modified time ===
```

Test 3:



```
>A 36
=A 35
<A 34
*ABCDEFGHIJKLMNOPQR 33
;A 33
)ABCDEFGHIJKLMNOPQR 32
:A 32
(ABCDEFGHIJKLMNOPQR 31
9A 31
'ABCDEFGHIJKLMNOPQR 30
8A 30
&ABCDEFGHIJKLMNOPQR 29
7A 29
6A 28
5A 27
4A 26
3A 25
2A 24
b.txt 12
```

Test 4:



```
1024-block-0108
1024-block-0109
1024-block-0110
1024-block-0111
1024-block-0112
1024-block-0113
1024-block-0114
1024-block-0115
1024-block-0116
1024-block-0117
1024-block-0118
1024-block-0119
1024-block-0120
1024-block-0121
1024-block-0122
1024-block-0123
1024-block-0124
1024-block-0125
```