CSC3150 Project 4 Report

1. Program Environment

- CentOS Linux release 7.5.1804
- CUDA version: 11.7
- GPU: Nvidia Quadro RTX 4000 GPU x 1

2. Execution Steps

Before executing, make sure your device has CUDA.

source

Under `/source` directory:

```
sbatch ./slurm.sh
cat result.out
```

bonus

Under `/bonus` directory:

```
sbatch ./slurm.sh
cat result.out
```

3. Design Of Program

MACRO

```
typedef unsigned char uchar;
typedef uint32_t u32;
// op code
#define G_WRITE 1
#define G_READ 0
#define LS_D 0
#define LS_S 1
#define RM 2
#define MKDIR 3
#define CD 4
#define CD_P 5
#define RM_RF 6
#define PWD 7

// bit operations for fcb
#define RESET(x) ((x) = (0))

#define VALID(x) (x & 0x10000)
#define SET_VALID(x) ((x) |= (0x10000))
#define RESET_VALID(x) ((x) &= (0xfffeffff))
```

```
#define DIR(x) ((x & 0x20000))
#define SET_DIR(x) ((x) |= (0x20000))
#define RESET_DIR(x) ((x) &= (0xfffdffff))

#define WD(x) (x & 0x40000)
#define SET_WD(x) ((x) |= (0x40000))
#define RESET_WD(x) ((x) &= (0xfffbffff))

#define ROOT(x) (x & 0x80000)
#define SET_ROOT(x) ((x) |= (0x80000))
#define RESET_ROOT(x) ((x) &= (0xfff7ffff))

#define PARENT(x) ((unsigned int)x >>22)
#define SET_PARENT(x, p) {\
    (x) &= 0x003fffff;\
    (x) |= (p) << 22;\
}

// bit operations for file descriptor
#define WRITE(x) ((x) & (0x10000000))
#define SET_WRITE(x) ((x) |= (0x10000000))
#define RESET_WRITE(x) ((x) &= (0xefffffff))

#define READ(x) ((x) & (0x20000000))
#define SET_READ(x) ((x) |= (0x20000000))
#define RESET_READ(x) ((x) &= (0xdfffffff))

// Min Max
#define MAX(x, y) (((x) > (y)) ? (x) : (y))
#define MIN(x, y) (((x) < (y)) ? (x) : (y))
```

Data Structures

```
// file control block
typedef struct FCB {
    // 20 + 3 * 4 = 32 bytes
    char name[20]; // file name
    int size; // 0-15 bits size, 16-31 bits addr
    int create_modified; // 0-15 bits modified time, 16-31 bits create time
    int address; // 0-15 bits address, 16 valid bit, 17 directory bit, 18
working directory bit, 19 root directory bit, 22-31 bits parent index.
} FCB;

struct FileSystem {
    // Instead of using volume, we allocate memory space when creating the
FileSystem.
    int SUPERBLOCK[1024]; // 1024 * 4 bytes
    FCB FCBS[1024]; // 1024 * 32 bytes
    uchar BLOCKS[32768][32]; // 32768 * 32 bytes

    int SUPERBLOCK_SIZE;
    int FCB_SIZE;
    int FCB_ENTRIES;
```

```
    int STORAGE_SIZE;
    int STORAGE_BLOCK_SIZE;
    int MAX_FILENAME_SIZE;
    int MAX_FILE_NUM;
    int MAX_FILE_SIZE;
    int FILE_BASE_ADDRESS;
};
```

Operations

- Open
  - Find the corresponding FCB vie file name
  - If found, return the FBC's index
  - If not, find a empty FCB to store this newly opened file's info
  - Update FCB's create time and modified time, its parent.
- Read
  - Check if it is a valid read size.
  - Simply read each blocks' info
- Write
  - Check if it is a valid read size
  - Simply write to each block
  - Update FBC's size and modified time
- LS (by Date / Size)
  - Declare a new array for sorted FBCs
  - Copy all available FBCs to this array
  - Use bubble sort to sort this new array (by Date / Size)
  - Print info of the sorted array
- RM
  - Re-initialize its corresponding FCB to original state (this denotes that this FCB is not used)
  - Re-initialize its corresponding bitmap to zero.
- RM-RF
  - Use a queue to recursively remove the directory and all its sub files.
- MKDIR
  - simliar as `open` , but set directory bit in the corresponding fcb.
- CD
  - reset working directory bit of old fcb
  - set working directory bit of current fcb
- CD_P
  - set the current FS to its parent.
- PWD
  - Print out the path of working directory

Features

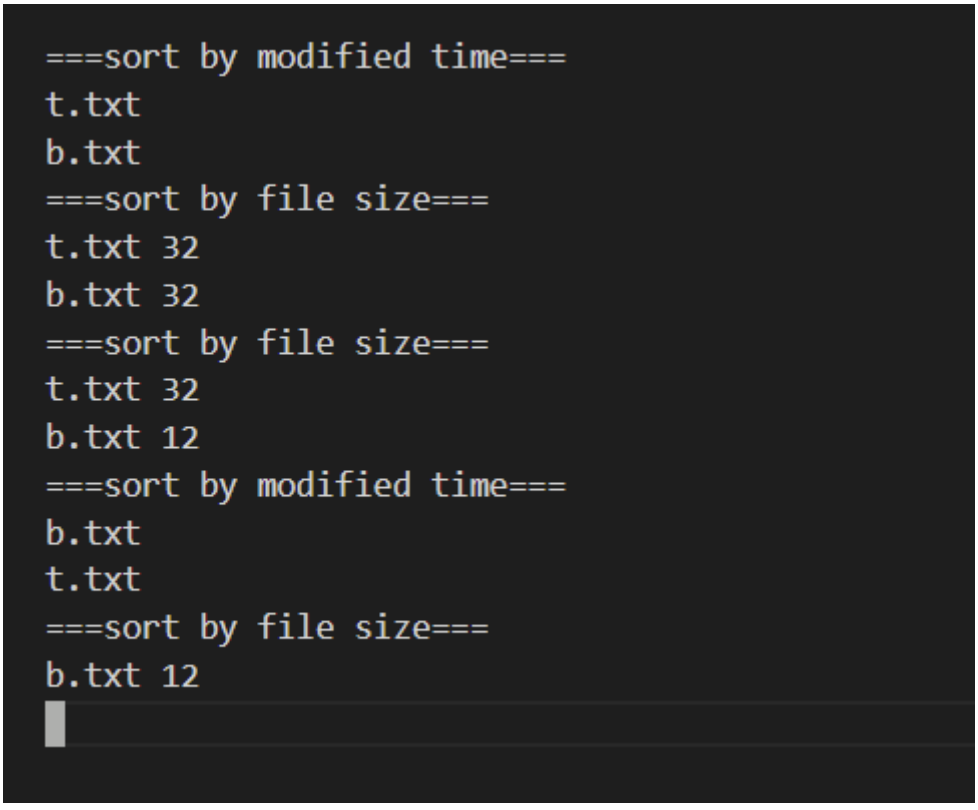
- no use extra space: only allocate a array to sort when needed. all information is stored in fcbs and superblocks.
- permission check: before read and write, check file descriptor's permission.
- support 1024kb file: max size of a file is 1024kb.
- support compact blocks: if there no more space to write, do a compact to collect segments to create continuous big enough space to write new files.
- support infinite operations: create time and modified time only has a max of $2^{16}-1$. If `gtime` reaches the max, we will sort fcbs by their create time and modified time to reduce the current `gtime` to a number less or equal to 1024(max file nums). As a reuslt, we support infinite operations without overflow.
- invalid operation warning: we automatically check invalid operations and stop the operations, including but not limited to read non-written blocks, remove a non-exist file, mkdir a already exist directory.

4. Problem Encountered

There are lots of details to notice. It takes much time to debug.

5. Screenshots

- test case 1



- test case 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===
b.txt
t.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===
*ABCDEFGHIJKLMNOPQR
)ABCDEFGHIJKLMNOPQR
(ABCDEFGHIJKLMNOPQR
'ABCDEFGHIJKLMNOPQR
&ABCDEFGHIJKLMNOPQR
b.txt
```

- test case 3

```
===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===
```

```
b.txt
t.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===
*ABCDEFGHIJKLMNOPQR
)ABCDEFGHIJKLMNOPQR
(ABCDEFGHIJKLMNOPQR
'ABCDEFGHIJKLMNOPQR
&ABCDEFGHIJKLMNOPQR
b.txt
===sort by file size===
~ABCDEFGHIJKLM 1024
}ABCDEFGHIJKLM 1023
|ABCDEFGHIJKLM 1022
{ABCDEFGHIJKLM 1021
zABCDEFGHIJKLM 1020
yABCDEFGHIJKLM 1019
xABCDEFGHIJKLM 1018
wABCDEFGHIJKLM 1017
vABCDEFGHIJKLM 1016
uABCDEFGHIJKLM 1015
tABCDEFGHIJKLM 1014
sABCDEFGHIJKLM 1013
rABCDEFGHIJKLM 1012
qABCDEFGHIJKLM 1011
pABCDEFGHIJKLM 1010
oABCDEFGHIJKLM 1009
...
=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
===sort by file size===
EA 1024
~ABCDEFGHIJKLM 1024
aa 1024
bb 1024
cc 1024
dd 1024
ee 1024
ff 1024
gg 1024
hh 1024
ii 1024
jj 1024
kk 1024
ll 1024
mm 1024
nn 1024
oo 1024
pp 1024
qq 1024
}ABCDEFGHIJKLM 1023
|ABCDEFGHIJKLM 1022
{ABCDEFGHIJKLM 1021
zABCDEFGHIJKLM 1020
yABCDEFGHIJKLM 1019
xABCDEFGHIJKLM 1018
wABCDEFGHIJKLM 1017
...
=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 case 4

```
triggering gc
===sort by modified time===
1024-block-1023
1024-block-1022
1024-block-1021
1024-block-1020
1024-block-1019
1024-block-1018
1024-block-1017
1024-block-1016
1024-block-1015
1024-block-1014
1024-block-1013
```

CSC3150_2022FALL > Assignment4 > source > result.out

```
1032    1024-block-0014
1033    1024-block-0013
1034    1024-block-0012
1035    1024-block-0011
1036    1024-block-0010
1037    1024-block-0009
1038    1024-block-0008
1039    1024-block-0007
1040    1024-block-0006
1041    1024-block-0005
1042    1024-block-0004
1043    1024-block-0003
1044    1024-block-0002
1045    1024-block-0001
1046    1024-block-0000
1047
```

- bonus case

```
72
73  ===sort by modified time===
74  t.txt
75  b.txt
76  ===sort by file size===
77  t.txt 32
78  b.txt 32
79  ===sort by modified time===
80  app d
81  t.txt
82  b.txt
83  ===sort by file size===
84  t.txt 32
85  b.txt 32
86  app 0 d
87  ===sort by file size===
88  ===sort by file size===
89  a.txt 64
90  b.txt 32
91  soft 0 d
92  ===sort by modified time===
93  soft d
94  b.txt
95  a.txt
96  /app/soft
97  ===sort by file size===
98  B.txt 1024
99  C.txt 1024
100 D.txt 1024
101 A.txt 64
102 ===sort by file size===
103 a.txt 64
104 b.txt 32
105 soft 24 d
106 /app
107 ===sort by file size===
108 t.txt 32
109 b.txt 32
110 app 17 d
111 ===sort by file size===
112 a.txt 64
113 b.txt 32
114 ===sort by file size===
115 t.txt 32
116 b.txt 32
117 app 12 d
118
```

Things learned

I learned the structure of filesystem.