Report for CSC3150 assignment 4

Feng Yutong 120090266

Design

- Intro

  This program implements the file system management. It supports open/read/write file, implement filesystem consisting of volumn control block, directory structure(bonus), FCB, contents of files. This program uses contiguous allocate and save the space of bitmap.

- Implementation

  - Abstract Data Type representation

    - Contents of files (Storage block: 128k): `fs->SB`
    - File control block (30k): `fs->FCB`
      - source

```
 struct FCB {
  u16 create; // create time
  u16 modified; // last modified time
  u16 priority; // order in sorting
  u16 size; // file size
  u16 st_block; // start block index in storage block
  char file_name[20]; // file name
};
```

      - bonus (actual use: 35k, using the space of VCB)

```
struct FCB {
  u16 create;
  u16 modified;
  u16 priority;
  u32 size;
  u16 st_block;
  bool dir; // whether is directory
  char file_name[20];
  u16 parent; // parent file descriptor
};
```

- Volummn control block (4k): `fs->VCB` (actual use: 4byte)
  - source & bonus

```
struct VCB {
  u16 tot_file_num; // total file number in storage block
  u16 empty_st; // the start of empty block
  // no bitmap
};
```

- Directory structure (bouns)
  - Tree structure store in FCB
  - File system

```
fs->root (FCB *): root directory
fs->current_dir (FCB *): which directory user is at
fs->FCB_now (FCB *): the last node file/directory under
the current directory
```

- Theoretical time complexiy
  - open/read/write: O(L)
  - sort: O(L^2)
- Function implement (Source)
  - `fs_open`
    1. Find file with the same name, return file descriptor
    2. Under write mode, file not found: create a new file, update VCB, FCB, storage block
    3. Otherwise return error
  - `fs_read`
    4. Use file descriptor to find the start block and block number
    5. Write data to storage block
  - `fs_write`
    6. Use file descriptor to find the start block and block number
    7. Calculate the change block number (new block number - orignal block number); Move Storage by the change block using `memcpy`

8. Update VCB and FCB
9. Find location in strorage block and copy data to output buffer

- `fs_gsys(LS_S / LS_D)` 10. Set every block to have a invalid priority 11. Iterate i form 0 to tot_file_number: find a file with invalid priority is the maximum according to rules (LS_S: maximum is file with maximum size; LS_D: maximum is file with lastest modified time), and set its priority to be i 12. Print out file name (and size) by the increasing order of priority.

- `fs_gsys(RM)` 13. Find the file by name and calculate the change block number 14. Update VCB and FCB 15. Move storage block by change block number using `memcpy` and `memset`

- Feature (source)

  - Since the storage block guarantees contiguous allocation and do compaction after each write and remove operation, there's no need to use bitmap. We can just use empty block number to indicate the storage situation. Here we add two more feature: total file number and empty block start position for easiness and debugging of the program.
  - This design always has files stored in `early create early start location` rule. Since we do compaction after each operation, the file created earlier will have a smaller file descriptor. We also do not change this feature in sorting. We do not change file location in storage block and VCB, we only use `priority` to indicate its position in sorting. Thus, when we change a file size, it only affects the file stored in the block after it. So we can just use `memset` to move file storage.

- Function implement (Bonus)

  - `fs_open` 16. Find file with the same name, return file descriptor and under current directory 17. Under write mode, file not found: create a new file 18. Update VCB, FCB, SB 19. Otherwise return error
  - `fs_read`
    - Same as source
  - `fs_write`
    - Same as source
  - `fs_gsys(LS_S / LS_D)`
    - Main idea is same as source, but we only print file under `fs->current_dir`
  - `fs_gsys(RM)`
    - Main idea is same as source, but we only delete file under `fs->current_dir`
  - `fs_gsg(RM_RF)`
    - We recursively delete files and directories under it.

            - For file, call `fs_gsg(RM)`
            - For directory, call `fs_gsg(RM_RF)`
        - At last, we delte this directory as a file by calling `fs_gsg(RM)`
        - Update `fs->current_dir` and `gparent` to indicate the directory level
    - `fs_gsg(MKDIR)`
        - Similar to creation in `fs_open`, but we set a directory to have a fixed block number 1 no matter how much size it actually has. We also set the flag `dir` to be true indicate it's a directory
    - `fs_gsg(CD)` and `fs_gsg(CD_P)`
        - Change `fs->current_dir` with parent information
    - `fs_gsg(PWD)`
        - Since maximum depth is 3, we only need to a char[60] to store filename, recursively move up `fs->current_dir->parent` and store the directory names in char array
        - Print out the names in char array in reverse order

- Feature (Bonus)

    - For each file, we add a `parent` attribute point to its father's file descriptor and use `fs->current_dir` to deal with the same name under different directory.
    - Same as source: `early create early start location`. Compaction only consider file/directory with larger file descriptor
    - Directory has fixed block number 1. When creating/deleting, change the parent directory size. Since the block number is fixed, no need to do compact because of the change block number of directory.

Environment and execution

- Environment

    - OS: CentOS Linux release 7.5.1804 (Core)
    - NVIDIA-SMI 515.65.01
    - Driver Version: 515.65.01
    - CUDA Version: 11.7

- Execution

    Modify `user_program` and run `bash slurm.sh`. It will print result in terminal and dump a binary file `snapshot.bin`. Use vim to check for correctness. Same is for bonus.

Output

- Case 1

- Output

```
===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
```

We can see that the page fault number is 4096 + 1 + 4096 = 8193

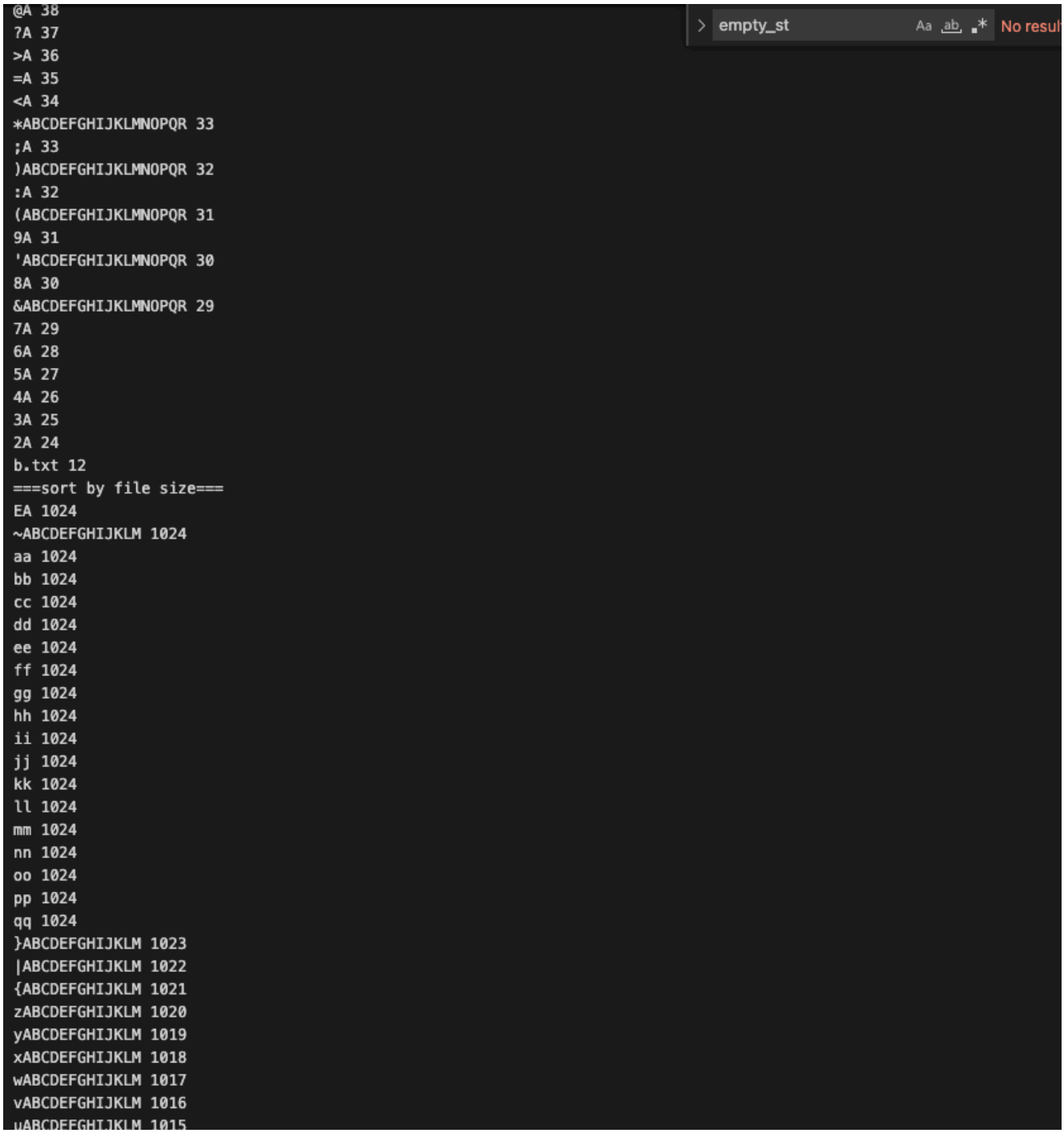
- Case 2
  - Output

```
===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
```

- Case 3
- Partial output

```
===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
```

...

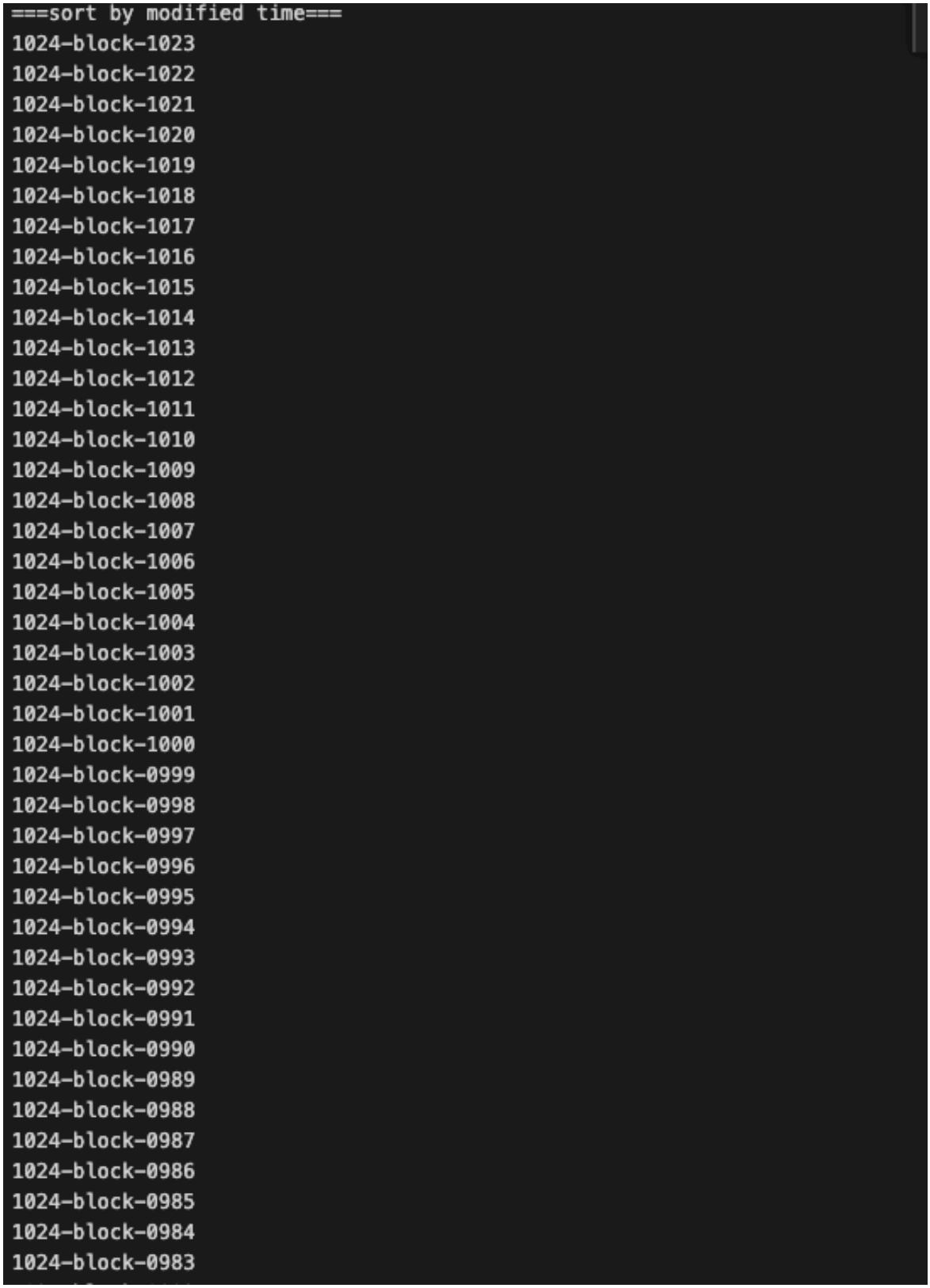

@A 38
?A 37
>A 36
=A 35
<A 34
*ABCDEFGHIJKLMNOPQR 33
;A 33
)ABCDEFGHIJKLMNOPQR 32
:A 32
(ABCDEFGHIJKLMNOPQR 31
9A 31
'ABCDEFGHIJKLMNOPQR 30
8A 30
&ABCDEFGHIJKLMNOPQR 29
7A 29
6A 28
5A 27
4A 26
3A 25
2A 24
b.txt 12
===sort by file size===
EA 1024
~ABCDEFGHIJKLM 1024
aa 1024
bb 1024
cc 1024
dd 1024
ee 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
vABCDEFGHIJKLM 1016
empty_st
Aa
No resul

...

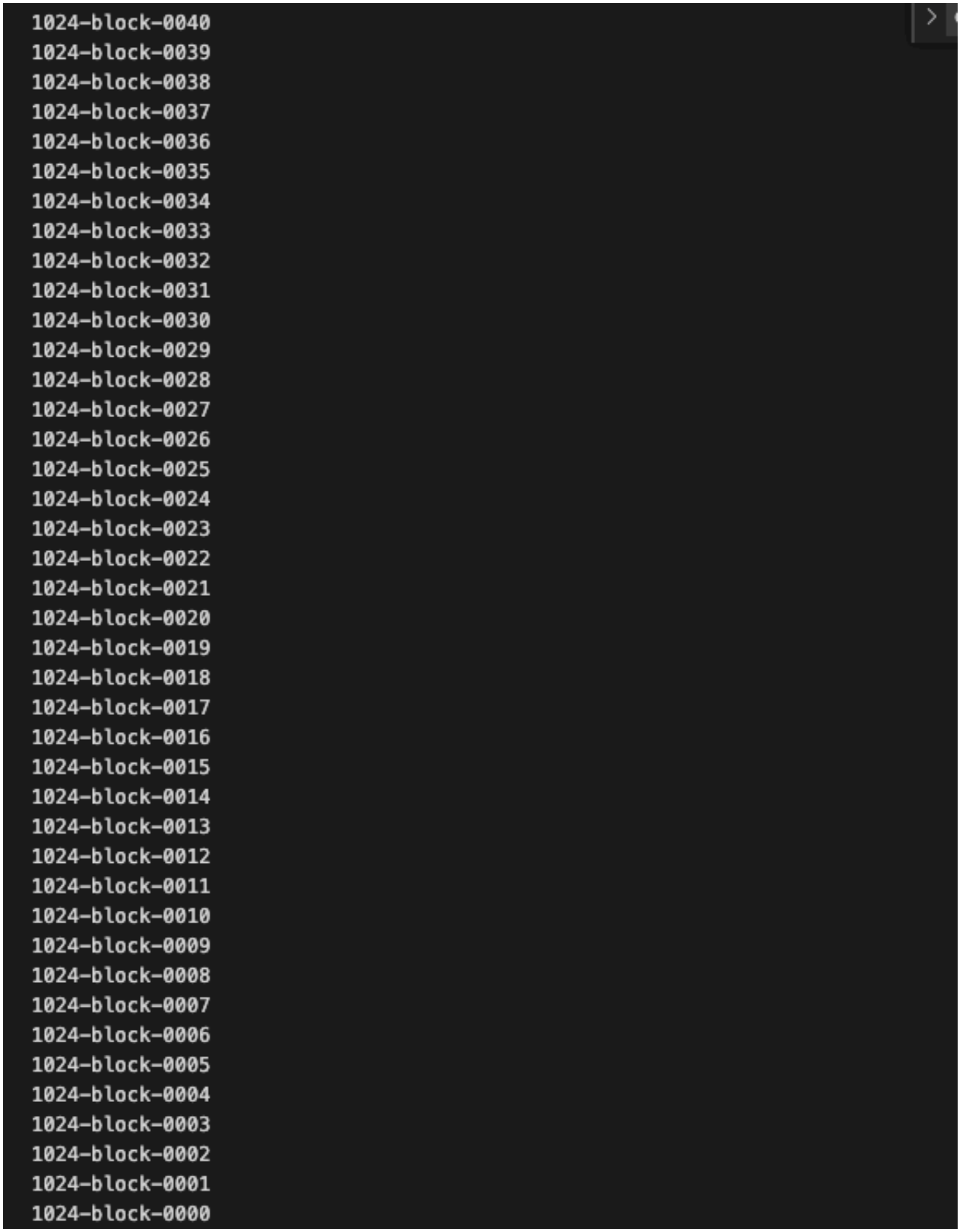
```
\A 66
[A 65
ZA 64
YA 63
XA 62
WA 61
VA 60
UA 59
TA 58
SA 57
RA 56
QA 55
PA 54
OA 53
NA 52
MA 51
LA 50
KA 49
JA 48
IA 47
HA 46
GA 45
FA 44
DA 42
CA 41
BA 40
AA 39
@A 38
?A 37
>A 36
=A 35
<A 34
*ABCDEFGHIJKLMNOPQR 33
;A 33
)ABCDEFGHIJKLMNOPQR 32
:A 32
(ABCDEFGHIJKLMNOPQR 31
9A 31
'ABCDEFGHIJKLMNOPQR 30
8A 30
&ABCDEFGHIJKLMNOPQR 29
7A 29
6A 28
5A 27
4A 26
3A 25
2A 24
b.txt 12
```

- Case 4

- Partial output



...


1024-block-0040
1024-block-0039
1024-block-0038
1024-block-0037
1024-block-0036
1024-block-0035
1024-block-0034
1024-block-0033
1024-block-0032
1024-block-0031
1024-block-0030
1024-block-0029
1024-block-0028
1024-block-0027
1024-block-0026
1024-block-0025
1024-block-0024
1024-block-0023
1024-block-0022
1024-block-0021
1024-block-0020
1024-block-0019
1024-block-0018
1024-block-0017
1024-block-0016
1024-block-0015
1024-block-0014
1024-block-0013
1024-block-0012
1024-block-0011
1024-block-0010
1024-block-0009
1024-block-0008
1024-block-0007
1024-block-0006
1024-block-0005
1024-block-0004
1024-block-0003
1024-block-0002
1024-block-0001
1024-block-0000

- bonus

```
===sort by modified time===
t.txt
b.txt
===sort by file size===
t.txt 32
b.txt 32
===sort by modified time===
app d
t.txt
b.txt
===sort by file size===
t.txt 32
b.txt 32
app 0 d
===sort by file size===
===sort by file size===
a.txt 64
b.txt 32
soft 0 d
===sort by modified time===
soft d
b.txt
a.txt
/app/soft
===sort by file size===
B.txt 1024
C.txt 1024
D.txt 1024
A.txt 64
===sort by file size===
a.txt 64
b.txt 32
soft 24 d
/app
===sort by file size===
t.txt 32
b.txt 32
app 17 d
===sort by file size===
a.txt 64
b.txt 32
===sort by file size===
t.txt 32
b.txt 32
app 12 d
```

Learning

When implementing the basic version, I learned to use `reinterpret_cast` to use the memory of superblock as struct, so the programming is more intuitive (value name is more intuitive than the bit

length presentation). I also improve the VCB memory use, using 10 bytes instead of building a bitmap. To finish the program, I write a `dump` function to print out the whole file system information. Each operation I call `dump` to see the change, which helps me to debug efficiently.

When implementing the bonus, I add a global index `fs->current_dir` to simplify programming and avoid same file names under different directories. I can still use `dump` to fix it one by one.