CSC3150 Project 4 File System

Tong Zhen 120090694

1. Program Environment

Linux Kernel Version

```
[120090694@node21 HM3]$ uname -r
3.10.0-862.el7.x86_64
```

cuda version

```
[120090694@node21 HM3]$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Jun__8_16:49:14_PDT_2022
Cuda compilation tools, release 11.7, V11.7.99
Build cuda_11.7.r11.7/compiler.31442593_0
```

2. Program Structure Design

In this assignment, a range of volume array is used to simulate the volume control block, file control block, and contents of the file. The program used cuda memory storage to simulate the concepts below.

Operating System Concept	Size & Type Description
Disk (1060KB) = VCB + FCB + Storage	`uchar[1085440]`
VCB (Super blocks) (4KB)	`uchar [fs->SUPERBLOCK_SIZE]`
FCB Blocks (32KB)	`uchar [fs->FCB_ENTRIES*fs->FCB_SIZE]`
FCB (32 Bytes)	`uchar [32]`
Storage blocks (1024KB)	`uchar [1048576]`
Storage block (32 Bytes)	`uchar [32]`

Bit Map Structure

The VCB(Super blocks) is build in Bit Map format. A Bit Map is an array of number, whose every bit represent for a content storage condition. For example, `01001001` represents for 8 blocks of storage, the 1st, 3rd, 4th, 6th, 7th are empty storage block, and 2nd, 5th, 8th block are occupied. Each `uchar` number is a 8 bit representation for 8 blocks.

FCB Structure

The File Control Block (FCB), is a structure used to link logical file conception and physical data storage. This project design the FCB by 32 `uchar` number in this way:

1. The 0-20 number is the file name, each number represents for a letter
2. Data larger than 255 is represented by 2 `uchar` number, because the `uchar` is limited to max = 255

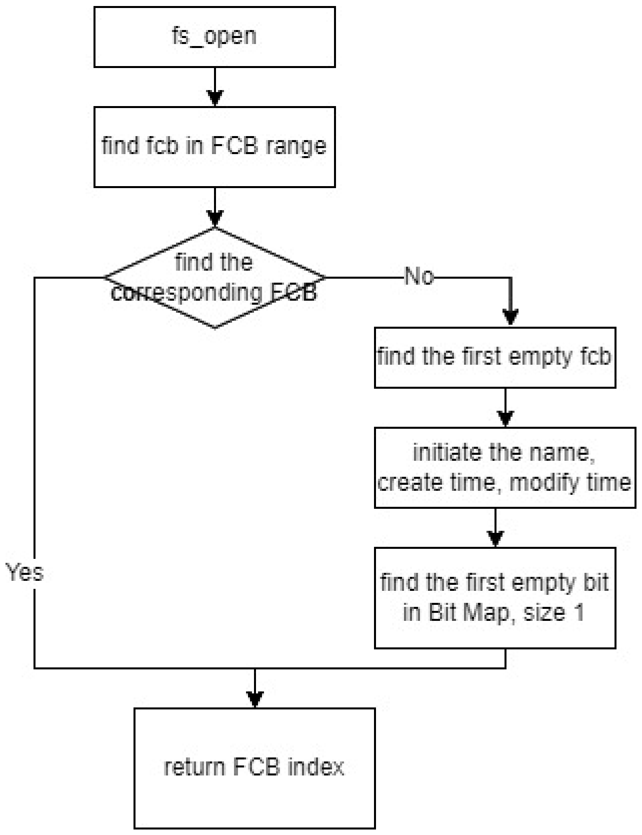
$$data = uchar_1 \times 256 + uchar_2$$

   The create time, modify time, file size, map bit size, map bit location are all data larger than 255. They were recored by the 20-21, 22-23, 24-25, 26-27, 28-29 `uchar` number in one FCB block.

```
uchar *fcb=start_fcb + fcb_idx * fs->FCB_SIZE;
uchar *fcb_name = target_fcb;
uchar *create_time = target_fcb + 20;
uchar *modify_time = target_fcb + 22;
uchar *file_size = target_fcb + 24;
uchar *bit_size = target_fcb + 26;
uchar *bit_location = target_fcb + 28;
```

Function

fs_open(FileSystem *fs, char *s, int op)

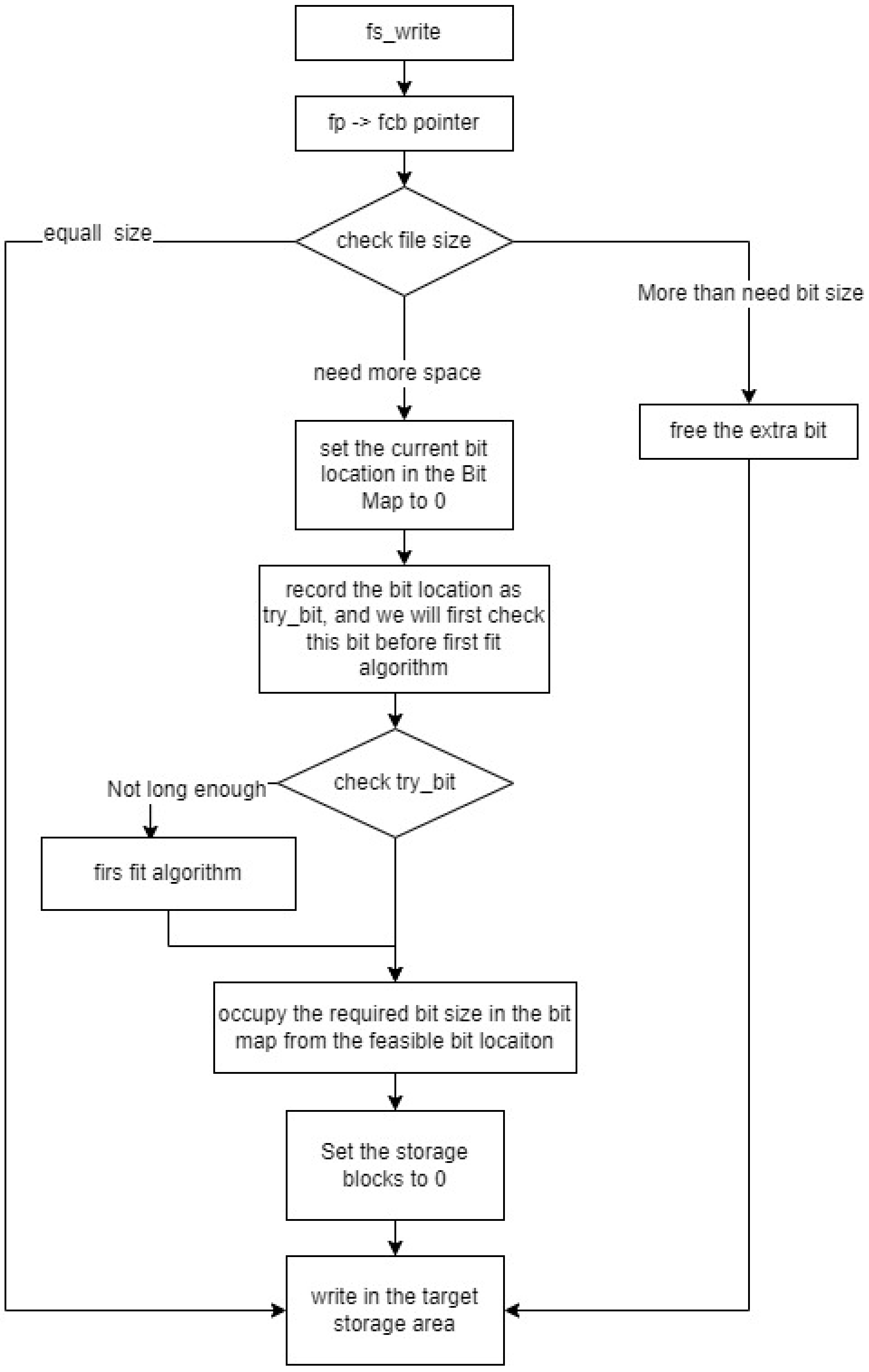

fs_open
find fcb in FCB range
find the corresponding FCB
No
Yes
find the first empty fcb
initiate the name, create time, modify time
find the first empty bit in Bit Map, size 1
return FCB index

When the fcb is not found in the FCB region. We will allocate one spare fcb for it, and find the first empty bit in the Bit Map for it.

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

With the input fp as a FCB index, we can find the FCB pointer in the volume. The FCB pointer will give the location of the data and how many blocks it last.

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


fs_write
fp -> fcb pointer
check file size
equall size
More than need bit size
need more space
free the extra bit
set the current bit location in the Bit Map to 0
record the bit location as try_bit, and we will first check this bit before first fit algorithm
check try_bit
Not long enough
firs fit algorithm
occupy the required bit size in the bit map from the feasible bit locaiton
Set the storage blocks to 0
write in the target storage area

When writing a file, the current allocated size may not fit. If more storage were given, set the extra bit map to zero. When the current allocated storage size is less than the required size, we need first set the allocated storage to zero.

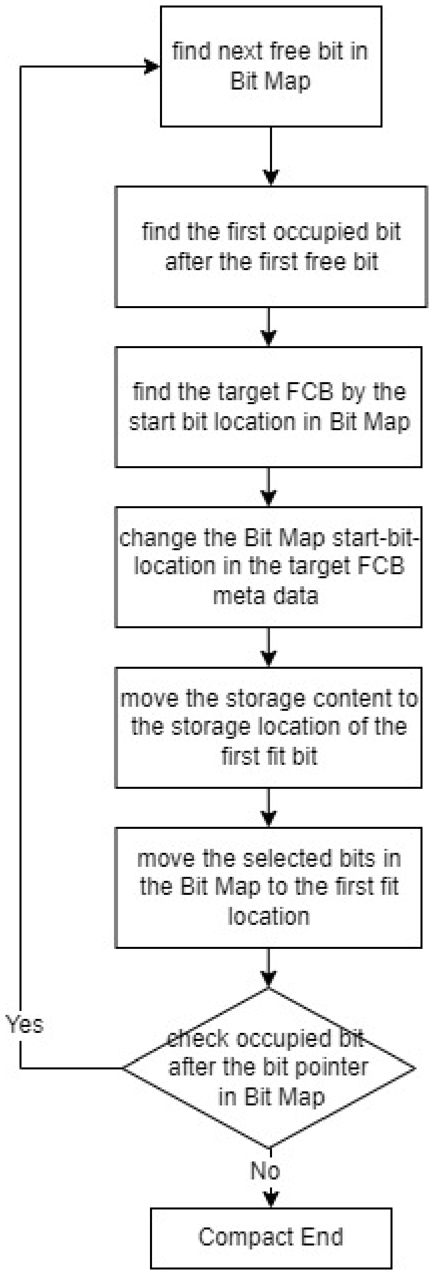
find_first_fit(FileSystem *fs, int bits_num)

As we mentioned above, the suitable bit location with a certain bit size should be found by first fit (first fit has similar performance with best fit algorithm). We search the Bit Map, and find the first contiguous given size bit location.

find 5 contiguous bit in BIT MAP

0	0	1	1	1	1	0	0
0	0	0	1	1	1	0	1
1	1	1	1	0	0	0	1
1	1	0	1	0	1	1	0

compact_storage(FileSystem *fs)


find next free bit in Bit Map
find the first occupied bit after the first free bit
find the target FCB by the start bit location in Bit Map
change the Bit Map start-bit-location in the target FCB meta data
move the storage content to the storage location of the first fit bit
move the selected bits in the Bit Map to the first fit location
check occupied bit after the bit pointer in Bit Map
Yes
No
Compact End

When the process encounters no more suitable bit in the Bit Map for writing data, a compact occur would occur. Its motivation is to squeeze the occupied bit to the front of the VCB tightly. Things worth to be mention:

1. The FCB corresponding to the start-bit location is search in FCB region with O(N) time.
2. The physical storage of the file should be moved to the new indicated region.

fs_gsys(FileSystem *fs, int op)

The `op` can be `LS_D`(modified time), or `LS_S`(file size). The program first scan all the fcbs in the FCB reigion, add them into an array, and do bubble sort in descending order.

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

The `op` is `RM` here, do remove. With given file name, the file system find the corresponding fcb. The bit map meta data is set to zero, the storage of the fcb is set to zero, and the FCB it self is freed.

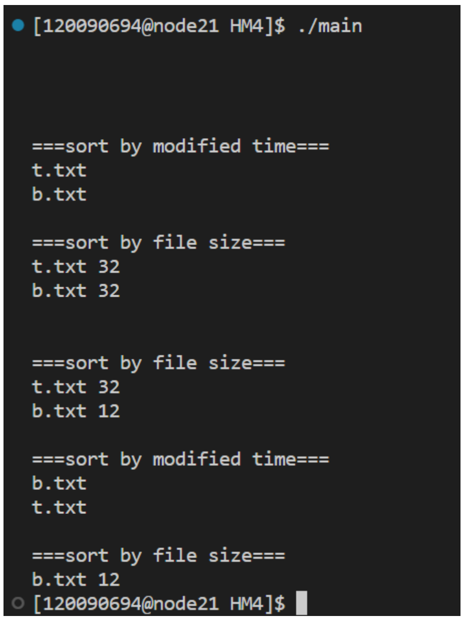
Run

Two way to run the program

```
make
./test
```

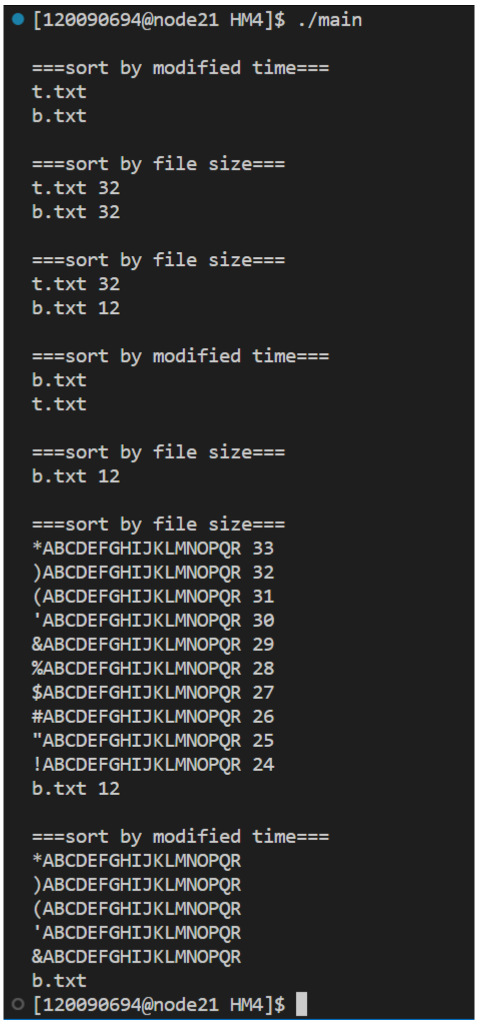
```
./slurm.sh
./test
```

case 1



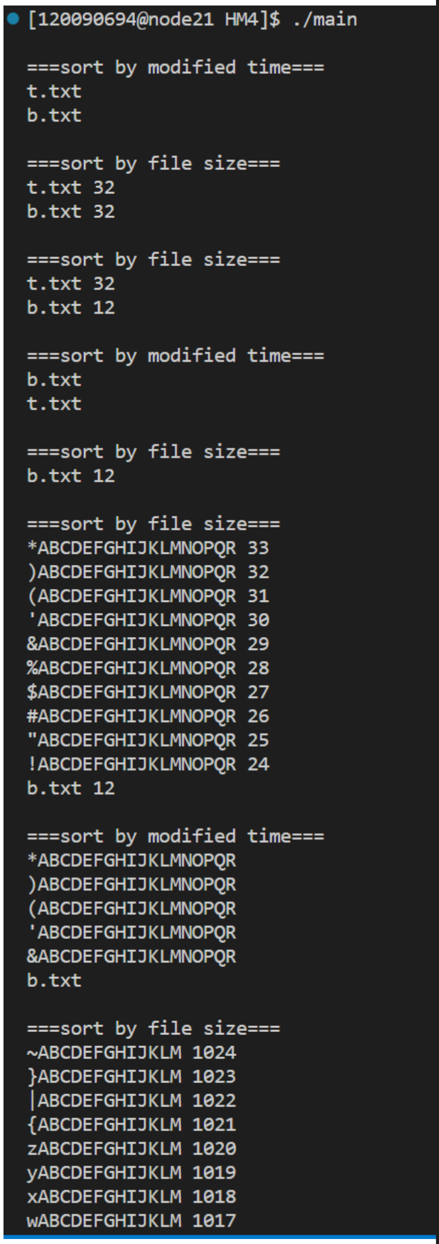
```
[120090694@node21 HM4]$ ./main



===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
[120090694@node21 HM4]$
```

case 2

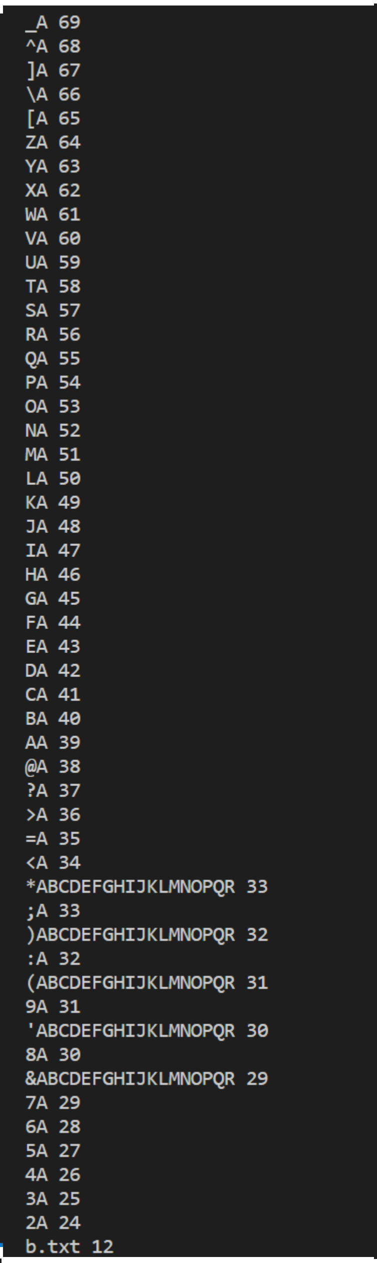


```
● [120090694@node21 HM4]$ ./main

  ===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
○ [120090694@node21 HM4]$ 
```

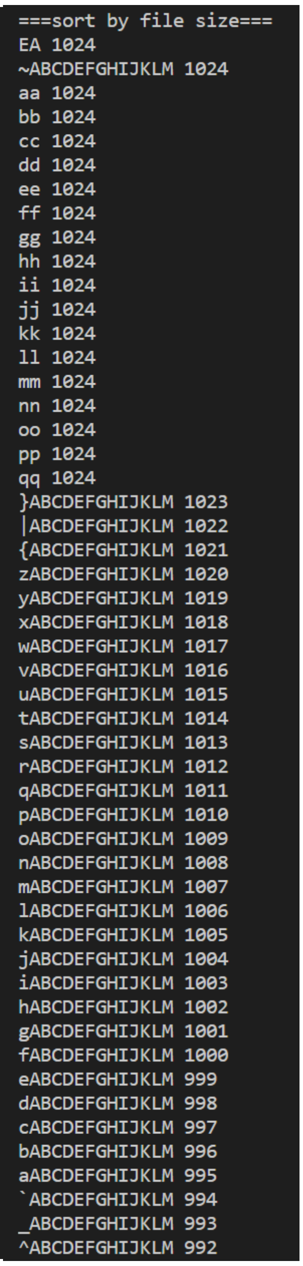
case 3



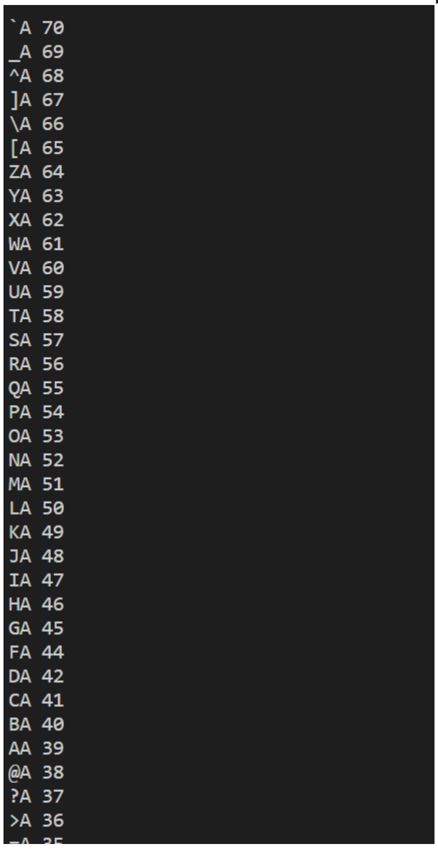
```
● [120090694@node21 HM4]$ ./main

===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
```



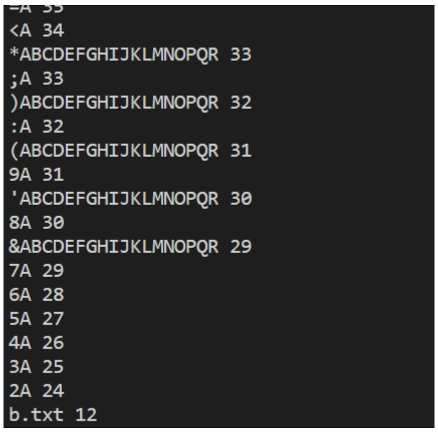
```
_A 69
^A 68
]A 67
\A 66
[A 65
ZA 64
YA 63
XA 62
WA 61
VA 60
UA 59
TA 58
SA 57
RA 56
QA 55
PA 54
OA 53
NA 52
MA 51
LA 50
KA 49
JA 48
IA 47
HA 46
GA 45
FA 44
EA 43
DA 42
CA 41
BA 40
AA 39
@A 38
?A 37
>A 36
=A 35
<A 34
*ABCDEFGHIJKLMNOPQR 33
;A 33
)ABCDEFGHIJKLMNOPQR 32
:A 32
(ABCDEFGHIJKLMNOPQR 31
9A 31
'ABCDEFGHIJKLMNOPQR 30
8A 30
&ABCDEFGHIJKLMNOPQR 29
7A 29
6A 28
5A 27
4A 26
3A 25
2A 24
b.txt 12
```



```
===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
uABCDEFGHIJKLM 1015
tABCDEFGHIJKLM 1014
sABCDEFGHIJKLM 1013
rABCDEFGHIJKLM 1012
qABCDEFGHIJKLM 1011
pABCDEFGHIJKLM 1010
oABCDEFGHIJKLM 1009
nABCDEFGHIJKLM 1008
mABCDEFGHIJKLM 1007
lABCDEFGHIJKLM 1006
kABCDEFGHIJKLM 1005
jABCDEFGHIJKLM 1004
iABCDEFGHIJKLM 1003
hABCDEFGHIJKLM 1002
gABCDEFGHIJKLM 1001
fABCDEFGHIJKLM 1000
eABCDEFGHIJKLM 999
dABCDEFGHIJKLM 998
cABCDEFGHIJKLM 997
bABCDEFGHIJKLM 996
aABCDEFGHIJKLM 995
`ABCDEFGHIJKLM 994
_ABCDEFGHIJKLM 993
^ABCDEFGHIJKLM 992
```

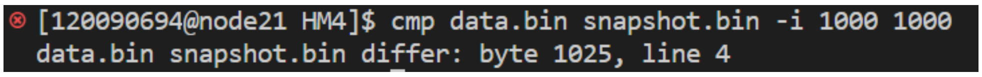


```
`A 70
_A 69
^A 68
]A 67
\A 66
[A 65
ZA 64
YA 63
XA 62
WA 61
VA 60
UA 59
TA 58
SA 57
RA 56
QA 55
PA 54
OA 53
NA 52
MA 51
LA 50
KA 49
JA 48
IA 47
HA 46
GA 45
FA 44
DA 42
CA 41
BA 40
AA 39
@A 38
?A 37
>A 36
```

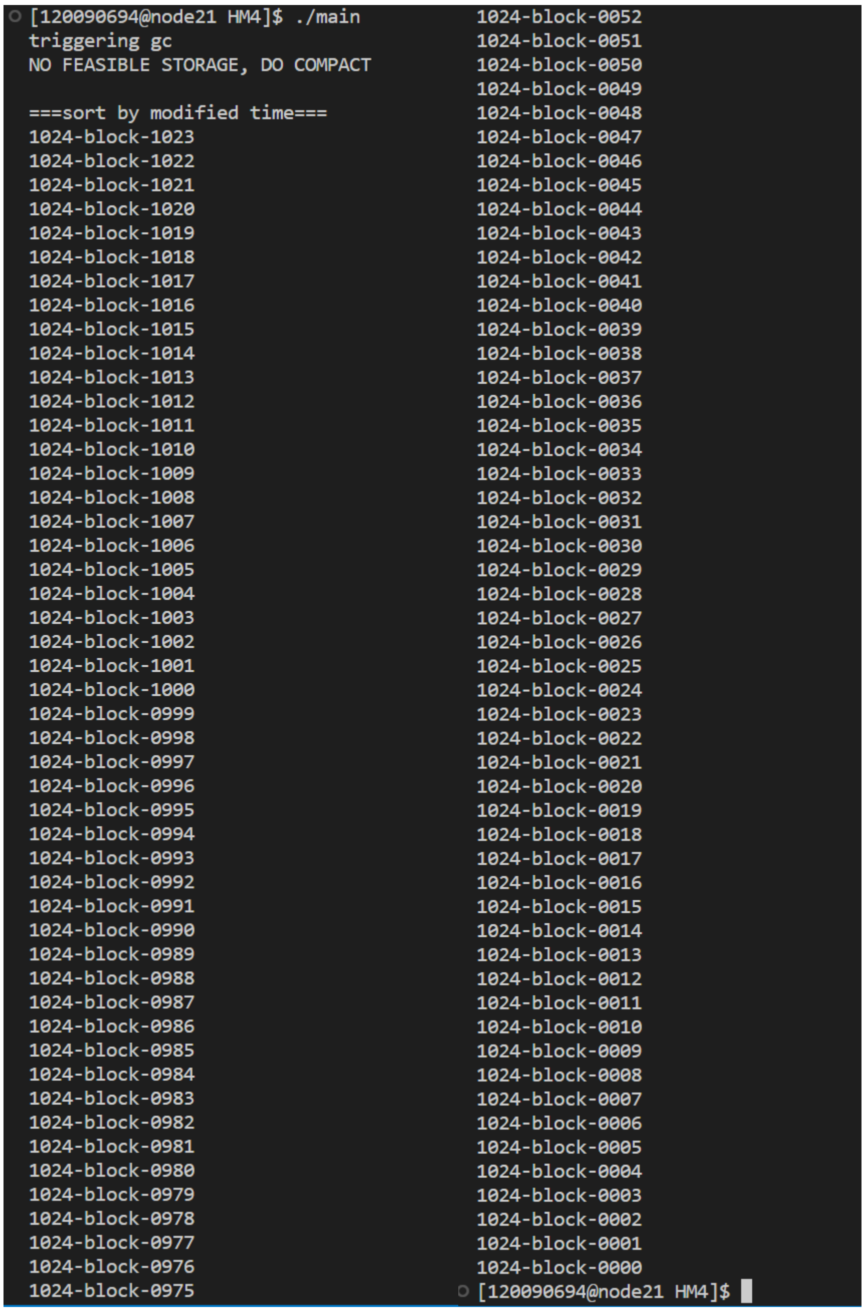

```
=A 35
<A 34
*ABCDEFGHIJKLMNOPQR 33
;A 33
)ABCDEFGHIJKLMNOPQR 32
:A 32
(ABCDEFGHIJKLMNOPQR 31
9A 31
'ABCDEFGHIJKLMNOPQR 30
8A 30
&ABCDEFGHIJKLMNOPQR 29
7A 29
6A 28
5A 27
4A 26
3A 25
2A 24
b.txt 12
```

Compare part of the written and read data with the input.

```
cmp data.bin snapshot.bin -i 1000 1000
```


```
[120090694@node21 HM4]$ cmp data.bin snapshot.bin -i 1000 1000
data.bin snapshot.bin differ: byte 1025, line 4
```

case 4

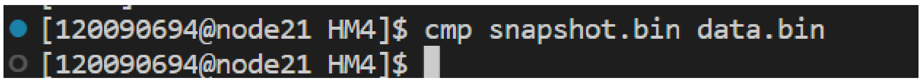


```
[120090694@node21 HM4]$ ./main
triggering gc
NO FEASIBLE STORAGE, DO COMPACT

===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
1024-block-1012
1024-block-1011
1024-block-1010
1024-block-1009
1024-block-1008
1024-block-1007
1024-block-1006
1024-block-1005
1024-block-1004
1024-block-1003
1024-block-1002
1024-block-1001
1024-block-1000
1024-block-0999
1024-block-0998
1024-block-0997
1024-block-0996
1024-block-0995
1024-block-0994
1024-block-0993
1024-block-0992
1024-block-0991
1024-block-0990
1024-block-0989
1024-block-0988
1024-block-0987
1024-block-0986
1024-block-0985
1024-block-0984
1024-block-0983
1024-block-0982
1024-block-0981
1024-block-0980
1024-block-0979
1024-block-0978
1024-block-0977
1024-block-0976
1024-block-0975
```

```
1024-block-0052
1024-block-0051
1024-block-0050
1024-block-0049
1024-block-0048
1024-block-0047
1024-block-0046
1024-block-0045
1024-block-0044
1024-block-0043
1024-block-0042
1024-block-0041
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
[120090694@node21 HM4]$
```

Compare the output `snapshot.bin` with the input.

```
[120090694@node21 HM4]$ cmp snapshot.bin data.bin
```



```
[120090694@node21 HM4]$ cmp snapshot.bin data.bin
[120090694@node21 HM4]$
```

Bonus

Directory Structure:

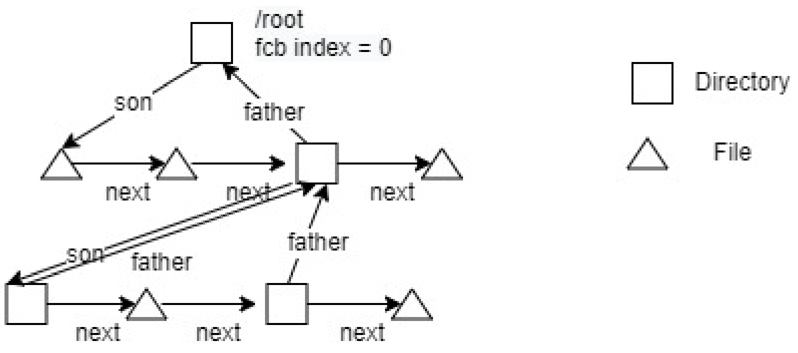
The directory concept is similar to file concept, except that it doesn't need to store data. The name, create time and modify time meta data location in FCB are the same. The difference is as follows:

1. The `son_fcb_idx` record the first son fcb index in the FCB region.
2. the son number is the length of the singly linked list of sons (max length 50)
3. `father` is the fcb index of the directory's parent directory
4. `next_bro` is the next sibling's fcb index.

5. The File System has a new variable called `cur_DIRfcb_idx` a `int` indicate the current directory fcb index.

```
                  uchar *create_time = dir_fcb + 20;
                  uchar *modify_time = dir_fcb + 22;
                  gtime++;
                  *create_time = gtime/256;
                  *(create_time + 1) = gtime%256;
                  *modify_time = gtime/256;
                  *(modify_time + 1) = gtime%256;

                  uchar *son_fcb_idx = dir_fcb + 24;
                  uchar *son_number = dir_fcb + 26;
                  uchar *father = dir_fcb + 28;
                  *father = fs->cur_DIRfcb_idx/256;
                  *(father +1) = fs->cur_DIRfcb_idx%256;
                  uchar *next_bro = dir_fcb + 30;
```

The File System structure with directory tree is:



The affiliation of the tree structure is constructed by singly linked list. The sort and remove operation are based on this structure.

Functions

MKDIR

Under the current directory , find a empty fcb and initialize the father of the new directory-fcc as `cur_DIRfcb_idx`.

CD

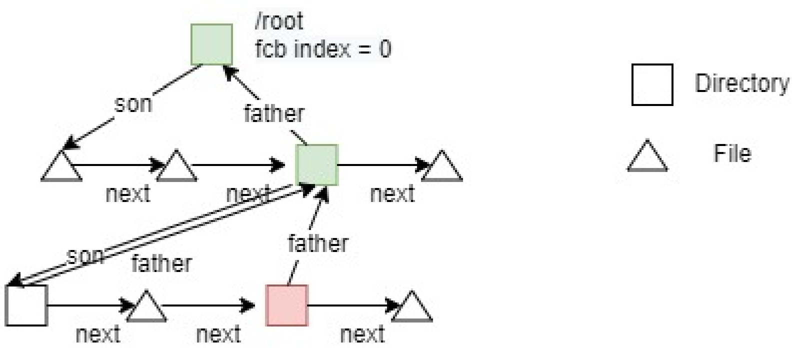
Set the `cur_DIRfcb_idx` to the given fcb index

CD_P

Find the father of the `cur_DIRfcb_idx` fcb and set the `cur_DIRfcb_idx` to the father fcb index

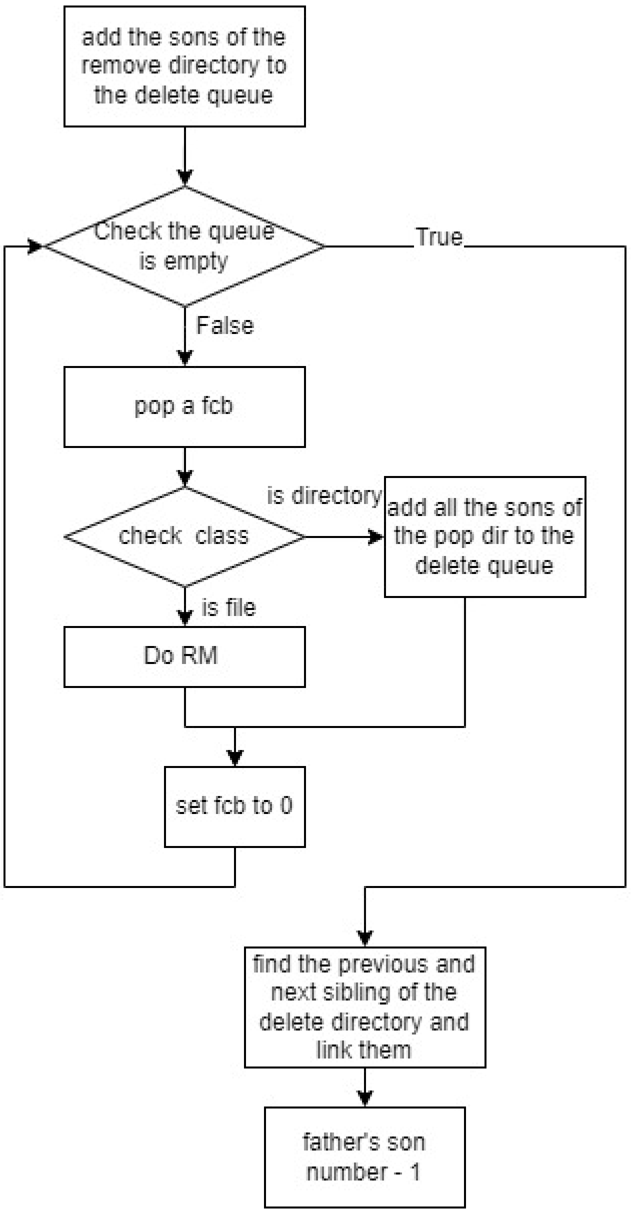
PWD

find the father of the directory-fcb pointer until its fcb index == 0 (the /root fcb when the File System initialed).



RM_RF

Remove the given directory and all its subdirectories and files recursively. Instead of really do it recursively, the program builds a queue to remove with while loop.


add the sons of the remove directory to the delete queue
Check the queue is empty
True
False
pop a fcb
check class
is directory
add all the sons of the pop dir to the delete queue
is file
Do RM
set fcb to 0
find the previous and next sibling of the delete directory and link them
father's son number - 1

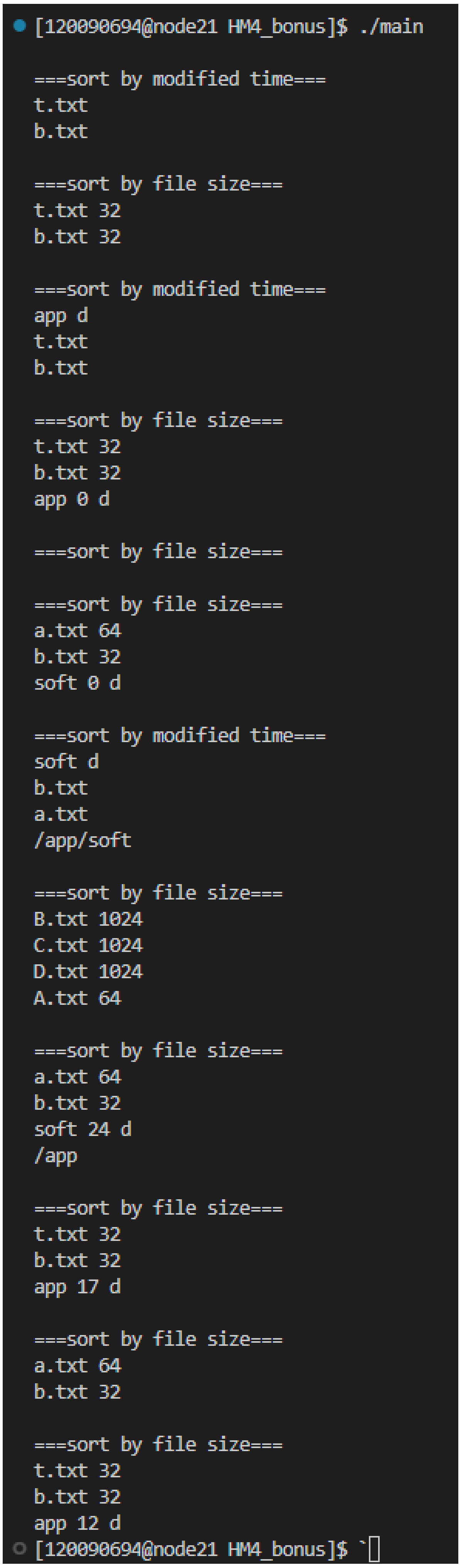
Run

Two way to run the bonus

```
make
./test
```

```
./slurm.sh
./test
```

Output



```
● [120090694@node21 HM4_bonus]$ ./main

===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
○ [120090694@node21 HM4_bonus]$ `
```