CSC3150 Assignment 4: File System

120090211

Tao Chujun

Overview:

1. Environment information

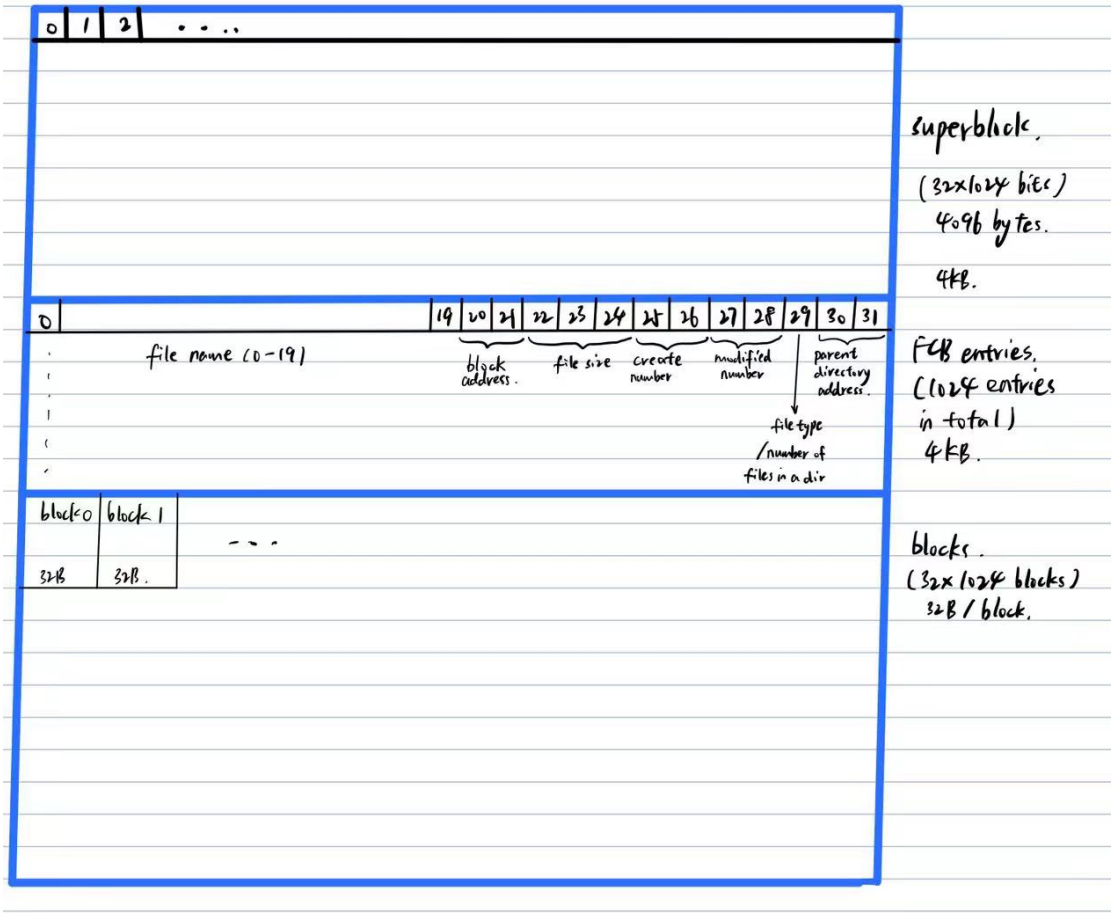
Item	Configuration / Version
System Type	x86_64
Opearing System	CentOS Linux release 7.5.1804
CPU	Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz 20 Cores, 40 Threads
Memory	100GB RAM
GPU	Nvidia Quadro RTX 4000 GPU x 1
CUDA	11.7
GCC	Red Hat 7.3.1-5
CMake	3.14.1

2. Execution step

```
sh slurm.sh
```

3. File system design

1) Volume Partition



In the standard part, the volume is divided into 3 parts. The superblock takes the first 4096 bytes in the volume. Every bit is a sign to record whether the block is in use or not.
The second part stores FCB.
For each FCB entry:
Byte 0-19: Record the file name.
Byte 20-21: The address of the first block used by the file.
Byte 22-24: The file size. (Specifically, byte 22 will be set as 0xff if it is invalid)
Byte 25-26: The create time.
Byte 27-28: The modified time.
Byte 29: Set as 0xff to indicate it is a file.
Initialization:

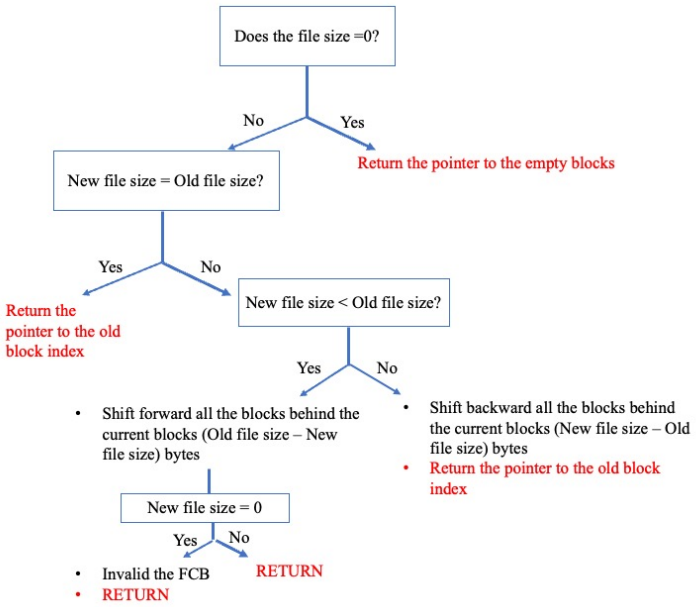
```
for (int i=0;i<FCB_ENTRIES;i++){
  // set file size to 0xffffff
    fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * i + 22] = 0xff;
  // set the file type
    fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * i + 29] = 0xff;
}
```

All the FCB entries will be invalid and file type set as "file".
2) Flow Charts

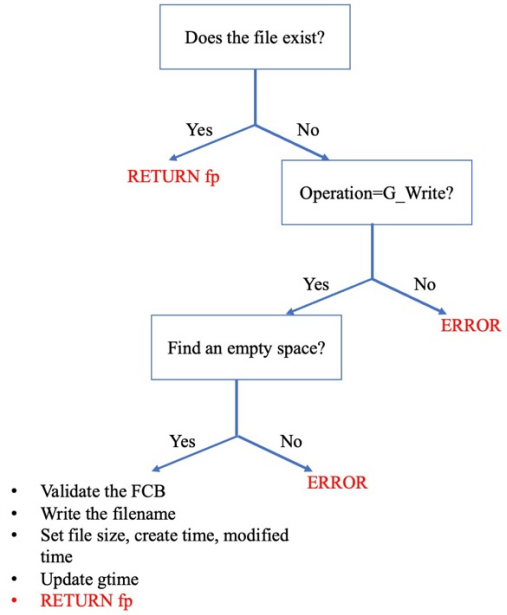
- gtime

It is a global variable to store the number of operations.

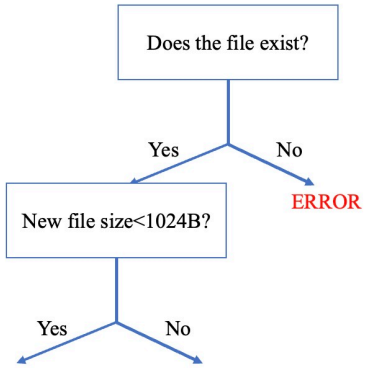
- *fs_allocate_blocks* (Compact Allocation Implementation)



- fs_open

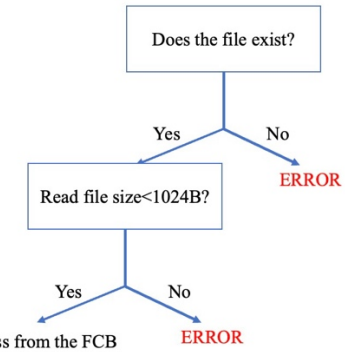


- fs_write



- Get the block address from the FCB entries
- Allocate the blocks according to the old file size and the new file size
- Write (new file size) of bytes to the blocks
- Set file size, modified time
- Update gtime
- RETURN

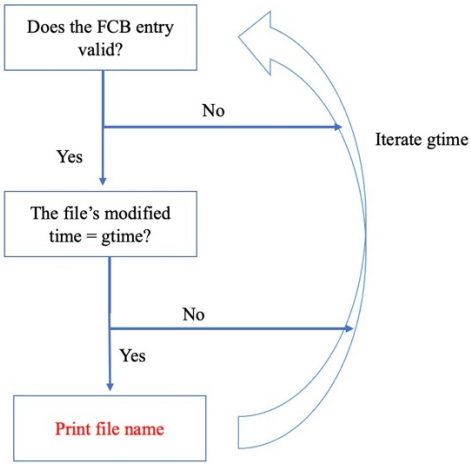
➢ *fs_read*



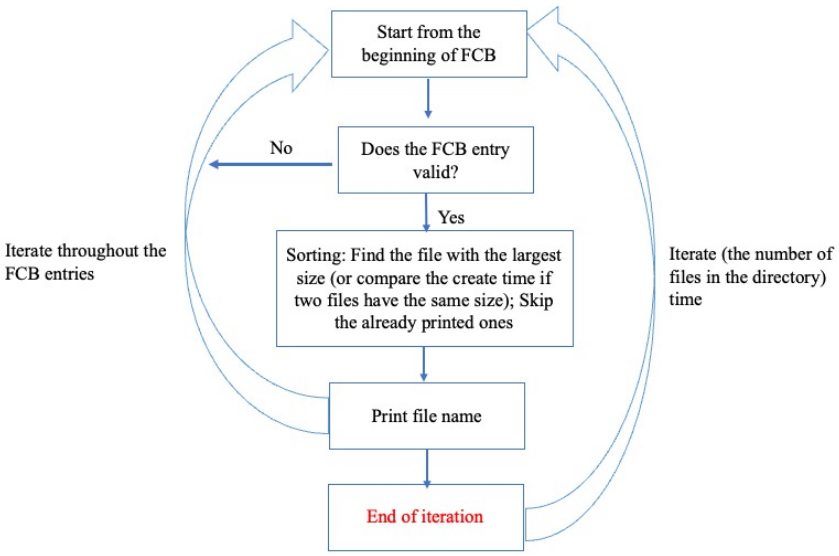
- Get the block address from the FCB entries
- Read (file size) of bytes to the buffer
- RETURN

➢ *fs_gsys*

- LS_D



- LS_S



- RM

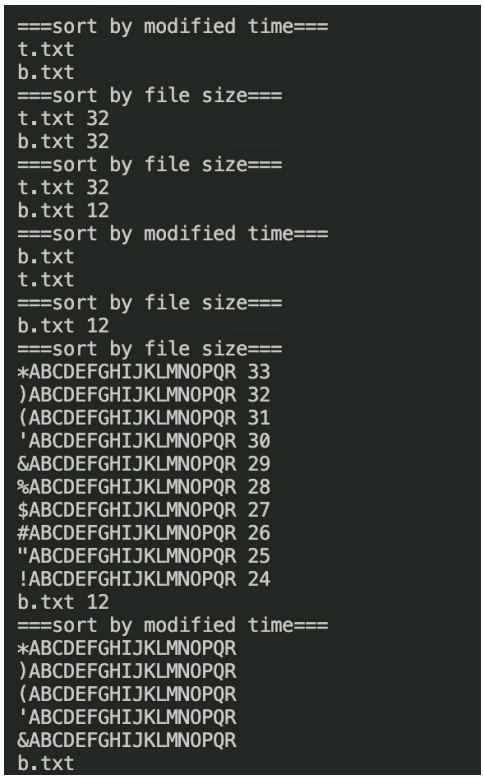
Use the $fs\_allocate\_file$ function with the new file size = 0 and the file pointer of the file to be deleted.

4. Sample Output

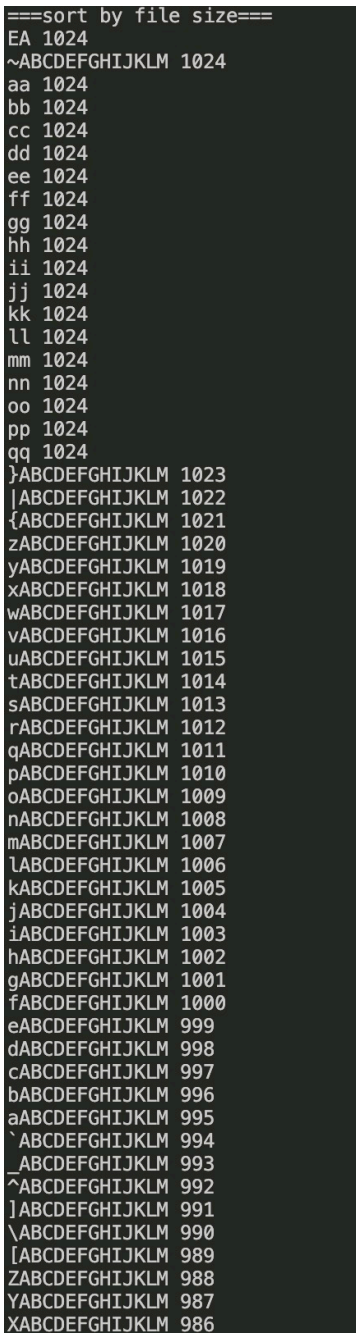
- Task 1

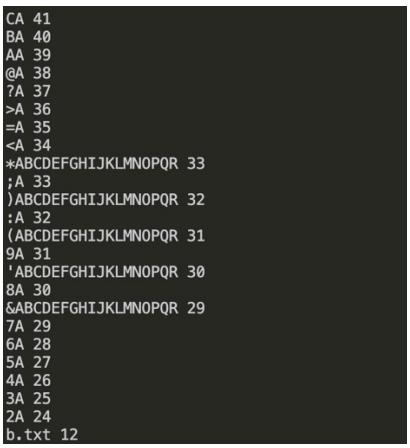
```
===sort by modified time===
t.txt
b.txt
===sort by file size===
t.txt 32
b.txt 32
===sort by file size===
t.txt 32
b.txt 12
===sort by modified time===
b.txt
t.txt
===sort by file size===
b.txt 12
```

- Task 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

- Task 3


===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
]ABCDEFGHIJKLM 991
\ABCDEFGHIJKLM 990
[ABCDEFGHIJKLM 989
ZABCDEFGHIJKLM 988
YABCDEFGHIJKLM 987
XABCDEFGHIJKLM 986


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

- Task 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
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-0004
1024-block-0003
1024-block-0002
1024-block-0001
1024-block-0000
[120090211@node21 Assignment4]$ cmp snapshot.bin data.bin
[120090211@node21 Assignment4]$
```

5. Bonus

1) Tree directory structure implementation

```
__device__ __managed__ u32 level_2 = -1;
__device__ __managed__ u32 level_3 = -1;
```

Global variables are used to record the current directory index. Since in this task, only 3 layers of directory is required, therefore we only use two global variables. Initially, they are set as -1.

2) Volume Partition in Bonus

In the bonus part, except from the setting in the standard part, we will use the last 3 bytes of the FCB to record the directory status.
Byte 29: Record the number of files in the directory, which should be less than 50, or 0xff if it is not a directory.
Byte 30-31: Record the parent directory of the current file. For example, if the parent directory is the root, these two bytes should be 0.
Initialization:

```
// initilize a root directory
  fs->volume[fs->SUPERBLOCK_SIZE  + 29] = 0;
  fs->volume[fs->SUPERBLOCK_SIZE  + 30] = 0xff;
  fs->volume[fs->SUPERBLOCK_SIZE  + 31] = 0xff;
  fs_gsys(fs, MKDIR, "root\0");
```

3) Modification on previous functions and new functions implementation

- fs_open

With level_2 and level_3, we know the current directory. When finding the file, we will specify whether the parent directory of the file with the target name is the same as the current directory. Also, when the file is not found eventually, we will add a new file to the current directory after checking the current number of files is less than 50. The file size, and modified time of the parent directory will be updated after a new file is created.

- fs_write

Except for the operation in the standard part, the parent directory will be checked during the file search. The modified time of the parent directory will also be updated.

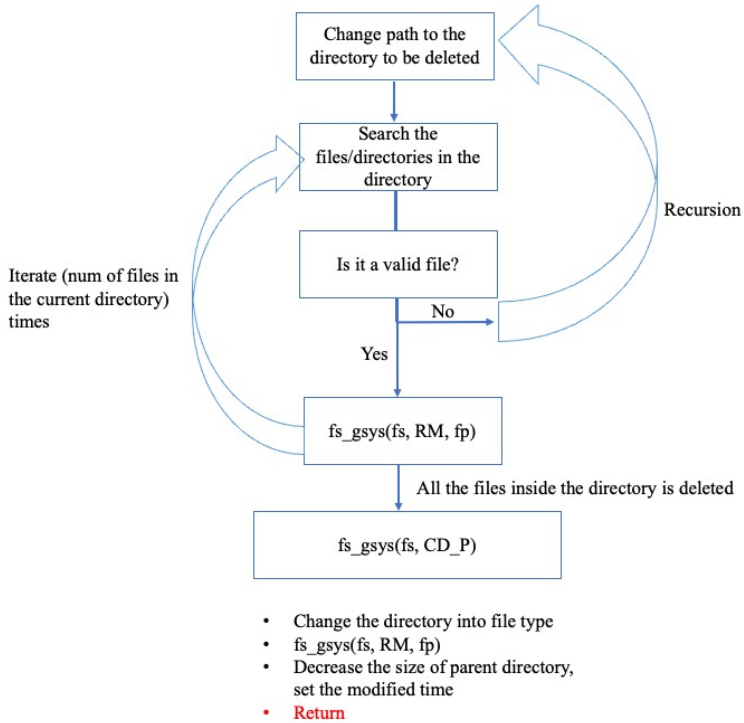
- fs_gsys

MKDIR: First, it will check whether the current directory is already in level 3 because we cannot add a directory in the third level of the directory structure. The rest of the operation is much alike to create a file, except that the 29th byte of the FCB will be set as 0, indicating the number of files in this directory is 0.

CD: The subdirectory will be searched by comparing the name and the parent directory. Once the directory is found, level_2 or level_3 or both will be updated.

CD_P: level_2 and level_3 will be updated with the parent directory stored in the target FCB.

RM_RF: The flow chart of rm_rf is shown below.



4) Sample Output