CSC3150 Assignment 4 Report

Environment

I use the centos OS on the computer of TC301

How to compile my program

1. "make" to makefiles
2. "./fs" to execute
3. "makeclean" to clean

How did I Design My Program

My FCB structure:

```
/*
 * my FCB structure
 * |0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|   20|21   | 22|23 | 24 | 25|26  | 27 |28  |29 | 30|  31|
 * |                  file name                    |       location    |      size        |create_t|modify_t|
 */
struct My_FCB
{
  char file_name[20];
  u32 location;
  u32 size;
  int create_time;
  int modified_time;
}Current_FCB;
```

function: fs_open

in my fs_open function:

1. search for the location of file name s if it exists, if it does not exist in the FCB we return -1 to denote.
2. in the read mode if we find the match of file s we simply record the location of its block in FCB by variable file_exist and find its start location by looking at its 20th to 23rd bits , in the end we return its pointer start_location in the superblock. recall that in my program the info of start _location is stored from 20-13 in each FCB.
3. in the write mode if no such file name can be found, we will just create a new zero byte file. If at the end of the day we find the match we clear the old content in the storage as will as modified the superblock to 0. return the start_location pointer in the FCB

```
__device__ u32 fs_open(FileSystem *fs, char *s, int op)
{
    /* Implement open operation here */
    //if not exist
    u32 file_exist = search_FCB(fs, s);
    if (op == G_READ) {
        if (file_exist == -1) {
            printf("error: no such file to read\n");
            return -1;
        }else{
            current_FCB_position = search_FCB(fs, s);
            u32 start_block = (fs->volume[current_FCB_position + 20] << 24) + (fs->volume[current_FCB_position + 21] << 16) + (fs->volume[current_FCB_position + 22] << 8) + (fs->volume[cur
            return start_block;
        }
    }

    if  (op == G_WRITE){
        if(file_exist == -1){
            file_info_store(fs, s);

        }
        else{
            gtime++;
            current_FCB_position = search_FCB(fs, s);
            u32 start_block = (fs->volume[current_FCB_position + 20] << 24) + (fs->volume[current_FCB_position + 21] << 16) + (fs->volume[current_FCB_position + 22] << 8) + (fs->volume[cur
            u32 size = (fs->volume[current_FCB_position + 24] << 24) + (fs->volume[current_FCB_position + 25] << 16) + (fs->volume[current_FCB_position + 26] << 8) + (fs->volume[current_FC
            for (int i = 0; i < size; i++) {
                fs->volume[start_block * 32 + i + fs->FILE_BASE_ADDRESS] = 0;
            }

            //clean the old file in block
            for (int i = 0; i < (size - 1) / 32 + 1; i++) {
                u32 super_file_start_location = start_block + i;
                int shift_number = super_file_start_location % 8;
                fs->volume[super_file_start_location / 8] = fs->volume[super_file_start_location / 8] - (1 << shift_number);
            }

            //uptime FCB time
            fs->volume[current_FCB_position + 30] = gtime >> 8;
            fs->volume[current_FCB_position + 31] = gtime;

            //uptime the time

            return start_block;
        }



    }
```

function: fs_read:

in my fs_read function:

1. we want to check if fp is a valid. if no, return an error.
2. so for the starting area to the size we put the content of fs->volume[fp * 32 + i + fs->FILE_BASE_ADDRESS] to the output buffer

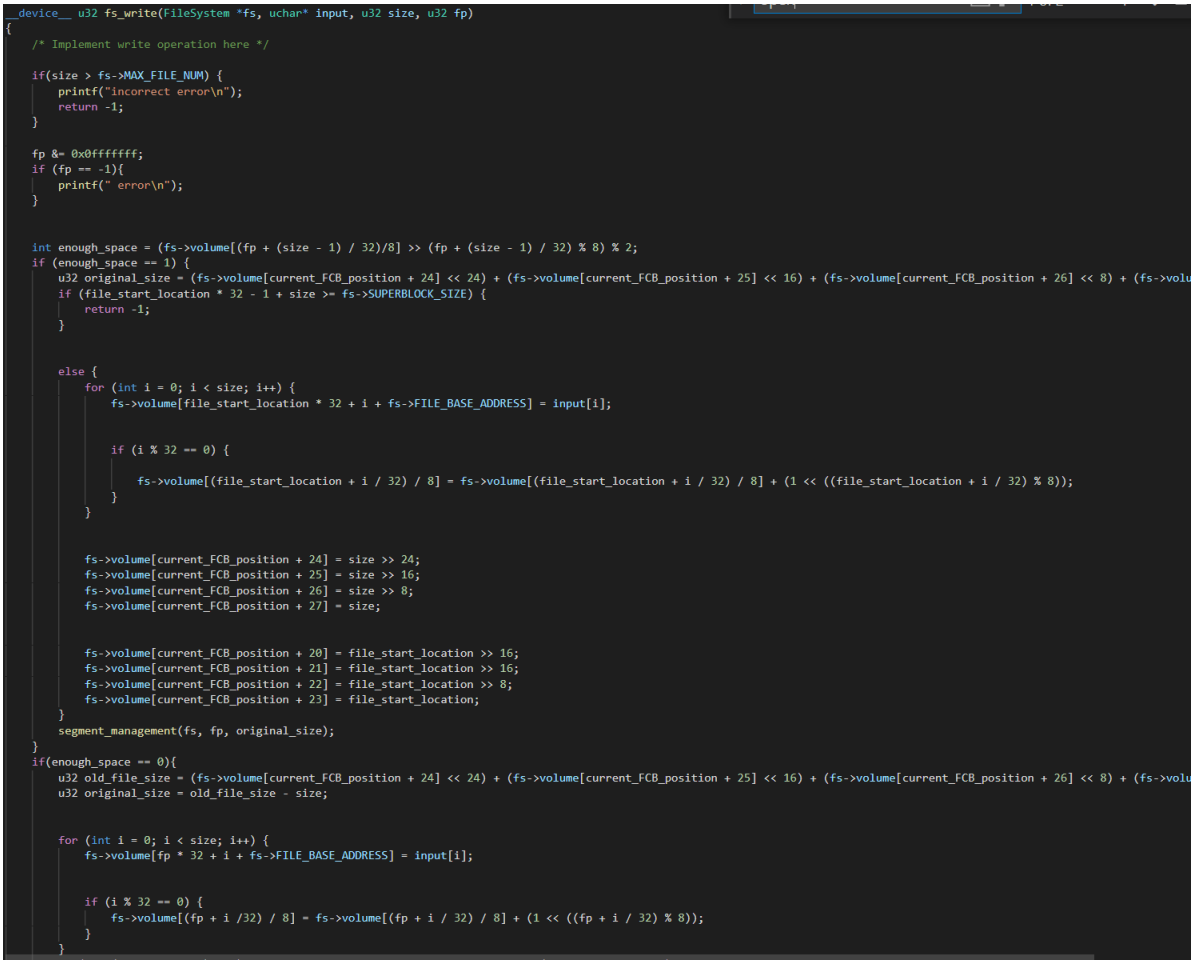
```
__device__ void fs_read(FileSystem *fs, uchar *output, u32 size, u32 fp)
{

    if(fp == -1){
        printf("%d", size);
        printf("error\n");
    }

    for (int i = 0; i < size; i++) {
        output[i] = fs->volume[fp * 32 + i + fs->FILE_BASE_ADDRESS];
    }
}
```

function: fs_write:

in my write function we can divide this problem into two cases:
1.the original file space is larger than the size of input written file:
we clear the file contents and directly input the things in input buffer to the storage, interate from 0 to size, then update the start location in the super block.

2. the space in the storage is not enough:
   allocate a new area for the input file, load the data to the new place as well as update the FCB and superblock

in the end if externel segmentation occured during this process, remove them as will as update the FCB and Superblock



```
__device__ u32 fs_write(FileSystem *fs, uchar* input, u32 size, u32 fp)
{
    /* Implement write operation here */

    if(size > fs->MAX_FILE_NUM) {
        printf("incorrect error\n");
        return -1;
    }

    fp &= 0x0fffffff;
    if (fp == -1){
        printf(" error\n");
    }


    int enough_space = (fs->volume[(fp + (size - 1) / 32)/8] >> (fp + (size - 1) / 32) % 8) % 2;
    if (enough_space == 1) {
        u32 original_size = (fs->volume[current_FCB_position + 24] << 24) + (fs->volume[current_FCB_position + 25] << 16) + (fs->volume[current_FCB_position + 26] << 8) + (fs->volu
        if (file_start_location * 32 - 1 + size >= fs->SUPERBLOCK_SIZE) {
            return -1;
        }


        else {
            for (int i = 0; i < size; i++) {
                fs->volume[file_start_location * 32 + i + fs->FILE_BASE_ADDRESS] = input[i];


                if (i % 32 == 0) {

                    fs->volume[(file_start_location + i / 32) / 8] = fs->volume[(file_start_location + i / 32) / 8] + (1 << ((file_start_location + i / 32) % 8));
                }
            }


            fs->volume[current_FCB_position + 24] = size >> 24;
            fs->volume[current_FCB_position + 25] = size >> 16;
            fs->volume[current_FCB_position + 26] = size >> 8;
            fs->volume[current_FCB_position + 27] = size;


            fs->volume[current_FCB_position + 20] = file_start_location >> 16;
            fs->volume[current_FCB_position + 21] = file_start_location >> 16;
            fs->volume[current_FCB_position + 22] = file_start_location >> 8;
            fs->volume[current_FCB_position + 23] = file_start_location;
        }
        segment_management(fs, fp, original_size);
    }
    if(enough_space == 0){
        u32 old_file_size = (fs->volume[current_FCB_position + 24] << 24) + (fs->volume[current_FCB_position + 25] << 16) + (fs->volume[current_FCB_position + 26] << 8) + (fs->volu
        u32 original_size = old_file_size - size;


        for (int i = 0; i < size; i++) {
            fs->volume[fp * 32 + i + fs->FILE_BASE_ADDRESS] = input[i];


            if (i % 32 == 0) {
                fs->volume[(fp + i /32) / 8] = fs->volume[(fp + i / 32) / 8] + (1 << ((fp + i / 32) % 8));
            }
        }
```

function: fs_gsys LS_D LS_S:

first we search where the file system is going to end

then according to the option we choose we do bubble sort, then we display the file name by the cretirea of the sort

```
__device__ void fs_gsys(FileSystem *fs, int op)
{
    u32 end_point;

    /* Implement LS_D and LS_S operation here */
    for (u32 i = 4096; i - 32 < 36863; i = i + 32) {
        u32 size = (fs->volume[i + 24] << 24) + (fs->volume[i + 25] << 16) + (fs->volume[i + 26] << 8) + (fs->volume[i + 27]);
        if (size == 0) {
            size = (fs->volume[4096 + 24] << 24) + (fs->volume[4096 + 25] << 16) + (fs->volume[4096 + 26] << 8) + (fs->volume[4096 + 27]);
            end_point = i - 32;
            break;
        }
        end_point = i - 32;
    }


    if (end_point < 4096) printf("error: no file in FCB \n");


    if (op == 0) {
        for (int i = 4096; i < end_point; i = i + 32) {
            for (int j = 4096; j < end_point + 4096 - i; j += 32) {
                u32 j_time_previous =  (fs->volume[j + 30] << 8) + (fs->volume[j + 31]);
                u32 j_time_after = (fs->volume[j + 30 + 32] << 8) + (fs->volume[j + 31+ 32]);
                if (j_time_previous < j_time_after){
                    for (int k = 0; k < 32; k++) {
                        uchar tempt = fs->volume[j + k];
                        fs->volume[j + k] = fs->volume[j + k +32];
                        fs->volume[j + k +32] = tempt;
                    }
                }
            }
        }
    }


    else {
        for (int i = 4096; i < end_point; i = i + 32) {
            for (int j = 4096; j < end_point - i + 4096; j = j + 32) {
                u32 j_size_previous = (fs->volume[j + 24] << 24) + (fs->volume[j + 25] << 16) + (fs->volume[j + 26] << 8) + (fs->volume[j + 27]);
                u32 j_size_after = (fs->volume[j + 32 + 24] << 24) + (fs->volume[j + 25 + 32] << 16) + (fs->volume[j + 26 + 32] << 8) + (fs->volume[j + 27 + 32]);
                u32 j_time_previous = (fs->volume[j + 30] << 8) + (fs->volume[j + 31]);
                u32 j_time_after = (fs->volume[j + 30 + 32] << 8) + (fs->volume[j + 31 + 32]);
                if (j_size_previous < j_size_after){
                    for (int k = 0; k < 32; k++) {
                        uchar tempt = fs->volume[j + k];
                        fs->volume[j + k] = fs->volume[j + k +32];
                        fs->volume[j + k +32] = tempt;
                    }
                }
                if (j_size_after == j_size_previous && j_time_previous > j_time_after){
                    for (int k = 0; k < 32; k++) {
                        uchar tempt = fs->volume[j + k];
                        fs->volume[j + k] = fs->volume[j + k +32];
                        fs->volume[j + k +32] = tempt;
                    }
                }
            }
        }
    }
```

function: fs_gsys rm:

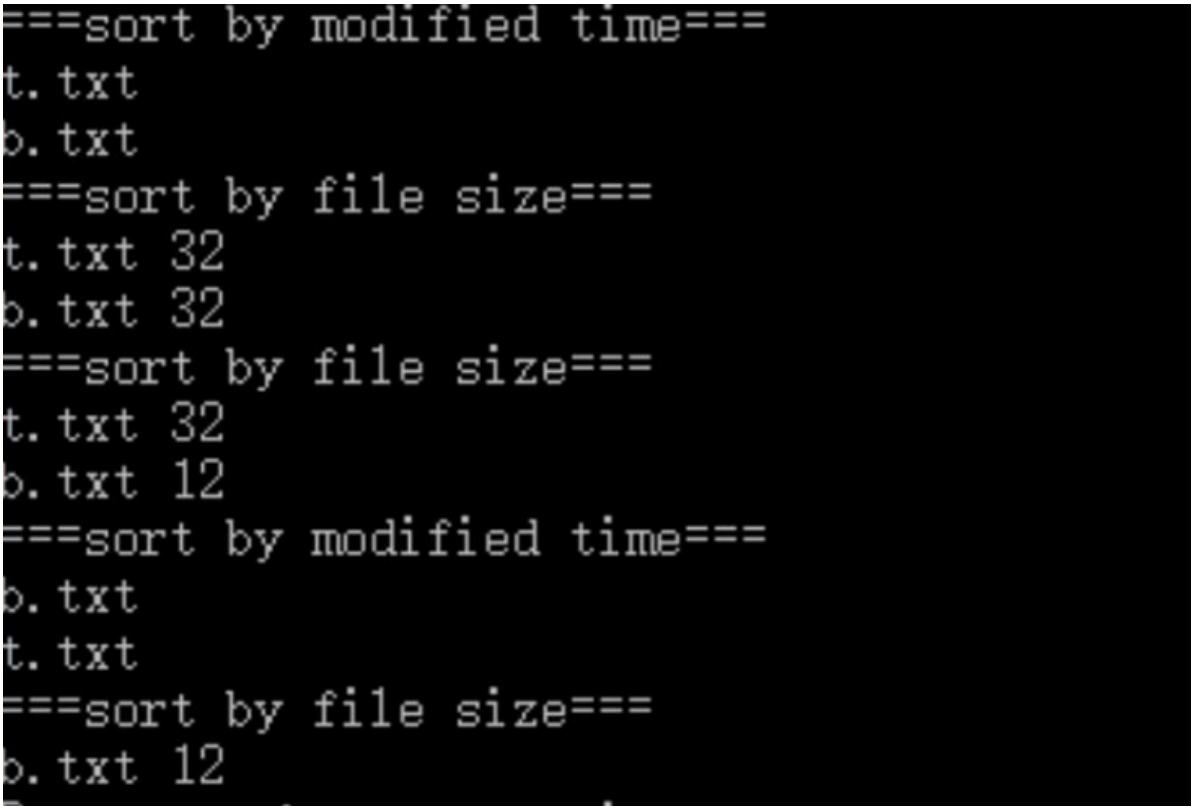
1. we find wether the file is exist in the file system. if yes we are trying to find its location in the file system.
2. find where its file start in the storage and its size
3. clear it from the storage as well as FCB and Superblock
4. Then we manage the externel segementation

```
__device__ void fs_gsys(FileSystem *fs, int op, char *s)
{
    /* Implement rm operation here */
    if (cuda_strlen(s) > 20)
    {
            printf("File Name Size Exceed The Largest Filename Limit \n");
            return;
    }


    if (op == RM){
        u32 file_exist = search_FCB(fs, s);
        if (file_exist == -1){
            printf("error: no such file to remove\n");}
        else {
            current_FCB_position = search_FCB(fs, s);

            // find the start block in FCB
            u32 start_block = (fs->volume[current_FCB_position + 20] << 24) + (fs->volume[current_FCB_position + 21] << 16) + (fs->volume[current_FCB_position + 22] << 8) + (fs->volume[cur

            //clean the old file in storage
            u32 size = (fs->volume[current_FCB_position+24] << 24) + (fs->volume[current_FCB_position + 25] << 16) + (fs->volume[current_FCB_position + 26] << 8) + (fs->volume[current_FCB_
            for (int i = 0; i < size; i++) {
                fs->volume[start_block * 32 + i + fs->FILE_BASE_ADDRESS] = 0;
            }

            //clean the file in superblock
            for (int i = 0; i < (size - 1) / 32 + 1; i++) {
                fs->volume[start_block + i] = 0;
            }

            //clear content in FCB
            for (int i = 0; i < 32; i++) {
                fs->volume[current_FCB_position + i] = 0;
            }
            segment_management(fs, start_block, size);

            for (int i = current_FCB_position; i < 36863; i = i + 32) {
            if (fs->volume[i + 32 + 24] == 0 && fs->volume[i + 32 + 25] == 0 && fs->volume[i + 32 + 26] == 0 && fs->volume[i + 32 + 27] == 0) break;
            for (int j = 0; j < 32; j++) {
                fs->volume[i + j] = fs->volume[i + j + 32];
                fs->volume[i + j + 32] = 0;
            }
        }
            FCB_position = FCB_position - 32;
        }
    }
}
```

Sample Output:

Input:

Output: test case1:



Output: test case2:

```
(base) [cuhksz@TC-301-10 zhouliang]$ ./fs
---sort by time---
b.txt
t.txt
---sort by file size---
t.txt 32
b.txt 32
---sort by file size---
t.txt 32
b.txt 12
---sort by 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 time---
*ABCDEFGHIJKLMNOPQR
)ABCDEFGHIJKLMNOPQR
(ABCDEFGHIJKLMNOPQR
'ABCDEFGHIJKLMNOPQR
&ABCDEFGHIJKLMNOPQR
```

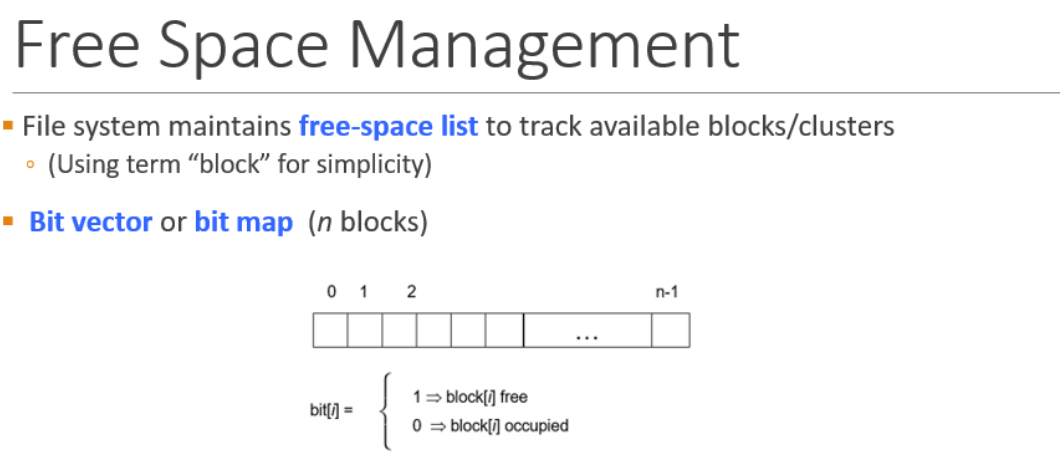
Output: test case3:

```
*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
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
```

The Problem I met

1. How should I do free space management
   we can do remove those externel segmentation while they occured, in my program:
   1. I remove a certain portion of file storage by setting corresponding volume[i] to be 0, and move the predessessor file upward.
   2. we set the corresponding area in superblock to be 0

3. update the file_start_location and get its quotient and reminder.
4. update the superblock and change the contents in FCB



2. I stuck in the problem when the size are the same, sort its modified time
   for those files with same sizes do the sort algorithm again for these files, but keep other files static
3. How to construct a tree-structure directory in bonus task :

   In my program I denote the file is a directory or file in the 28th and 29th bit to denote who is the dircetory/file's parent. this makes the tree structure feasible the structure of tree directory is:

```
struct directory{
    char name[20];
    u32 sibling;
    u32 parent;
    u32 child;
    u32 size;
    u32 modified date;
}directory[4];


/*
 * my FCB structure of tree directory
 * |0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|   20|21   | 22|23 | 24 | 25|26  | 27 |28      |29    | 30|  31 |
 * |                   file name                   |       location    |      size       |index   |parent|modify_t |
 */
```

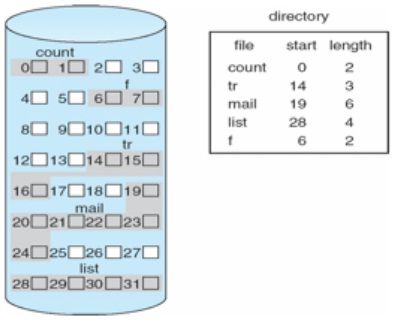
What I learnt from the task

1. The relationship between Superblock and contents of file:
   each bit in the volume[i] in superblock correpsonding to a unit block in the contents of files.
   in my work in the superblock we use 1 to denote the file exist, 0 to denote it doesn't exist.

2. How to do contiguous allocation

Contiguous Allocation

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

basicly we should know where the file start, and allocate the corresponding slots according to its size. when we are going to remove the file we put the predecessor file upwards have the same start_block with the file to be deleted.