Report

HuangPengxiang

11/24/2021

Contents

Overview

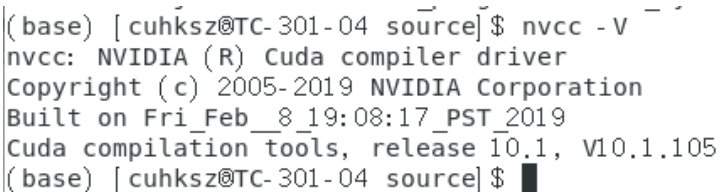
This is the Fourth Assignment of Operating System. In this assignment, The programming goal is to implement the the simulation of file system management via GPU's memory with single thread. This simulation is implemented in CUDA. It takes the global memory as the disk memory, and the operation of Open, Read, Write, Remove, List are implemented in my program. I also **implement bonus part**, which can have hierarchical directory with tree structure instead of using only root directory. This program will focus on the part of implementation of file operation and use command `ls` to print result out.

Some Declarations

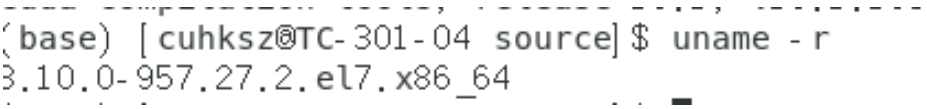
- There might be some Indentation unaccustomedness in your computer. I used Mac VSC to develop the program and test it on CenterOS, and I found the Indentation is not comfortable to view when I open it on test computer. I am sorry for that and I hope it won't affect my grade.
- I finish the part of bonus part, which including the tree structure building, `pwd` command to traverse the tree, `cd` , `cd_p` and `mkdir` and command to build up hierarchical directory, use `ls` and `pwd` to see to output. But there are still some bugs in my operation of `rm-rf` part, I didn't finish it untill the deadline approach.

Environment

- The Running Environment is showing below:

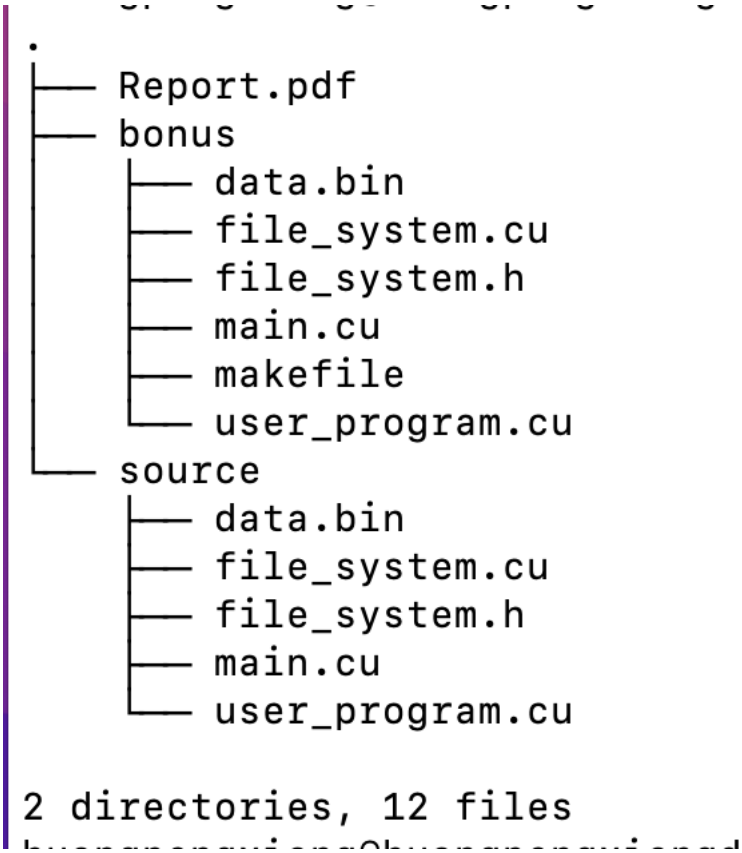


- The Test Environment is:



Running Guidance

The Tree of My Progarm



Detailed Steps to Run Program

I designed a makefile to test my program, so You can simply use this file to test my program

```
$ cd /* where the source or bonus located */
$ make     // this command will generate some warning like "char *" in main.o, choose to ignore it
$ make test // use this command to run
$ make clean // use this command to clean
```

Demo Output

```
[cuhksz@TC-201-22 source]$ make test
./Project
===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
[cuhksz@TC-201-22 source]$
```

Program Design

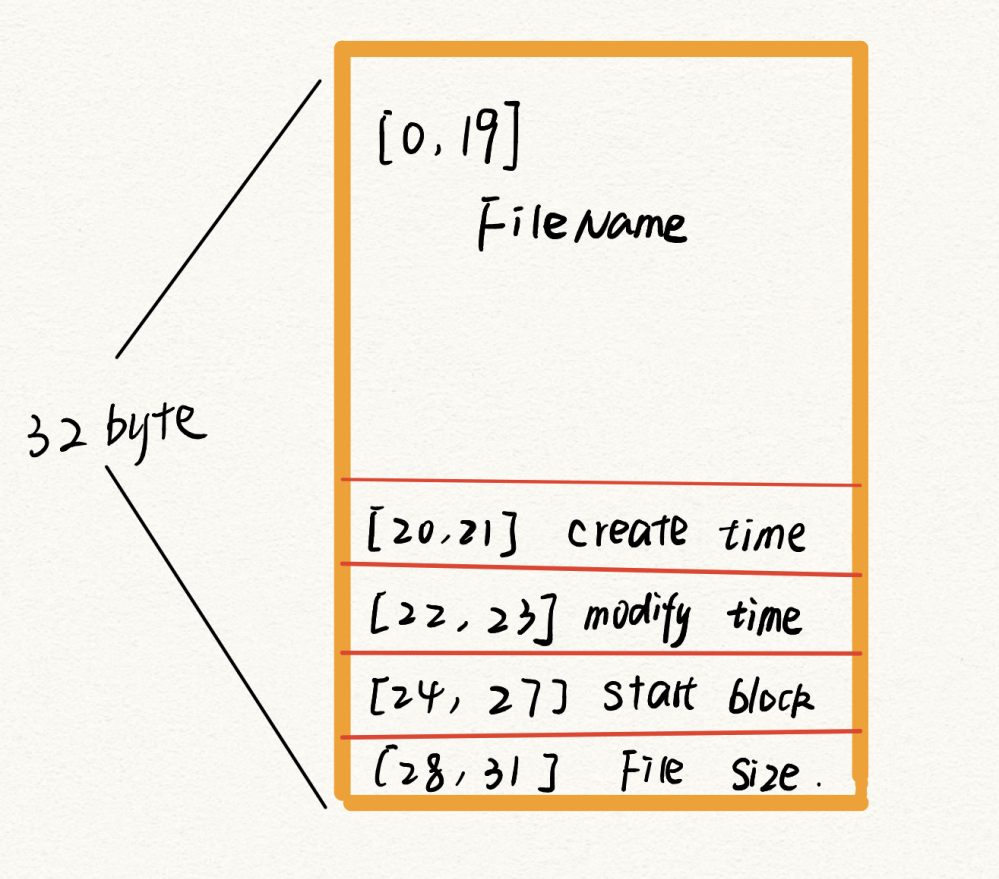
Source Design

The source program is designed to simulate the file system and implement the function including open, read, write and rm operations. I will show Those part one by one.

The structure of file storage is contiguous. All the files are stored in contiguous way and LCB information are also still in a contiguous allocation. Hence, All the operation will basically follow this rule and mantain the FCB order in the block

The FCB Structure

The first 20 byte is used to store the file name, and the end of the file name should be "\0". The [20,21] is used to store the file create time, and also [22,23] is used to store the modify time of the file. The [24,27] is used to store the location of the file strat block, and the remain byte is used to store the file size.



How I design operation: Open

In the Open part, When given a file pointer and the program will to check whether the file is exist in one of the FCB entry. If the file existed, then we need to return the file pointer of the start location for user to modify it. And The program should clean the information when user input write mode if the file existed for further write operation. Otherwise, If the file is not existed, The program should create a new file and return its start location to user. Hence, for this case, The program should create a new fcb entry and store all the initial information in it, including the file name, modify time, create time, and so on.

How I design operation: Read

In the Read part, simply this operation must be executed after the open operation. Since the Open operation will judge the existance of the file, for the read part, we can simply just read the information from the file block to the output buffer.

How I design operation: Write

In the Write part. This is basically the most difficult one for me to design. First we need to check whether there are enough space for file to write. Since the open operation will give the start pointer for the file. and write operation need to check whether the wirte information will occupied other file's space. If it won't occupied othe file's space, then the file will write the information into disk volume. otherwise, we need to write all the information in the new space for store. Finally, we need to check the external fragmentation and do the compact to eliminate it.

How I design operation: Ls

In the LS part. First we need to check if it is LS by size or by time. And we need to find the stop block in the file. and from the initail location to the stop position, we list all the file. and we compare all the file by name or by size. Finally, The program will do the output part.

How I design operation: RM

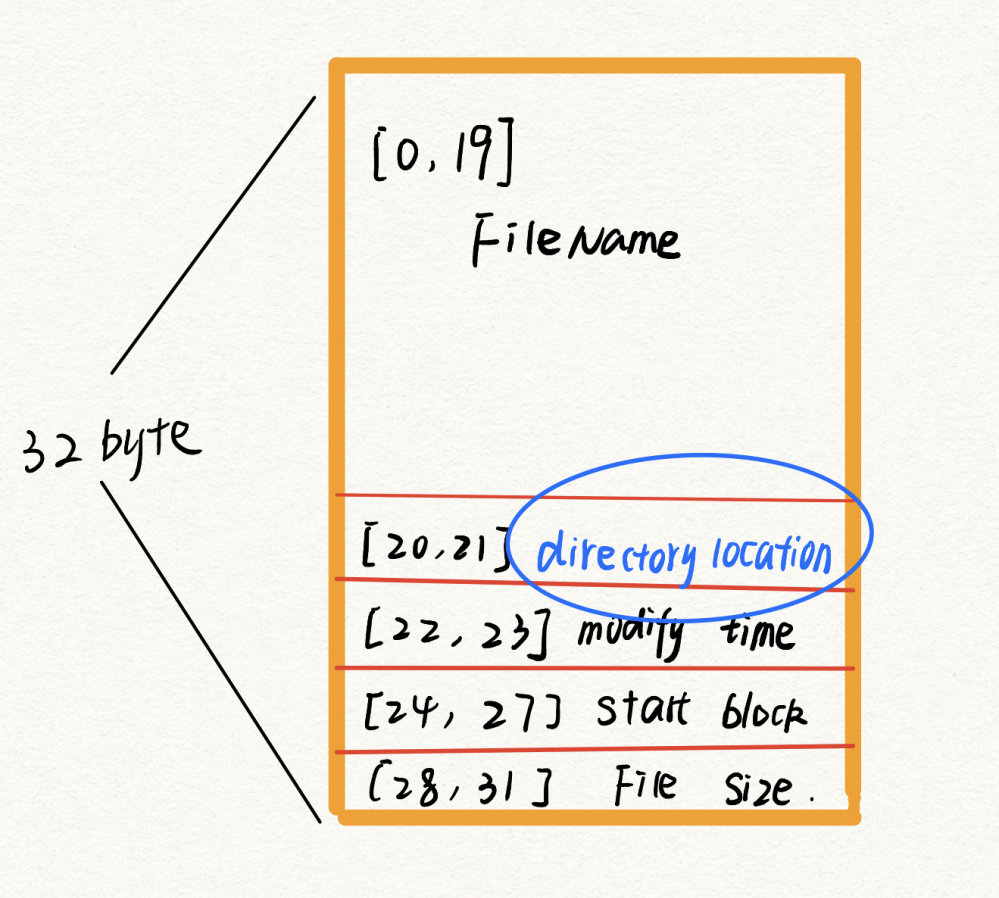
In The RM part, We first need to check whether the file is existed in the fcb block. If it is exised, then we find its fcb entry and all the information in it. then we remove file from disk, and delete this fcb entry. and finally, we also do the compact to eliminate the external fragmentation.

How I design compact

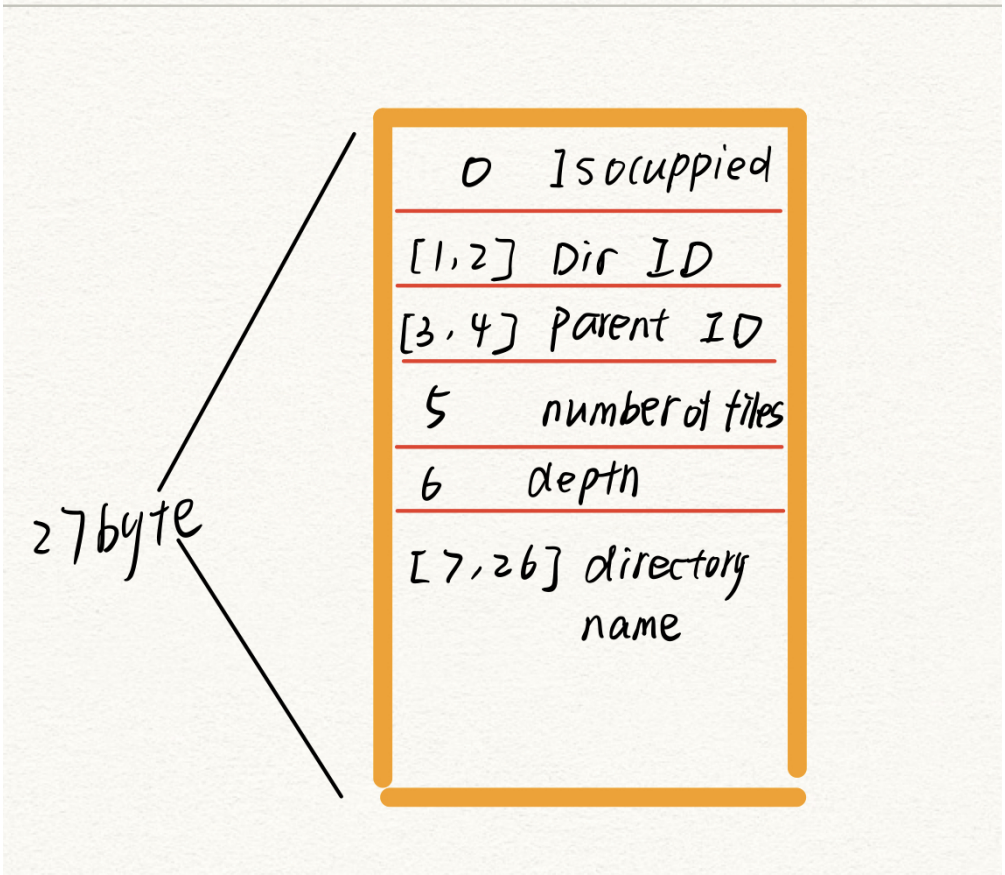
The compact is the function named `Update`. In this function, it need the parameter of file pointer which is poined to the start block. and the file size need to compact. First we move all the file in the disk forward to occupied the empty space. After that, update the super block, and update the fcb block to set the unsed one to 0.

Bonus Design

In the bonus part, The design of fcb enry is modified little bit. The create time is changed to the directory id, which means where the file belongs to which directory.



And I also create the directroy block to store the tree structure of the directory. The Desined of this is showing below:



How I design the tree structure

I used the dirctory block named `dir[]` to implement the structure of the tree. each block will **store the location id of its parent**. Hence, when I traverse the whole directory, I simply just need to traverse the tree from **ROOT to specific directory**. And The program will implement the operation of `cd` and `cd_p` based on the tree structure.

How I modify operations

I implement the operation of `mkdir`, `cd`, and `cd_p` based on the tree structure. and I implement the command `pwd` by traverse the whole tree and print the current location. for the `ls` command, I find the all the directory and check whether it is in the current parent director. If it is, also print it out. I did not finish the rm-rf part, since there are still some bugs i did not have the sufficient time to finish.

Program Implementation

Source Implementation

How I implement the Open:

```
  /* assign the position to current */
  current_FCB_pos = IsFileExist(fs,s);
  u32 start_block = (fs->volume[current_FCB_pos+24] << 24) + (fs->volume[current_FCB_pos+25] << 16)
                  +(fs->volume[current_FCB_pos+26] << 8) + (fs->volume[current_FCB_pos+27] );

  /* write mode */
  if (op == 1){
    u32 filesize = (fs->volume[current_FCB_pos+28] << 24) + (fs->volume[current_FCB_pos+29] << 16)
                  +(fs->volume[current_FCB_pos+30] << 8) + (fs->volume[current_FCB_pos+31]);

    /* clean the old file in disk */
    for (int i = 0; i < filesize; ++i){
      fs->volume[fs->FILE_BASE_ADDRESS + start_block * 32 + i] = 0;
    }

    /* update the super block */
    for (int i = 0; i < (filesize -1)/32 + 1; i++){
      u32 super_block = start_block + i;
      int shiftnum = super_block % 8;
      fs->volume[super_block/8] = fs->volume[super_block/8] - (1 << shiftnum); // modify one bit
    }


    /* update fcb */
    fs->volume[current_FCB_pos + 22] =  mod_time >> 8;
    fs->volume[current_FCB_pos + 23] = mod_time;

    mod_time++;

  }
  // printf("start block is %d  \n ", start_block);
  // printf("in open is .. %c \n", fs->volume[4096]);
  return start_block;
```

How I implement the Write:

```
/* if enough space to write */
if ( IsEnoughSpace(fs,fp,size) ){

  for (int i = 0; i < size; ++i){
    /* update the disk */
    fs->volume[fs->FILE_BASE_ADDRESS + fp * 32 + i] = input[i];
    /* update the super block */
    if ( i % 32 == 0){
      u32 super_block = fp + i/32;
      int shiftnum = super_block % 8;
      fs->volume[super_block/8] = fs->volume[super_block/8] + (1 << shiftnum); // modify one bit
    }
  }

  u32 pre_file_size =  (fs->volume[current_FCB_pos + 28] << 24) + (fs->volume[current_FCB_pos + 29] << 16)
                      +(fs->volume[current_FCB_pos + 30] << 8) + (fs->volume[current_FCB_pos + 31]);

  /**/
  u32 delta_size = pre_file_size - size;


  if ((int) delta_size < 0 ){
    block_pos = block_pos + (-delta_size - 1)/32 + 1;
  }

  /* update the size */
  fs->volume[current_FCB_pos + 28] = size >> 24;
  fs->volume[current_FCB_pos + 29] = size >> 16;
  fs->volume[current_FCB_pos + 30] = size >> 8;
  fs->volume[current_FCB_pos + 31] = size;




  if (delta_size > 0 && pre_file_size != 0 && fp != block_pos - 1){
    Update(fs, fp + (size -1)/32 + 1 , delta_size);
  }
  // printf(" current block is %d \n", block_pos);
```

How I implement the compact

```
__device__ u32 Update(FileSystem * fs, u32 fp, u32 size ){

u32 pos = fs->FILE_BASE_ADDRESS + fp * 32; // the intial position for file
u32 required_size = ((size -1)/32 + 1)*32;   // the required space for file including internal fragmentation

/* if write the file occupy other file's space, move them */
while ( (fs->volume[pos + required_size] != 0 || (pos+required_size)%32 != 0) && pos + size < fs->STORAGE_SIZE  ){
  fs->volume[pos] = fs->volume[pos + required_size];
  fs->volume[pos + required_size] = 0;
  pos++;
}

/* update the block */
for (int i = 0; i < block_pos/8 + 1; i++){
  // set it all to zero
  fs->volume[i] = 0;
}
block_pos = block_pos - (size-1)/32 -1;
u32 whole_block = block_pos/8;
u32 remain = block_pos%8;

// set the block before to 511(111111111)
for (int i = 0; i < whole_block && i < fs->SUPERBLOCK_SIZE ; i++) {
  fs->volume[i] = 511;
}

// set the remain bit to 0
for (int i = 0; i < remain; i++) {
  fs->volume[whole_block] = fs->volume[whole_block] + (1 << i); // modify one bit
}

/* modifty the fcb */
u32 fcb_temp_pos;

for (int i = 4096; i < 36863; i = i + 32){
  if (fs->volume[i+28] == 0 && fs->volume[i+29] && fs->volume[i+30] ==  0 && fs->volume[i=31] == 0){
    break; // search till empty
  }
  fcb_temp_pos = (fs->volume[i+24] << 24) + (fs->volume[i+25] << 16)
                 + (fs->volume[i+26] << 8) +  (fs->volume[i+27]);
  if (fcb_temp_pos > fp){
    // clear the external space
    fcb_temp_pos = fcb_temp_pos - (size-1)/32 - 1;
    fs->volume[i + 24] = fcb_temp_pos >> 24;
    fs->volume[i + 25] = fcb_temp_pos >> 16;
    fs->volume[i + 26] = fcb_temp_pos >> 8;
    fs->volume[i + 27] = fcb_temp_pos;
  }
```

Upload to Bin Upload to Bin

Bonus Implementation

How I implement the mkdir and cd

```
/* MKDIR */
if (op == 3){
  u32 myid;
  for (int i  = 0; i < 1024*27-1; i += 27){
    if (dir[i+7] == 0){
      myid = i;
      break;
    }
  }
  dir[myid] = 1; // represent dir
  dir[myid + 1] = myid >> 8;
  dir[myid + 2]= myid;
  dir[myid + 3]= current_dir >> 8;
  dir[myid + 4]= current_dir;
  dir[myid + 5] = 0; //number of file
  dir[myid + 6] = current_dep;
  for (int j = 0; j < 20; j++){
    dir[myid + 7 +j] = s[j];
  }
}

/* CD */
if ( op == 4){
  int flag;
  for (int i = 0; i < 1024*27 -1; i+=27){
    flag = 0;
    if (dir[i] == 0) break;
    for (int j = 0; j < 20; j++){
      if (dir[i + 7 +j] != s[j]) flag =1;
    }

    if (flag == 0){
      int parentid = (dir[i + 3] << 8) + dir[i + 4];
      if (parentid == current_dir){
        current_dir = (dir[i + 1] << 8) + dir[i+2];
      }
    }
  }
}
```

How I implement the ls and pwd

file_system.cu

```
/* PWD */
if (op == 7){
  int mylist[3];
  int size = 0;
  int id = current_dir;

  while (id != 0){
    mylist[size++] = id;
    id = (dir[id + 3] << 8) + dir[id + 4]; // parent id
  }

  if (size == 0){
    printf("/");
  }
  while (size > 0){
    id = mylist[--size];
    printf("/");
    for (int i = 0; i < 20; i++){

      if (dir[id + 7 +i] == 0) break;
      else{
        printf("%c",dir[id+7+i]);
      }
    }
  }
  printf("\n");
}

/* LS_D and LS_S */
else{
  u32 stop_pos;

  /* search the stop point */
  for ( int i = 4096 ; i < (4096 + 32*1024 - 1); i += 32 ){
    u32 file_size = (fs->volume[i + 28] <<  24) + (fs->volume[i + 29] <<  16)
                    +(fs->volume[i + 30] <<  8)  + (fs->volume[i + 31]);

    if (file_size == 0) break;
    stop_pos = i ;
  }


  if (stop_pos <=  4096){
    printf("LS Error: No file in FCB \n");
  }

  Sort(fs, 4096, stop_pos, op);
```

The problem I have meet

- Most of the information in the FCB file need 2 bytes or more to store the data like the block start position, creation date, modified date. However, each chart in unsigned character array can only store 1 byte. My solution is to use the shift operation (« and ») to cut the long data into 2 or 4 parts with each part has one byte. Store the 1-byte information in the array.

- When I first try to sort the information in FCB, I used the quicksort hoping to reduce the execution

time. However, the program will automatically down afterwards.My solution is to change the quicksort into bubblesort and avoid any recursion.

- **The biggest problem I have ever met is the compact.** It is difficult to move all the fcb and file information. Finally, I implement this by design a function to move the external fragmentation, and transfer the comapct problem to move external fragmentation.
- **The problem I have not solved yet is the rm_rf operation.** since this operation need to update the information both in directory and fcb enry and also file disk. And It is esay to have bugs for mapping those information. I may fixed it in the later.

What I have learned from this program

- Free space management: there are two possible ways to manage the external segment. You can do the segment management when the storage is out of available space. You can also do the segment management when the external segment occurreds
- How the contiguous allocation work in the file system. You need to have a bitmap or bitvector in the super block to record the used block. You need to find a new block at the end of the used block sequences if the original block size is not enough for new file.
- Better understanding of the implementation of file related system call and how they work together to attain some user operations. How does each part work together: volume structure, super block, file control block (FCB), free-space management etc.

Demo Screenshots

Source Test Case 1

```
[cuhksz@TC-201-22 source]$ make test
./Project
===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
[cuhksz@TC-201-22 source]$
```

Source Test Case 2

```
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
$ABCDEFGHIJKLMNOPQR 27
#ABCDEFGHIJKLMNOPQR 26
"ABCDEFGHIJKLMNOPQR 25
!ABCDEFGHIJKLMNOPQR 24
b.txt 12
===sort by modified time===
*ABCDEFGHIJKLMNOPQR
)ABCDEFGHIJKLMNOPQR
(ABCDEFGHIJKLMNOPQR
'ABCDEFGHIJKLMNOPQR
&ABCDEFGHIJKLMNOPQR
b.txt
[cuhksz@TC-201-22 source]$
```

Source Test Case 3

```
@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
[cuhksz@TC-201-22 source]$
```

Bouns Test Case

```
./Project
===sort by modified time===
t.txt
b.txt
===sort by file size===
t.txt 32
b.txt 32
===sort by modified time===
t.txt
b.txt
appd
===sort by file size===
t.txt 32
b.txt 32
appd
===sort by file size===
===sort by file size===
a.txt 64
b.txt 32
softd
===sort by modified time===
b.txt
a.txt
softd
/app/soft
===sort by file size===
B.txt 1024
C.txt 1024
D.txt 1024
A.txt 64
soft===sort by file size===
a.txt 64
b.txt 32
softd
/app
===sort by file size===
a.txt 64
b.txt 32
softd
/app
app===sort by file size===
t.txt 32
b.txt 32
appd
===sort by file size===
t.txt 32
b.txt 32
appd
/app
(base) [cuhksz@TC-301-04 bonus]$
```