Assignment 4

CSC3150

Student Number: 117010032

Student Name: 陈雅茜

香 港 中 文 大 學（深 圳）

The Chinese University of Hong Kong, Shenzhen

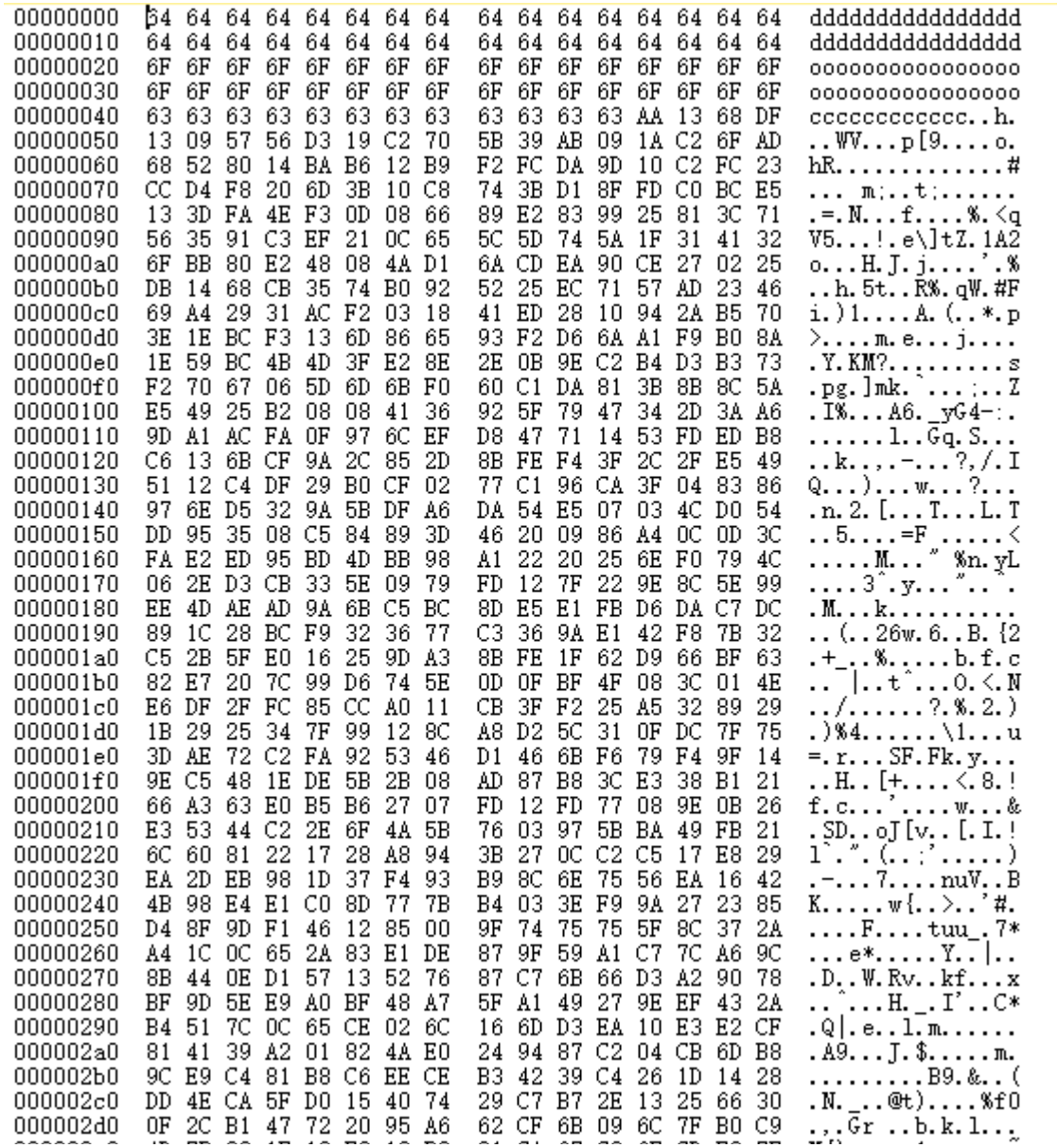
Introduction: Environment and Steps to Compile

This program is written to simulate the file system based on contiguous allocagion. The program is implemented on CUDA and tested on the windows OS with CUDA version 9.2.148, VS version 2017, GPU for NVIDIA GeForce GTX 1060 6GB. The user should input the user program and a binary file, the program will automatically put all the data read into the snapshot.bin file.

The input program should be like

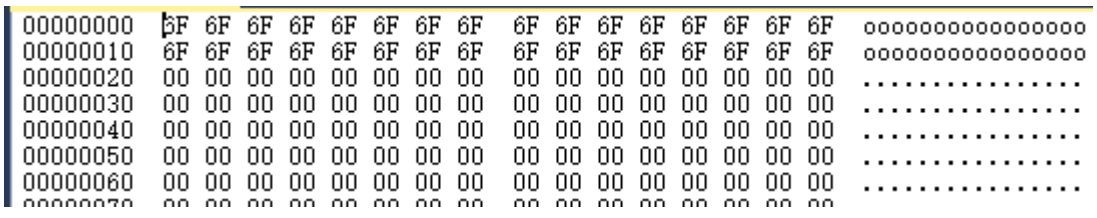
```
/////////////// Test Case 3  ///////////////
u32 fp = fs_open(fs, "t.txt\0", G_WRITE);
fs_write(fs, input, 64, fp);
fp = fs_open(fs, "b.txt\0", G_WRITE);
fs_write(fs, input + 32, 32, fp);
fp = fs_open(fs, "t.txt\0", G_WRITE);
fs_write(fs, input + 32, 32, fp);
fp = fs_open(fs, "t.txt\0", G_READ);
fs_read(fs, output, 32, fp);
fs_gsys(fs, LS_D);
fs_gsys(fs, LS_S);
fp = fs_open(fs, "b.txt\0", G_WRITE);
fs_write(fs, input + 64, 12, fp);
fs_gsys(fs, LS_S);
fs_gsys(fs, LS_D);
fs_gsys(fs, RM, "t.txt\0");
fs_gsys(fs, LS_S);

char fname[10][20];
for (int i = 0; i < 10; i++)
{
    fname[i][0] = i + 33;
    for (int j = 1; j < 19; j++)
        fname[i][j] = 64 + j;
    fname[i][19] = '\0';
}

for (int i = 0; i < 10; i++)
{
    fp = fs_open(fs, fname[i], G_WRITE);
    fs_write(fs, input + i, 24 + i, fp);
}

fs_gsys(fs, LS_S);


for (int i = 0; i < 5; i++)
    fs_gsys(fs, RM, fname[i]);
```

The input data file should be like

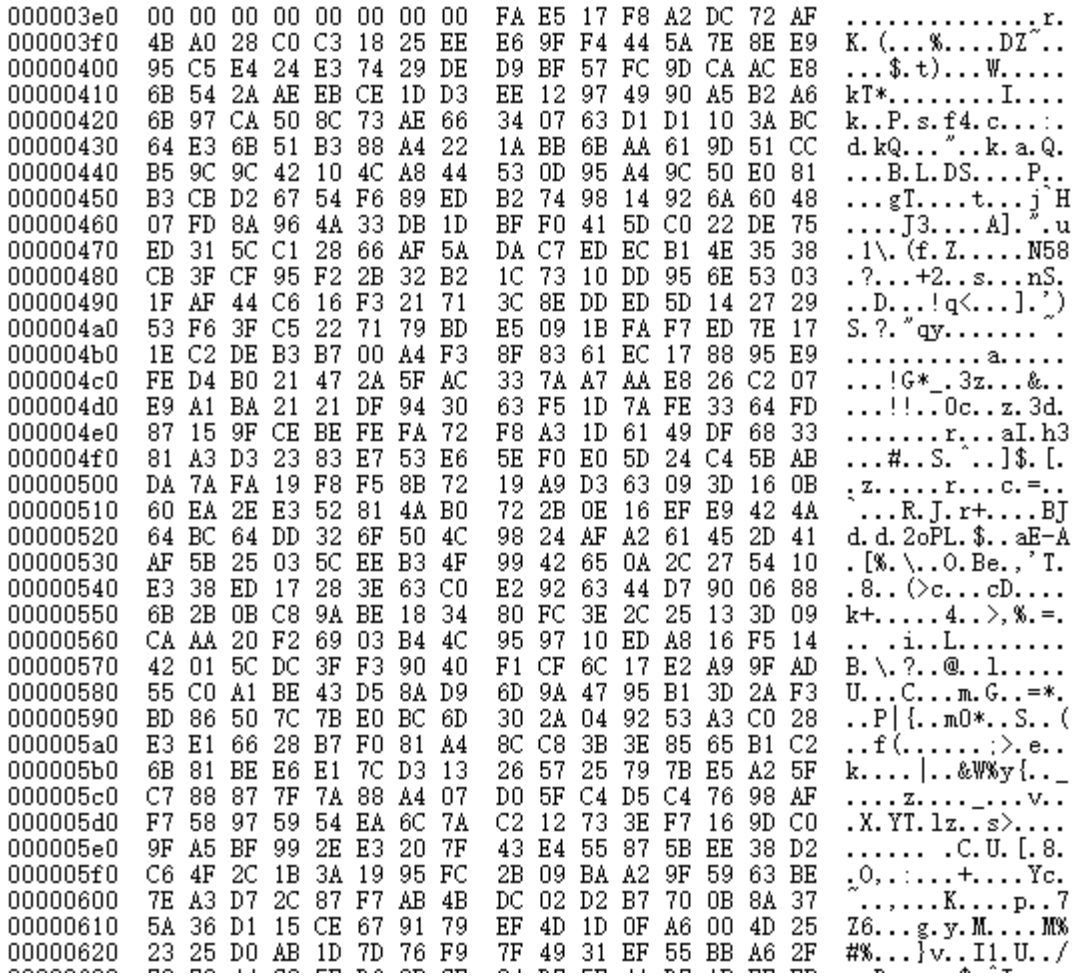


```
00000000  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
00000010  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
00000020  6F 6F 6F 6F 6F 6F 6F 6F  6F 6F 6F 6F 6F 6F 6F 6F  oooooooooooooooo
00000030  6F 6F 6F 6F 6F 6F 6F 6F  6F 6F 6F 6F 6F 6F 6F 6F  oooooooooooooooo
00000040  63 63 63 63 63 63 63 63  63 63 63 63 AA 13 68 DF  cccccccccccc..h.
00000050  13 09 57 56 D3 19 C2 70  5B 39 AB 09 1A C2 6F AD  ..WV...p[9....o.
00000060  68 52 80 14 BA B6 12 B9  F2 FC DA 9D 10 C2 FC 23  hR.............#
00000070  CC D4 F8 20 6D 3B 10 C8  74 3B D1 8F FD C0 BC E5  ... m;..t;......
00000080  13 3D FA 4E F3 0D 08 66  89 E2 83 99 25 81 3C 71  .=.N...f....%.<q
00000090  56 35 91 C3 EF 21 0C 65  5C 5D 74 5A 1F 31 41 32  V5...!.e\]tZ.1A2
000000a0  6F BB 80 E2 48 08 4A D1  6A CD EA 90 CE 27 02 25  o...H.J.j....'.%
000000b0  DB 14 68 CB 35 74 B0 92  52 25 EC 71 57 AD 23 46  ..h.5t..R%.qW.#F
000000c0  69 A4 29 31 AC F2 03 18  41 ED 28 10 94 2A B5 70  i.)1....A.(..*.p
000000d0  3E 1E BC F3 13 6D 86 65  93 F2 D6 6A A1 F9 B0 8A  >....m.e...j....
000000e0  1E 59 BC 4B 4D 3F E2 8E  2E 0B 9E C2 B4 D3 B3 73  .Y.KM?.........s
000000f0  F2 70 67 06 5D 6D 6B F0  60 C1 DA 81 3B 8B 8C 5A  .pg.]mk.`...;..Z
00000100  E5 49 25 B2 08 08 41 36  92 5F 79 47 34 2D 3A A6  .I%...A6._yG4-:.
00000110  9D A1 AC FA 0F 97 6C EF  D8 47 71 14 53 FD ED B8  ......l..Gq.S...
00000120  C6 13 6B CF 9A 2C 85 2D  8B FE F4 3F 2C 2F E5 49  ..k..,.-...?,/.I
00000130  51 12 C4 DF 29 B0 CF 02  77 C1 96 CA 3F 04 83 86  Q...)...w...?...
00000140  97 6E D5 32 9A 5B DF A6  DA 54 E5 07 03 4C D0 54  .n.2.[...T...L.T
00000150  DD 95 35 08 C5 84 89 3D  46 20 09 86 A4 0C 0D 3C  ..5....=F .....<
00000160  FA E2 ED 95 BD 4D BB 98  A1 22 20 25 6E F0 79 4C  .....M...” %n.yL
00000170  06 2E D3 CB 33 5E 09 79  FD 12 7F 22 9E 8C 5E 99  ....3^.y...”..^.
00000180  EE 4D AE AD 9A 6B C5 BC  8D E5 E1 FB D6 DA C7 DC  .M...k..........
00000190  89 1C 28 BC F9 32 36 77  C3 36 9A E1 42 F8 7B 32  ..(..26w.6..B.{2
000001a0  C5 2B 5F E0 16 25 9D A3  8B FE 1F 62 D9 66 BF 63  .+_..%.....b.f.c
000001b0  82 E7 20 7C 99 D6 74 5E  0D 0F BF 4F 08 3C 01 4E  .. |..t^...O.<.N
000001c0  E6 DF 2F FC 85 CC A0 11  CB 3F F2 25 A5 32 89 29  ../.......?.%.2.)
000001d0  1B 29 25 34 7F 99 12 8C  A8 D2 5C 31 0F DC 7F 75  .)%4......\1...u
000001e0  3D AE 72 C2 FA 92 53 46  D1 46 6B F6 79 F4 9F 14  =.r...SF.Fk.y...
000001f0  9E C5 48 1E DE 5B 2B 08  AD 87 B8 3C E3 38 B1 21  ..H..[+....<.8.!
00000200  66 A3 63 E0 B5 B6 27 07  FD 12 FD 77 08 9E 0B 26  f.c...'....w...&
00000210  E3 53 44 C2 2E 6F 4A 5B  76 03 97 5B BA 49 FB 21  .SD..oJ[v..[.I.!
00000220  6C 60 81 22 17 28 A8 94  3B 27 0C C2 C5 17 E8 29  l`.”.(..;'.....)
00000230  EA 2D EB 98 1D 37 F4 93  B9 8C 6E 75 56 EA 16 42  .-...7....nuV..B
00000240  4B 98 E4 E1 C0 8D 77 7B  B4 03 3E F9 9A 27 23 85  K.....w{..>..'#.
00000250  D4 8F 9D F1 46 12 85 00  9F 74 75 75 5F 8C 37 2A  ....F....tuu_.7*
00000260  A4 1C 0C 65 2A 83 E1 DE  87 9F 59 A1 C7 7C A6 9C  ...e*.....Y..|..
00000270  8B 44 0E D1 57 13 52 76  87 C7 6B 66 D3 A2 90 78  .D..W.Rv..kf...x
00000280  BF 9D 5E E9 A0 BF 48 A7  5F A1 49 27 9E EF 43 2A  ..^...H._.I'..C*
00000290  B4 51 7C 0C 65 CE 02 6C  16 6D D3 EA 10 E3 E2 CF  .Q|.e..l.m......
000002a0  81 41 39 A2 01 82 4A E0  24 94 87 C2 04 CB 6D B8  .A9...J.$.....m.
000002b0  9C E9 C4 81 B8 C6 EE CE  B3 42 39 C4 26 1D 14 28  .........B9.&..(
000002c0  DD 4E CA 5F D0 15 40 74  29 C7 B7 2E 13 25 66 30  .N._..@t)....%f0
000002d0  0F 2C B1 47 72 20 95 A6  62 CF 6B 09 6C 7F B0 C9  .,.Gr ..b.k.l...
```

The output file should be like



```
00000000  6F 6F 6F 6F 6F 6F 6F 6F  6F 6F 6F 6F 6F 6F 6F 6F  oooooooooooooooo
00000010  6F 6F 6F 6F 6F 6F 6F 6F  6F 6F 6F 6F 6F 6F 6F 6F  oooooooooooooooo
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

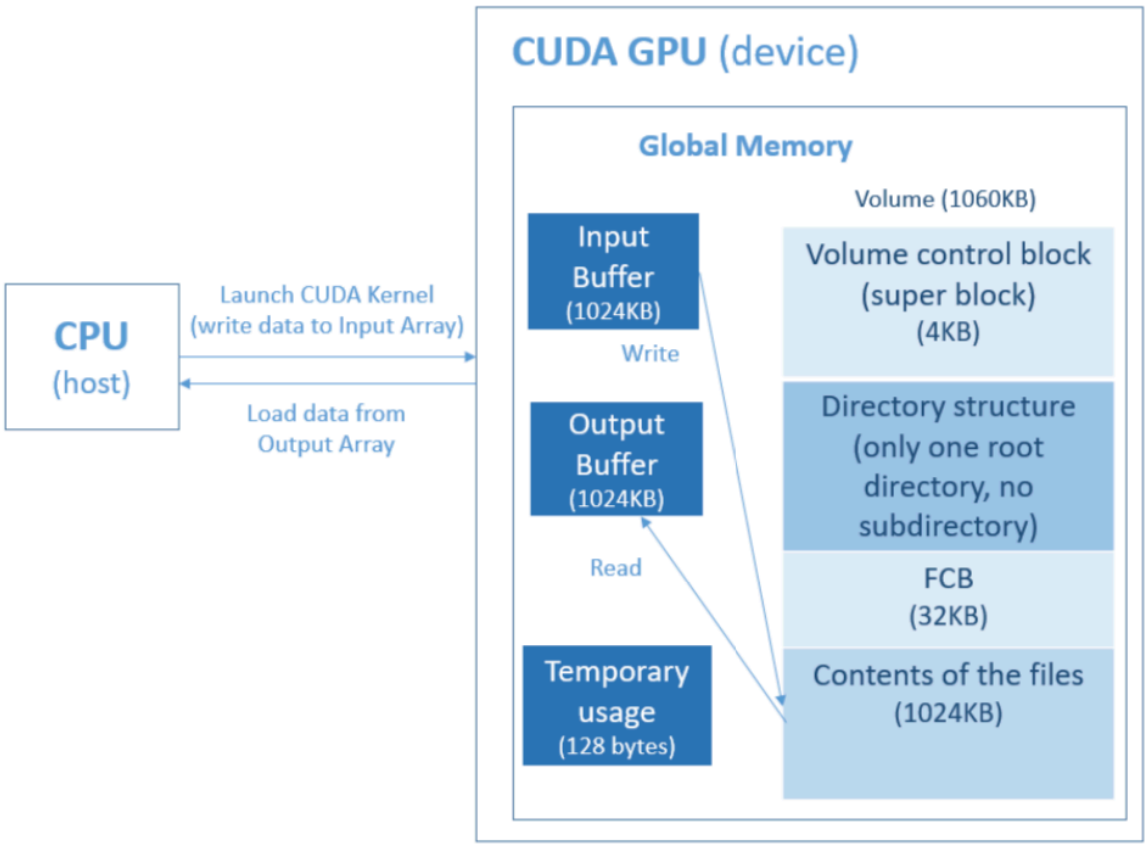


```
000003e0  00 00 00 00 00 00 00 00  FA E5 17 F8 A2 DC 72 AF  ..............r.
000003f0  4B A0 28 C0 C3 18 25 EE  E6 9F F4 44 5A 7E 8E E9  K.(...%....DZ~..
00000400  95 C5 E4 24 E3 74 29 DE  D9 BF 57 FC 9D CA AC E8  ...$.t)...W.....
00000410  6B 54 2A AE EB CE 1D D3  EE 12 97 49 90 A5 B2 A6  kT*........I....
00000420  6B 97 CA 50 8C 73 AE 66  34 07 63 D1 D1 10 3A BC  k..P.s.f4.c...:.
00000430  64 E3 6B 51 B3 88 A4 22  1A BB 6B AA 61 9D 51 CC  d.kQ...".k.a.Q.
00000440  B5 9C 9C 42 10 4C A8 44  53 0D 95 A4 9C 50 E0 81  ...B.L.DS....P..
00000450  B3 CB D2 67 54 F6 89 ED  B2 74 98 14 92 6A 60 48  ...gT....t...j`H
00000460  07 FD 8A 96 4A 33 DB 1D  BF F0 41 5D C0 22 DE 75  ....J3....A].".u
00000470  ED 31 5C C1 28 66 AF 5A  DA C7 ED EC B1 4E 35 38  .1\.(f.Z.....N58
00000480  CB 3F CF 95 F2 2B 32 B2  1C 73 10 DD 95 6E 53 03  .?...+2..s...nS.
00000490  1F AF 44 C6 16 F3 21 71  3C 8E DD ED 5D 14 27 29  ..D...!q<...].')
000004a0  53 F6 3F C5 22 71 79 BD  E5 09 1B FA F7 ED 7E 17  S.?."qy.......~.
000004b0  1E C2 DE B3 B7 00 A4 F3  8F 83 61 EC 17 88 95 E9  ..........a.....
000004c0  FE D4 B0 21 47 2A 5F AC  33 7A A7 AA E8 26 C2 07  ...!G*_.3z...&..
000004d0  E9 A1 BA 21 21 DF 94 30  63 F5 1D 7A FE 33 64 FD  ...!!..0c..z.3d.
000004e0  87 15 9F CE BE FE FA 72  F8 A3 1D 61 49 DF 68 33  .......r...aI.h3
000004f0  81 A3 D3 23 83 E7 53 E6  5E F0 E0 5D 24 C4 5B AB  ...#..S.^..]$.[.
00000500  DA 7A FA 19 F8 F5 8B 72  19 A9 D3 63 09 3D 16 0B  .z.....r...c.=..
00000510  60 EA 2E E3 52 81 4A B0  72 2B 0E 16 EF E9 42 4A  `...R.J.r+....BJ
00000520  64 BC 64 DD 32 6F 50 4C  98 24 AF A2 61 45 2D 41  d.d.2oPL.$..aE-A
00000530  AF 5B 25 03 5C EE B3 4F  99 42 65 0A 2C 27 54 10  .[%.\..O.Be.,'T.
00000540  E3 38 ED 17 28 3E 63 C0  E2 92 63 44 D7 90 06 88  .8..(>c...cD....
00000550  6B 2B 0B C8 9A BE 18 34  80 FC 3E 2C 25 13 3D 09  k+.....4..>,%.=.
00000560  CA AA 20 F2 69 03 B4 4C  95 97 10 ED A8 16 F5 14  .. .i..L........
00000570  42 01 5C DC 3F F3 90 40  F1 CF 6C 17 E2 A9 9F AD  B.\.?..@..l.....
00000580  55 C0 A1 BE 43 D5 8A D9  6D 9A 47 95 B1 3D 2A F3  U...C...m.G..=*.
00000590  BD 86 50 7C 7B E0 BC 6D  30 2A 04 92 53 A3 C0 28  ..P|{..m0*..S..(
000005a0  E3 E1 66 28 B7 F0 81 A4  8C C8 3B 3E 85 65 B1 C2  ..f(......;>.e..
000005b0  6B 81 BE E6 E1 7C D3 13  26 57 25 79 7B E5 A2 5F  k....|..&W%y{.._
000005c0  C7 88 87 7F 7A 88 A4 07  D0 5F C4 D5 C4 76 98 AF  ....z...._...v..
000005d0  F7 58 97 59 54 EA 6C 7A  C2 12 73 3E F7 16 9D C0  .X.YT.lz..s>....
000005e0  9F A5 BF 99 2E E3 20 7F  43 E4 55 87 5B EE 38 D2  ...... .C.U.[.8.
000005f0  C6 4F 2C 1B 3A 19 95 FC  2B 09 BA A2 9F 59 63 BE  .O,.:...+....Yc.
00000600  7E A3 D7 2C 87 F7 AB 4B  DC 02 D2 B7 70 0B 8A 37  ~..,...K....p..7
00000610  5A 36 D1 15 CE 67 91 79  EF 4D 1D 0F A6 00 4D 25  Z6...g.y.M....M%
00000620  23 25 D0 AB 1D 7D 76 F9  7F 49 31 EF 55 BB A6 2F  #%...}v..I1.U../
```

You should use the “Ctrl” + “F7” to compile each CUDA file and use the “Ctrl” + “F5” to execute the program.

Flow: How Did I Design My Program

1. We use the volume to store the meta data and the file data for the file system. The structure for volume should be like below



The super block part will contain 4KB, each bit represents to one block, if the block contains data, the bit will be 1, otherwise the bit will be 0.

The FCB file will contain the file information including size, name, create date, modified date and the block position for the start point. For each file the size information stores in the 1-4 bytes of the FCB file, the name stores in the 5-24, create date stores in the 25-26 bytes, the modified date stores in the 27-28 blocks and the block position stores in the 29-32 bytes. Each file in total occupies 32 bytes in the FCB parts.

The last 1024KB is used to store the data information. We can maximumly contain 1024 files and the total size for the files should not exceed 1024KB.

2. While you want to read or write the file, you firstly call the open operation. In the open operation,
(1) we will firstly check is the file has already existed or not using the input file name.
(2) if the file doesn't exist, we will find a new space in the FCB, store the name, create date, modified date, start block position in it, find a new pace in the storage and return the block position.

(3) If the file existed, we will find its FCB position and in the FCB position we will find its block position for the start point.
(4) We just return the block position if the user only wants to read the file.
(5) If the user requires to write data in the existing file, we will clean the original file data in the storage and change the bit for corresponding blocks in to 0 in the super block.
(6) pay attention that the user can’t call the read operation if the file doesn’t originally exist in the file system. If the users do so, the program will return the error signal.

```
__device__ u32 fs_open(FileSystem *fs, char *s, int op)
{
    /* Implement open operation here */
    //if not exist
    if (if_exist(fs, s) == -1) {
        if (op == 0) {
            printf("can not find the file to read error\n");
            return -1;
        }

        //store the name
        current_FCB_position = FCB_position;
        //printf("for name in open = ");
        for (int i = 4; i < 24; i++) {
            fs->volume[FCB_position + i] = s[i - 4];
        }

        //store the create date
        fs->volume[FCB_position + 24] = gtime_create >> 8;
        fs->volume[FCB_position + 25] = gtime_create;

        //strore the modified date
        fs->volume[FCB_position + 26] = gtime >> 8;
        fs->volume[FCB_position + 27] = gtime;

        //store the start block
        fs->volume[FCB_position + 28] = block_position >> 24;
        fs->volume[FCB_position + 29] = block_position >> 16;
        fs->volume[FCB_position + 30] = block_position >> 8;
        fs->volume[FCB_position + 31] = block_position;

        //update the date
        gtime++;
        gtime_create++;

        //update FCB position
        FCB_position = FCB_position + 32;
        return block_position;
    }

    //if exist
    else {
        current_FCB_position = if_exist(fs, s);
        u32 start_block = (fs->volume[current_FCB_position + 28] << 24) + (fs->volume[current_FCB_position + 29]
            << 16) + (fs->volume[current_FCB_position + 30] << 8) + (fs->volume[current_FCB_position + 31]);

        //if write
        if (op == 1) {

            //clean the old file in volume
            u32 size = (fs->volume[current_FCB_position] << 24) + (fs->volume[current_FCB_position + 1] << 16) +
                (fs->volume[current_FCB_position + 2] << 8) + (fs->volume[current_FCB_position + 3]);
            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_block_position = start_block + i;
                int shift_number = super_block_position % 8;
                fs->volume[super_block_position / 8] = fs->volume[super_block_position / 8] - (1 << shift_number);
            }
```

3. The user may want to read the file by calling the read operation
   In the read operation, we just read the file from the storage from the fp position. Write the data in the output file.

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

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

4. Then the user may want to write in the file by calling the write operation.
   (1) In the write the operation, we will first check if there are enough space to write the file.
   (2) If the space is not enough, the program will assign a new position for the file.
   (3) If the data exceed the storage space the program will return the error signal.
   (4) Then the system will load the data in the valid position in the storage and corresponding load or change the size information in the FCB.
   (5) If the file is assigned the new block position, the program will change the start block information in FCB as well.
   (6) After the write, if the external segmentation occurred, the program will do the segmentation management automatically to make sure that there is no external segment in the storage.

```
__device__ u32 fs_write(FileSystem *fs, uchar* input, u32 size, u32 fp)
{
    /* Implement write operation here */
    //enough space
    if ((fs->volume[(fp + (size - 1) / 32)/8] >> (fp + (size - 1) / 32) % 8) % 2 == 0) {
        u32 old_file_size = (fs->volume[current_FCB_position] << 24) + (fs->volume[current_FCB_position + 1] <<
            16) + (fs->volume[current_FCB_position + 2] << 8) + (fs->volume[current_FCB_position + 3]);
        u32 original_size = old_file_size - size;

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

        //update block
            if (i % 32 == 0) {
                u32 super_block_position = fp + i / 32;
                int shift_number = super_block_position % 8;
                fs->volume[(fp + i /32) / 8] = fs->volume[(fp + i / 32) / 8] + (1 << shift_number);
            }
        }
        if (int (original_size) < 0) block_position = block_position + (-original_size - 1) / 32 + 1;

        //update size
        fs->volume[current_FCB_position] = size >> 24;
        fs->volume[current_FCB_position + 1] = size >> 16;
        fs->volume[current_FCB_position + 2] = size >> 8;
        fs->volume[current_FCB_position + 3] = size;
        if (original_size > 0 && old_file_size != 0 && fp != block_position - 1) segment_management(fs, fp +
            (size - 1) / 32 + 1, original_size);
    }
```

```
    //out of space
    else {
        u32 original_size = (fs->volume[current_FCB_position] << 24) + (fs->volume[current_FCB_position + 1] <<
            16) + (fs->volume[current_FCB_position + 2] << 8) + (fs->volume[current_FCB_position + 3]);
        if (block_position * 32 - 1 + size >= fs->SUPERBLOCK_SIZE) {
            return -1;
        }

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

                //update block
                if (i % 32 == 0) {
                    u32 super_block_position = block_position + i / 32;
                    int shift_number = super_block_position % 8;
                    fs->volume[(block_position + i / 32) / 8] = fs->volume[(block_position + i / 32) / 8] + (1 <<
                        shift_number);
                }
            }

            //update size
            fs->volume[current_FCB_position] = size >> 24;
            fs->volume[current_FCB_position + 1] = size >> 16;
            fs->volume[current_FCB_position + 2] = size >> 8;
            fs->volume[current_FCB_position + 3] = size;

            //update block position
            fs->volume[current_FCB_position + 28] = block_position >> 16;
            fs->volume[current_FCB_position + 29] = block_position >> 16;
            fs->volume[current_FCB_position + 30] = block_position >> 8;
            fs->volume[current_FCB_position + 31] = block_position;
        }
        segment_management(fs, fp, original_size);
    }
}
```

5. The program also provides the LS_D operation and the LS_S operation.
   (1) In the LS_D operation, the file will be sorted and displayed by the modified time. The recent modified file will be displayed first.
   (2) In the LS_S operation, the file will be sorted by the size. The larger size file will be displayed at first. For the files with same size, the file with earlier create time will be displayed at first.

```
__device__ void fs_gsys(FileSystem *fs, int op)
{
    u32 stop_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) + (fs->volume[i + 1] << 16) + (fs->volume[i + 2] << 8) + (fs->volume[i +
            3]);
        if (size == 0) {
            size = (fs->volume[4096] << 24) + (fs->volume[4096 + 1] << 16) + (fs->volume[4096 + 2] << 8) +
                (fs->volume[4096 + 3]);
            stop_point = i - 32;
            break;
        }
        stop_point = i - 32;
    }

    //if there is no file
    if (stop_point < 4096) printf("no file in FCB error\n");

    //sort by date
    if (op == 0) {
        bubblesort(fs, 4096, stop_point, 0);
        display(fs, stop_point, 0);
    }

    //sort by size
    else {
        bubblesort(fs, 4096, stop_point, 1);
        display(fs, stop_point, 1);
    }
}
```

6. The program also provides the RM operation to remove the file from the file system.
   (1) You firstly will find the FCB position and the block position for the corresponding file.
   (2) You clean the file data in the storage and clean the information stored in the FCB and change the bit for the file to 0 in the super block.
   (3) Pay attention that you can't remove the file that is not originally in the file system. If the system doesn't find the file, it will print the file not found error.
   (4) After the write, if the external segmentation occurred, the program will do the segmentation management automatically to make sure that there is no external segment in the storage.
   (5) The program will also delete the external segment in the FCB.

```
__device__ void fs_gsys(FileSystem *fs, int op, char *s)
{
    /* Implement rm operation here */
    if (if_exist(fs, s) == -1) printf("no such file founded error\n");
    else {
        current_FCB_position = if_exist(fs, s);

        //change volume
        u32 start_block = (fs->volume[current_FCB_position + 28] << 24) + (fs->volume[current_FCB_position + 29]
            << 16) + (fs->volume[current_FCB_position + 30] << 8) + (fs->volume[current_FCB_position + 31]);

        //clean the old file in volume
        u32 size = (fs->volume[current_FCB_position] << 24) + (fs->volume[current_FCB_position + 1] << 16) +
            (fs->volume[current_FCB_position + 2] << 8) + (fs->volume[current_FCB_position + 3]);
        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++) {
            fs->volume[start_block + i] = 0;
        }

        //clean the FCB
        for (int i = 0; i < 32; i++) {
            fs->volume[current_FCB_position + i] = 0;
        }
        segment_management(fs, start_block, size);
        segment_management_FCB(fs, current_FCB_position);
        FCB_position = FCB_position - 32;
    }
}
```

Functions: How did I Design My Program

__device__ void segment_management_FCB: The function is used to delete the external segment in the FCB to make sure the data in FCB are contiguous.

```
__device__ void segment_management_FCB(FileSystem *fs, u32 fp) {
    for (int i = fp; i < 36863; i = i + 32) {
        if (fs->volume[i + 32] == 0 && fs->volume[i + 32 + 1] == 0 && fs->volume[i + 32 + 2] == 0 && fs->volume[i
            + 32 + 3] == 0) break;
        for (int j = 0; j < 32; j++) {
            fs->volume[i + j] = fs->volume[i + j + 32];
            fs->volume[i + j + 32] = 0;
        }
    }
}
```

__device__ void segment_management: The function is used to delete the external segment in storage. It will also automatically change the information in FCB for any file stored after the external segment.

```
__device__ void segment_management(FileSystem *fs, u32 fp, u32 original_size) {

    //manage the volume
    u32 position = fs->FILE_BASE_ADDRESS + fp * 32;
    u32 size = ((original_size - 1) / 32 + 1) * 32;
    while ((fs->volume[position + size] != 0 || (position + size) %32 != 0)&& position + original_size <
        fs->STORAGE_SIZE) {
        fs->volume[position] = fs->volume[position + size];
        fs->volume[position + size] = 0;
        position++;
    }
```

__device__ void display: The function is used to sequentially print the information for each file.

```
__device__ void display(FileSystem*fs, u32 stop_position, int op) {
    //display date
    char name[20];
    if (op == 0) {
        printf("===sort by modified time===\n");
        for (u32 i = 4096; i <= stop_position; i = i + 32) {
            for (int j = 4; j < 24 ; j++) {
                name[j - 4] = fs->volume[i + j];
            }
            printf("%s\n",name);
        }
    }
    else {
        u32 size;
        printf("===sort by file size===\n");
        for (u32 i = 4096; i <= stop_position; i = i + 32) {
            for (int j = 4; j < 24 ; j++) {
                name[j - 4] = fs->volume[i + j];
            }
            size = (fs->volume[i] << 24) + (fs->volume[i + 1] << 16) + (fs->volume[i + 2] << 8) + (fs->volume[i + 3]);
            printf("%s %d\n", name, size);
        }
    }
}
```

__device__ void swap: The function is used to help the bubblesort function to swap the FCB positions for two files.

```
__device__ void swap(FileSystem* fs, u32 x, u32 y) {
    for (int i = 0; i < 32; i++) {
        uchar tempt = fs->volume[x + i];
        fs->volume[x + i] = fs->volume[y + i];
        fs->volume[y + i] = tempt;
    }
}
```

__device__ void bubblesort: The function is used to sort the original FCB by modified time if the operation is LS_D or by file size if the operation is LS_S. The basic mechanism for sorting is bubble sort.

```
__device__ void bubblesort(FileSystem *fs, u32 left, u32 right, int op) {

    // sort by date
    if (op == 0) {
        for (int i = left; i < right; i = i + 32) {
            for (int j = left; j < right - i + left; j = j + 32) {
                u32 j_date_previous =  (fs->volume[j + 26] << 8) + (fs->volume[j + 27]);
                u32 j_date_after = (fs->volume[j + 26 + 32] << 8) + (fs->volume[j + 27 + 32]);
                if (j_date_previous < j_date_after) swap(fs, j, j + 32);
            }
        }
    }
    else {
        for (int i = left; i < right; i = i + 32) {
            for (int j = left; j < right - i + left; j = j + 32) {
                u32 j_size_previous = (fs->volume[j] << 24) + (fs->volume[j + 1] << 16) + (fs->volume[j + 2] <<
                    8) + (fs->volume[j + 3]);
                u32 j_size_after = (fs->volume[j + 32] << 24) + (fs->volume[j + 1 + 32] << 16) + (fs->volume[j +
                    2 + 32] << 8) + (fs->volume[j + 3 + 32]);
                u32 j_date_previous = (fs->volume[j + 24] << 8) + (fs->volume[j + 25]);
                u32 j_date_after = (fs->volume[j + 24 + 32] << 8) + (fs->volume[j + 25 + 32]);
                if (j_size_previous < j_size_after) swap(fs, j, j + 32);
                if (j_size_after == j_size_previous && j_date_previous > j_date_after) swap(fs, j, j + 32);
            }
```

__device__ u32 if_exist: The function is used to find the corresponding FCB position by name. The function will return the FCB position if the file is found and return -1 if the file is not found.

```
__device__ u32 if_exist(FileSystem *fs, char *s) {
    //return FCB position
    int flag;
    for (int i = 4096; i < 36863; i = i + 32) {
        flag = 0;
        if (fs->volume[i] == 0 && fs->volume[i + 1] == 0 && fs->volume[i + 2] == 0 && fs->volume[i + 3] == 0) {
            break;
        }
        for (int j = 4; j < 24; j++) {
            if (fs->volume[i + j] != s[j - 4]) {
                flag = 1;
                break;
            }
        }
        if (flag == 1) continue;
        if (flag == 0) return i;
    }
    return -1;
}
```

__device__ u32 fs_open: The function is used to do the open operation while the user called. It will assign a starting block for the file and store the basic information except size.

```
__device__ u32 fs_open(FileSystem *fs, char *s, int op)
{
    /* Implement open operation here */
    //if not exist
    if (if_exist(fs, s) == -1) {
        if (op == 0) {
            printf("can not find the file to read error\n");
            return -1;
        }

        //store the name
        current_FCB_position = FCB_position;
        //printf("for name in open = ");
        for (int i = 4; i < 24; i++) {
            fs->volume[FCB_position + i] = s[i - 4];
        }

        //store the create date
        fs->volume[FCB_position + 24] = gtime_create >> 8;
        fs->volume[FCB_position + 25] = gtime_create;

        //strore the modified date
        fs->volume[FCB_position + 26] = gtime >> 8;
        fs->volume[FCB_position + 27] = gtime;

        //store the start block
        fs->volume[FCB_position + 28] = block_position >> 24;
        fs->volume[FCB_position + 29] = block_position >> 16;
        fs->volume[FCB_position + 30] = block_position >> 8;
        fs->volume[FCB_position + 31] = block_position;

        //update the date
        gtime++;
        gtime_create++;

        //update FCB position
        FCB_position = FCB_position + 32;
        return block_position;
    }
```

```
//if exist
else {
    current_FCB_position = if_exist(fs, s);
    u32 start_block = (fs->volume[current_FCB_position + 28] << 24) + (fs->volume[current_FCB_position + 29]
        << 16) + (fs->volume[current_FCB_position + 30] << 8) + (fs->volume[current_FCB_position + 31]);

    //if write
    if (op == 1) {

        //clean the old file in volume
        u32 size = (fs->volume[current_FCB_position] << 24) + (fs->volume[current_FCB_position + 1] << 16) +
            (fs->volume[current_FCB_position + 2] << 8) + (fs->volume[current_FCB_position + 3]);
        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_block_position = start_block + i;
            int shift_number = super_block_position % 8;
            fs->volume[super_block_position / 8] = fs->volume[super_block_position / 8] - (1 << shift_number);
        }

        //update FCB date
        fs->volume[current_FCB_position + 26] = gtime >> 8;
        fs->volume[current_FCB_position + 27] = gtime;

        //update the date
        gtime++;
    }
    return start_block;
}
```

__device__ void fs_read: The function is to implement the read operation while the user called. It will continuously put the data from the pointer position from the storage into the output buffer.

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

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

__device__ u32 fs_write: The function is used to implement the write operation while the user called. It will write the data from the input buffer in to the storage in the valid block.

```
__device__ u32 fs_write(FileSystem *fs, uchar* input, u32 size, u32 fp)
{
    /* Implement write operation here */
    //enough space
    if ((fs->volume[(fp + (size - 1) / 32)/8] >> (fp + (size - 1) / 32) % 8) % 2 == 0) {
        u32 old_file_size = (fs->volume[current_FCB_position] << 24) + (fs->volume[current_FCB_position + 1] <<
            16) + (fs->volume[current_FCB_position + 2] << 8) + (fs->volume[current_FCB_position + 3]);
        u32 original_size = old_file_size - size;

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

        //update block
            if (i % 32 == 0) {
                u32 super_block_position = fp + i / 32;
                int shift_number = super_block_position % 8;
                fs->volume[(fp + i /32) / 8] = fs->volume[(fp + i / 32) / 8] + (1 << shift_number);
            }
        }
        if (int (original_size) < 0) block_position = block_position + (-original_size - 1) / 32 + 1;

        //update size
        fs->volume[current_FCB_position] = size >> 24;
        fs->volume[current_FCB_position + 1] = size >> 16;
        fs->volume[current_FCB_position + 2] = size >> 8;
        fs->volume[current_FCB_position + 3] = size;
        if (original_size > 0 && old_file_size != 0 && fp != block_position - 1) segment_management(fs, fp +
            (size - 1) / 32 + 1, original_size);
    }

    //out of space
    else {
        u32 original_size = (fs->volume[current_FCB_position] << 24) + (fs->volume[current_FCB_position + 1] <<
            16) + (fs->volume[current_FCB_position + 2] << 8) + (fs->volume[current_FCB_position + 3]);
        if (block_position * 32 - 1 + size >= fs->SUPERBLOCK_SIZE) {
            return -1;
        }

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

                //update block
                if (i % 32 == 0) {
                    u32 super_block_position = block_position + i / 32;
                    int shift_number = super_block_position % 8;
                    fs->volume[(block_position + i / 32) / 8] = fs->volume[(block_position + i / 32) / 8] + (1 <<
                        shift_number);
                }
            }

             //update size
             fs->volume[current_FCB_position] = size >> 24;
             fs->volume[current_FCB_position + 1] = size >> 16;
             fs->volume[current_FCB_position + 2] = size >> 8;
             fs->volume[current_FCB_position + 3] = size;

             //update block position
             fs->volume[current_FCB_position + 28] = block_position >> 16;
             fs->volume[current_FCB_position + 29] = block_position >> 16;
             fs->volume[current_FCB_position + 30] = block_position >> 8;
             fs->volume[current_FCB_position + 31] = block_position;
        }
        segment_management(fs, fp, original_size);
```

__device__ void fs_gsys: The function is used to implement the LS_S and LS_D operation while called. It will firstly find the index range for the valid file system information and call the bubblesort function to sort the FCB according to size or date.

```
__device__ void fs_gsys(FileSystem *fs, int op)
{
    u32 stop_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) + (fs->volume[i + 1] << 16) + (fs->volume[i + 2] << 8) + (fs->volume[i +
            3]);
        if (size == 0) {
            size = (fs->volume[4096] << 24) + (fs->volume[4096 + 1] << 16) + (fs->volume[4096 + 2] << 8) +
                (fs->volume[4096 + 3]);
            stop_point = i - 32;
            break;
        }
        stop_point = i - 32;
    }

    //if there is no file
    if (stop_point < 4096) printf("no file in FCB error\n");

    //sort by date
    if (op == 0) {
        bubblesort(fs, 4096, stop_point, 0);
        display(fs, stop_point, 0);
    }

    //sort by size
    else {
        bubblesort(fs, 4096, stop_point, 1);
        display(fs, stop_point, 1);
    }
}
```

__device__ void fs_gsys: The function is used to implement the remove operation while called. It will clean the old file data in the storage, clean the block in the super block, clean the file information in the FCB. It will also call the segment_management and segment_management_FCB functions to solve the external segment problem.

```
__device__ void fs_gsys(FileSystem *fs, int op, char *s)
{
    /* Implement rm operation here */
    if (if_exist(fs, s) == -1) printf("no such file founded error\n");
    else {
        current_FCB_position = if_exist(fs, s);

        //change volume
        u32 start_block = (fs->volume[current_FCB_position + 28] << 24) + (fs->volume[current_FCB_position + 29]
            << 16) + (fs->volume[current_FCB_position + 30] << 8) + (fs->volume[current_FCB_position + 31]);

        //clean the old file in volume
        u32 size = (fs->volume[current_FCB_position] << 24) + (fs->volume[current_FCB_position + 1] << 16) +
            (fs->volume[current_FCB_position + 2] << 8) + (fs->volume[current_FCB_position + 3]);
        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++) {
            fs->volume[start_block + i] = 0;
        }

        //clean the FCB
        for (int i = 0; i < 32; i++) {
            fs->volume[current_FCB_position + i] = 0;
        }
        segment_management(fs, start_block, size);
        segment_management_FCB(fs, current_FCB_position);
        FCB_position = FCB_position - 32;
    }
}
```

Results: Screen Shot of My Output Result

Result for case 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
```

Result for case 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
```

Result for case 3

```
<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
```

Results: What Are the Problems I Met in this Assignment and My Solutions

1. 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.

2. 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.

3. If we write the file in the existing file and the new file size exceed the old file size, we need to reallocate the block for the new file. This will cause the external segment problem in the storage.
   My solution is to call the segmentation management every time we rewrite a file and remove the file to reuse the external segmentation.

4. We have to change each bit while one block is token in the super block. However, the basic unit in the unsigned character array is one byte.
   My solution: let the (block position / 8) be the number of array charts in the super block and let the (block position % 8) be shift number. We will each time add the 2 to the power of the shift number in the corresponding chart. This will allow each bit to represent one block condition.

Conclusion: What I have learnt from this program

1. 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.

2. When the recursion occurred in the cuda, the data will be stored in stack which may cause conflict if the later allocate other arrays.

3. The structure for the file system should contain super block, which is used to store the bitmap, FCB, which is used to store file information, storage which is used to store the data in the file.

4. 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 occurred.