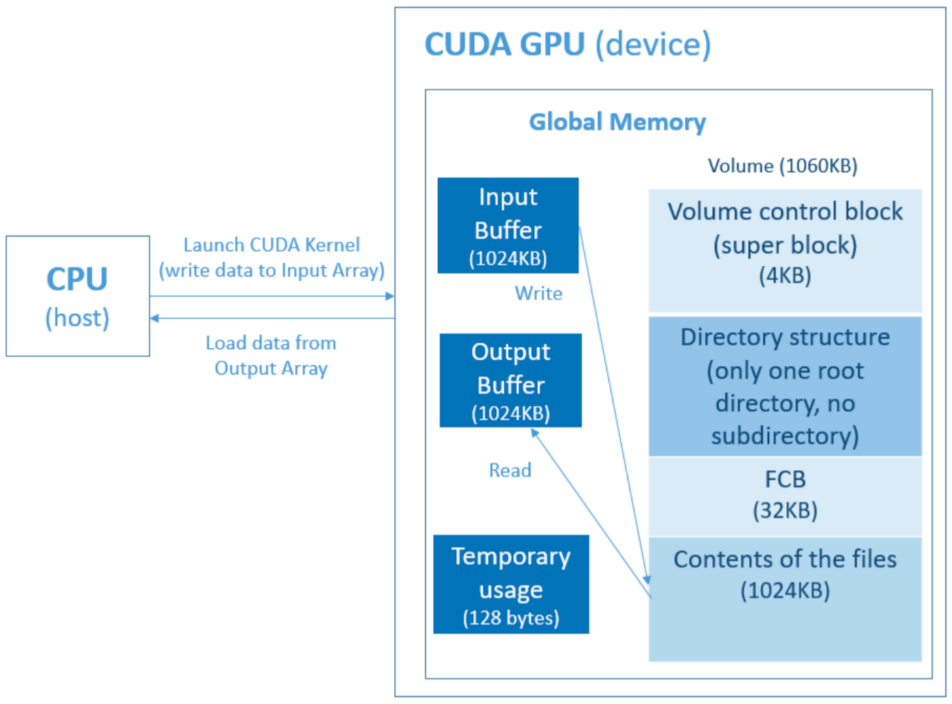
CSC3150 Assignment4 Report

Introduction

In this assignment, a simple mechanism of file system management was implemented via GPU's memory with single thread. There are five file operations implemented: *fs_open, fs_write, fs_read, fs_gsys(RM) and fs_gsys(LS_D/LS_S)* . The source code of this file system implementation has been attached with this report, which is in the file *"file_system.cu"*.

Specification:

- The global memory was taken as a volume
- Contiguous allocation was adopt
- The size of volume - 1060 KB
- The total size of files - 1024 KB
- The maximum number of file - 1024
- The maximum size of a file - 1 KB (1024 bytes)
- The maximum size of a file name - 20 bytes (file name ends with '\0')
- FCB size - 32 bytes
- FCB entries - 32 KB / 32 bytes = 1024
- Storage block size - 32 bytes


CUDA GPU (device)
Global Memory
Volume (1060KB)
Input Buffer (1024KB)
Write
Output Buffer (1024KB)
Read
Temporary usage (128 bytes)
Volume control block (super block) (4KB)
Directory structure (only one root directory, no subdirectory)
FCB (32KB)
Contents of the files (1024KB)
CPU (host)
Launch CUDA Kernel (write data to Input Array)
Load data from Output Array

Program design

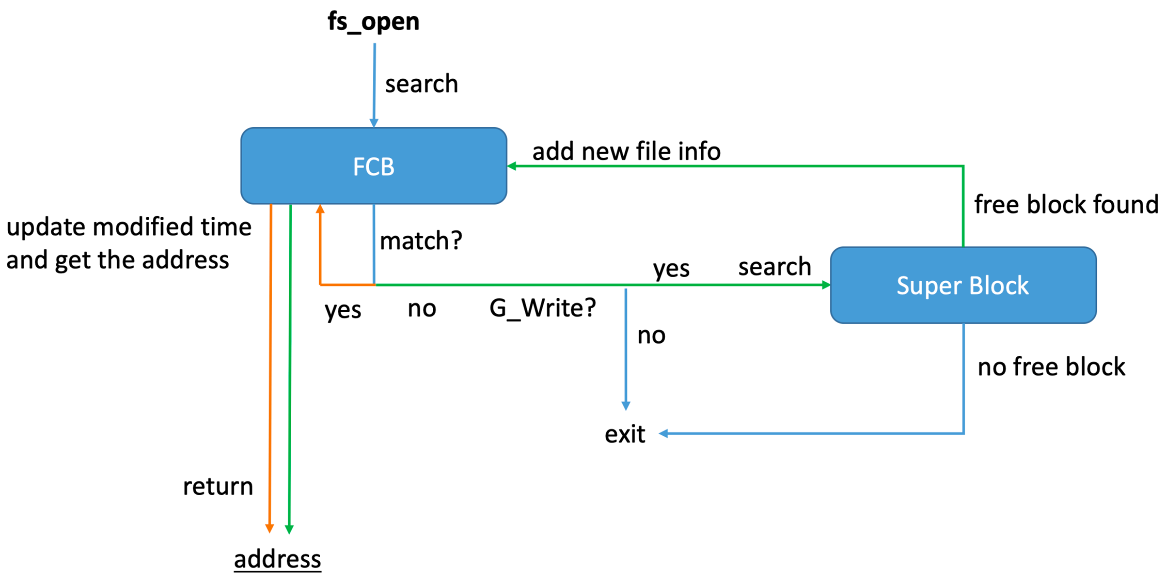
Initialization of the super block

- The super block was intialized in the function *fs_init*:

```
// initialize the Super Block
for (int i = 0; i < SUPERBLOCK_SIZE; ++ i){
    volume[i] = 255;  // 255 means 1111 1111
}
```

where 255 means 1111 1111 in binary, since each uchar could represent a 8-bit bit map for 8 blocks.

fs_open



As the logic graph above shows, when the function *fs_open()* is called, the following steps was processed:

1. Seach for the FCB entry based on the input file name:

```
// serch the file info in FCB
for (i = fs->SUPERBLOCK_SIZE; i < fs->FILE_BASE_ADDRESS; i += fs-
>FCB_SIZE){
    is_match = true;
    // comparing the file name
    for (j = 0; j < fs->MAX_FILENAME_SIZE; ++j) {
        if (s[j] != fs->volume[i + j]) {
            is_match = false;
            break;
        }
    }
    if (is_match) {
        index = i;  // The FCB entry address of the target file
        break;
    }
}
```

2. Return the physical address if file name matched:

```
if (index != -1) {     // the file exists
    // update the modified time
    fs->volume[index + 31] = gtime % 256;
    fs->volume[index + 30] = (gtime >> 8) % 256;
    // get address from FCB
    u32 address = fs->volume[index+23] + (fs->volume[index+22]<<8) + (fs-
>volume[index+21]<<16) + (fs->volume[index+20]<<24);
    return address;
}
```

3. If no matched file name and G_Write is input op, then search in Super Block:

```
for (i = 0; i < fs->SUPERBLOCK_SIZE; i += 4){
    if (fs->volume[i] == 255){  // every uchar represents 8 bits(block)
        fs->volume[i] = 127;
        index = i * 8;          // #start block of the file
        break;
    }
} // end for
```

4. If there is a free block, calculate the entry address of FCB and the file information:

```
base = fs->SUPERBLOCK_SIZE + index;
start = fs->FILE_BASE_ADDRESS + index * (fs->STORAGE_BLOCK_SIZE);  // the
physical address of the file in byte
size = 0;           // the size of the file in in byte
created = gtime;    // get current time as created time
modified = gtime;   // get current time as modified time
```

5. Store the information to FCB and return the physical address of the file:

```
// store the file name
for (j = 0; j < fs->MAX_FILENAME_SIZE; ++j) {
    fs->volume[base + j] = s[j];
    if (s[j] == '\0') break;
}

// store the physical start address of the file (4B)
fs->volume[base + 23] = start % 256;
fs->volume[base + 22] = (start >> 8) % 256;
fs->volume[base + 21] = (start >> 16) % 256;
fs->volume[base + 20] = (start >> 24) % 256;

// store the size of the file (4B)
fs->volume[base + 27] = size % 256;
fs->volume[base + 26] = (size >> 8) % 256;
```

```
// store the created time
fs->volume[base + 29] = created % 256;
fs->volume[base + 28] = (created >> 8) % 256;

// store the modified time
fs->volume[base + 31] = modified % 256;
fs->volume[base + 30] = (modified >> 8) % 256;

return start;
```

fs_read

For *fs_read()*, simply read the data from physical storage based on the input file pointer and copy the data to the output buffer:

```
__device__ void fs_read(FileSystem *fs, uchar *output, u32 size, u32 fp)
{
  for (int i = 0; i < size; ++ i) {
    output[i] = (char)fs->volume[fp++];
  }
}
```

fs_write

For file writing:

1. Increment the timing since the file is going to be modified:

```
gtime++;
```

2. Based on the input file pointer, calculate the corresponding entry address of FCB:

```
u32 index, base, file_size;
index = (fp - fs->FILE_BASE_ADDRESS) >> 10;  // the index of the entry of
FCB
base = fs->SUPERBLOCK_SIZE + index * fs->FCB_SIZE;  // address of the
entry
```

3. Get the size of the old file for later storage cleanup use:

```
file_size = fs->volume[base+27] + (fs->volume[base+26]<<8) + (fs-
>volume[base+25]<<16) + (fs->volume[base+24]<<24);
```

4. Update the size of the file and the modified time:

```
// update the modified time
fs->volume[base+31] = gtime % 256;
fs->volume[base+30] = (gtime >> 8) % 256;

// update the size of file
fs->volume[base + 27] = size % 256;
fs->volume[base + 26] = (size >> 8) % 256;
```

5. Write the physical storage, update the super block, and cleanup the contents of the old file based on the input file pointer:

```
// write the physical storage
for (int i = 0; i < size; ++ i){
  // update the super block everytime writing a new block
  if (i % 32 == 0) {
    int block_order = i / 32;
    uchar target = fs->volume[index * 4 + block_order / 8];
    if (target / (1 << (7 - (block_order % 8))) % 2 == 1)
      target -= (1 << (7 - (block_order % 8)));         // target block: 1
-> 0
    fs->volume[index * 4 + block_order / 8] = target;
  }
  // write
  fs->volume[fp++] = input[i];
}

// cleanup the older contents left
if (file_size > size) {
  for (int i = fp; i < fp + file_size - size; ++i){
    fs->volume[i] = '\0';
  }
}
```

6. Return the file pointer.

fs_gsys(LS_D/LS_S)

Basically, regardless of the printing information, the logic of the implementation of LS_D and LS_S are the same, except that LS_D sorts the modified time and LS_S sorts the file size. Thus here just elaborated the implementation of LS_D only. The insertion sort was adopt as the sorting method, which means it put the newly added elements to the right position everytime add a new element into the array.

1. Prepare the array for sorting:

```
u32 sort_arr[fs->MAX_FILE_NUM];
int arr_size = 0;
```

2. Get modified time (for LS_S, size) information from FCB and do the sorting:

```
for (int i = 0; i < fs->FCB_ENTRIES; ++ i){
    base = fs->SUPERBLOCK_SIZE + i * fs->FCB_SIZE;
    // FCB entry is not empty
    if (fs->volume[base] != '\0'){
        entry = i;
        modified = fs->volume[base+31] + (fs->volume[base+30] << 8);
        int ptr = arr_size - 1;
        sort_arr[arr_size++] = (entry << 16) + modified; // add to the
sort array
        // insertion sort
        while (ptr >= 0 && modified < sort_arr[ptr] % (1 << 16)) {
            sort_arr[ptr + 1] = sort_arr[ptr];
            sort_arr[ptr] = (entry << 16) + modified;
            ptr--;
        } // end while
    } // end if
} // end for
```

   Notice that, during the sorting, the information of the FCB entry and the modified time were translated as a *u32* data, since *u32* data type takes 4 bytes and the FCB entry and modified time take 2 bytes each. Here, the most significant two bytes represents FCB entry and the least significant two bytes represents modified time. The sorting only focus on the modified time (least siginificant two bytes) and the FCB entry would be extracted for locating the file name after sorting.

3. Print the sorting results:

```
printf("===sort by modified time===\n");
while (--arr_size >= 0) {
    entry = sort_arr[arr_size] >> 16;
    printf("%s\n", &fs->volume[fs->SUPERBLOCK_SIZE + entry * fs-
>FCB_SIZE]);
}
```

fs_gsys(RM)

For file removement:

1. Search for the FCB entry based on the input file name, which is the same as what in the *fs_open()*

```
int index = -1;
for (i = fs->SUPERBLOCK_SIZE; i < fs->FILE_BASE_ADDRESS; i += fs-
>FCB_SIZE) {
    is_match = true;
    // comparing the file name
    for (j = 0; j < fs->MAX_FILENAME_SIZE; ++j) {
        if (s[j] != fs->volume[i + j]) {
            is_match = false;
            break;
        }
    }
    if (is_match) {
        index = i;  // address of the FCB entry
        break;
    }
} // end for
```

2. If the file is found, get the address and the size of the file:

```
u32 address = fs->volume[index + 23] + (fs->volume[index + 22] << 8) +
(fs->volume[index + 21] << 16) + (fs->volume[index + 20] << 24);
u32 size = fs->volume[index + 27] + (fs->volume[index + 26] << 8);
```

3. Release the space of the file based on the address:

```
for (i = 0; i < size; ++i) {
    fs->volume[address + i] = '\0';
}
```

4. Release the corresponding FCB:

```
for (i = 0; i < fs->FCB_SIZE; ++i) {
    fs->volume[index + i] = '\0';
}
```

5. Update the super block:

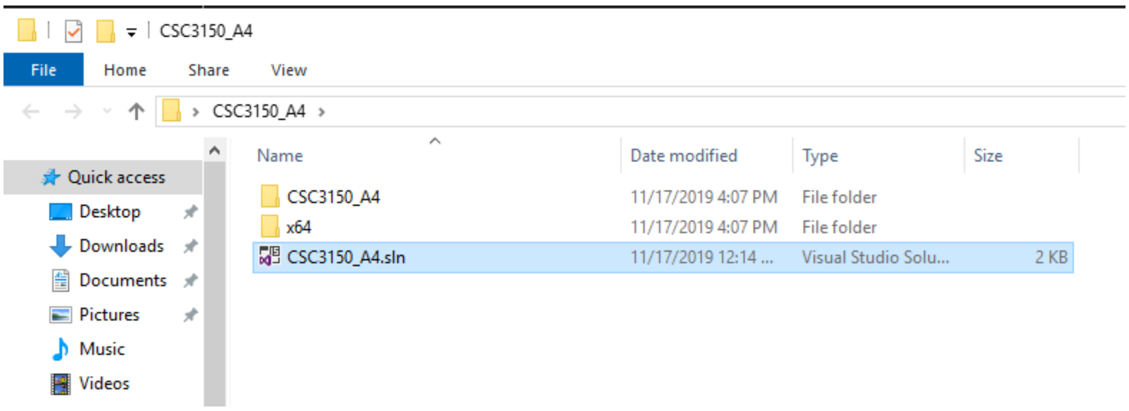
```
index = (index - fs->SUPERBLOCK_SIZE) / fs->FCB_SIZE;
for (i = 0; i < 4; ++i) {
    fs->volume[index * 4 + i] = 255;
}
```

Running environment

- Windows 10 Enterprise
- Microsoft Visual Studio 2017
- CUDA 9.2
- NVIDIA Geforce GTX 1060 6GB
- Compute capacity: 6.1

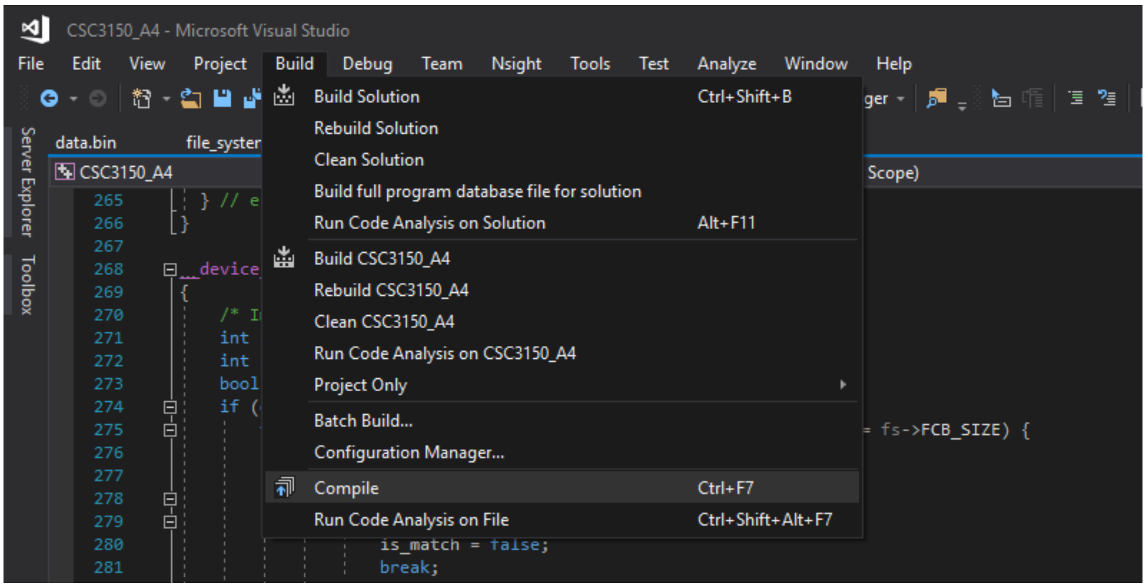
Steps to run the program

1. In the folder of "CSC3150_A4", open "CSC3150_A4.sln" with VS 2017:



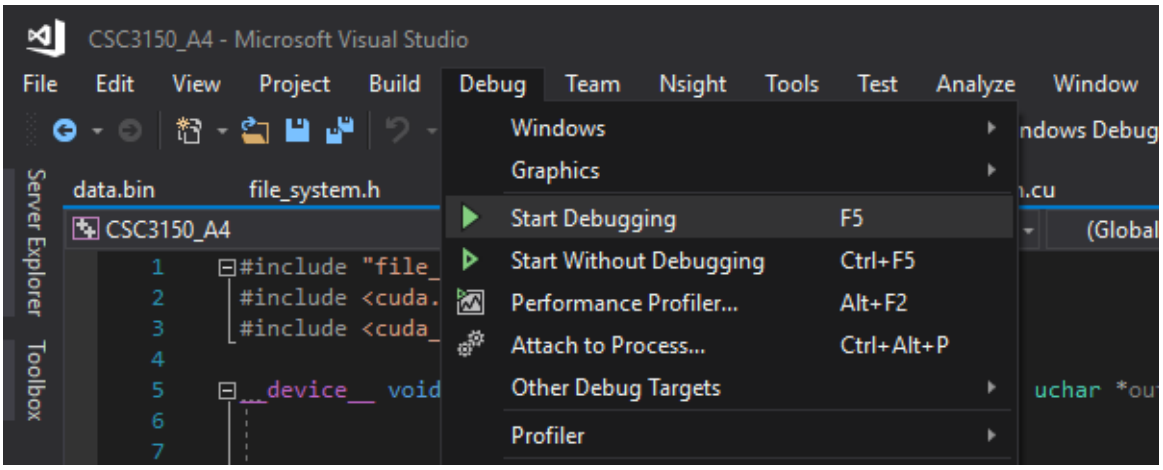
```
Name                Date modified          Type                  Size
CSC3150_A4          11/17/2019 4:07 PM     File folder
x64                 11/17/2019 4:07 PM     File folder
CSC3150_A4.sln      11/17/2019 12:14 ...   Visual Studio Solu...  2 KB
```

2. In the VS 2017, compile all the file end with ".cu" by pressing 'Ctrl' + 'F7' :



```
    } // e
}

__device
{
    /* I
    int
    int
    bool
    if (
                      fs->FCB_SIZE) {
        is_match = false;
        break;
```

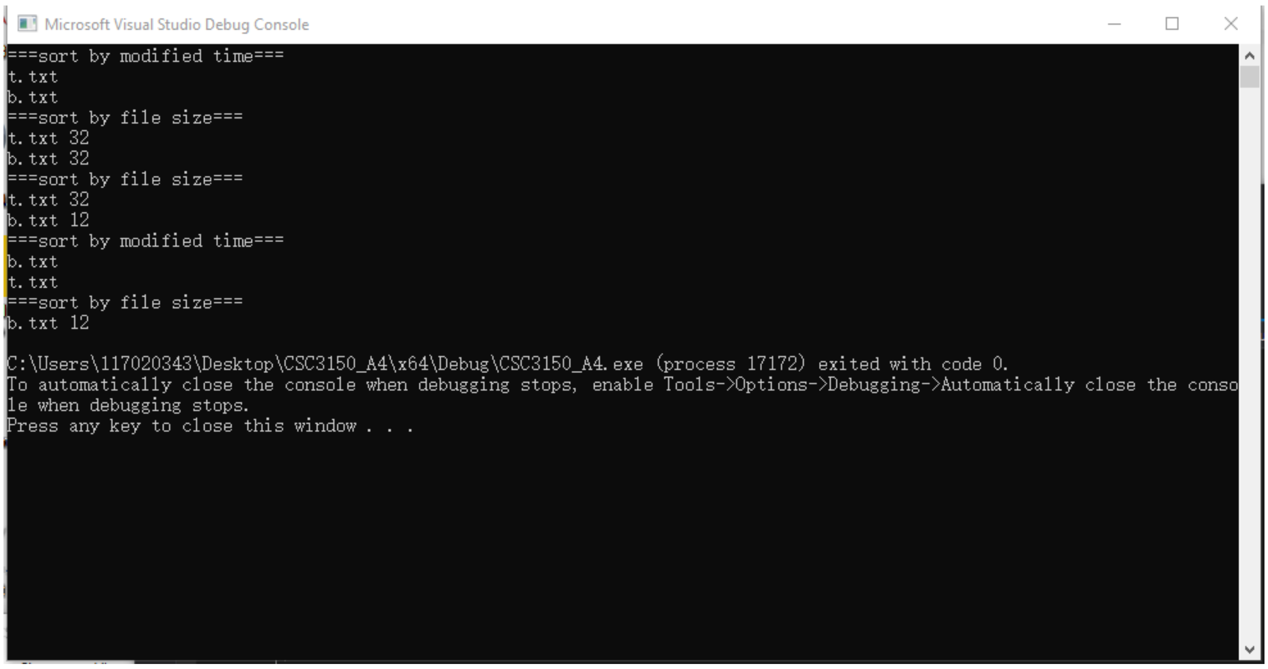
3. Select "Debug" item in the menu and click "Start Debugging" to run the program (or directly press F5):



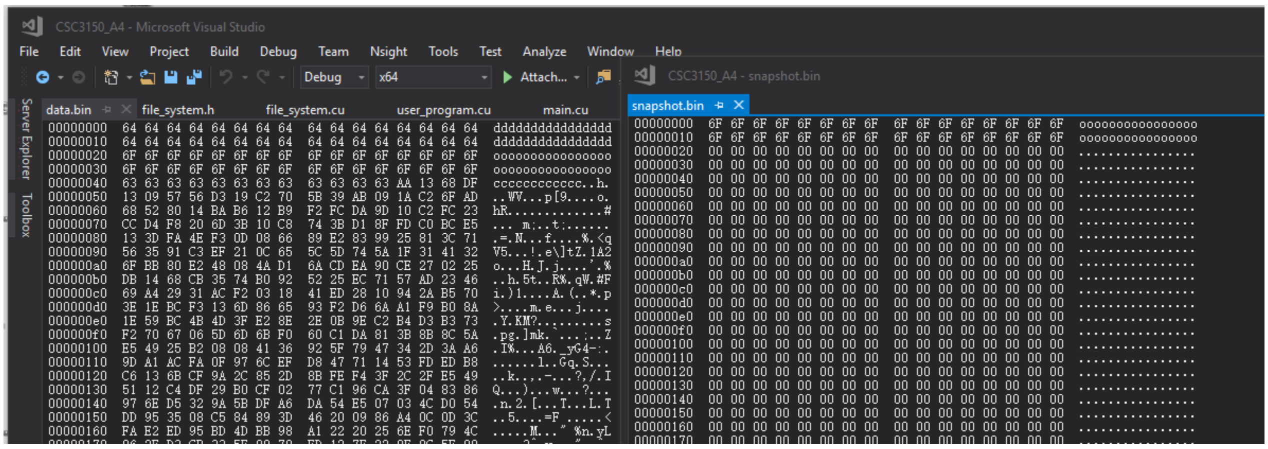
```
#include "file
#include <cuda.
#include <cuda_

__device__ void                         uchar *ou
```

Sample output

Test case 1



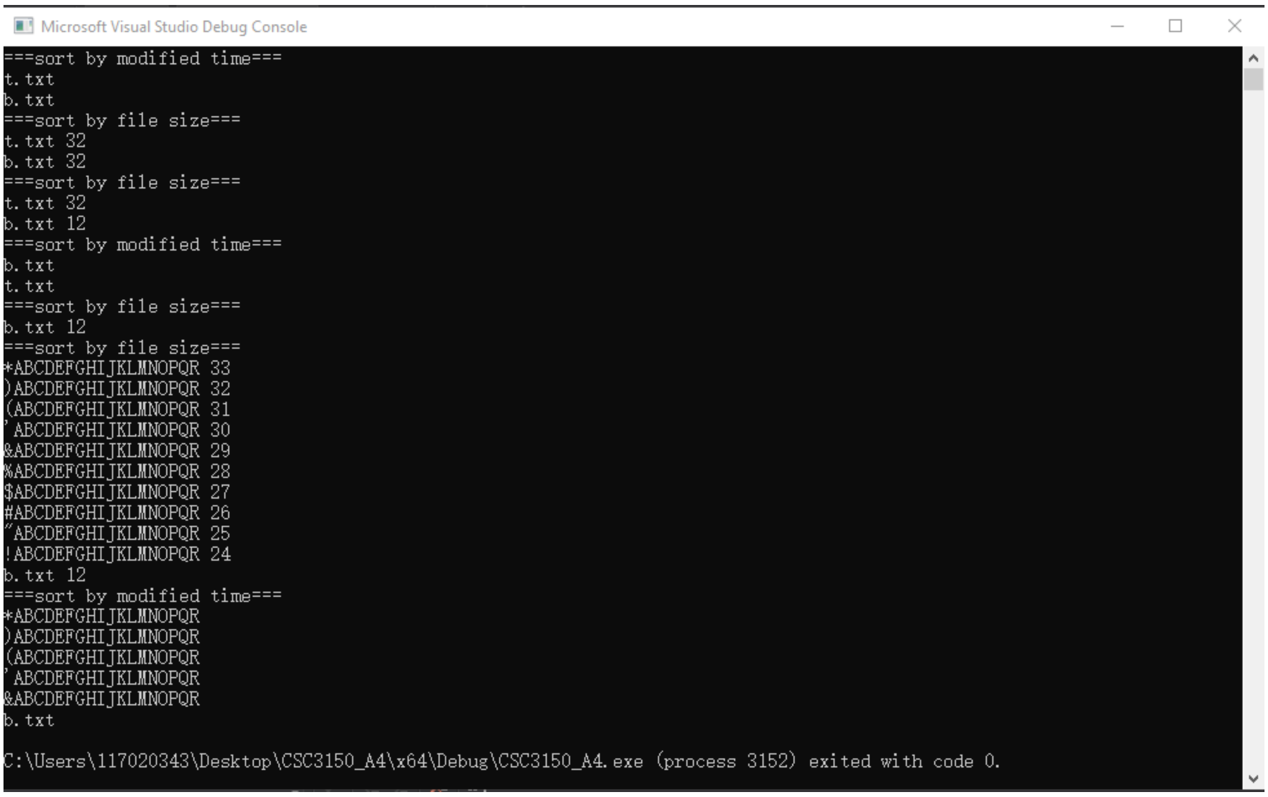
```
===sort by modified time===
t.txt
b.txt
===sort by file size===
t.txt 32
b.txt 32
===sort by file size===
t.txt 32
b.txt 12
===sort by modified time===
b.txt
t.txt
===sort by file size===
b.txt 12

C:\Users\117020343\Desktop\CSC3150_A4\x64\Debug\CSC3150_A4.exe (process 17172) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```



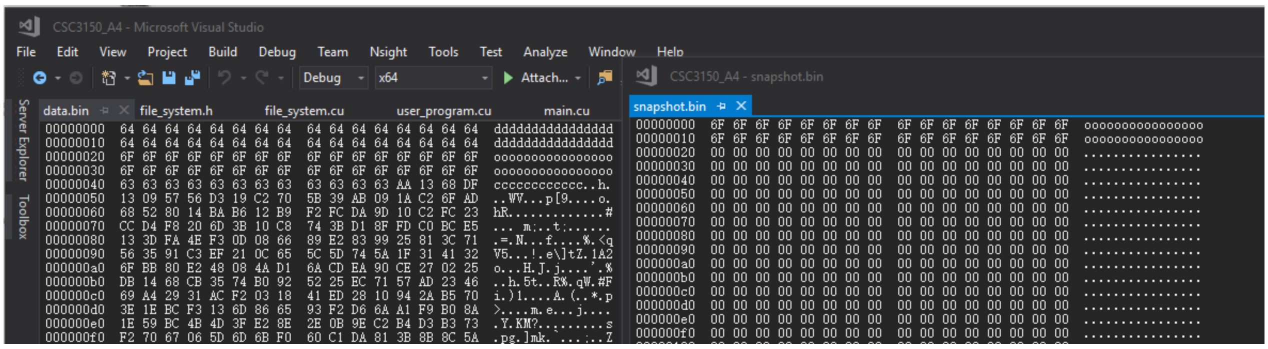
```
00000000  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
00000010  64 64 64 64 64 64 64 64  64 64 64 64 64 64 64 64  dddddddddddddddd
00000020  6F 6F 6F 6F 6F 6F 6F 6F  6F 6F 6F 6F 6F 6F 6F 6F  oooooooooooooooo
00000030  6F 6F 6F 6F 6F 6F 6F 6F  6F 6F 6F 6F 6F 6F 6F 6F  oooooooooooooooo
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
```

```
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  ................
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Test 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

C:\Users\117020343\Desktop\CSC3150_A4\x64\Debug\CSC3150_A4.exe (process 3152) exited with code 0.
```



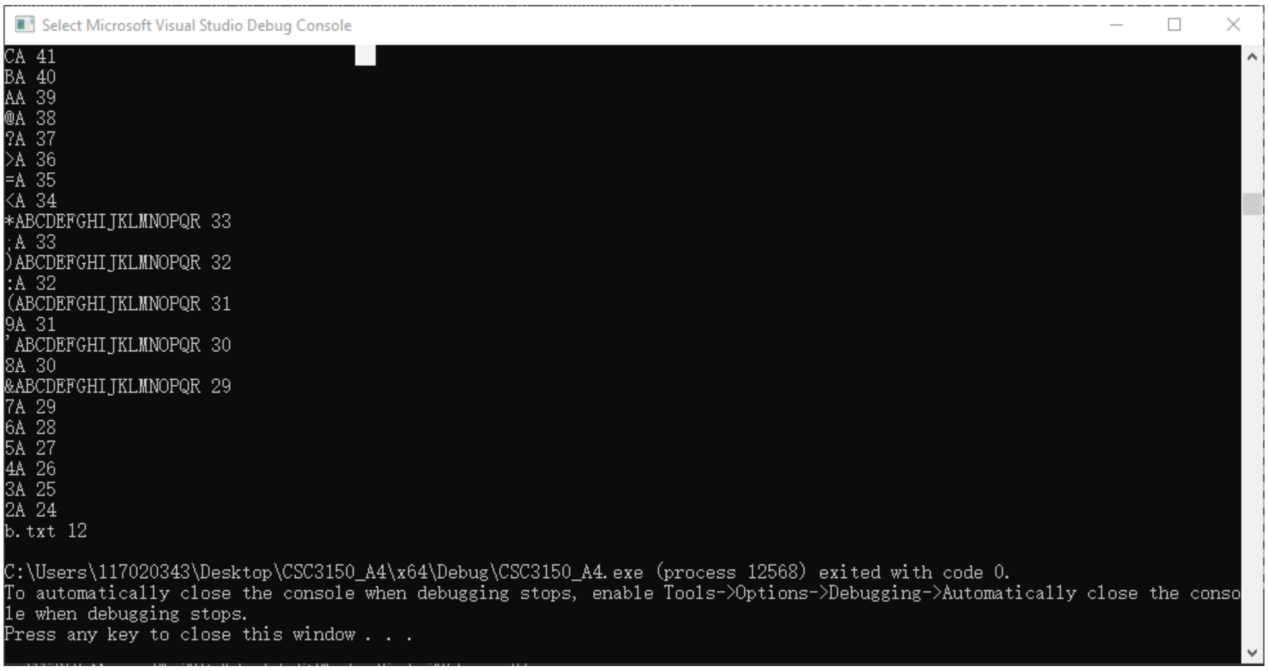
CSC3150_A4 - Microsoft Visual Studio
File Edit View Project Build Debug Team Nsight Tools Test Analyze Window Help
Debug x64 Attach...
CSC3150_A4 - snapshot.bin
data.bin file_system.h file_system.cu user_program.cu main.cu
snapshot.bin
Server Explorer Toolbox

```
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
```

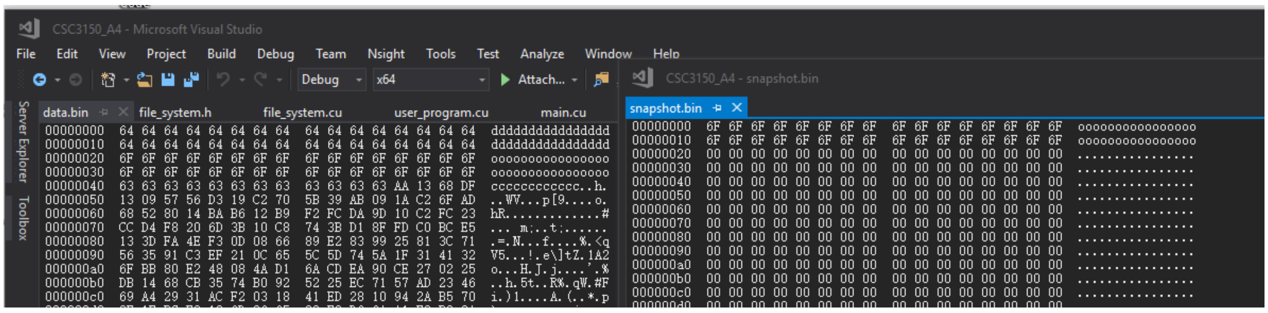
```
00000000  6F 6F 6F 6F 6F 6F 6F 6F  6F 6F 6F 6F 6F 6F 6F 6F  oooooooooooooooo
00000010  6F 6F 6F 6F 6F 6F 6F 6F  6F 6F 6F 6F 6F 6F 6F 6F  oooooooooooooooo
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
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  ................
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Test case 3 (part)

The output for Test case 3 has thousands of lines, here only show the last several lines.



```
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

C:\Users\117020343\Desktop\CSC3150_A4\x64\Debug\CSC3150_A4.exe (process 12568) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```



CSC3150_A4 - Microsoft Visual Studio
File Edit View Project Build Debug Team Nsight Tools Test Analyze Window Help
Debug x64 Attach...
CSC3150_A4 - snapshot.bin
data.bin file_system.h file_system.cu user_program.cu main.cu
snapshot.bin
Server Explorer Toolbox

```
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
```

```
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  ................
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Encountered problems

Problem

The given FCB size is limited to 32 bytes and each byte is stored in form of *uchar*, but each kind of information to be stored is larger than 1 bytes and inform of u32 or other types.

Solution

To make the most of each bytes in FCB, the distribution was made carefully. For each entry of the FCB, it has 32 bytes and was specified as follows:

- File name - 20 bytes

- The physical address of the file - 4 bytes
- The size of the file - 2 bytes
- The created time - 2 bytes
- The modified time - 2 bytes.

Other information of files is not covered since it is not used in this task.

To store the information into FCB or extract the information from FCB, some transformations are required:

- For file names, since it is in form of *char* with 1 byte each, simply copy the name character by character, byte by byte.
- For the other information, they are in form of *u32*, which is 4 bytes each, so bit operations are needed. The integer was regarded as 32-bit binary number and was partitioned as several 8-bit numbers (1 byte each) to store. And these 8-bit number was combined together again when the information was extracted from the FCB.

Set the physical address of files as an example:

Store:

```
// store the physical start address of the file (4B)
fs->volume[base + 23] = addr % 256;
fs->volume[base + 22] = (addr >> 8) % 256;
fs->volume[base + 21] = (addr >> 16) % 256;
fs->volume[base + 20] = (addr >> 24) % 256;
```

Extract:

```
u32 address = fs->volume[index+23] + (fs->volume[index+22]<<8) + (fs-
>volume[index+21]<<16)                    + (fs->volume[index+20]<<24);
```

Problem

When the program run the test case 1, it exited unexpectedly.

Solution

This may be caused by segment fault, which refers to the problem of memory overflow. Function *printf()* was used to keep track of the variables during program running and the output showed that some variables went wrong before the program exited. To locate the problem, more *printf()* was used to track. Finally the problem was found at the bit operation. As for the priority of the operators, “+", "-", "*", "/", "%" are higher than "<<" and ">>" , but the operations of "<<" and ">>" in my code were not put in brackets, which resulted in the problem.

What I learned

Better understanding of the mechanism of file system

- How does each part work together: volume structure, super block, file control block (FCB), free-space management etc.

- How to allocate the file into physical storage: the implementation of contiguous allocation
- To make best of the space, some operation might needed to translate the data.
- Better understanding of the implementation of file related system call and how they work together to attain some user operations.
- ...

Bonus

Tree structure

To implement the directory, tree structure was adopted and a new block in the volume was taken to store the information of the tree. Since there are at most 1024 files, so totally 1024 entries for the directory and each entry takes 27 bytes with following distribution:

Entry	is_dir	id	parent_id	num_files	depth	file_name
Bytes taken	1	2	2	1	1	20

where:

- **is_dir** is to identify whether the file is a directory or not.
- **id** is the id of the file, which is equal to the index of the entry, so id is smaller than 1024.
- **parent_id** records the id of the parent of the file. (for root directory, its parent id was set as 1024)
- **num_files** records how many files (including directories) contained in this directory, for non-directory, this variable is 0. This is to avoid the files under a directory exceeds 50.
- **depth** records the depth of the file in the tree structure, this is to avoid the depth of tree exceeds 3.

To keep the track of the current direcory, a global variable current_dir was set and initilized as 0:

```
__device__ __managed__ u32 current_dir = 0;
```

To implement the directory, we just need to maintain the structure above and the global variable above.

Root directory initialization

In the function *fs_init()*:

```
// dir flag
// 1 means directory file, 0 means non-directory file
fs->volume[fs->DIR_BASE_ADDRESS] = 1;

// dir id
fs->volume[fs->DIR_BASE_ADDRESS+2] = 0;
fs->volume[fs->DIR_BASE_ADDRESS+1] = 0;
```

```
// parent id
fs->volume[fs->DIR_BASE_ADDRESS+4] = 1024 % 256; // since no id would be 1024,
fs->volume[fs->DIR_BASE_ADDRESS+3] = 1024 >> 8;  // set parent as 1024
represents root dir

// number of files contained
fs->volume[fs->DIR_BASE_ADDRESS+5] = 0;

// depth
fs->volume[fs->DIR_BASE_ADDRESS+6] = 0;

// name
fs->volume[fs->DIR_BASE_ADDRESS+7] = '/';
fs->volume[fs->DIR_BASE_ADDRESS+8] = '\0';
```

fs_gsys(MKDIR)

1. Get the depth of the current directory in the tree structure, and check whether it is smaller than 3. If is smaller than 3, continue; otherwise, exit.

   ```
   int depth = fs->volume[fs->DIR_BASE_ADDRESS + current_dir*fs->DIR_SIZE +
   6];
   ```

2. Search in the directory strucutre, find an empty entry:

   ```
   for (int i = 0; i < fs->DIR_ENTRIES; ++ i){
       // find empty dir entry
       if (fs->volume[fs->DIR_BASE_ADDRESS + i*fs->DIR_SIZE] == '\0'){
           new_id = i;
           break;
       }
   }
   ```

3. Add the information of the new directory into directory structure:

   ```
   int base = fs->DIR_BASE_ADDRESS + new_id * fs->DIR_SIZE;  // base address

   // dir flag
   fs->volume[base] = 1;

   // dir id
   fs->volume[base+2] = new_id % 256;
   fs->volume[base+1] = new_id >> 8;

   // parent id
   fs->volume[base+4] = current_dir % 256; // not id would be 1024,
   ```

```
fs->volume[base+3] = current_dir >> 8;  // set parent as 1024 represents
root dir

// number of files contained
fs->volume[base+5] = 0;

// depth
fs->volume[base+6] = depth + 1;

// name
for (int j = 0; j < fs->MAX_FILENAME_SIZE; ++j) {
    fs->volume[base+7+j] = s[j];
    if (s[j] == '\0') break;
}
```

4. Update the FCB and the physical storage of files:

```
// update FCB and file space
fs_open(fs, s, 1);
```

5. Update the parent directory in directory structure:

```
// increment the num_files of the parent directory
fs->volume[fs->DIR_BASE_ADDRESS+dir_index*fs->DIR_SIZE+5] += 1;
```

fs_gsys(CD)

Search the directory structure for the target directory, by focusing the subdirectories of the current directory. After find the id of the target directory, update the *current_dir* so as to enter the directory:

```
int parent, base;
bool is_match;
// serch in directory
for (int i = 0; i < fs->DIR_ENTRIES; ++i){
    base = fs->DIR_BASE_ADDRESS + i * fs->DIR_SIZE;
    parent = fs->volume[base + 4] + (fs->volume[base+3] << 8);
    // only focus on subdirectory
    if (parent == current_dir){
        is_match = true;
        for (int j = 0; j < fs->MAX_FILENAME_SIZE; ++j) {
            if (s[j] != fs->volume[i + j]) {
                is_match = false;
                break;
            }
        }
        // update current directory
        if (is_match) {
            current_dir = i;
            break;
        }
    }
} // end for
```

fs_gsys(CD_P)

```
// get the base address of current directory
int base = fs->DIR_BASE_ADDRESS + current_dir * fs->DIR_SIZE;
// get the id of parent directory from directory structure
int parent = fs->volume[base+4] + (fs->volume[base+3] << 8);
// update the current directory
current_dir = parent;
```

fs_gsys(PWD)

```
int print_list[3];
int size = 0;
int id = current_dir;
int base, name_address;

// get all the id through the path
while (id != 0){
    print_list[size++] = id;
    base = fs->DIR_BASE_ADDRESS + id * fs->DIR_SIZE;
    id = fs->volume[base+4] + (fs->volume[base+3] << 8); // parent id
}

// print the result
if (size == 0) printf('/'); // root directory
while (size > 0){
    id = print_list[--size];
    base = fs->DIR_BASE_ADDRESS + id * fs->DIR_SIZE;
    name_address = fs->volume[base+7];
    printf("/%s", &name_address);
}
```