香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

CSC3150 Operating System

Assignment 4

File System Implementation

Ziqi Gao (高梓骐)

School of Data Science

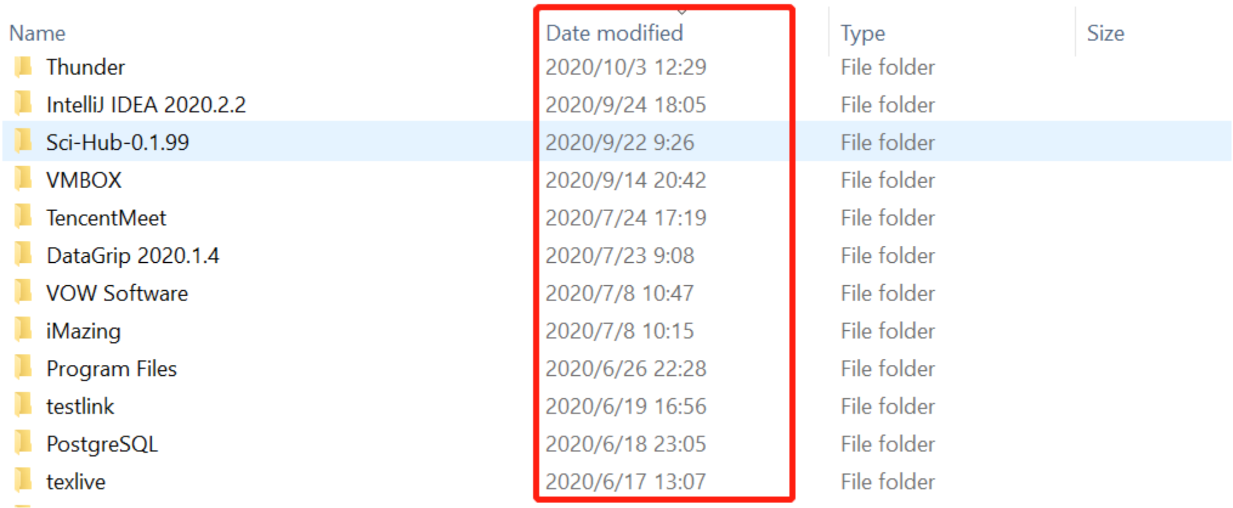
The Chinese University of Hong Kong, Shenzhen

118010077@link.cuhk.edu.cn

1. Introduction

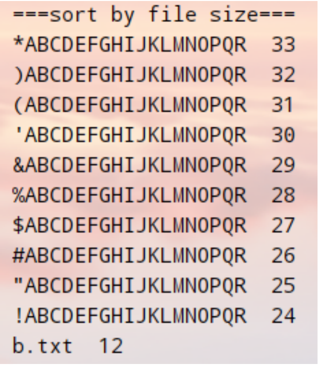
In this assignment, we will finish two tasks. The first one is we need give users the interface to manipulate their files. In this project, I will give 4 File System APIs.

a. fs_open: User should give the name about the file that he needs, and the access-mode (read/write). Then this method will return a pointer to the file that the user wants.
b. fs_write: This method will read data from the input buffer to the disk with the help of the file pointer returned from fs_open.
c. fs_read: This method will read data from the contents of the files and write those data into the output buffer.
d. fs_gsys(): This methods have three operation modes:
   The first one is RM: This method will delete the file given the file name from the users.
   The second one is LS_D: This mode will output a list of files' names ordered by their modified time. Just like what windows 10 do for us when we clicked the "*Date Modified Button*".



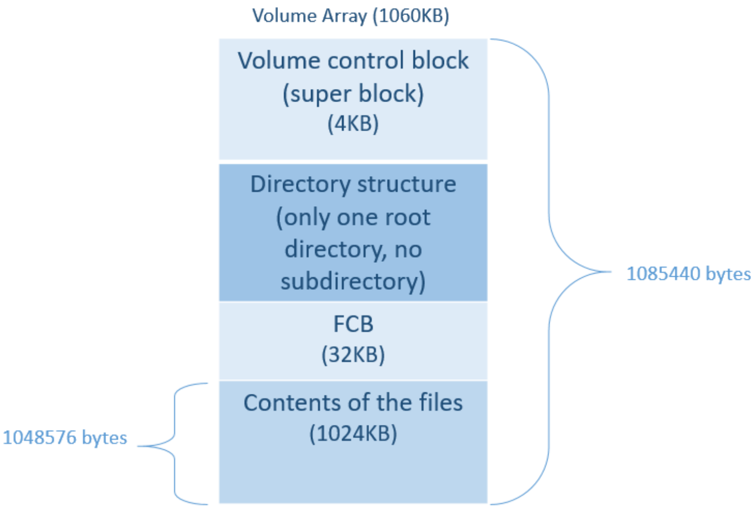
Name	Date modified	Type	Size
Thunder	2020/10/3 12:29	File folder	
IntelliJ IDEA 2020.2.2	2020/9/24 18:05	File folder	
Sci-Hub-0.1.99	2020/9/22 9:26	File folder	
VMBOX	2020/9/14 20:42	File folder	
TencentMeet	2020/7/24 17:19	File folder	
DataGrip 2020.1.4	2020/7/23 9:08	File folder	
VOW Software	2020/7/8 10:47	File folder	
iMazing	2020/7/8 10:15	File folder	
Program Files	2020/6/26 22:28	File folder	
testlink	2020/6/19 16:56	File folder	
PostgreSQL	2020/6/18 23:05	File folder	
texlive	2020/6/17 13:07	File folder	

The third one is LS_S: This mode will output a list of files' names with the file's size in the same row ordered by their data sizes. A demo output is like this:



```
===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
```

The second task is to create algorithms and data structures to save and deal with all the information that those APIs need. The core structure in this file system is this volume: We have a 4KB volume control block to save which blocks in the "contents of the files" are taken up and which blocks are free using the bit-map structure.

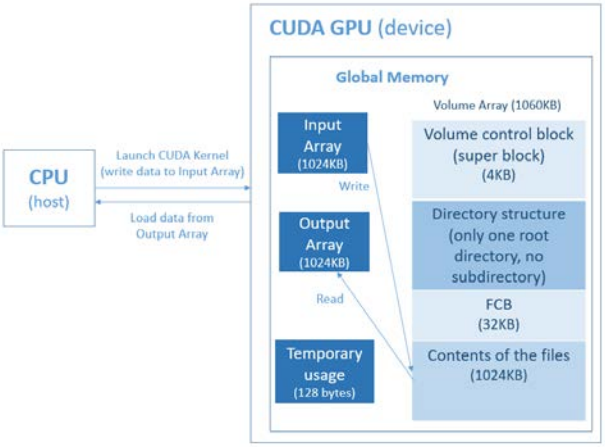


Then, we have a 32KB FCB table, each entry has a FCB of a certain file (32bytes). This table can hold 1024 entries, which means this file system can have at most 1024 files. FCB saves the metadata for each file. The detail about FCB will be introduced later.

Finally, we have a 1024KB district called "Contents of the files" to save all the data from the input buffer. Each FCB has a pointer to this area to find the data related to the files.

2. About GPU

In this experiment, I will use the global memory of one GPU core to simulate one volume of the disk. Since there is no OS to operate the GPU, we need to simulate a file system to simulate the work of OS file system.



Besides the volume array we introduced before. We also simulate the input buffer (1MB), output buffer (1MB), and a temp array for sorting (128 B).

3. Implementation Details

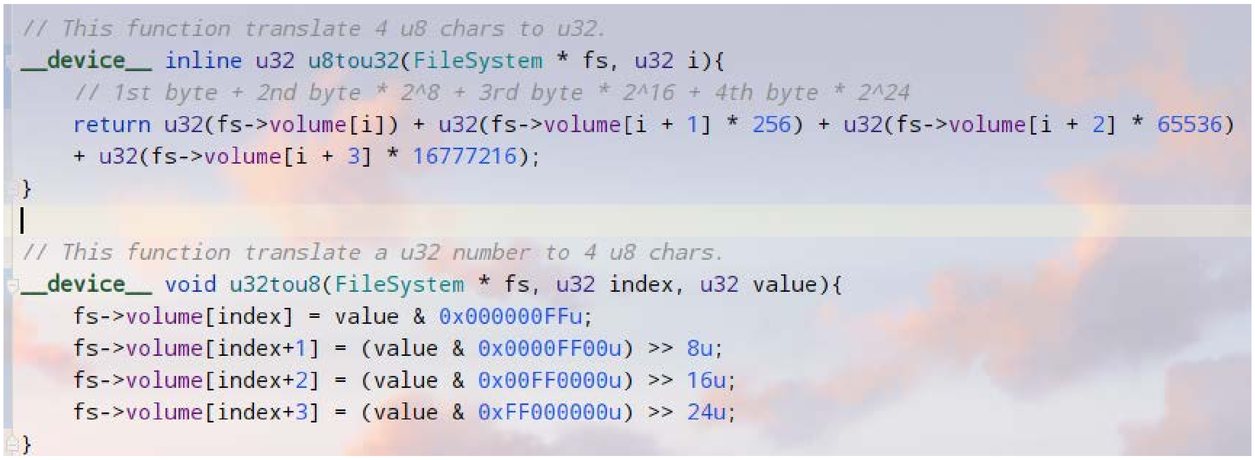
In this part, I will introduce the implementation of each API and the corresponding data structure.

FCB (32KB): I consider each 32 continuous elements as one entry of FCB. And this 32bytes can be divided into 4 parts like the following table shows:

INDEX	FILE_NAME(20 bytes)	START_ADDR(4 bytes)	FILE_SIZE (4 bytes)	MODIF_TIME(4bytes)
0	Example.txt\0	MIN: 36864	64	1
32	Example2.txt\0	36896		2
64	Example3.txt\0	MAX:1085439	MAX:1024	3

Note that FILE_SIZE is about the concise bytes size of this file, which is not calculated from the number of blocks it takes up.

We use 20 bytes to save the file's name, and the file name cannot exceed 20. Start address of the file, file size, and the modify time are all 4 bytes. However, volume is an array of unsigned character instead of the integer. We have to do the translation between four 1-byte data and a 4-byte data in order to let them express the same integer value. This can be done with the following two functions:



```
// This function translate 4 u8 chars to u32.
__device__ inline u32 u8tou32(FileSystem * fs, u32 i){
    // 1st byte + 2nd byte * 2^8 + 3rd byte * 2^16 + 4th byte * 2^24
    return u32(fs->volume[i]) + u32(fs->volume[i + 1] * 256) + u32(fs->volume[i + 2] * 65536)
    + u32(fs->volume[i + 3] * 16777216);
}

// This function translate a u32 number to 4 u8 chars.
__device__ void u32tou8(FileSystem * fs, u32 index, u32 value){
    fs->volume[index] = value & 0x000000FFu;
    fs->volume[index+1] = (value & 0x0000FF00u) >> 8u;
    fs->volume[index+2] = (value & 0x00FF0000u) >> 16u;
    fs->volume[index+3] = (value & 0xFF000000u) >> 24u;
}
```

u8tou32 method will read four 1-byte data and translate it to a 4-bytes integer. I conduct a hex decimal to decimal procedure here.

u32tou8 method read a 4-bytes integer and an index. Then it will change the value of FCB given an index. I conduct some bitwise operations to decide the values of the 1-byte data one by one.

Super Block (Bit Map, 4KB):

This data structure is also a compressed data structure because we do not have enough space to save the information of those $\frac{1024 \times 1024}{32} = 32768\ blocks$ for only 4KB array. If we use 4 bytes to show whether one block is taken up or free, the space will run up fast. As a result, since a 4KB array have 32768 bits, we can use each bit to show whether the block is used or free. For example, if the first element's value is 0x00000001, this means the first 7 blocks have already been used, and the last block (8th block) is free.
Lots of bitwise operations are used here to translate the integer information to the bitwise information.

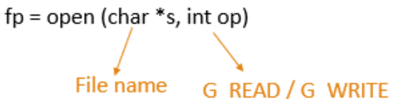
Temporary Usage (128 bytes):

Here we also take these 128 bytes space as an unsigned character array with 128 elements. This array is helpful in the sort (by size/modified time). This array saves the status of whether a file has already been sorted out. Since we may have 1024 files at most, it is not enough for us save all the data if we use one byte for one file. As a result, the technique here is very similar to what we use when we design the bit map. We also use one bit to show whether one file is sorted out.

Also, when we want to allocate new memory space to the disk, we have to look through the bit map to check where is the first unused blocks. We only to find the first unused blocks because, in my file system, there will be no external fragmentation.

fs_open:

This method receives two parameters. The first parameter is the name of the file that we want to use. And the second parameter is the access-mode. This access mode can be G_READ or G_WRITE.

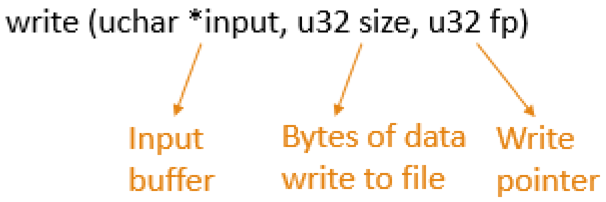


This method firstly looks through the FCB table to check whether this file has already existed in the disk. If we are in the G_READ mode and the file is not found. This method will return -1 and print out that NO FILE IS FOUND. If we are in the G_WRITE mode, then we look through the bit map to find free blocks for the new file.

If we find the file, this method will just return a pointer pointing to the disk. Then user can read or write data to the disk with this pointer.

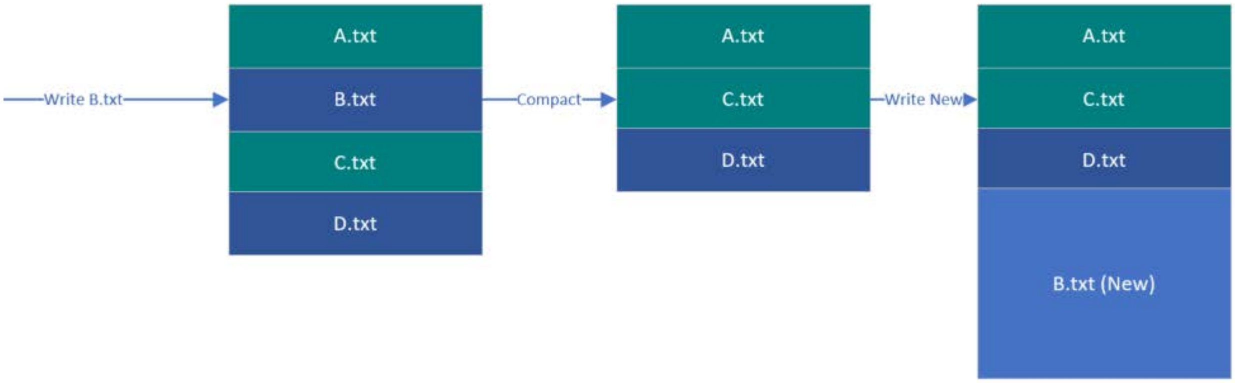
fs_write:

This method receives three arguments. The first argument is a pointer to the input buffer. The second argument is the bytes of data that will be written to the volume. The third argument is the pointer pointing to the volume. This method has two situations.


write (uchar *input, u32 size, u32 fp)
Input buffer
Bytes of data write to file
Write pointer

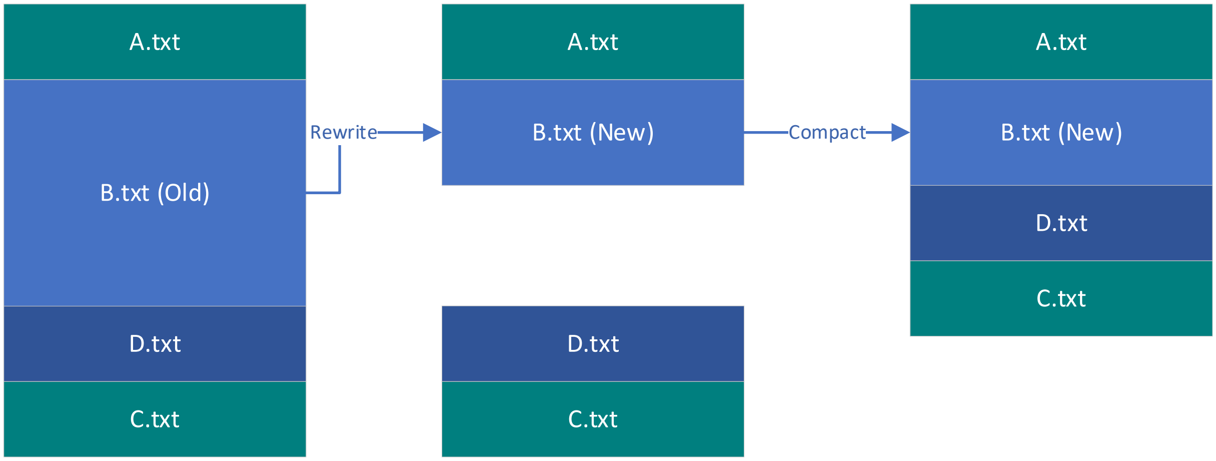
Before each write operation, we will update the modified time by adding 1 to gtime and save this gtime to file's FCB.

First Situation: In this situation, the file's size is larger than the allocated space (or when we want to create a new file). This will happen when we rewrite the data to the same file. We have to compact the old file's space before we write the new data just like this graph shows when we want to write *B.txt*:



Besides the memory space reallocation, we need to compact the FCB table, corresponding FCB entries, and the bit map. In order to update the bit map, we will free all the blocks in the bit map (although the space in the disk is not freed at all) in order to simplify the following compaction operations. You can check my code and the remarks for more detail in fs_write method.

Second Situation: In this situation, the allocated old space is larger than what we need to write. As a result, we will meet internal fragmentation. To deal with this problem, we still need to conduct the compaction. However, this case is simpler since we do not need to update the FCB table except the entries data, and we do not need to free the blocks in the bit map.



fs_read:

This function is straightforward, we just need to read data from the disk into the output buffer with the help of the file pointer returned from the pointer.

fs_gsys(RM):

This method will remove one file from the disk. Actually, this function can be considered as one part of the 1st situation of the fs_write where we also delete the old file and conduct the compaction. After those operations, we will free the relative FCB entries with null ('\0').

fs_gsys(LS_D/LS_S):

This method will print out a list of files saved in the disk ordered by the modified time or they may be ordered by the files' sizes. We use bubble sort here since we only have one temp array to record which files are already sorted out. As a result, quick sort and merge sort are not applicable in this situation.

One thing I need to mention is a special situation where some files may have the same sizes. In this situation, we need to print out those files one by one from the first created file to the last created file. Also, rewrite is not considered as a create operation.

How to decide the create time:
Since we have used up the FCB table, we need to use the structure properties to show the creation order. As a result, what we need to do is the "compaction" of FCB table whenever we want to remove some files so that the latest created file's FCB's index is the biggest.

INDEX	FILE_NAME(20 bytes)	START_ADDR(4 bytes)	FILE_SIZE (4 bytes)	MODIF_TIME(4bytes)
0	Example.txt\0	MIN: 36864	64	1
32	Example2.txt\0	36896		2
64	Example3.txt\0	MAX:1085439	MAX:1024	3

If we want to remove the Example.txt, we will first empty this FCB entry like this:

INDEX	FILE_NAME(20 bytes)	START_ADDR(4 bytes)	FILE_SIZE (4 bytes)	MODIF_TIME(4bytes)
32	Example2.txt\0	36896		2
64	Example3.txt\0	MAX:1085439	MAX:1024	3

If we do not "compact" the FCB table, Example4.txt will be written to the table with 0 index. This is what we do not want to see:

INDEX	FILE_NAME(20 bytes)	START_ADDR(4 bytes)	FILE_SIZE (4 bytes)	MODIF_TIME(4bytes)
0	Example4.txt\0	xxxxxx	xxxx	xxxx
32	Example2.txt\0	36896		2
64	Example3.txt\0	MAX:1085439	MAX:1024	3

As a result, we conduct the compaction here then add the new file's FCB entries to the last row of the FCB.

INDEX	FILE_NAME(20 bytes)	START_ADDR(4 bytes)	FILE_SIZE (4 bytes)	MODIF_TIME(4bytes)
0	Example2.txt\0	36896		2
32	Example3.txt\0	MAX:1085439	MAX:1024	3
64	Example4.txt\0	xxxxxxx	xxxx	xxxx

4. Execution

Versions of OS: Windows 10 Pro - 1909

Compilation Toolchains:

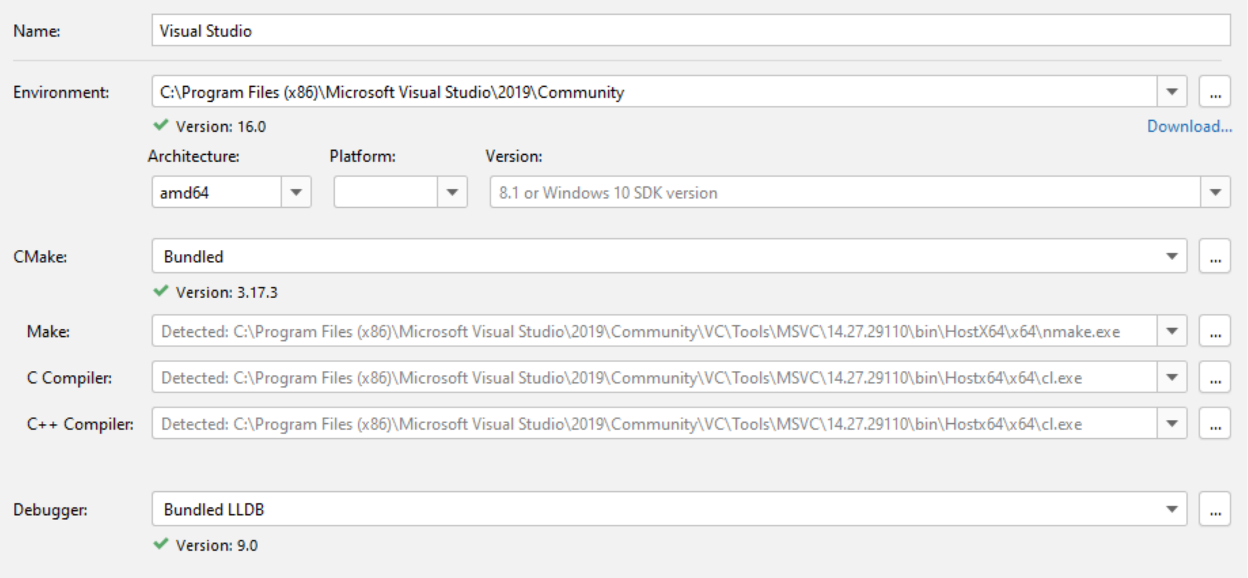
Make: MSVC-14.27.29110 amd64 architecture (Hostx64\x64\nmaek.exe)

C Compiler: MSVC-14.27.29110 amd64 architecture (Hostx64\x64\cl.exe)

C++ Compiler: MSVC-14.27.29110 amd64 architecture (Hostx64\x64\cl.exe)

CUDA Compiler: NVCC 11

IDE: CLion



Name: Visual Studio

Environment: C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

✓ Version: 16.0

Architecture: amd64

Platform:

Version: 8.1 or Windows 10 SDK version

CMake: Bundled

✓ Version: 3.17.3

Make: Detected: C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.27.29110\bin\HostX64\x64\nmake.exe

C Compiler: Detected: C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.27.29110\bin\Hostx64\x64\cl.exe

C++ Compiler: Detected: C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.27.29110\bin\Hostx64\x64\cl.exe

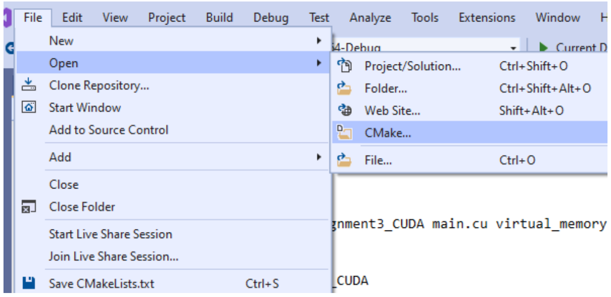
Debugger: Bundled LLDB

✓ Version: 9.0

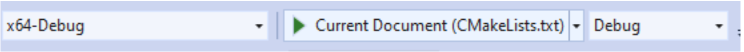
Also, my project is managed by CMake system. Save the data.bin file under the cmake-build-debug directory before test my program. Then, you can find my snapshot.bin under the cmake-build-debug directory after the execution of my program.

For Visual Studio Users:

File >> Open >> CMake >> Open my CMakeLists.txt



After you import the cmake file, please wait for a moment and click the Current Document (CMakeLists.txt) to run my code. (Please copy the data.bin to the debug file, otherwise the program cannot find this file.)

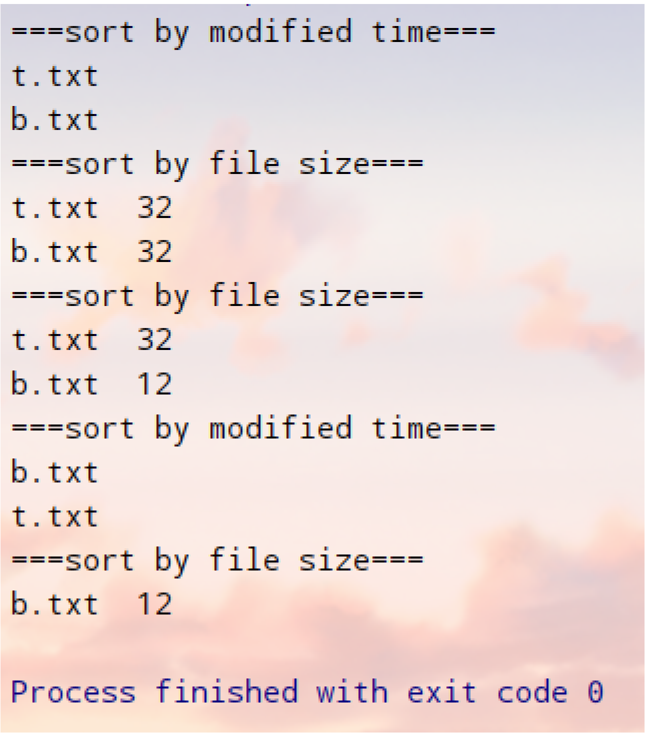


For CLion Users:

Click open a new project and choose my CMakeLIsts.txt to load my project.

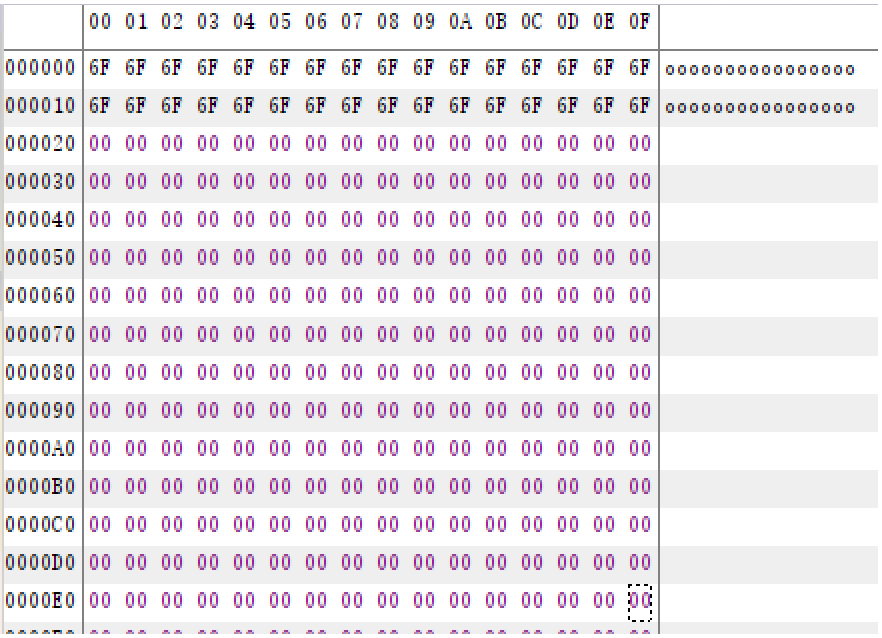
The output of my program is as follows:

Test Case 1:



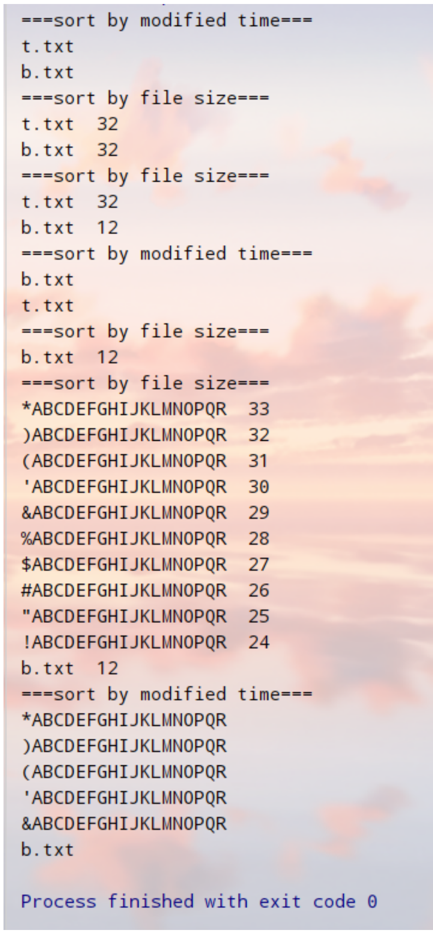
```
===sort by modified time===
t.txt
b.txt
===sort by file size===
t.txt  32
b.txt  32
===sort by file size===
t.txt  32
b.txt  12
===sort by modified time===
b.txt
t.txt
===sort by file size===
b.txt  12

Process finished with exit code 0
```

snapshot.bin:



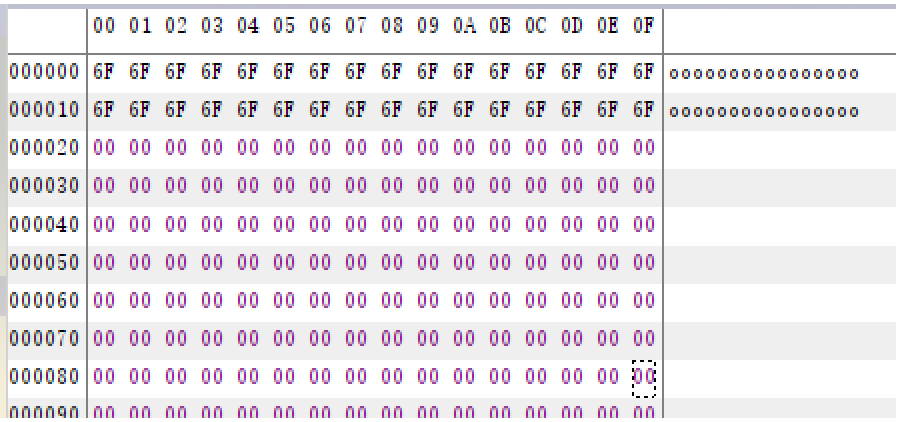
```
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000000 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F oooooooooooooooo
000010 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F oooooooooooooooo
000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Test Case 2:



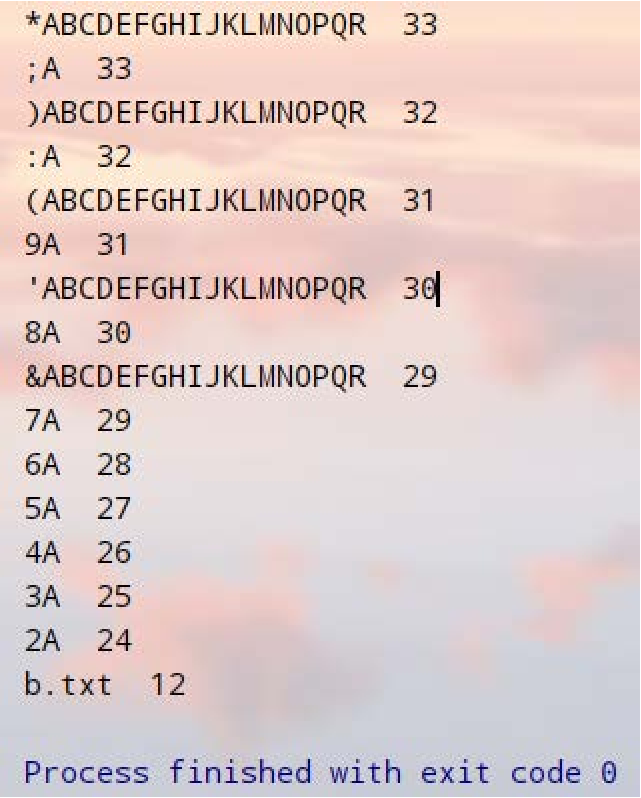
```
===sort by modified time===
t.txt
b.txt
===sort by file size===
t.txt  32
b.txt  32
===sort by file size===
t.txt  32
b.txt  12
===sort by modified time===
b.txt
t.txt
===sort by file size===
b.txt  12
===sort by file size===
*ABCDEFGHIJKLMNOPQR  33
)ABCDEFGHIJKLMNOPQR  32
(ABCDEFGHIJKLMNOPQR  31
'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

Process finished with exit code 0
```

snapshot.bin:



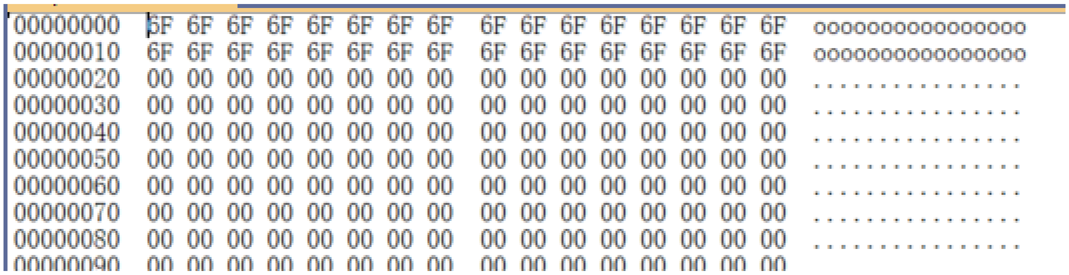
```
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000000 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F oooooooooooooooo
000010 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F 6F oooooooooooooooo
000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Test Case 3:

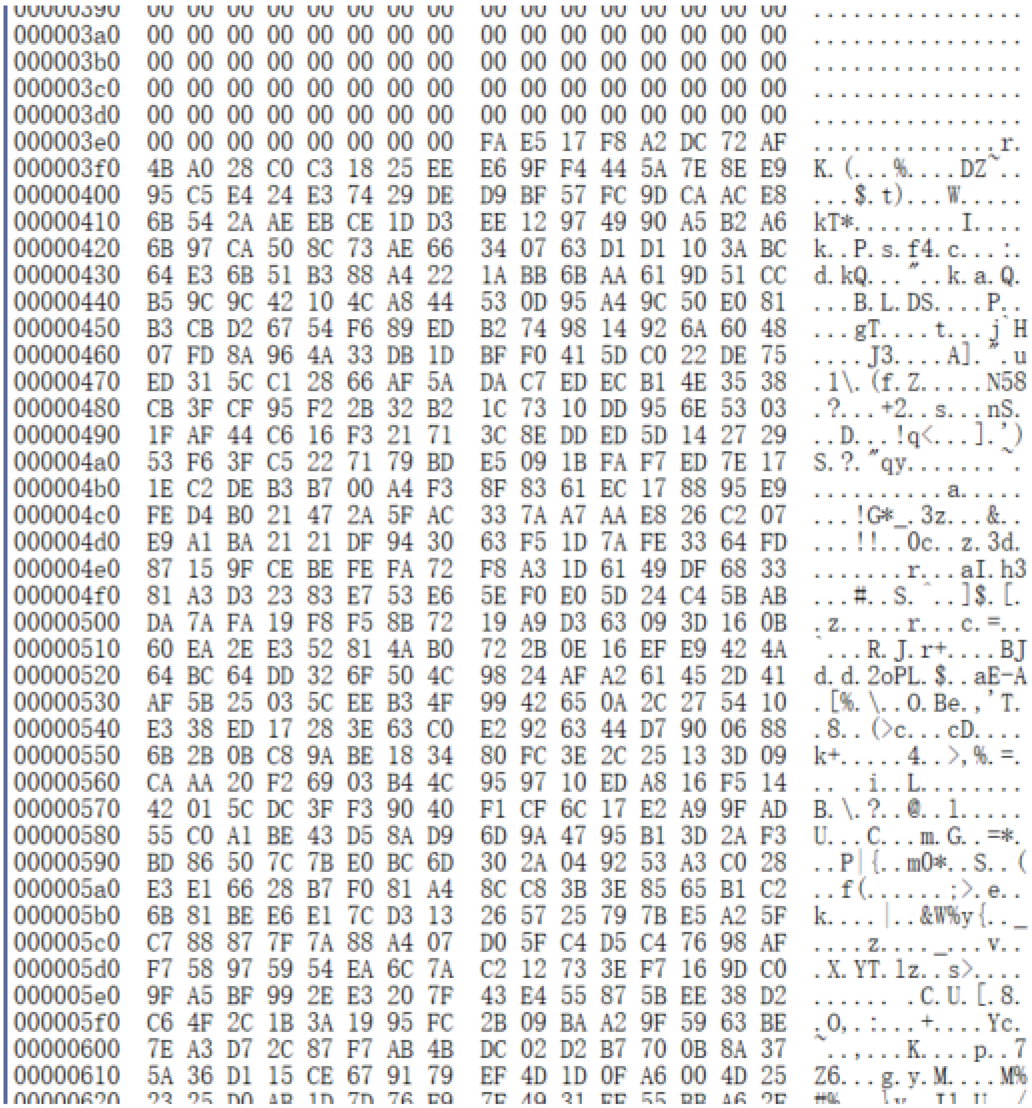


```
*ABCDEFGHIJKLMNOPQR  33
;A  33
)ABCDEFGHIJKLMNOPQR  32
:A  32
(ABCDEFGHIJKLMNOPQR  31
9A  31
'ABCDEFGHIJKLMNOPQR  30
8A  30
&ABCDEFGHIJKLMNOPQR  29
7A  29
6A  28
5A  27
4A  26
3A  25
2A  24
b.txt  12

Process finished with exit code 0
```

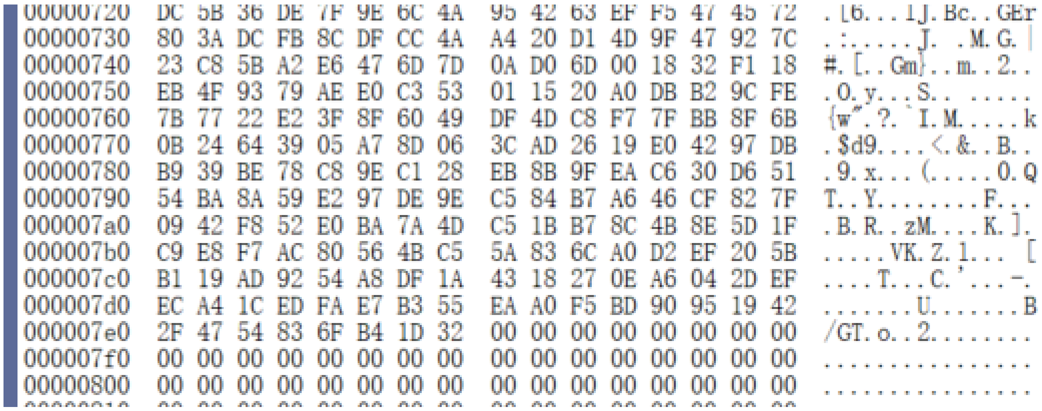
snapshot.bin:



```
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
```



```
00000390  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000003a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000003b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000003c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000003d0  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 E9  7F 49 31 EF 55 BB A6 2F
```



```
00000720  DC 5B 36 DE 7F 9E 6C 4A  95 42 63 EF F5 47 45 72  .[6...lJ.Bc..GEr
00000730  80 3A DC FB 8C DF CC 4A  A4 20 D1 4D 9F 47 92 7C  .:.....J. .M.G.|
00000740  23 C8 5B A2 E6 47 6D 7D  0A D0 6D 00 18 32 F1 18  #.[..Gm}..m..2..
00000750  EB 4F 93 79 AE E0 C3 53  01 15 20 A0 DB B2 9C FE  .O.y...S.. .....
00000760  7B 77 22 E2 3F 8F 60 49  DF 4D C8 F7 7F BB 8F 6B  {w".?.`I.M.....k
00000770  0B 24 64 39 05 A7 8D 06  3C AD 26 19 E0 42 97 DB  .$d9....<.&..B..
00000780  B9 39 BE 78 C8 9E C1 28  EB 8B 9F EA C6 30 D6 51  .9.x...(.....0.Q
00000790  54 BA 8A 59 E2 97 DE 9E  C5 84 B7 A6 46 CF 82 7F  T..Y........F...
000007a0  09 42 F8 52 E0 BA 7A 4D  C5 1B B7 8C 4B 8E 5D 1F  .B.R..zM....K.].
000007b0  C9 E8 F7 AC 80 56 4B C5  5A 83 6C A0 D2 EF 20 5B  .....VK.Z.l... [
000007c0  B1 19 AD 92 54 A8 DF 1A  43 18 27 0E A6 04 2D EF  ....T...C.'...-.
000007d0  EC A4 1C ED FA E7 B3 55  EA A0 F5 BD 90 95 19 42  .......U.......B
000007e0  2F 47 54 83 6F B4 1D 32  00 00 00 00 00 00 00 00  /GT.o..2........
000007f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000800  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000810  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

5. What I learned

The implementation of a simple file system makes me have a better understanding of how the control blocks like FCB, Super Block works in the file system. Also, this assignment is different from the projects I have done before. In this assignment, I am limited to very small space to hold data. We have to take the strategy time for space. For example, we use bit map instead of byte map. Or we use bubble sort instead of quick or merge sort. It is easy to figure out a time for space strategy, but the implementation is tedious actually. In this assignment, I conduct lots of bitwise operations to save the limited space.

Further Work: This time, I do not have enough time to do the bonus work that is very helpful because a file system is not powerful enough without directory structure. I will spend some time on this work in future.