CSC3150 Project4 Report

Zhixuan LIU

118010202

1 Introduction

This CSC3150 Assignment 1 submission, in compliance with the project instructions, implements the programs for the one required tasks and one bonus task. In the first task, we should implement a file system with a single directory in CUDA GPU memory. In the bonus task, we should implement a tree-structured directories.

2 Original Task

2.1 Overview of my Structure

In the first task, i use CUDA memory to implement my file system. My volume structure is as shown in the figure:

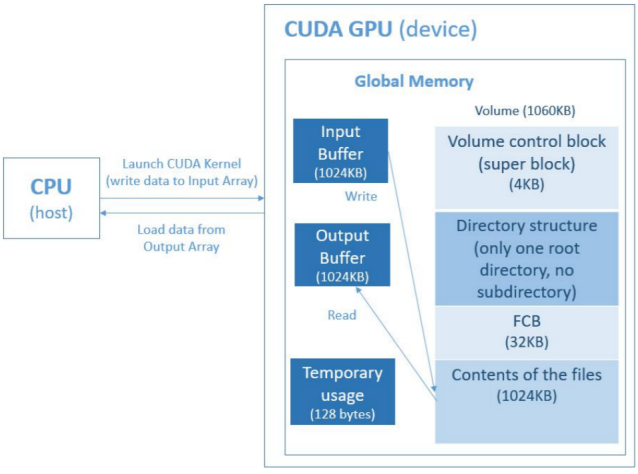

Figure 1: The volume structure of my file system

Here follows the detailed information of each part of my structure.

2.1.1 Volume control block

I use bit-map strategies to realize the volume control block. I have 4KB in total for volume control block, and i use one bit (0 or 1) to represent whether the block in the Contents is available or not (0 means the block is free and vice versa). By using this strategies, i can stalk 4k*8 = 32k blocks, which is exactly the same number of blocks in the file content part.

2.1.2 File Control Block

There are 32KB for FCB part. Each FCB entry is 32bytes, therefore, there are 1024 FCB entries in total. FCB contains the file information. My FCB structure is as follows:

bit 0−9: File name
bit 20+21: The starting position of block containing file contents
bit 22-23: The size of the file
bit 24: Mode of the file (read mode or write mode)
bit 25+26: Create time of the file
bit 27+28: Modified time of the file

0-19	20 21	22 23	24	25 26	27 28
Folder name	Location	Size	Mode	Create time	Last modified time

Figure 2: The volume structure of my file system

2.1.3 Contents of the files

There are 1024KB for the contents of the file part. The minimum unit is a block with 32bytes, therefore, there are 30K blocks in the Content of the files in total.

2.2 The design of each functions

In the file system, I have to implement several functions. The design details are as follows:

2.2.1 fs_open

The open function is used to open a file and return the address of its corresponding fcb entry. There are two mode for this function: G_READ mode and G_WRITE mode. If it is write mode and the file name we want to open is not in the file system, then we need to automatically create such a file for the file system. In the following paragraphs, I will explain how do I design those two functions in detail.

G_WRITE :

First, i go through all the entries in FCB to check whether the file name user wants to open has already exit in the file system.

If the answer is yes, which means i find this file in my file system, only two step will be further taken: 1) change the *mode* information in FCB to write mode; 2) return the FCB entry address i find then finish.

If the answer is no, which means this file is not in the file system. Several steps will be taken: 1) check whether there is available block in the file system by using bitmaps; 2) If there are still spaces in the file system, then find the first empty block using bitmaps strategies; 3) store the information of the file into a new FCB entry. Return the address of the FCB entry to user.

G_READ :

Similar to write mode, in read mode first we go through all the entries in FCB to find whether the file is in the file system. If so, change the mode to read mode and return the address of the FCB entry that is founded.

2.2.2 fs_write

This function is used to write a file. A file pointer points to the address of the FCB entries which contents the information of the target file. Firstly, I check whether this file has been modified by checking whether the size information in the FCB entry is valid nor not.

If the file has not been modified, then I check whether the remaining blocks is enough for the size we want to write. After that, I rewrite the bitmaps, write the content in the file system and change the FCB entry's information.

If the file has already been modified, which means there are old contents in the file system and i need to remove it. First, i checked whether the block number of the original contents equals to that of the new contents. If so, simply replace the old contents with the new one and modify the information in FCB entries.

However, if the number of blocks does not equal to each other, then we need to implement the compact strategy: First remove all the contents of the original file. Next, move all the blocks in the file contents part upwards to avoid fragment. I also change the corresponding bitmaps. Finally, find a new place for the new contents similar to the previous write method for file which has not been modified.

2.2.3 fs_read

This function is used to read a file. A file pointer points to the address of the FCB entries which contents the information of the target file. Then I go to the right location and read the information to the output buffer.

2.3 RM

To remove the file, First go through all the FCB entries to find the FCB that contents the information of the target file. Then clean all the information in the

file contents part in the file system. Finally, use the conpact method described in the fs_write paragraph.

2.4 LS_S and LS_D

In this function, first i use bubble sort (which is a in-placed sorting algorithm to sort) the FCB entries. For LS_S, I sort FCB entries according to their file size. If their sizes are the same, i sort them by their modified time. For LS_D, I bubble sort them by modified time. Then I display the name of file from the top of FCB to the bottom.

2.5 Results

There are tree test cases. The screen shots are shown at the end of the report.

3 Bonus

In the bonus part, I change the structure of my FCB, while other parts do not have some important changes.

Because there are maximally 1024 files and folders, I use FCB to store the information of each files and folders as shown as follows:

For files:
bit 0−9: File name
bit 20+21: The starting position of block containing file contents
bit 22-23: The size of the file
bit 24: Mode of the file (read mode or write mode)
bit 25+26: Create time of the file
bit 27+28: Modified time of the file
bit 29+30: Folder number where the file is under
bit 31: 0, which is used to distinguish from folder
For folders:
bit 0−9: Folder name
bit 20+21: Folder number, each folder has a unique index
bit 22-23: The size of the folder
bit 24: NA
bit 25+26: Create time of the folder
bit 27+28: Modified time of the folder
bit 29+30: Folder number where the folder is under
bit 31: 255, which is used to distinguish from folder

0-19	20 21	22 23	24	25 26	27 28	29 30	31
File name	Location	Size	Mode	Create time	Modified Time	Folder number	255

Figure 3: FCB entry structure for files

0-19	20 21	22 23	24	25 26	27 28	29 30	31
Folder name	number	Size		Create time	Modified Time	Supper Folder number	0

Figure 4: FCB entry structure for folders

In my bonus structure, I have a global variable called $current\_folder$ to stalks the current file number. If the current folder is root, then the $current\_folder = 0$. In the following section, i introduce some new functions in bonus task.

3.1 MKDIR

This function is to create a directory under current folder. First I will store the information of the folder in a new FCB entries, with supper folder number equals $current\_folder$. I also store the create time of the folder in the FCB entry. If the folder is not under root directory, I increase the current folder size corresponding to the newly created folder name.

3.2 CD

This function is to enter into a folder. In my system, i just simply go through the FCB entries and get the folder number of the target folder, and change my global variable $current\_folder$ to the target folder number.

3.3 CD_P

This function is to go back to the parent folder. I go through the FCB system to find the target FCB entry by the current folder number. Then I can get the super folder of the current folder. Just simply change the $current\_folder$ to the super folder number, we can get to the parent folder.

3.4 RM_RF

First, I go through the FCB entries and find the folder I want to remove. Then I first marked the target folder number, then remove all the information about that folder in the FCB. If the folder user wants to remove is not in the root directory, I also change the size of the supper folder. Then I go through FCB to find all the files under target folder using target folder number. Then use RM function to remove the files. I also changes RM function as well as open function in the original task a little bit to change the size of folders.

3.5 PWD

First, I have the current folder number referring to $current\_folder$. Then I find the super folder step by step. Then I displayed them from top to bottom.

3.6 LS_S and LS_D

I still use bubble sort to sort all the FCB entries. Therefore, the file FCB entries and folder FCN entries can all be sorted. Then I display them.

3.7 Results

For the test cases, the screen shot is at the end of the report.

4 Environment of Running my Program

I use the computer in TC301, with the following environment:
Win 10
VS2017
CUDA9.2
NVIDIA Geforce GTX 1060
Compute capacity: 6.1

5 Execution Step to Run my Program

In VS2017, press "Ctrl + F7" to compile my programs, and press "Ctrl + F5" to run my program.

6 What did I Learned

In this program, I learnt how file system works. How to implement bitmaps, File Control Blocks and how to implement compact methods. I also learnt how to implement a tree-structured for multiple directories.

7 Screen Shots

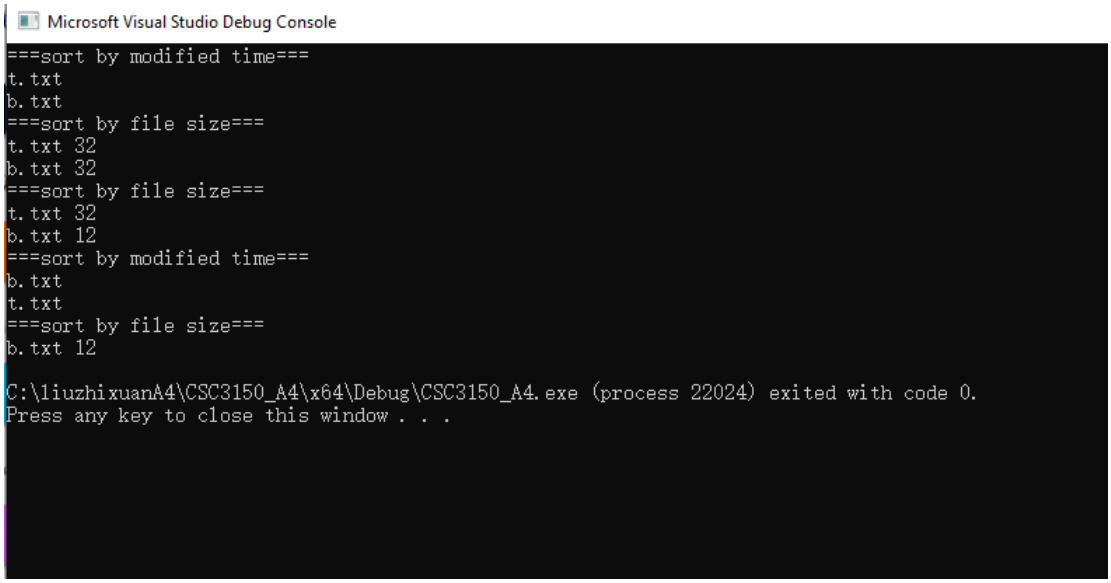

Figure 5: Terminal output for case 1

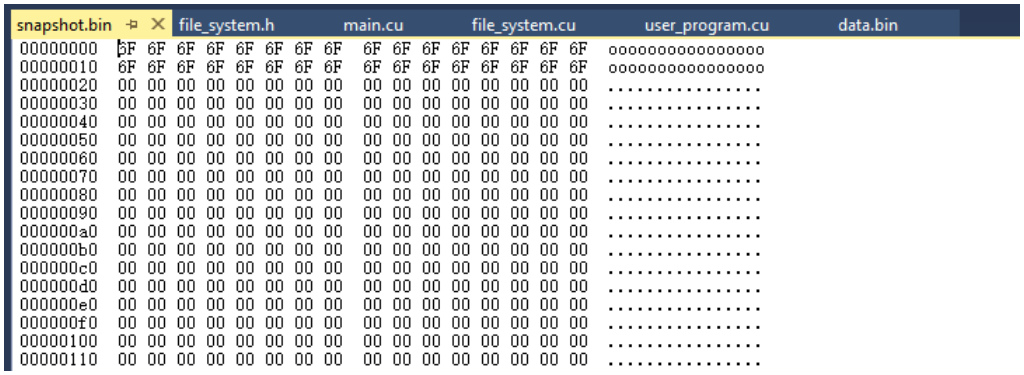

Figure 6: Snapshot.bin for case 1

Figure 7: Terminal output for case 2

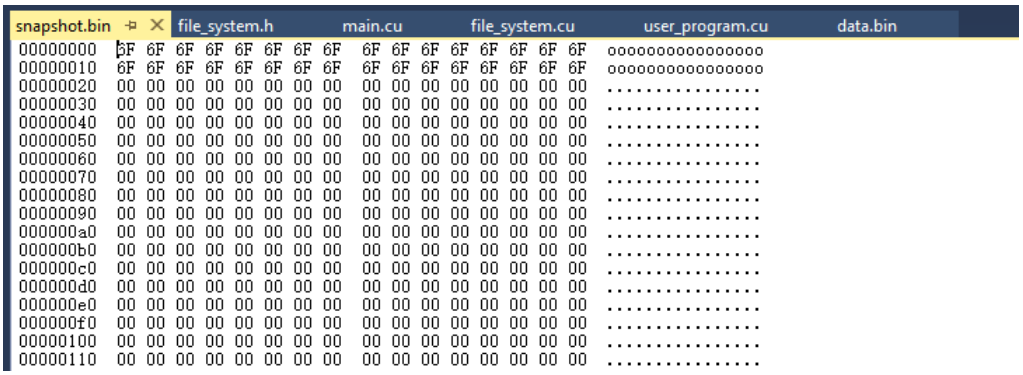

Figure 8: Snapshot.bin for case 2

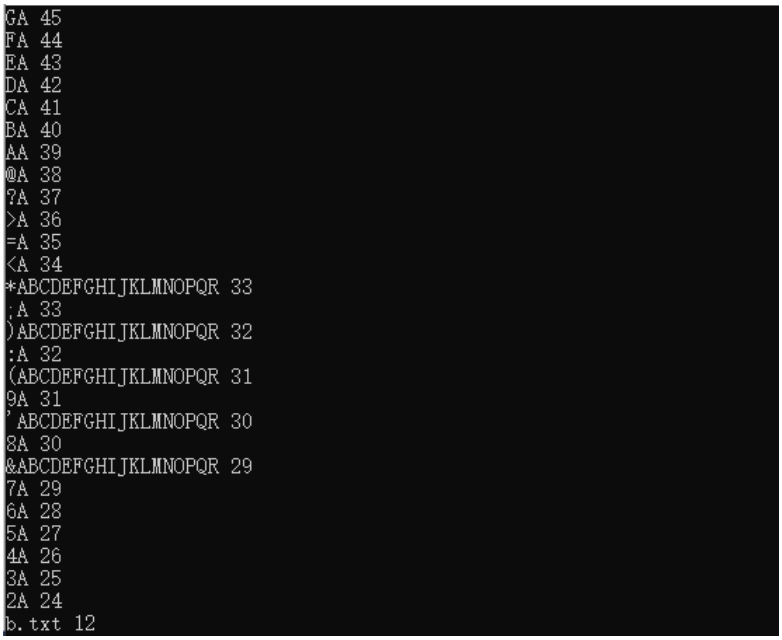

Figure 9: Terminal output for case 2

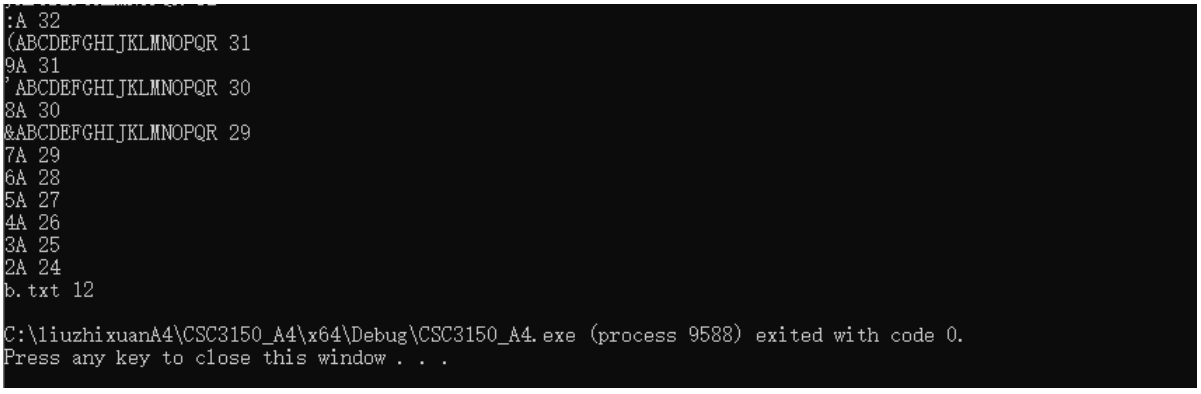

Figure 10: Terminal output for case 2

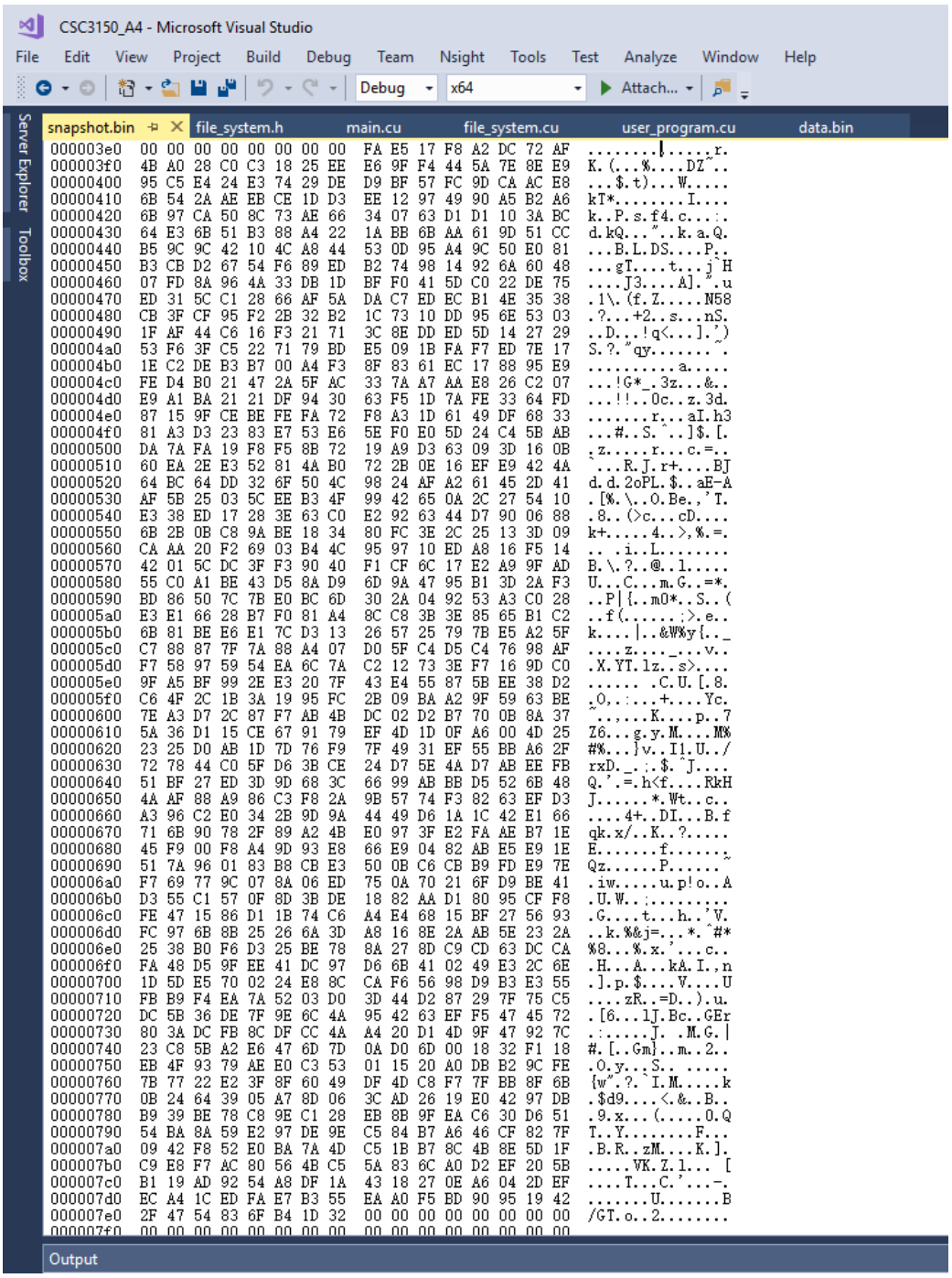

Figure 11: Snapshot.bin for case 2

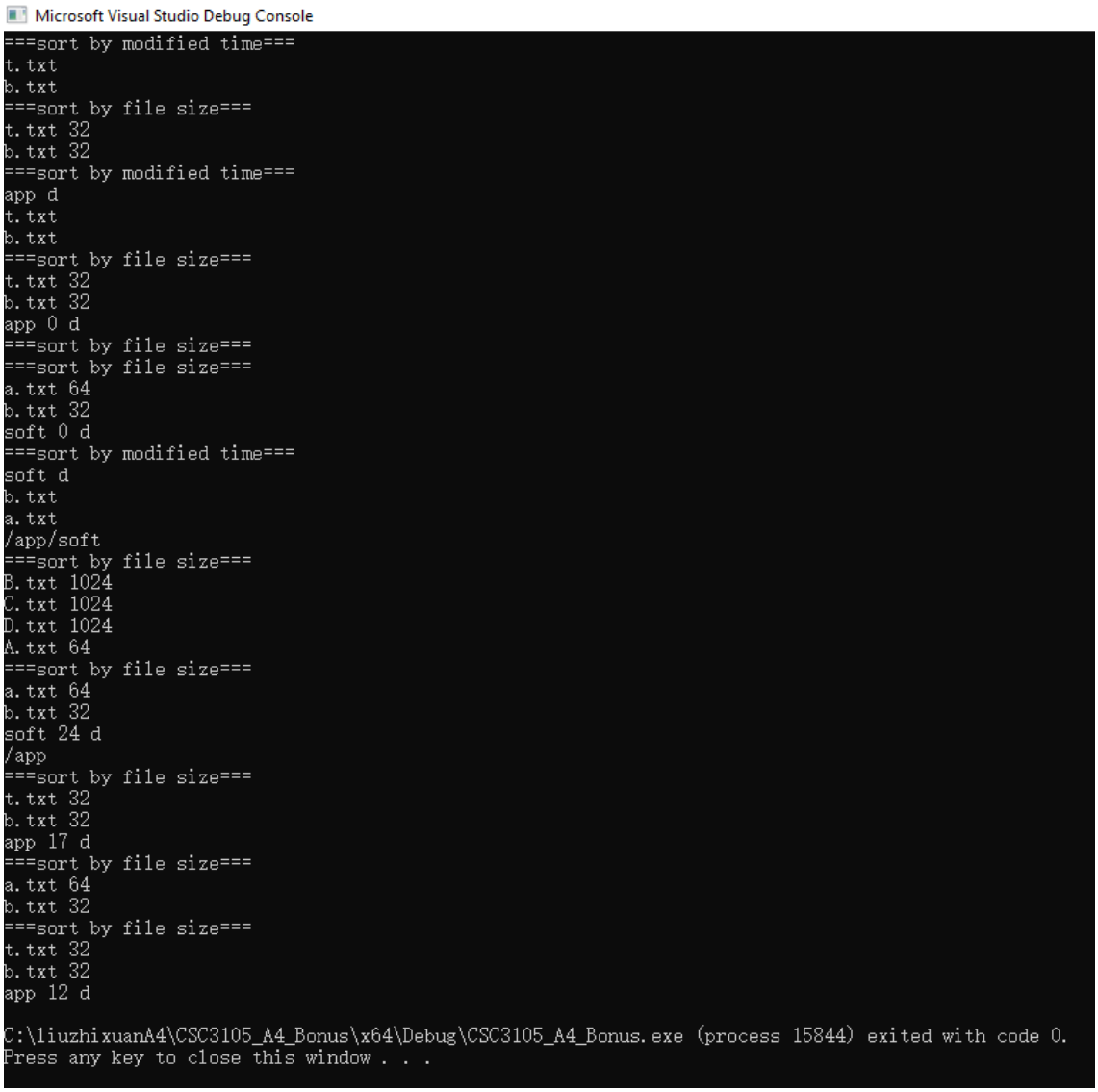

Figure 12: Terminal output message for Bonus task