CSC3150 Assignment 4

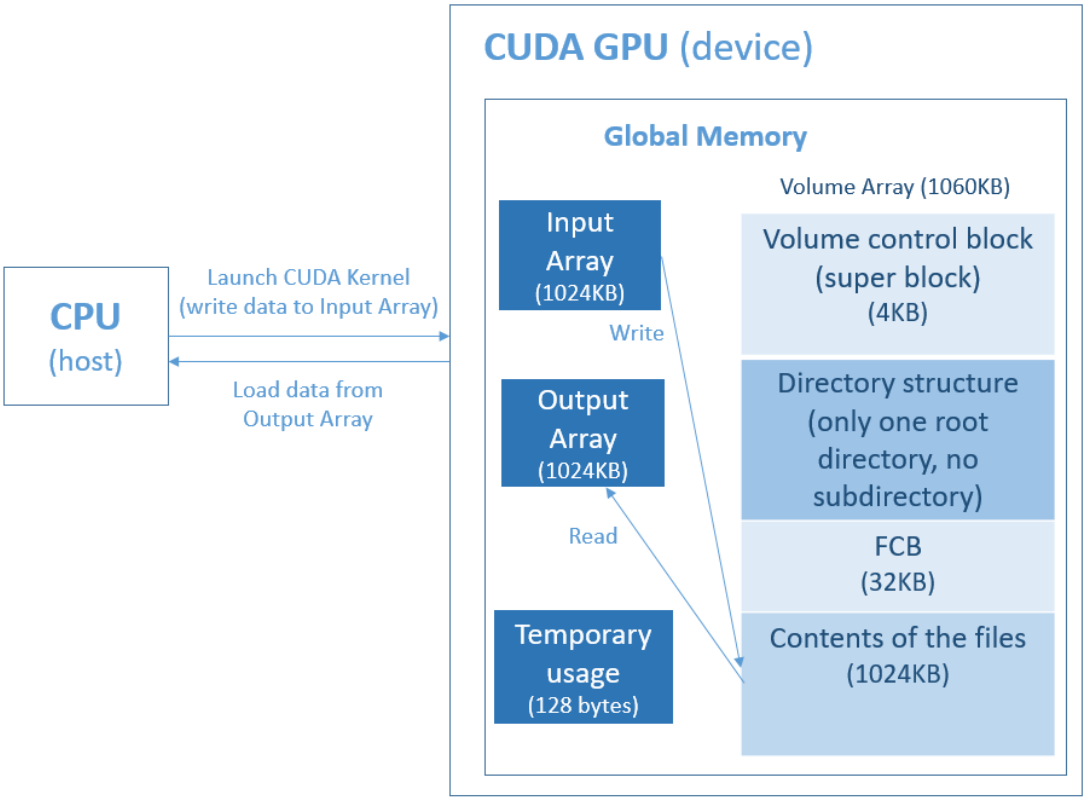
In Assignment 4, you are required to simulate a mechanism of file system via GPU's memory.

Background:

- **File systems** provide efficient and convenient access to the disk by allowing data to be stored, located, and retrieved easily.
- A file system poses two quite different design problems. The first problem is defining how the file system should look to the user. This task involves defining a file with its attributes, the operations allowed on a file, and the directory structure for organizing files.
- The second problem is creating algorithms and data structures to map the logical file system on to the physical secondary-storage devices.
- The file-organization module knows about files and their logical blocks, as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file-organization module can translate logical block address to physical block address for the basic file system to transfer.
- Each file's logical blocks are numbered from 0 (or 1) through N. Since the physical blocks containing the data usually do not match the logical numbers, a translation is required to locate each block.
- The **file-organization** module also includes the **free-space manager**, which tracks unallocated blocks and provides these blocks to the file-organization module when requested.
- The **logical file system** manages the directory structure to provide the file-organization module with the information the latter needs, given a symbolic file name. It maintains file structure via file-control blocks.
- A **file-control block** (**FCB**) (an inode in UNIX file systems) **contains information about the file**, including ownership, permissions, and location of the file contents.
- Then we can try to implement a simple file system in CUDA GPU with single thread, and limit global memory as volume.

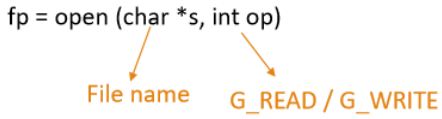
The GPU File System we need to design:

- We take the **global memory as a volume** (logical drive) from a hard disk.
- No directory structure stored in volume, **only one root directory**, no subdirectory in this file system.
- **A set of file operations should be implemented.**
- In this project, **we use only one of GPU memory, the global memory as a volume. We don't create the shared memory as physical memory for any data structures stored in, like system-wide open file table in memory**.
- In this simple file system, we just directly take the information from a volume (in global memory) by single thread.



Specification:

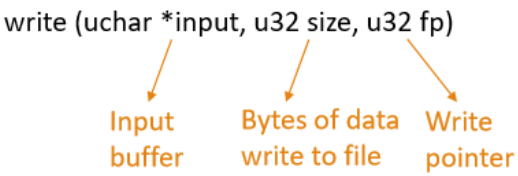
- The **size of volume** is **1085440** bytes (1060KB).
- The **size of files** total is **1048576** bytes (1024KB).
- The maximum number of file is 1024.
- The **maximum size of a file** is **1024 bytes** (1KB).
- The **maximum size of a file name** is **20 bytes**.
- File name end with "\0".
- **FCB size** is **32 bytes**.
- FCB entries is 32KB/ 32 bytes = 1024.
- **Storage block size** is **32 bytes**.
- **open:**
  - Open a file
  - Give a file pointer to find the file's location.
  - Space in the file system must be found for the file.
  - An entry for the new file must be made in the directory.
  - Also accept access-mode information: read/write
  - When to use write mode, if no such file name can be found, create a new zero byte file.
  - Return a write/read pointer.
  - Function definition:



  - Demo usage:

```
fp = open("b.txt\0", G_WRITE);

fp = open("t.txt\0", G_READ);
```

- **write:**
  - To write a file.
  - There is a write pointer to identify the location in the file.
  - If the file has existed, cleanup the older contents of the file and write the new contents.
  - Take the **input** buffer to write bytes data to the file.
  - Function definition:



- Demo usage:

```
// start from input[0], write 64 bytes into t.txt
fp = open("t.txt\0", G_WRITE);
write(input, 64, fp);

// start from input[32], write 32 bytes into b.txt
fp = open("b.txt\0", G_WRITE);
write(input + 32, 32, fp);
```

- **read:**
  - To read contents from a file.
  - There is a read pointer to identify the location in the file.
  - To read bytes data from the file to the **output** buffer.
  - The offset of the opened file associated with the read pointer is 0 (always read the file from head).
  - Function definition:

```
read(uchar *output, u32 size, u32 fp)
```

Output buffer — Bytes of data read from file — Read pointer

  - Demo usage:

```
// start from begining of t.txt, read 32 bytes and write into output
fp = open("t.txt\0", G_READ);
read(output, 32, fp);
```

- **rm:**
  - To delete a file and release the file space.
  - Search the directory for the named file.
  - Implement **gsys()** to pass the **RM** command.
  - Function definition.

```
gsys(int op, char *s)
```

Delete command: RM — File name you want to delete

  - Demo usage

```
// remove the file t.txt
gsys(RM, "t.txt\0");
```

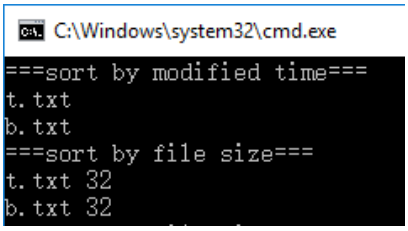
- **ls:**
  - List information about files.
  - Implement **gsys()** to pass the **LS_D/LS_S** commands.
  - **LS_D** list all files name in the directory and order by modified time of files.
  - **LS_S** list all files name and size in the directory and order by size.
  - If there are several files with the same size, then first create first print.
  - Function definition

  - Demo usage

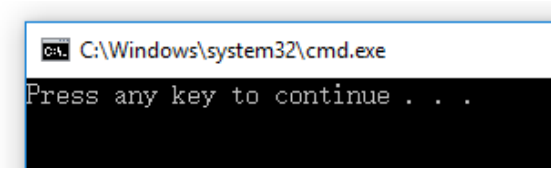
```
//list all files sort by modified time
gsys(LS_S);

//list all files sort by file size
gsys(LS_D);
```

  - Demo output



Template structure:

- Create your cuda project in VS, and add “data.bin” to your project. Pres “Ctrl”+”F5” to run the template. If it successes, will have output like:



- The storage size of the simulated file system is already definded as:

```
9   #define DATAFILE "./data.bin"
10  #define OUTFILE "./snapshot.bin"
11
12  #define SUPERBLOCK_SIZE 4096 //4KB
13  #define FCB_SIZE 32 //32 bytes per FCB
14  #define FCB_ENTRIES 1024
15  #define STORAGE_SIZE 1085440 //1060KB
16  #define STORAGE_BLOCK_SIZE 32
17
18  #define MAX_FILENAME_SIZE 20 //20 bytes
19  #define MAX_FILE_NUM 1024
20  #define MAX_FILE_SIZE 1048576 //1024KB
21
22  typedef unsigned char uchar;
23  typedef uint32_t u32;
24
25  __device__ uchar volume_d[STORAGE_SIZE];
```

- At first, load the binary file, named **“data.bin”** to **input** buffer (via “load_binarary_file()”) before kernel launch.

```
122     // load binary file from data.bin
123     load_binaryFile(DATAFILE, input_h, MAX_FILE_SIZE);
```

- Launch to GPU kernel with single thread.

```
129     mykernel << <1, 1 >> >(input, output);
```

- In kernel function, simulate file operations for testing. **We will replace this part with different test cases.**

```
55  __global__ void mykernel(uchar *input, uchar *output)
56  {
57      //kernel test start
58
59      /* Complete your test case for file opereation here! */
60
61      // kernel test end
62  }
```

- You should complete the file operations for open/write/read/rm/ls_d/ls_s.

```
28  __device__ u32 open(char *s, int op)
29  {
30      /* Implement open operation here */
31  }
32
33
34  __device__ void read(uchar *output, u32 size, u32 fp)
35  {
36      /* Implement read operation here */
37  }
38
39  __device__ u32 write(uchar* input, u32 size, u32 fp)
40  {
41      /* Implement write operation here */
42  }
43
44  __device__ void gsys(int op)
45  {
46      /* Implement LS_D and LS_S operation here */
47  }
48
49  __device__ void gsys(int op, char *s)
50  {
51      /* Implement rm operation here */
52  }
53
```

- In CPU(host) main function, the output buffer is copied in device, and it is written into “snapshot.bin” (via write_binarary_file()).

```
133     // dump output array to snapshot.bin
134     write_binaryFile(OUTFILE, output_h, MAX_FILE_SIZE);
```

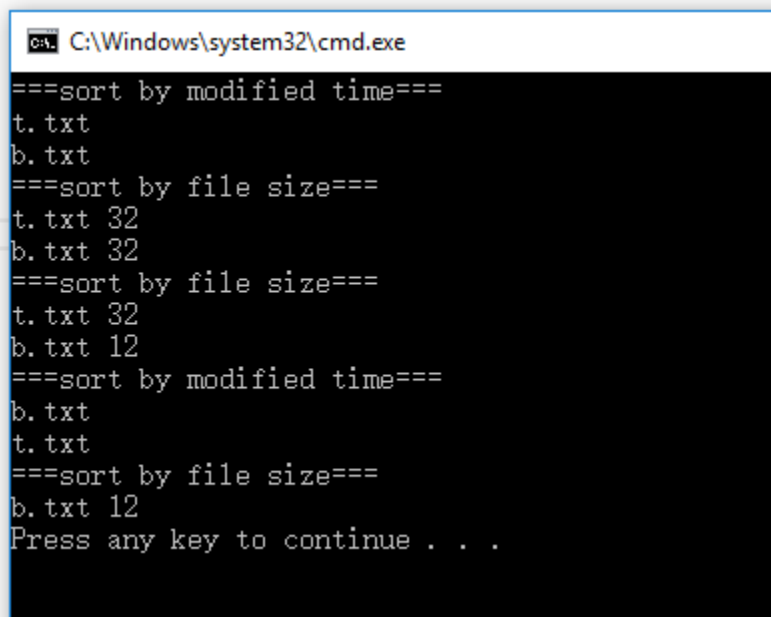
Function Requirements (90 points):

- Implement file volume structure. (10 points)
- Implement free space management. (For example, Bit-Vector / Bit-Map). (10 points)
- Implement contiguous allocation. (10 points)
- Implement open operation (10 points)
- Implement write operation (10 points)
- Implement read operation (10 points)
- Implement rm operation (10 points)
- Implement LS_D operation (10 points)
- Implement LS_S operation (10 points)

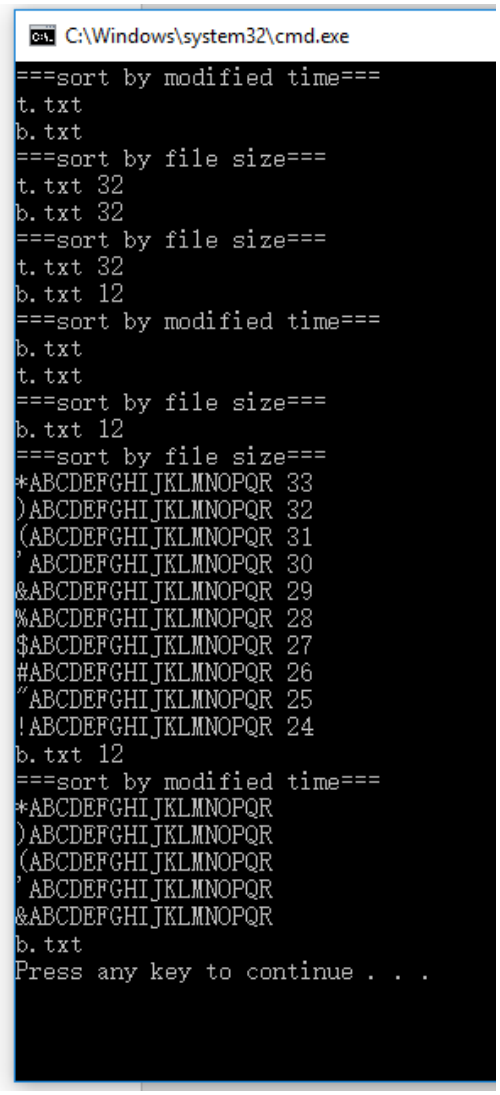
Demo Output:

In the “CSC3150_Assignment_4/Test Case”, we provided three test cases, you could copy them from txt file and replace them in kernel test part in template code.

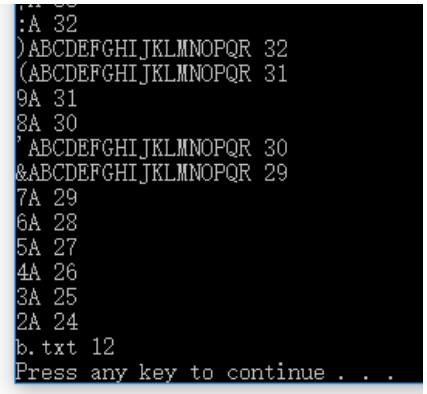
- Test Case 1



- Test Case 2



- Test Case 3



Bonus (10 points)

Referring to Chapter 11 Section 11.4 and 11.5, redesign the file system with linked allocation.

- The **maximum size of a file** is **1024 bytes** (1KB).
- The **maximum size of a file name** is **20 bytes**.
- File name end with “\0”.
- You could extend the volume size with **extra 32KB**.
- You could redesign the volume storage structure.
- Compare the performance of these two file systems.

Report (10 points)

Write a report for your assignment, which should include main information as below:

- How did you design your program?
- What problems you met in this assignment and what is your solution?
- The steps to execute your program.
- Screenshot of your program output.
- What did you learn from this assignment?

Submission

- Please submit the file as package with directory structure as below:
  - **CSC3150_Assignment_4_(Student ID)**
    - Source
      - main.cu
      - data.bin
      - snapshot.bin
      - bonus.cu (if you implement bonus part)
    - Report
- Due date: End (23:59) of 29 Nov, 2018

Grading rules

Completion	Marks
Report	10 points
Bonus	10 points
Completed with good quality	80 ~ 90
Completed accurately	80 +
Fully Submitted (compile successfully)	60 +
Partial submitted	0 ~ 60
No submission	0
Late submission	**Not allowed**