Assignment4 Report

Name: Zhang Yixuan

Student No: 120010058

Environment of running

OS: Linux

VS version: 1.73.0

Cuda version: CUDA_11.7.1

GPU information: GeForce MX350

Execution steps of running your program

In terminal:

First use "**nvcc --relocatable-device-code=true main.cu user_program.cu file_system.cu -o test**" to compile the cu file.

Then use "**srun ./test**" to run the test file.

How did I design my program?

The code is constructed by five main function:

__device__ u32 fs_open(FileSystem *fs, char *s, int op)

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

__device__ u32 fs_write(FileSystem *fs, uchar* input, u32 size, u32 fp)

__device__ void fs_gsys(FileSystem *fs, int op)

__device__ void fs_gsys(FileSystem *fs, int op, char *s)

Each of them implements the open, read, write, sort, and delete function of file system.

Except for the five main functions, there is a function helping find the filename: **compare_file_name**. The pointers to the head of two filename are the input. A for loop is implemented to check whether two chars are the same and move to the next

char. Once a pair of chars are different, the function will return false.

In the open function, first check whether the file exists or not. If it exists, give file _pointer the value of the index of **FCB** in the volume (start from 0). Notice that all **FCB** is behind a Superblock. If the file_pointer is not -1, that means the file already exists and the modified time will be updated and the open function returns **file_pointer**.

```
u32 file_pointer = -1;
for (int i = 0; i < fs->FCB_ENTRIES; i++) {
  u32 addr_entry = i * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE;
  if (fs->volume[addr_entry] != '\0') {
    bool check_similarity = compare_file_name(s, (char*) &fs->volume[addr_entry]);
    if (check_similarity) {
      file_pointer = i;
      break;
    }
  }
}

if (file_pointer != -1) {
  //update modified time
  modified_time++;
  fs->volume[file_pointer * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE + 22] = modified_time / 256;
  fs->volume[file_pointer * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE + 23] = modified_time % 256;
  return file_pointer;
```

If the value is -1 and the operation is **G_WRITE**, the new file should be write in the volume. First find a empty **FCB** which is start from '\0', meaning no filename. Again the **file_pointer** is set as the index of **FCB**. Then calculate the exact starting position of the **FCB** by recording

file_pointer * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE

as **file_entry**. Starting from **fs->volume[file_entry]**, copy the chars of filename until it reaches '\0'. After copying, add a '\0' behind the filename.

```
int length = 0;
int file_entry = file_pointer * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE;
while (s[length] != '\0') {
  fs->volume[file_entry + length] = s[length];
  length++;
  if (length == fs->MAX_FILENAME_SIZE) {
    printf("filename length exceeds max size.");
    break;
  }
}
fs->volume[file_entry + length] = '\0';
```

After the filename, initialize the information of the file, including **create_time**, **modified_time**, **size**, and **address**. **Create_time** and **modified_time** are global variable starting from 0. **fs->FILE_ADDING_ADDRESS** is the record of position of head of file starting from base.

```
/*
index:
0-19 filename
20-21 create_time
22-23 modified_time
24-27 size
28-29 address
*/

//set create_time
fs->volume[file_entry + 20] = create_time / 256;
fs->volume[file_entry + 21] = create_time % 256;
create_time++;

//set modified_time
fs->volume[file_entry + 22] = modified_time / 256;
fs->volume[file_entry + 23] = modified_time % 256;
modified_time++;

//set size
u32 size = 0;
fs->volume[file_entry + 24] = size % 256;
fs->volume[file_entry + 25] = (size>>8) % 256;
fs->volume[file_entry + 26] = (size>>16) % 256;
fs->volume[file_entry + 27] = (size>>24) % 256;
//set address
fs->volume[file_entry + 28] = fs->FILE_ADDING_ADDRESS / 256;
fs->volume[file_entry + 29] = fs->FILE_ADDING_ADDRESS % 256;

return file_pointer;
```

At the end, **file_pointer** will be returned.

In the read function, it first checks whether the **fp** is valid (!= -1), and whether the **FCB** is empty (start from '\0'). If read operation is ok, starting from the address stored in **FCB**, read bit by bit into output until the number of chars reach **size**.

```
__device__ void fs_read(FileSystem *fs, uchar *output, u32 size, u32 fp)
{
  /* Implement read operation here */
  if (fp != -1 && fs->volume[fp * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE] != '\0') {
    u32 addr_entry = fs->volume[fp * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE + 28] * 256 + fs->volume[fp * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE + 29];
    for (int i = 0; i < size; i++) {
      output[i] = fs->volume[addr_entry + i];
    }
  }
}
```

In the write function, first find the head of **FCB** by using **fp * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE**. Then get the address (**file_pointer**) and size (**old_size**) of the old file, making a record. Then update the new size by using the new writing **size**. There are several conditions of writing in data.

```
u32 addr_entry = fp * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE;
u32 file_pointer = fs->volume[addr_entry + 28] * 256 + fs->volume[addr_entry + 29];
u32 old_size = fs->volume[addr_entry + 24] + fs->volume[addr_entry + 25]<<8 + fs->volume[addr_entry + 26]<<16 + fs->volume[addr_entry + 27]<<24;
//set new size
fs->volume[addr_entry + 24] = size % 256;
fs->volume[addr_entry + 25] = (size>>8) % 256;
fs->volume[addr_entry + 26] = (size>>16) % 256;
fs->volume[addr_entry + 27] = (size>>24) % 256;
```

If file is empty, meaning **fs->FILE_ADDING_ADDRESS == fs->FILE_BASE_ADDRESS**, no new file size adds to adding address. Here directly copy the

chars from **input** to volume starting from **file_pointer**, for '**size**' times. Then the adding address will add the size.

```
if (fs->FILE_ADDING_ADDRESS == fs->FILE_BASE_ADDRESS) {
  for (int i = 0; i < size; i++) {
    fs->volume[file_pointer + i] = input[i];
  }
  fs->FILE_ADDING_ADDRESS += size;
```

If adding address is exactly the starting address plus **old_size**, then write in data from **input** for **size** chars. Be careful that if new size is smaller than old size, then the space after new size but before the end of old size will be stored '\0'. The adding address will minus old size and add new size.

```
} else if (fs->FILE_ADDING_ADDRESS == fp * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE + old_size) {
  for (int i = 0; i < size; i++) {
    fs->volume[file_pointer + i] = input[i];
  }
  if (size < old_size) {
    for (int i = file_pointer + old_size-size; i < file_pointer + old_size; i++) {
      fs->volume[i] = '\0';
    }
  }
```

In other conditions, since old file will be cleaned, new content will start from **fs->FILE_ADDING_ADDRESS-old_size**. When the old file is being cleaned, move content at index **i + old_size** to **i**, moving them forward for **old_size** distance until the end of adding address. The new file content will add behind them. Remember if the new file size is smaller than old file size, the difference should be stored as '\0'. Also, for the file after **file_pointer** which address has be decreased for value **old_size**, their address stored in **FCB** should be updated. After updating the existing files, update the new file address and adding address.

```
} else {
  u32 new_addr = fs->FILE_ADDING_ADDRESS-old_size;
  u32 i = file_pointer;
  while (i < new_addr) {
    fs->volume[i] = fs->volume[i + old_size];
    i++;
  }
  for (int j = 0; j < size; j++) {
    fs->volume[new_addr + j] = input[j];
  }

  if (size < old_size) {
    for (int k = 0; k < old_size-size; k++) {
      fs->volume[fs->FILE_ADDING_ADDRESS-k] = '\0';
    }
  }
  for (int i = 0; i < fs->FCB_ENTRIES; i++) {
    u32 FCB_start = i * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE;
    u32 addr = fs->volume[FCB_start + 28] * 256 + fs->volume[FCB_start + 29]-old_size;
    if (i != addr_entry && addr >= file_pointer) {
      if (fs->volume[FCB_start] != '\0') {
        fs->volume[FCB_start + 28] = addr / 256;
        fs->volume[FCB_start + 29] = addr % 256;
      }
    }
  }
  fs->volume[addr_entry + 28] = new_addr / 256;
  fs->volume[addr_entry + 29] = new_addr % 256;

  fs->FILE_ADDING_ADDRESS = new_addr + size;
}
```

In the sorting\listing function, I use insertion sort to do sorting. In **LS_D**, starting from the second **FCB**, compare with the file in front of it. Use **current_modified_time** and **previous_modified_time** to represent the target file's modified time and that of its previous file. Use index to represent the position of the target file. If current one is larger than previous one, exchange the two **FCB** and update **index** by minus 1.

```
if (op == LS_D) {
  printf("===sort by modified time===\n");
  int temp_modified_time = 0;
  int temp_fcb[32];
  for (int i = 1; i < fs->FCB_ENTRIES; i++) {
    u32 addr_entry = i * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE;
    if (fs->volume[addr_entry] != '\0') {
      int current_modified_time = fs->volume[addr_entry + 22] * 256 + fs->volume[addr_entry + 23];
      int index = i;
      for (int j = i-1; j >= 0; j--) {
        u32 addr_entry2 = j * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE;
        if (fs->volume[addr_entry2] != '\0') {
          int previous_modified_time = fs->volume[addr_entry2 + 22] * 256 + fs->volume[addr_entry2 + 23];
          if (previous_modified_time < current_modified_time) {
            for (int k = 0; k < fs->FCB_SIZE; k++) {
              temp_fcb[k] = fs->volume[index * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE + k];
              fs->volume[index * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE + k] = fs->volume[addr_entry2 + k];
              fs->volume[addr_entry2 + k] = temp_fcb[k];
            }
            index--;
          }
        }
      }
    }
  }
```

After sorting, print out file name using char*.

In **LS_S**, I use **current_file_size** and **previous_file_size** to represent the file size of target file size and previous one, **current_create_time** and **previous_create_time** to represent target create time and previous one. Still starting from the second file, comparing the file size, if current is larger than previous or when they are the same, current create time is smaller than previous, exchange the **FCB**.

```
} else if (op == LS_S) {
  printf("===sort by file size===\n");
  int temp_file_size = 0;
  int temp_fcb[32];
  for (int i = 1; i < fs->FCB_ENTRIES; i++) {
    u32 addr_entry = i * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE;
    if (fs->volume[addr_entry] != '\0') {
      int current_file_size = fs->volume[addr_entry + 24] + (fs->volume[addr_entry + 25]<<8) + (fs->volume[addr_entry + 26]<<16) + (fs->volume[addr_entry + 27]<<24);
      int current_create_time = fs->volume[addr_entry + 20] * 256 + fs->volume[addr_entry + 21];
      int index = i;
      for (int j = i-1; j >= 0; j--) {
        u32 addr_entry2 = j * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE;
        if (fs->volume[addr_entry2] != '\0') {
          int previous_file_size = fs->volume[addr_entry2 + 24] + (fs->volume[addr_entry2 + 25]<<8) + (fs->volume[addr_entry2 + 26]<<16) + (fs->volume[addr_entry2 + 27]<<
          int previous_create_time = fs->volume[addr_entry2 + 20] * 256 + fs->volume[addr_entry2 + 21];
          if (previous_file_size < current_file_size || (previous_file_size == current_file_size && current_create_time < previous_create_time)) {
            for (int k = 0; k < fs->FCB_SIZE; k++) {
              temp_fcb[k] = fs->volume[index * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE + k];
              fs->volume[index * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE + k] = fs->volume[addr_entry2 + k];
              fs->volume[addr_entry2 + k] = temp_fcb[k];
            }
            index--;
          }
        }
      }
    }
  }
```

After sorting, print out the filename plus file size using char*.

In the deleting function, check whether the operation is **RM**. If yes, use **compare_file_name** to find the index of **FCB**. If **file_pointer** is -1, then no file found. Otherwise, use **addr_entry** to store the head of **FCB**. Count address and size stored in the **FCB**. Starting from address to adding address, move chars at **i + file_size** to **i**, deleting the first file in the range until **i + file_size** equals adding address. Then substitute the following chars as '\0'. Then let adding address minus **file_size**, meaning deleting a file. Then updating the address of the moved file and set chars in the deleted FCB as '\0'.

```
if (op == RM) {
  //check whether the file exists
  u32 file_pointer = -1;
  for (int i = 0; i < fs->FCB_ENTRIES; i++) {
    u32 addr_entry = i * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE;
    if (fs->volume[addr_entry] != '\0') {
      bool check_similarity = compare_file_name(s, (char*) &fs->volume[addr_entry]);
      if (check_similarity) {
        file_pointer = i;
        break;
      }
    }
  }

  u32 addr_entry = file_pointer * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE;
  if (file_pointer == -1) {
    printf("file %s does not exist!\n", s);
  } else {
    //clean the content
    u32 addr = fs->volume[addr_entry + 28] * 256 + fs->volume[addr_entry + 29];
    u32 file_size = fs->volume[addr_entry + 24] + fs->volume[addr_entry + 25]<<8 + fs->volume[addr_entry + 26]<<16 + fs->volume[addr_entry + 27]<<24;
    for (u32 i = addr; i < fs->FILE_ADDING_ADDRESS; i++) {
      if (i < fs->FILE_ADDING_ADDRESS-file_size) {
        fs->volume[i] = fs->volume[i + file_size];
      } else if (i >= fs->FILE_ADDING_ADDRESS-file_size && i < fs->FILE_ADDING_ADDRESS) {
        fs->volume[i] = '\0';
      }
    }
```

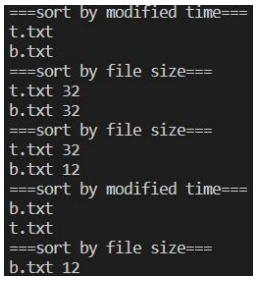
```
    fs->FILE_ADDING_ADDRESS -= file_size;
    for (int k = 0; k < fs->FCB_ENTRIES; k++) {
      u32 addr_entry2 = k * fs->FCB_SIZE + fs->SUPERBLOCK_SIZE;
      u32 new_addr = fs->volume[addr_entry2 + 28] * 256 + fs->volume[addr_entry2 + 29];
      if (fs->volume[addr_entry2] != '\0' && addr_entry2 != addr_entry) {
        if (new_addr-addr >= file_size) {
          fs->volume[addr_entry2 + 28] = (new_addr-file_size) / 256;
          fs->volume[addr_entry2 + 29] = (new_addr-file_size) % 256;
        }
      }
    }
    for (int i = 0; i < fs->FCB_SIZE; i++) {
      fs->volume[addr_entry + i] = '\0';
    }
}
```

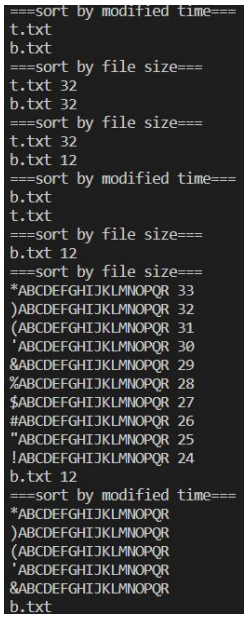
What problems you met in the assignment and what are your solution?

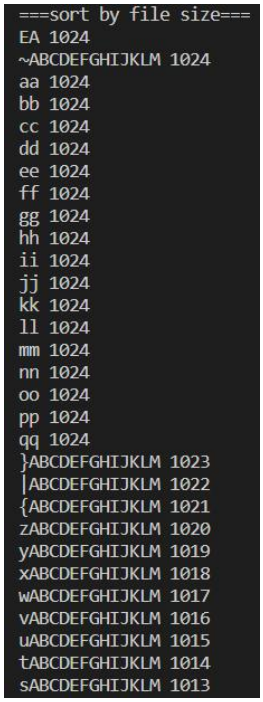
One problem I met in finishing the assignment is how the file pointer sending from open function can be used in the following functions. Here I use the index of FCB as fp so in the following functions, I need to change fp into the exact index (address). In testing, I sometimes failed to printout filename sometimes because of the

problem.

Screenshot of your program output







```
>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
```

```
triggering gc
===sort by modified time===
1024-block-1023
1024-block-1022
1024-block-1021
1024-block-1020
1024-block-1019
1024-block-1018
1024-block-1017
1024-block-1016
1024-block-1015
1024-block-1014
1024-block-1013
1024-block-1012
```

```
1024-block-0012
1024-block-0011
1024-block-0010
1024-block-0009
1024-block-0008
1024-block-0007
1024-block-0006
1024-block-0005
1024-block-0004
1024-block-0003
1024-block-0002
1024-block-0001
1024-block-0000
```

What did you learn from this assignment?

The assignment is about how to implement file system by implementing open, write, read, list, and delete function. By operating content of fs->volume, I know how FCB works and the basic structure of volume.