CSC3150 Project4 Report 120090263

1. Environment

```
OS: CentOS Linux release 7.5.1804
VSC version: 1.73.0
CUDA version: 11.7
GPU information: Nvidia Quadro RTX 4000 GPU
```

Item	Configuration / Version
System Type	x86_64
Opearing System	CentOS Linux release 7.5.1804
CPU	Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz 20 Cores, 40 Threads
Memory	100GB RAM
GPU	Nvidia Quadro RTX 4000 GPU x 1
CUDA	11.7
GCC	Red Hat 7.3.1-5
CMake	3.14.1

2. Execution step

```
1. Set corresponding test case in "user_program.cu"
2. Enter "nvcc -rdc=true file_system.cu main.cu user_program.cu -o file_system"
3. Enter "./file_system"
```

3. Design

In this program, I implement a file system which has only one root directory. It includes several operations----open, write, read, remove and display file information.

1. Open

Process:

In Open operation, it gives a file name to find the file's location. If file is found, return the file locaiton, that is, the File Control Block number (this is used as pointer). If in WRITE mode no file is found, a new file with size 0 will be create in the FCB. Its file attributes including size, name, create time, modified time and valid bit will be updated. However, it won't be allocated storage spcae until Write operation.

Details:

1. How to design FCB?
   In my program, there are 32 Byte for each FCB. The 1st byte is used as valid bit. If the file is written, it will be 1. If the file is removed, it will be 0. The 2-21 byte is file name. The 22-24 Byte is file size. I used address base as 256 to make it compact and sufficient. The 25-26 Byte is create time. The 27-28 is modified time. The 29-32 Byte is file address. (Note that Byte 24 and 32 is actually not used)

```
/* Overview of FCB------
  0 : valid bit
  1-20: file name
  21-23: file size
  24-25: create time
  26-27: modified time
  28-31: file address  /存的是在storage中的block number
*/
```

2. How to check file name?
   Compare each char by increment.

```
//check whether aim file name is correspond with file in storage
__device__ bool check_filename(char *n1, char *n2){
  while(*n1 == *n2){
    if(*n1 != '\0'){
      n1++;
      n2++;
    }
    else if(*n1 == '\0'){
      return true;
    }
  }
  return false;
}
```

2. Read

Process:

This operation read the content of file to OUTPUT Buffer. It use a read pointer to identify the location in the file and always read from the head. In my program, I first check the valid bit. Then use the pointer to get the file address from FCB. From the volunme to read the byte data to OUTPUT Buffer.

```
__device__ void fs_read(FileSystem *fs, uchar *output, u32 size, u32 fp)
{
  /* Implement read operation here */
  gtime++;
  if(!check_validbit(fs, fp)){
    printf("The file is not valid!\n");
    return ;
  }
  else{
    int storage_addr_ptr = get_storage_addr(fs, fp, ADDRESS_BASE);
    for (int i = 0; i < size; i++) {
      output[i] = fs->volume[storage_addr_ptr + i];
    }
  }

}
```

3. Write

Process:

This operation gives a file pointer to identify the file location in the volume. The file must be found since in the Open operation, I have create the FCB for it. I divided the case into 2 cases. If the found file size is 0, which means that it is newly created, then can allocate storage for it. If the size is not 0, it means that this is an old file existed. Then it has to be cleaned and write new content, and then allocate the storage. At last, update the storage pointer.

Detail

1. How to clean the old file?

I defined a new funciton named "do_compaction".

This function acts as a clean function and it is also used in Remove operation.

In this function, it will clean up the pointed file, update FCB and bitmap, compact the storage since it uses contiguous allocaiton.

There are several steps.

First, set the cleaned file's valid bit to be 0.

Second, obtain the shift-block-numbers after cleaning it.

Third, deal with FCB and VCB. Traverse all the file and compare their storage address with thr pointed file. If it is after the cleaned file, then it needs to be shifted forward. Also update its FCB including address information. Update its superblock information which means that set its previous storage blocks bitmap to be 0, while set its current storage blocks bitmap to be 1.

Fourth, shift the storage. Traverse to obtain the block numbers after the cleaned file, move all of them forward for shift-block-numbers steps.

```
__device__ void do_compaction(FileSystem *fs, u32 fp){//fp是要删除的文件的pointer（block number）

  // printf("This is one compactions\n");
  int shift_addr = get_storage_addr(fs, fp, ADDRESS_BASE); //要删除文件的实际地址
  int shift_block = get_block_addr(fs, fp, ADDRESS_BASE); //要删除文件的block起始位置
  int fp_size =  fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * fp +21] * ADDRESS_BASE + fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * fp +22];
  int block_takeup = fp_size/fs->STORAGE_BLOCK_SIZE;
  if((fp_size % fs->STORAGE_BLOCK_SIZE)!=0) block_takeup++;
  // printf("This is blocktake up %d", block_takeup);
  //设置这个文件的bitmap为0
  for(int i=0; i<block_takeup; i++){
    update_bitmap(fs, shift_block + i, 0);
  }

  //检查这个文件之后有多少个block要移动
  int storage_block_tomove = 0;//storage中要move的block数量
  for(int i = shift_block + block_takeup; i< 1024; i++){
    if(get_bitmap(fs, i) == 1) storage_block_tomove ++;
  }
  //遍历所有的文件，如果在这个removefile之后，则更新其VCB，FCB，并且shift storage
  for (int i = 0; i < fs->FCB_ENTRIES; i++) {
    if (i != fp && fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * i] != 0){
      int cur_addr = get_storage_addr(fs, i, ADDRESS_BASE); //当前遍历文件的实际地址
      int cur_block = get_block_addr(fs, i, ADDRESS_BASE); //当前遍历文件的block起始位置
      int cur_block_takeup =  fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * i +21] * ADDRESS_BASE + fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * i +22];
      //对比storage addr的大小
      if(cur_addr > shift_addr){
        //更新FCB中的addr
        int new_block = cur_block - shift_block;
        set_address(fs, i, new_block);
        //更新VCB
        //1.删除移动前的block bit为0
        for (int j = 0; j < cur_block_takeup; j++) {
          update_bitmap(fs, cur_block + j, 0);
        }
        //2.设置移动后的block bit为1
        for (int j = 0; j < cur_block_takeup; j++) {
          update_bitmap(fs, new_block + j, 1);
        }

      }
    }
```

更新FCB

更新VCB

```
  //移动storage----
  int tmp_add = shift_addr;
  for (int i = 0; i <  storage_block_tomove; i++) {//要移动storage_block_tomove这么多次
    for (int j = 0; j < fs->STORAGE_BLOCK_SIZE; j++){//每一个block要移动size次
      fs->volume[tmp_add] = fs->volume[tmp_add + block_takeup * fs->STORAGE_BLOCK_SIZE];
      tmp_add++;
    }
  }

  //更新global storage pointer
  gstorage_ptr = gstorage_ptr - block_takeup * fs->STORAGE_BLOCK_SIZE;
}
```

2. How to allocate storage?

I set a global variable "gstorage_ptr", which always points at the first free block address. Since I do storage compaction, it always points at the lowest position in the storage. Therefore, obtain the file size to calculate the blocks it will occupy and start from "gstorage_ptr" to allocate the storage.

```
int start_addr = gstorage_ptr + fs->FILE_BASE_ADDRESS;
int start_block = gstorage_ptr/ fs->STORAGE_BLOCK_SIZE; //不用减去filebase addr，因为直接从storage 0开始
for(int j=0; j<block_num; j++){
  update_bitmap(fs, start_block + j, 1);
}


//写进storage
for(int i=0; i<size; i++){
  fs->volume[start_addr++] = input[i];
}
```

4.Remove ---- fs_gsys(RM)

Process:

The operation search the file by its name. If found, delete the file and update file information.

Detail

1. How to delete the file?
   In this process, I apply "do_compaction" funciton to clean up the original file and compact the storage.
2. How to update the informaiton?
   The bitmap and FCB upadte is done in "do_compaction" funciton. However, one difference is that in Write operaiton, after cleaning the file I dont change the valid bit because it is over-written and is still in use. But in Remove operation, the valid bit of the cleaned file will be set to 0, indicating that this file is not valid, that is, the file is removed and cannot be accessed.

```
__device__ void fs_gsys(FileSystem *fs, int op, char *s)
{
  /* Implement rm operation here */
  gtime++;
  if(op == RM){
    //在FCB中找到要remove的file
    int removefile_pos = -1;
    for (int i = 0; i < fs->FCB_ENTRIES; i++) {
      if (check_filename(s, (char *) &fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * i +1])) {
        removefile_pos = i;
        break;
      }
    }

    //更新FCB---更改valid bit为0
    fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * removefile_pos] = 0;
    //移除文件并做compaction(包含更新移动了的file的VCB)
    do_compaction(fs, removefile_pos);

  }
}
```

5. Display ---- fs_gsys(LS_D/LS_S)

Process

If the parameter is LS_D, that is list all file names in the order of modified time. If it is LS_S, then list all file names in the order of size. When 2 files have the same size, sort it by create time which fisrt create first print.

Detail

How to sort the file?

Answer: Bubble sort

I used Bubble sort to sacrifice time for space since it can be down in the volume. Therefore, it does not require extra space. Each time I only have to swap the FCB entries, that is only sort FCB entries and then print the file names from 1st FCB entry to the last valid entry. The bitmap and storage is not changed.

```
if(op == LS_D){//用bubble sort来排序
  for(int i=0; i< fs->FCB_ENTRIES -1; i++){
    for(int j=0; j< fs->FCB_ENTRIES -1-i; j++){
      if(check_validbit(fs, j) && check_validbit(fs,j+1)){
        int left_modified_time = fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * j +26] *256 + fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * j +27];
        int right_modified_time = fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * (j+1) +26] *256 + fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * (j+1) +27];
        if(left_modified_time < right_modified_time){
          swap_FCB(fs, j, j+1);//只需要调整FCB的顺序
        }
      }
```

```
for(int i=0; i< fs->FCB_ENTRIES -1; i++){
  for(int j=0; j< fs->FCB_ENTRIES -1-i; j++){
    if(check_validbit(fs, j) && check_validbit(fs, j+1)){
      int left_size = fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * j +21] *256 + fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * j +22];
      int right_size = fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * (j+1) +21] *256 + fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * (j+1) +22];
      if(left_size < right_size){
        swap_FCB(fs, j, j+1);
      }
      else if(left_size == right_size){//若size相同，则按照create time 来排序
        int left_create_time = fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * j +24] *256 + fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * j +25];
        int right_create_time = fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * (j+1) +24] *256 + fs->volume[fs->SUPERBLOCK_SIZE + fs->FCB_SIZE * (j+1) +25];
        if(left_create_time > right_create_time){
          swap_FCB(fs, j, j+1);//只需要调整FCB的顺序
        }
      }
    }
  }
```

create time

4. Problems

1. I encounter the problem that when I was running test 3, when it is writing the 1001 files into the storage, after correctly writing several files it will stop and print error that the file system is full. So I checked write and open operation and found that I misused the global storage pointer, that is I set initially set it to be 0, so when I write files I have to add File Base Address to obtain the real storage address. Also in the Remove operation, I wrongly used the start block address to be the shift-block-number, which cause the file system problems.

5. Screen shot

case1

```
[120090263@node21 A4-template]$ ./file_system
===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
[120090263@node21 A4-template]$
```

case2

```
[120090263@node21 A4-template]$ ./file_system
===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
$ABCDEFGHIJKLMNOPQR 27
#ABCDEFGHIJKLMNOPQR 26
"ABCDEFGHIJKLMNOPQR 25
!ABCDEFGHIJKLMNOPQR 24
b.txt 12
===sort by modified time===
*ABCDEFGHIJKLMNOPQR
)ABCDEFGHIJKLMNOPQR
(ABCDEFGHIJKLMNOPQR
'ABCDEFGHIJKLMNOPQR
&ABCDEFGHIJKLMNOPQR
b.txt
[120090263@node21 A4-template]$
```

case3(partial)

```
GA 45
FA 44
DA 42
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
[120090263@node21 A4-template]$
```

case4(partial)

```
1024-block-0019
1024-block-0018
1024-block-0017
1024-block-0016
1024-block-0015
1024-block-0014
1024-block-0013
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
[120090263@node21 A4-template]$
```

6. What I learned from this project.

In this program, I implement a file system which make me know better about the file system structure and how the file is written, read and removed in our computers. I have a better understanding of how

operating system control the files and storage, such as doing compaction. Also, I become more skillful in CUDA programming.

*ps: Bonus task is not accomplished in this progra