Assignment Report

CSC 3150 File-Systems

Wei Wu (吴畏)

118010335

November 15, 2020

The School of Data Science

香 港 中 文 大 學（深 圳）
The Chinese University of Hong Kong, Shenzhen

1. Environment of Running My Program. (e.g., OS, VS Version, CUDA Version, GPU Information etc.)

a. OS: Microsoft Windows [Version 10.0.19041.610]

b. VS: Microsoft Visual Studio Community 2017 [Version 15.9.28]

c. VS Toolkit: Visual Studio 2015 (v140)

d. CUDA: NVIDIA CUDA 11.1

e. GPU: NVIDIA GeForce GTX 1070

f. GPU Driver: NVIDIA GeForce Game Ready Driver 456.81

2. How Did I Design My Program?

2.1. Basic Program (Source)

2.1.1. Introduction

This program is a file system based on CUDA. It takes the global memory as a volume and does not support directory structure. It implements a set of basic file operations: fs_open, fs_write, fs_read, fs_gsys(RM), fs_gsys(LS_D), and fs_gsys(LS_S). Specification of this file system is given in the following tables.

Item	Size (Bytes)
Volume	1085440
All Files	0 - 1048576
A File	0 - 1024
A Filename	0 - 20

File-control Block (FCB)	32
Storage Block	32

Item	Number of Items
File	0 - 1024
File-control Block (FCB)	0 - 1024

2.1.2. Structures

a. struct FileSystem

Two additional data fields are added. file_count is the current count of files while block_count is the current count of blocks occupied.

```
struct FileSystem {
    uchar *volume;
    int SUPERBLOCK_SIZE;
    int FCB_SIZE;
    int FCB_ENTRIES;
    int STORAGE_SIZE;
    int STORAGE_BLOCK_SIZE;
    int MAX_FILENAME_SIZE;
    int MAX_FILE_NUM;
    int MAX_FILE_SIZE;
    int FILE_BASE_ADDRESS;
    int MAX_BLOCK_NUM;

    int file_count;
    int block_count;
};
```

b. struct FCB

filename[20] is a string of the name of a file, address is the address of the starting block of a file, size is the size of a file in bytes, and time is the modified time of a file.

```
struct FCB { // file-control block: 32 bytes
    char filename[20]; // filename: 20 bytes
    uint16_t address = 0xFFFF; // block address: 2 bytes
    uint16_t size = 0; // size: 2 bytes
    uint32_t time = 0; // modified time: 4 bytes
};
```

2.1.3. Utility Functions

To make the entire program more concise and efficient, I designed a lot of utility functions. Most of them are called in multiple places.

a. __device__ void init_volume(FileSystem *fs)

Initialize the volume by setting all bytes to 0.

```
// initialize the volume by setting all bytes to 0
__device__ void init_volume(FileSystem *fs) {
    for (int i = 0; i < fs->STORAGE_SIZE; i++)
        fs->volume[i] = 0;
}
```

b. __device__ void read_filename(FileSystem *fs, int addr, char *dest)

__device__ void write_filename(FileSystem *fs, int addr, char *s)

Read a filename in the volume into a char* / write a filename into the volume.

```
// read a filename in the volume into a char*
__device__ void read_filename(FileSystem *fs, int addr, char *dest) {
    int offset = 0;
    while (fs->volume[addr + offset]) {
        dest[offset] = fs->volume[addr + offset];
        offset++;
    }
    dest[offset] = '\0';
}

// write a filename into the volume
__device__ void write_filename(FileSystem *fs, int addr, char *s) {
    int i = 0;
    while (s[i]) {
        fs->volume[addr + i] = s[i];
        i++;
    }
    fs->volume[addr + i] = '\0';
}
```

c. __device__ uint32_t read_word(FileSystem *fs, int addr)

__device__ void write_word(FileSystem *fs, int addr, uint32_t value)

__device__ uint16_t read_halfword(FileSystem *fs, int addr)

__device__ void write_halfword(FileSystem *fs, int addr, short value)

Read a word (halfword) in the volume / Write a word (halfword) into the volume.

```
// read a word in the volume
__device__ uint32_t read_word(FileSystem *fs, int addr) {
    uint32_t result = 0;
    for (int i = 0; i < 4; i++)
        result += fs->volume[addr + i] << (24 - 8 * i);
    return result;
}

// write a word into the volume
__device__ void write_word(FileSystem *fs, int addr, uint32_t value) {
    for (int i = 0; i < 4; i++)
        fs->volume[addr + i] = value >> (24 - 8 * i);
}

// read a halfword in the volume
__device__ uint16_t read_halfword(FileSystem *fs, int addr) {
    uint16_t result = 0;
    for (int i = 0; i < 2; i++)
        result += fs->volume[addr + i] << (8 - 8 * i);
    return result;
}

// write a word into the volume
__device__ void write_halfword(FileSystem *fs, int addr, short value) {
    for (int i = 0; i < 2; i++)
        fs->volume[addr + i] = value >> (8 - 8 * i);
}
```

d. __device__ void update_bitmap(FileSystem *fs)

Update the bit map according to the current file count.

```
// update the bitmap
__device__ void update_bitmap(FileSystem *fs) {
    // update the superblock (bit map)
    int filled_bytes_num = fs->block_count / 8;
    // filled bytes = 0b 1111 1111
    for (int i = 0; i < filled_bytes_num; i++)
        fs->volume[i] = 0b11111111;
    // half-filled byte = 0b ???? ???? (could be 0)
    int half_filled_byte = 0;
    for (int i = 0; i < fs->block_count % 8; i++)
        half_filled_byte += 1 << (7 - i);
    fs->volume[filled_bytes_num] = half_filled_byte;
    // unfilled bytes = 0b 0000 0000
    for (int i = filled_bytes_num + 1; i < fs->SUPERBLOCK_SIZE; i++)
        fs->volume[i] = 0;
}
```

e. __device__ int compact(FileSystem *fs, int frag_start, int frag_size)

Compact the volume by eliminating the fragment indicated by frag_start and frag_size. In the meantime, the FCBs corresponding to moved files are updated.

```
// compact the volume
__device__ int compact(FileSystem *fs, int frag_start, int frag_size) {
    int frag_end = frag_start + frag_size - 1;
    int move_start = (frag_end + 1) * fs->STORAGE_BLOCK_SIZE  + fs->FILE_BASE_ADDRESS;
    int move_size = (fs->block_count - 1 - frag_end) * fs->STORAGE_BLOCK_SIZE;
    // move the subsequent data to fill up the fragment
    for (int i = 0; i < move_size; i++) {
        int from = move_start + i;
        int to = frag_start * fs->STORAGE_BLOCK_SIZE + fs->FILE_BASE_ADDRESS + i;
        fs->volume[to] = fs->volume[from];
    }
    // update the FCBs
    for (int i = 0; i <= fs->file_count; i++) {
        int fcb_addr = read_FCB_address(fs, i);
        if (fcb_addr > frag_start) {
            int fcb_addr_new = fcb_addr - frag_size;
            write_FCB_address(fs, i, fcb_addr_new);
        }
    }
    // update the block count
    fs->block_count -= frag_size;
    // update the bit map
    update_bitmap(fs);
}
```

f. __device__ int get_length(const char* ptr)

Get the length of a string (including the '/0').

```
// get the length of a string
__device__ int get_length(const char* ptr) {
    int length = 0;
    while (*ptr++)
        length++;
    length++;
    return length;
}
```

g. __device__ int find_filename(FileSystem *fs, const char *filename)

Search the FCBs

```
// search the FCBs for a given filename, return its address if found
__device__ int find_filename(FileSystem *fs, const char *filename) {
    int filename_length = get_length(filename);
    if (filename_length > 20) {
        printf("Error: filename \"%s\" is over %d characters\n", filename, fs->MAX_FILENAME_SIZE)
    }
    // search among the FCBs
    char *fcb_filename = (char *)malloc(20 * sizeof(char));
    for (int i = 0; i < fs->file_count; i++) {
        int base_addr = fs->SUPERBLOCK_SIZE + i * fs->FCB_SIZE;
        read_filename(fs, base_addr, fcb_filename);
        bool found = true;
        for (int i = 0; i < filename_length; i++) {
            if (fcb_filename[i] != filename[i]) {
                found = false;
                break;
            }
        }
        if (found)
            return i;
    }
    free(fcb_filename);
    // if not found
    return -1;
}
```

h. __device__ void write_FCB_filename(FileSystem *fs, int fcb_num, char* filename)

__device__ void write_FCB_address(FileSystem *fs, int fcb_num, uint16_t address)

__device__ void write_FCB_size(FileSystem *fs, int fcb_num,

uint16_t size)

__device__ void write_FCB_mod_time(FileSystem *fs, int fcb_num, uint32_t time)

Update FCB data fields.

```
// update the FCB filename
__device__ void write_FCB_filename(FileSystem *fs, int fcb_num, char* filename) {
    int base_addr = fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE;
    write_filename(fs, base_addr, filename);
}

// update the FCB address
__device__ void write_FCB_address(FileSystem *fs, int fcb_num, uint16_t address) {
    write_halfword(fs, fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE, address);
}

// update the FCB size
__device__ void write_FCB_size(FileSystem *fs, int fcb_num, uint16_t size) {
    write_halfword(fs, fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE + sizeof(uint16_t), size);
}

// update the FCB modified time
__device__ void write_FCB_mod_time(FileSystem *fs, int fcb_num, uint32_t time) {
    write_word(fs, fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE + sizeof(uint32_t), time);
}
```

i. __device__ void read_FCB_filename(FileSystem *fs, int fcb_num, char* dest)

__device__ uint16_t read_FCB_address(FileSystem *fs, int fcb_num)

__device__ uint16_t read_FCB_size(FileSystem *fs, int fcb_num)

__device__ uint32_t read_FCB_mod_time(FileSystem *fs, int fcb_num)

Read FCB data fields.

```
// read the FCB filename into a char*
__device__ void read_FCB_filename(FileSystem *fs, int fcb_num, char* dest) {
    int base_addr = fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE;
    read_filename(fs, base_addr, dest);
}

// read the FCB address
__device__ uint16_t read_FCB_address(FileSystem *fs, int fcb_num) {
    return read_halfword(fs, fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE);
}

// read the FCB size
__device__ uint16_t read_FCB_size(FileSystem *fs, int fcb_num) {
    return read_halfword(fs, fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE + sizeof(uint16_t));
}

// read the FCB modified time
__device__ uint32_t read_FCB_mod_time(FileSystem *fs, int fcb_num) {
    return read_word(fs, fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE + sizeof(uint32_t));
}
```

2.1.4. Major Functions

a. __device__ void fs_init(FileSystem *fs, uchar *volume, int SUPERBLOCK_SIZE, int FCB_SIZE, int FCB_ENTRIES, int VOLUME_SIZE, int STORAGE_BLOCK_SIZE, int MAX_FILENAME_SIZE, int MAX_FILE_NUM, int MAX_FILE_SIZE, int FILE_BASE_ADDRESS)

Initialize the file system.

```
// initialize the file system
__device__ void fs_init(FileSystem *fs, uchar *volume, int SUPERBLOCK_SIZE,
    int FCB_SIZE, int FCB_ENTRIES, int VOLUME_SIZE,
    int STORAGE_BLOCK_SIZE, int MAX_FILENAME_SIZE,
    int MAX_FILE_NUM, int MAX_FILE_SIZE, int FILE_BASE_ADDRESS) {
    // init variables
    fs->volume = volume;
    fs->file_count = 0;
    fs->block_count = 0;

    // init constants
    fs->SUPERBLOCK_SIZE = SUPERBLOCK_SIZE;
    fs->FCB_SIZE = FCB_SIZE;
    fs->FCB_ENTRIES = FCB_ENTRIES;
    fs->STORAGE_SIZE = VOLUME_SIZE;
    fs->STORAGE_BLOCK_SIZE = STORAGE_BLOCK_SIZE;
    fs->MAX_FILENAME_SIZE = MAX_FILENAME_SIZE;
    fs->MAX_FILE_NUM = MAX_FILE_NUM;
    fs->MAX_FILE_SIZE = MAX_FILE_SIZE;
    fs->FILE_BASE_ADDRESS = FILE_BASE_ADDRESS;
    fs->MAX_BLOCK_NUM = MAX_FILE_SIZE / STORAGE_BLOCK_SIZE;

    // init volume
    init_volume(fs);
}
```

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

Open a file and return a file pointer pointing at the starting block of that file. It first calls find_filename() to searches for the filename among all FCBs. If the filename exists, it return the file's address directly. If not, then it checks the op code. If op is G_READ, return an error. If op is G_WRITE and the file count is less than the maximum, it creates a new file by updating the FCB and file count. Then, it returns the current block count as the address of the new file.

```
// open a file and return a file pointer
__device__ u32 fs_open(FileSystem *fs, char *s, int op) {
    // search the FCBs for the filename
    int fcb_num = find_filename(fs, s);
    // if the filename exists
    if (fcb_num != -1) {
        int file_addr = read_FCB_address(fs, fcb_num);
        return file_addr;
    }
    // else if the filename does not exist
    else if (fcb_num == -1) {
        // if operation is write, create a new file
        if (op == G_WRITE) {
            if (fs->file_count >= fs->MAX_FILE_NUM) {
                printf("Error: the number of files reaches %d\n", fs->MAX_FILE_NUM);
                return 0x80000000;
            }
            // update the FCB
            write_FCB_filename(fs, fs->file_count, s);
            write_FCB_address(fs, fs->file_count, fs->block_count);
            write_FCB_size(fs, fs->file_count, 0);
            write_FCB_mod_time(fs, fs->file_count, gtime);
            // increase gtime by 1
            gtime++;
            // increase file count by 1
            fs->file_count++;
            // return a pointer to the next free block
            return fs->block_count;
        }
        // if operation is read
        else if (op == G_READ) {
            printf("Error: file \"%s\" does not exist.\n", s);
            return 0x80000000;
        }
    }
}
```

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

Read the content of a file into the result buffer if the combination of fp and size is valid.

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

Write the content of the input buffer into a file. There are several invalid cases: fp is over the boundary, fp points to some free block

other than the first one, size is greater than the maximum size of a single file, FCB does not exist, and volume is not sufficient. If valid, there are three cases. The first case is that fp points to the first free block, which means the file to write is an empty file. In this case, just write the data and do some updates. The second case is that fp points to an occupied block (a non-empty file) and the blocks allocated to that file are sufficient. Then, after writing the data, the program needs to detect and eliminate external fragments. The third case is the same as the second one except that the blocks of the original file are not sufficient. To deal with this, the program should compact the volume (removing the original file) and then write the new data into the volume.

```
// write the content of the input buffer into a file
__device__ u32 fs_write(FileSystem *fs, uchar* input, u32 size, u32 fp) {
    int blocks_occupied = (size - 1) / fs->STORAGE_BLOCK_SIZE + 1;
    // invalid case 1: if fp is over the boundary
    if (fp > fs->MAX_BLOCK_NUM) { ... }
    // invalid case 2: if fp points to some free block other than the first one
    if (fp > fs->block_count) { ... }
    // invalid case 3: if size > the max size of a file
    if (size > fs->MAX_FILE_SIZE / fs->MAX_FILE_NUM) { ... }
    // find the corresponding FCB
    int fcb_num = -1;
    for (int i = 0; i < fs->file_count; i++) { ... }
    // invalid case 4: FCB does not exist
    if (fcb_num == -1) { ... }
    // valid case 1: if fp points to the first free block, then this is an empty file
    if (fp == fs->block_count) { ... }
    // if fp points to a occupied block, then this is a non-empty file
    else if (fp < fs->block_count) {
        int file_size = read_FCB_size(fs, fcb_num);
        int file_blocks_occupied = (file_size - 1) / fs->STORAGE_BLOCK_SIZE + 1;
        // valid case 2: if the blocks of the original file are enough
        if (blocks_occupied <= file_blocks_occupied) { ... }
        // valid case 3: if the blocks of the original file are not enough
        else { ... }
    }
    // update the FCB
    write_FCB_size(fs, fcb_num, size);
    write_FCB_mod_time(fs, fcb_num, gtime);
    // increase gtime by 1
    gtime++;
    // update the bit map
    update_bitmap(fs);
}
```

e. __device__ void fs_gsys(FileSystem *fs, int op)

List out all the files in the volume sorted by modified time or file size. If there are several FCBs with the same file size, then first create first print. Refer to Section 3.a of this Report for more details.

```
// list out the files
__device__ void fs_gsys(FileSystem *fs, int op) {
    bool *printed = (bool *)malloc(fs->file_count * sizeof(bool));
    char *filename = (char *)malloc(fs->MAX_FILENAME_SIZE * sizeof(char));
    for (int i = 0; i < fs->file_count; i++)
        printed[i] = false;
    // sort by modified time
    if (op == LS_D) { ... }
    // sort by file size
    else if (op == LS_S) { ... }
    free(printed);
    free(filename);
}
```

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

Remove an existing file. After removing, compact both file volume and FCB volume.

```
// remove a file
__device__ void fs_gsys(FileSystem *fs, int op, char *s) {
    if (op == RM) {
        int fcb_num = find_filename(fs, s);
        //if the file exists
        if (fcb_num != -1) {
            // get the file size and number of blocks occupied
            int file_size = read_FCB_size(fs, fcb_num);
            int block_occupied = (file_size - 1) / fs->STORAGE_BLOCK_SIZE + 1;
            // get the file address
            int file_address = read_FCB_address(fs, fcb_num);
            // compact the volume
            compact(fs, file_address, block_occupied);
            // compact the FCBs
            for (int i = fcb_num; i < fs->file_count; i++) { ... }
            // update the file count
            fs->file_count--;
            // update the bit map
            update_bitmap(fs);
        }
        //if the file does not exist
        else if (fcb_num == -1)
            printf("Error: file \"%s\" does not exist.\n", s);
    }
}
```

2.2. Bonus Program

2.2.1. Introduction

The bonus program supports a tree-structured directory-file system. It adds a lot of new functions to the original file system: fs_gsys(MKDIR), fs_gsys(CD), fs_gsys(CD_P), fs_gsys(RM_RF), and fs_gsys(PWD). Also, many of the original functions, such as fs_gsys(LS_D), fs_gsys(LS_S), and fs_gsys(RM), are updated to support directory operations.

2.2.2. Structures

a. FileSystem

```
struct FileSystem {
    uchar *volume;
    int SUPERBLOCK_SIZE;
    int FCB_SIZE;
    int FCB_ENTRIES;
    int STORAGE_SIZE;
    int STORAGE_BLOCK_SIZE;
    int MAX_FILENAME_SIZE;
    int MAX_FILE_NUM;
    int MAX_FILE_SIZE;
    int FILE_BASE_ADDRESS;
    int MAX_BLOCK_NUM;

    int MAX_DIR_DEPTH;
    int MAX_DIR_FILE_NUM;

    int file_count;
    int block_count;

    int* current_dir;
};
```

Two constants and one variable are added. MAX_DIR_DEPTH is the

maximum depth of the tree-structured directories. MAX_DIR_FILE_NUM is the maximum number of files in a single directory. current_dir is an array of integers indicating the current working directory.

b. FCB

```
struct FCB { // file-control block: 32 bytes
    char filename[20]; // filename: 20 bytes
    uint16_t address = 0xFFFF; // block address: 2 bytes
    int16_t size = 0; // size: 2 bytes
    uint32_t time = 0; // modified time: 4 bytes
    int16_t parent = -1; // parent fcb number: 2 bytes
    int8_t depth = 0; // depth: 1 byte
    int8_t type = 0; // type: 1 byte
};
// byte 0-19: filename
// byte 20-21: address
// byte 22-23: size
// byte 24-27: modified time
// byte 28-29: parent
// byte 30: depth
// byte 31: type
```

Three additional data fields are added. parent is the FCB number of the parent directory. depth is the depth of the current file / directory in the tree structure. type indicates whether the object is a file or directory. Refer Section 3.b of this Report for more details.

2.2.3. Utility Functions

a. __device__ int get_current_dir_index(FileSystem *fs)

Get the index (FCB number) of the current (working) directory.

```
// get the index of the current directory
__device__ int get_current_dir_index(FileSystem *fs) {
    for (int i = fs->MAX_DIR_DEPTH - 1; i >= 0; i--) {
        if (fs->current_dir[i] != -1)
            return fs->current_dir[i];
    }
    return -1;
}
```

b. __device__ int get_current_depth(FileSystem *fs)

Get the depth of the current working directory in the tree structure.

```
// get the current directory's depth
__device__ int get_current_depth(FileSystem *fs) {
    int depth = 0;
    for (int i = fs->MAX_DIR_DEPTH - 1; i >= 0; i--) {
        if (fs->current_dir[i] != -1)
            depth++;
    }
    return depth;
}
```

c. __device__ void write_FCB_parent(FileSystem *fs, int fcb_num, uint16_t parent)

__device__ void write_FCB_depth(FileSystem *fs, int fcb_num, uint8_t depth)

__device__ void write_FCB_type(FileSystem *fs, int fcb_num, uint8_t type)

Update the FCB data fields.

```
// update the FCB parent
__device__ void write_FCB_parent(FileSystem *fs, int fcb_num, uint16_t parent) {
    write_halfword(fs, fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE + sizeof(uint32_t) + sizeof(uint32_t), parent);
}

// update the FCB depth
__device__ void write_FCB_depth(FileSystem *fs, int fcb_num, uint8_t depth) {
    fs->volume[fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE + sizeof(uint32_t) + sizeof(uint32_t) + sizeof(uint16_t)] = depth;
}

// update the FCB type
__device__ void write_FCB_type(FileSystem *fs, int fcb_num, uint8_t type) {
    fs->volume[fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE + sizeof(uint32_t) + sizeof(uint32_t) + sizeof(uint16_t) + sizeof(uint8_t)] = type;
}
```

d. __device__ int16_t read_FCB_parent(FileSystem *fs, int fcb_num)

__device__ uint8_t read_FCB_depth(FileSystem *fs, int fcb_num)

__device__ uint8_t read_FCB_type(FileSystem *fs, int fcb_num)

Read the FCB data fields.

```
// read the FCB parent
__device__ int16_t read_FCB_parent(FileSystem *fs, int fcb_num) {
    return read_halfword(fs, fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE + sizeof(uint32_t) + sizeof(uint32_t));
}

// read the FCB depth
__device__ uint8_t read_FCB_depth(FileSystem *fs, int fcb_num) {
    return fs->volume[fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE + sizeof(uint32_t) + sizeof(uint32_t) + sizeof(uint16_t)];
}

// read the FCB type
__device__ uint8_t read_FCB_type(FileSystem *fs, int fcb_num) {
    return fs->volume[fs->SUPERBLOCK_SIZE + fcb_num * fs->FCB_SIZE + fs->MAX_FILENAME_SIZE + sizeof(uint32_t) + sizeof(uint32_t) + sizeof(uint16_t) + sizeof(uint8_t)];
}
```

e. __device__ int get_current_dir_file_count(FileSystem *fs)

Get the number of files in the current directory.

```
// get the number of files in the current directory
__device__ int get_current_dir_file_count(FileSystem *fs) {
    int current_dir = get_current_dir_index(fs);
    int count = 0;
    for (int i = 0; i < fs->file_count; i++) {
        if (read_FCB_parent(fs, i) == current_dir)
            count++;
    }
    return count;
}
```

f. __device__ void compact_FCBs(FileSystem *fs, int fcb_num)

Compact the FCB volume. Refer to Section 3.c of this Report for more details.

```
// compact the FCBs
__device__ void compact_FCBs(FileSystem *fs, int fcb_num) {
    // update all FCB parents
    for (int i = 0; i < fs->file_count; i++) {
        int parent = read_FCB_parent(fs, i);
        // change all FCB parents which are equal to fcb_num to -2 (dangling)
        if (parent == fcb_num)
            write_FCB_parent(fs, i, -2);
        // decrease all FCB parents which are greater than fcb_num by 1
        else if (parent > fcb_num)
            write_FCB_parent(fs, i, parent - 1);
    }
    // compact the FCBs
    for (int i = fcb_num; i < fs->file_count; i++) {
        for (int j = 0; j < fs->FCB_SIZE; j++)
            fs->volume[fs->SUPERBLOCK_SIZE + i * fs->FCB_SIZE + j] = fs->volume[fs->SUPERBLOCK_SIZE + (i + 1) * fs->FCB_SIZE + j];
    }
}
```

g. __device__ void remove_file_recursively(FileSystem *fs, int fcb_num, char *s)

Remove a directory and its contents. Refer to Section 3.d of this Report for more details.

```
// remove a directory and its content
__device__ void remove_file_recursively(FileSystem *fs, int fcb_num, char *s) {
    // update the file count
    fs->file_count--;
    // if this is a file
    if (read_FCB_type(fs, fcb_num) == FILE_TYPE) {
        // get the file size and number of blocks occupied
        int file_size = read_FCB_size(fs, fcb_num);
        int block_occupied = (file_size - 1) / fs->STORAGE_BLOCK_SIZE + 1;
        // get the file address
        int file_address = read_FCB_address(fs, fcb_num);
        // compact the volume
        compact_files(fs, file_address, block_occupied);
        // update the bit map
        update_bitmap(fs);
    }
    // update the parent's FCB size
    int file_parent = read_FCB_parent(fs, fcb_num);
    update_dir_size(fs, file_parent, -get_length(s));
    // compact the FCBs
    compact_FCBs(fs, fcb_num);
    // remove the next file
    for (int i = 0; i < fs->file_count; i++) {
        if (read_FCB_parent(fs, i) == -2) {
            remove_file_recursively(fs, i, s);
            return;
        }
    }
}
```

h. __device__ void update_dir_size(FileSystem *fs, int dir_fcb_num, int

delta_size)

Update the FCB size of a directory.

```
// update a directory's size
__device__ void update_dir_size(FileSystem *fs, int dir_fcb_num, int delta_size) {
    if (dir_fcb_num != -1) {
        int old_dir_size = read_FCB_size(fs, dir_fcb_num);
        write_FCB_size(fs, dir_fcb_num, old_dir_size + delta_size);
    }
}
```

2.2.4. Major Functions

a. __device__ void fs_gsys(FileSystem *fs, int op)

LS_D: list out all the files / directories sorted by modified time.

LS_S: list out all the files / directories sorted by size.

CD_P: move up to the parent directory of the current directory.

PWD: print the current path name.

```
// move up to the parent directory
else if (op == CD_P) {
    if (get_current_dir_index(fs) == -1) {
        printf("Error: already in the root directory.\n");
        return;
    }
    for (int i = fs->MAX_DIR_DEPTH - 1; i >= 0; i--) {
        if (fs->current_dir[i] != -1) {
            fs->current_dir[i] = -1;
            break;
        }
    }
}
// print the current path name
else if (op == PWD) {
    char *filename = (char *) malloc(20 * sizeof(char));
    for (int i = 0; i < fs->MAX_DIR_DEPTH; i++) {
        if (fs->current_dir[i] != -1) {
            read_FCB_filename(fs, fs->current_dir[i], filename);
            printf("/%s", filename);
        }
    }
    free(filename);
    printf("\n");
}
```

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

MKDIR: make a new directory under the current directory.

```
// make a new directory
if (op == MKDIR) {
    if (fs->file_count >= fs->MAX_FILE_NUM) {
        printf("Error: the number of files reaches %d.\n", fs->MAX_FILE_NUM);
        return;
    }
    int current_depth = get_current_depth(fs);
    if (current_depth >= fs->MAX_DIR_DEPTH) {
        printf("Error: the directory depth reaches %d.\n", fs->MAX_DIR_DEPTH);
        return;
    }
    // update the directory's FCB size
    int current_dir = get_current_dir_index(fs);
    update_dir_size(fs, current_dir, get_length(s));
    // update the FCB
    write_FCB_filename(fs, fs->file_count, s);
    write_FCB_address(fs, fs->file_count, 0x80000000);
    write_FCB_size(fs, fs->file_count, 0);
    write_FCB_mod_time(fs, fs->file_count, gtime);
    write_FCB_parent(fs, fs->file_count, current_dir);
    write_FCB_depth(fs, fs->file_count, current_depth + 1);
    write_FCB_type(fs, fs->file_count, DIR_TYPE);
    // increase gtime by 1
    gtime++;
    // increase file count by 1
    fs->file_count++;
}
```

CD: change the current directory to some subdirectory under the current directory.

```
// change directory
else if (op == CD) {
    int fcb_num = find_filename(fs, s);
    //if the directory exists in the current directory
    if (fcb_num != -1) {
        // if this is not a directory
        if (read_FCB_type(fs, fcb_num) != DIR_TYPE) {
            printf("Error: file \"%s\" is not a directory.\n", s);
            return;
        }
        for (int i = 0; i < fs->MAX_DIR_DEPTH; i++) {
            if (fs->current_dir[i] == -1) {
                fs->current_dir[i] = fcb_num;
                break;
            }
        }
    }
    //if the directory does not exist in the current directory
    else if (fcb_num == -1) {
        printf("Error: directory \"%s\" does not exist.\n", s);
        return;
    }
}
```

RM_RF: remove a directory and all its subdirectories and files. Note that this operation cannot remove a file.

```
// remove a directory
else if (op == RM_RF) {
    int fcb_num = find_filename(fs, s);
    //if the directory exists in the current directory
    if (fcb_num != -1) {
        // if this is not a directory
        if (read_FCB_type(fs, fcb_num) != DIR_TYPE) {
            printf("Error: file \"%s\" is not a directory.\n", s);
            return;
        }
        // remove the subdirectories and files recursively
        remove_file_recursively(fs, fcb_num, s);
    }
    //if the directory does not exist in the current directory
    else if (fcb_num == -1) {
        printf("Error: directory \"%s\" does not exist.\n", s);
        return;
    }
}
```

RM: remove a file. Note that this operation cannot remove a directory.

3. What Problems I Met in This Assignment and What Is My Solution?

a. When doing fs_gsys(LS_S), if there are several files with the same size, then first create first print. How to implement this?

In my FCB structure, the data field “time” stands for modified time. It appeared that to implement “first create first print”, I had to record a creation time for each file. This was straight-forward, but would take additional space. In particular, in my bonus program, where every byte of the FCB structure was taken advantage of, there was essentially no space for creation time. In fact, the way of creating, updating, and removing FCB has already assured that the order of the FCB entries in the volume would be exactly the same as the order of file creation.

Therefore, if I choose selection sort, when there are two consecutive FCBs with the same file size, I keep their relative order. In this way, “first create first print” is automatically guaranteed.

b. How to design the structure of the File-control Blocks?

Since a FCB can occupy at most 32 bytes of memory, the design of the data fields become challenging. First, the filename must occupy 20 bytes. Second, since the address of a file is in terms of blocks, and there are in total 32,768 storage blocks, 16 bits are sufficient to represent the address of any file. Third, the size of a file is at most 1,024 bytes, so 16 bits are enough. Fourth, I wish to allocate as much space as possible to the modified time data field. So, I give it 4 bytes. Fifth, since there are at most 1,024 files / directories, 16 bits are sufficient to represent the index of the parent of a file / directory. Sixth, given that the maximum depth of the tree structure is 3, 8 bits are used to record the depth. Last, there are only two types of object: file and directory. So, 8 bits are more than enough for type.

```
struct FCB { // file-control block: 32 bytes
    char filename[20]; // filename: 20 bytes
    uint16_t address = 0xFFFF; // block address: 2 bytes
    int16_t size = 0; // size: 2 bytes
    uint32_t time = 0; // modified time: 4 bytes
    int16_t parent = -1; // parent fcb number: 2 bytes
    int8_t depth = 0; // depth: 1 byte
    int8_t type = 0; // type: 1 byte
};
// byte 0-19: filename
// byte 20-21: address
// byte 22-23: size
// byte 24-27: modified time
// byte 28-29: parent
// byte 30: depth
// byte 31: type
```

c. How to compact FCB volume without messing up the links of the tree structure?

According to my design of FCB, the tree structure is maintained by a set of single links. However, when compacting the FCB volume, all FCBs after the fragment are moved by 1 FCB (32 bytes) forward. That is, their indexes are changed. Hence, the parent data fields of the FCBs whose parent FCBs are moved by the compaction should be updated. Besides, the children FCBs of the removed FCB become dangling. So, update their parent index to indicate that they should be removed in the future. Here, I choose -2 to represent dangling.

```
// compact the FCBs
__device__ void compact_FCBs(FileSystem *fs, int fcb_num) {
    // update all FCB parents
    for (int i = 0; i < fs->file_count; i++) {
        int parent = read_FCB_parent(fs, i);
        // change all FCB parents which are equal to fcb_num to -2 (dangling)
        if (parent == fcb_num)
            write_FCB_parent(fs, i, -2);
        // decrease all FCB parents which are greater than fcb_num by 1
        else if (parent > fcb_num)
            write_FCB_parent(fs, i, parent - 1);
    }
```

d. How to implement RM_RF? That is, how to remove a directory and all its subdirectories and files?

I designed a function called remove_file_recursively(). In one iteration, this function removes one file / directory and marks all its children as dangling. Then, it scans through the FCBs to find a dangling file / directory. If found, call itself to remove that file / directory and return. Otherwise, all the contents of the root subdirectory are removed.

```
// remove a directory and its content
__device__ void remove_file_recursively(FileSystem *fs, int fcb_num, char *s) {
    // update the file count
    fs->file_count--;
    // if this is a file
    if (read_FCB_type(fs, fcb_num) == FILE_TYPE) {
        // get the file size and number of blocks occupied
        int file_size = read_FCB_size(fs, fcb_num);
        int block_occupied = (file_size - 1) / fs->STORAGE_BLOCK_SIZE + 1;
        // get the file address
        int file_address = read_FCB_address(fs, fcb_num);
        // compact the volume
        compact_files(fs, file_address, block_occupied);
        // update the bit map
        update_bitmap(fs);
    }
    // update the parent's FCB size
    int file_parent = read_FCB_parent(fs, fcb_num);
    update_dir_size(fs, file_parent, -get_length(s));
    // compact the FCBs
    compact_FCBs(fs, fcb_num);
    // remove the next file
    for (int i = 0; i < fs->file_count; i++) {
        if (read_FCB_parent(fs, i) == -2) {
            remove_file_recursively(fs, i, s);
            return;
        }
    }
}
```

e. How to avoid deleting a file using RM_RF or deleting a directory using RM?

I add an additional parameter type to the function find_filename() as follows:

__device__ int find_filename(FileSystem *fs, const char *filename, uint8_t type)

This distinguishes a file from a folder when doing RM or RM_RF.

f. How to list out the files without using too much memory space? What algorithm should be used to sort the array of FCBs?

Although O(NlogN) sorting algorithms such as Merge Sort and Quick

Sort are efficient in terms of running time, their space complexity is relatively high. In CUDA, memory is very limited. Thus, I have chosen Selection Sort to lower memory occupation.

Since the CUDA memory is very limited, we cannot declare something like FCB[1024] on the stack or in the heap. To save temporary memory usage, I implement a modified version of Selection Sort. Briefly speaking, I print as I sort, without storing the whole array. In this way, the temporary memory usage is minimized.

```
// sort by modified time
if (op == LS_D) {
    printf("===sort by modified time===\n");
    int index;
    int current_mod_time;
    int max_mod_time;
    for (int i = 0; i < fs->file_count; i++) {
        max_mod_time = -1;
        for (int j = 0; j < fs->file_count; j++) {
            if (!printed[j]) {
                current_mod_time = read_FCB_mod_time(fs, j);
                if (current_mod_time > max_mod_time) {
                    max_mod_time = current_mod_time;
                    index = j;
                }
            }
        }
        read_FCB_filename(fs, index, filename);
        printf("%s\n", filename);
        printed[index] = true;
    }
}
```

g. How to indicate an invalid file address?

Given that fp is of type u32 and there are in total 32,768 storage blocks, I use 0x80000000 to represent an invalid address because it is naturally

out of boundary.

h. How to indicate an invalid file / directory / FCB index?

I use -1.

i. What should be the FCB address of a directory?

I use 0x8000.

4. The Steps to Execute My Program.

a. Open CSC3150_A4\CSC3150_A4.vcxproj using Visual Studio 2017

b. Do NOT change the Target Platform Version or the Toolkit.

c. Right click on and compile the cuda files (.cu). (Or use Ctrl+F7).

d. Press Ctrl+F5 to run the program.

5. Screenshot of My Program Output.

a. 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
```

b. 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
```

c. Test Case 3

```
===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
===sort by file size===
~ABCDEFGHIJKLM 1024
}ABCDEFGHIJKLM 1023
|ABCDEFGHIJKLM 1022
{ABCDEFGHIJKLM 1021
zABCDEFGHIJKLM 1020
yABCDEFGHIJKLM 1019
xABCDEFGHIJKLM 1018
wABCDEFGHIJKLM 1017
vABCDEFGHIJKLM 1016
uABCDEFGHIJKLM 1015
tABCDEFGHIJKLM 1014
sABCDEFGHIJKLM 1013
rABCDEFGHIJKLM 1012
qABCDEFGHIJKLM 1011
pABCDEFGHIJKLM 1010
```

```
oABCDEFGHIJKLM 1009
nABCDEFGHIJKLM 1008
mABCDEFGHIJKLM 1007
lABCDEFGHIJKLM 1006
kABCDEFGHIJKLM 1005
jABCDEFGHIJKLM 1004
iABCDEFGHIJKLM 1003
hABCDEFGHIJKLM 1002
gABCDEFGHIJKLM 1001
fABCDEFGHIJKLM 1000
eABCDEFGHIJKLM 999
dABCDEFGHIJKLM 998
cABCDEFGHIJKLM 997
bABCDEFGHIJKLM 996
aABCDEFGHIJKLM 995
`ABCDEFGHIJKLM 994
_ABCDEFGHIJKLM 993
^ABCDEFGHIJKLM 992
]ABCDEFGHIJKLM 991
\ABCDEFGHIJKLM 990
[ABCDEFGHIJKLM 989
ZABCDEFGHIJKLM 988
YABCDEFGHIJKLM 987
XABCDEFGHIJKLM 986
WABCDEFGHIJKLM 985
VABCDEFGHIJKLM 984
UABCDEFGHIJKLM 983
TABCDEFGHIJKLM 982
SABCDEFGHIJKLM 981
RABCDEFGHIJKLM 980
QABCDEFGHIJKLM 979
PABCDEFGHIJKLM 978
OABCDEFGHIJKLM 977
NABCDEFGHIJKLM 976
MABCDEFGHIJKLM 975
LABCDEFGHIJKLM 974
KABCDEFGHIJKLM 973
JABCDEFGHIJKLM 972
IABCDEFGHIJKLM 971
HABCDEFGHIJKLM 970
GABCDEFGHIJKLM 969
FABCDEFGHIJKLM 968
EABCDEFGHIJKLM 967
DABCDEFGHIJKLM 966
```

```
CABCDEFGHIJKLM 965
BABCDEFGHIJKLM 964
AABCDEFGHIJKLM 963
@ABCDEFGHIJKLM 962
?ABCDEFGHIJKLM 961
>ABCDEFGHIJKLM 960
=ABCDEFGHIJKLM 959
<ABCDEFGHIJKLM 958
;ABCDEFGHIJKLM 957
:ABCDEFGHIJKLM 956
9ABCDEFGHIJKLM 955
8ABCDEFGHIJKLM 954
7ABCDEFGHIJKLM 953
6ABCDEFGHIJKLM 952
5ABCDEFGHIJKLM 951
4ABCDEFGHIJKLM 950
3ABCDEFGHIJKLM 949
2ABCDEFGHIJKLM 948
~ABCDEFGHIJKL 947
}ABCDEFGHIJKL 946
|ABCDEFGHIJKL 945
{ABCDEFGHIJKL 944
zABCDEFGHIJKL 943
yABCDEFGHIJKL 942
xABCDEFGHIJKL 941
wABCDEFGHIJKL 940
vABCDEFGHIJKL 939
uABCDEFGHIJKL 938
tABCDEFGHIJKL 937
sABCDEFGHIJKL 936
rABCDEFGHIJKL 935
qABCDEFGHIJKL 934
pABCDEFGHIJKL 933
oABCDEFGHIJKL 932
nABCDEFGHIJKL 931
mABCDEFGHIJKL 930
lABCDEFGHIJKL 929
kABCDEFGHIJKL 928
jABCDEFGHIJKL 927
iABCDEFGHIJKL 926
hABCDEFGHIJKL 925
gABCDEFGHIJKL 924
fABCDEFGHIJKL 923
eABCDEFGHIJKL 922
```

```
dABCDEFGHIJKL 921
cABCDEFGHIJKL 920
bABCDEFGHIJKL 919
aABCDEFGHIJKL 918
`ABCDEFGHIJKL 917
_ABCDEFGHIJKL 916
^ABCDEFGHIJKL 915
]ABCDEFGHIJKL 914
\ABCDEFGHIJKL 913
[ABCDEFGHIJKL 912
ZABCDEFGHIJKL 911
YABCDEFGHIJKL 910
XABCDEFGHIJKL 909
WABCDEFGHIJKL 908
VABCDEFGHIJKL 907
UABCDEFGHIJKL 906
TABCDEFGHIJKL 905
SABCDEFGHIJKL 904
RABCDEFGHIJKL 903
QABCDEFGHIJKL 902
PABCDEFGHIJKL 901
OABCDEFGHIJKL 900
NABCDEFGHIJKL 899
MABCDEFGHIJKL 898
LABCDEFGHIJKL 897
KABCDEFGHIJKL 896
JABCDEFGHIJKL 895
IABCDEFGHIJKL 894
HABCDEFGHIJKL 893
GABCDEFGHIJKL 892
FABCDEFGHIJKL 891
EABCDEFGHIJKL 890
DABCDEFGHIJKL 889
CABCDEFGHIJKL 888
BABCDEFGHIJKL 887
AABCDEFGHIJKL 886
@ABCDEFGHIJKL 885
?ABCDEFGHIJKL 884
>ABCDEFGHIJKL 883
=ABCDEFGHIJKL 882
<ABCDEFGHIJKL 881
;ABCDEFGHIJKL 880
:ABCDEFGHIJKL 879
9ABCDEFGHIJKL 878
```

```
8ABCDEFGHIJKL 877
7ABCDEFGHIJKL 876
6ABCDEFGHIJKL 875
5ABCDEFGHIJKL 874
4ABCDEFGHIJKL 873
3ABCDEFGHIJKL 872
2ABCDEFGHIJKL 871
~ABCDEFGHIJK 870
}ABCDEFGHIJK 869
|ABCDEFGHIJK 868
{ABCDEFGHIJK 867
zABCDEFGHIJK 866
yABCDEFGHIJK 865
xABCDEFGHIJK 864
wABCDEFGHIJK 863
vABCDEFGHIJK 862
uABCDEFGHIJK 861
tABCDEFGHIJK 860
sABCDEFGHIJK 859
rABCDEFGHIJK 858
qABCDEFGHIJK 857
pABCDEFGHIJK 856
oABCDEFGHIJK 855
nABCDEFGHIJK 854
mABCDEFGHIJK 853
lABCDEFGHIJK 852
kABCDEFGHIJK 851
jABCDEFGHIJK 850
iABCDEFGHIJK 849
hABCDEFGHIJK 848
gABCDEFGHIJK 847
fABCDEFGHIJK 846
eABCDEFGHIJK 845
dABCDEFGHIJK 844
cABCDEFGHIJK 843
bABCDEFGHIJK 842
aABCDEFGHIJK 841
`ABCDEFGHIJK 840
_ABCDEFGHIJK 839
^ABCDEFGHIJK 838
]ABCDEFGHIJK 837
\ABCDEFGHIJK 836
[ABCDEFGHIJK 835
ZABCDEFGHIJK 834
```

```
YABCDEFGHIJK 833
XABCDEFGHIJK 832
WABCDEFGHIJK 831
VABCDEFGHIJK 830
UABCDEFGHIJK 829
TABCDEFGHIJK 828
SABCDEFGHIJK 827
RABCDEFGHIJK 826
QABCDEFGHIJK 825
PABCDEFGHIJK 824
OABCDEFGHIJK 823
NABCDEFGHIJK 822
MABCDEFGHIJK 821
LABCDEFGHIJK 820
KABCDEFGHIJK 819
JABCDEFGHIJK 818
IABCDEFGHIJK 817
HABCDEFGHIJK 816
GABCDEFGHIJK 815
FABCDEFGHIJK 814
EABCDEFGHIJK 813
DABCDEFGHIJK 812
CABCDEFGHIJK 811
BABCDEFGHIJK 810
AABCDEFGHIJK 809
@ABCDEFGHIJK 808
?ABCDEFGHIJK 807
>ABCDEFGHIJK 806
=ABCDEFGHIJK 805
<ABCDEFGHIJK 804
;ABCDEFGHIJK 803
:ABCDEFGHIJK 802
9ABCDEFGHIJK 801
8ABCDEFGHIJK 800
7ABCDEFGHIJK 799
6ABCDEFGHIJK 798
5ABCDEFGHIJK 797
4ABCDEFGHIJK 796
3ABCDEFGHIJK 795
2ABCDEFGHIJK 794
~ABCDEFGHIJ 793
}ABCDEFGHIJ 792
|ABCDEFGHIJ 791
{ABCDEFGHIJ 790
```

```
zABCDEFGHIJ 789
yABCDEFGHIJ 788
xABCDEFGHIJ 787
wABCDEFGHIJ 786
vABCDEFGHIJ 785
uABCDEFGHIJ 784
tABCDEFGHIJ 783
sABCDEFGHIJ 782
rABCDEFGHIJ 781
qABCDEFGHIJ 780
pABCDEFGHIJ 779
oABCDEFGHIJ 778
nABCDEFGHIJ 777
mABCDEFGHIJ 776
lABCDEFGHIJ 775
kABCDEFGHIJ 774
jABCDEFGHIJ 773
iABCDEFGHIJ 772
hABCDEFGHIJ 771
gABCDEFGHIJ 770
fABCDEFGHIJ 769
eABCDEFGHIJ 768
dABCDEFGHIJ 767
cABCDEFGHIJ 766
bABCDEFGHIJ 765
aABCDEFGHIJ 764
`ABCDEFGHIJ 763
_ABCDEFGHIJ 762
^ABCDEFGHIJ 761
]ABCDEFGHIJ 760
\ABCDEFGHIJ 759
[ABCDEFGHIJ 758
ZABCDEFGHIJ 757
YABCDEFGHIJ 756
XABCDEFGHIJ 755
WABCDEFGHIJ 754
VABCDEFGHIJ 753
UABCDEFGHIJ 752
TABCDEFGHIJ 751
SABCDEFGHIJ 750
RABCDEFGHIJ 749
QABCDEFGHIJ 748
PABCDEFGHIJ 747
OABCDEFGHIJ 746
```

```
NABCDEFGHIJ 745
MABCDEFGHIJ 744
LABCDEFGHIJ 743
KABCDEFGHIJ 742
JABCDEFGHIJ 741
IABCDEFGHIJ 740
HABCDEFGHIJ 739
GABCDEFGHIJ 738
FABCDEFGHIJ 737
EABCDEFGHIJ 736
DABCDEFGHIJ 735
CABCDEFGHIJ 734
BABCDEFGHIJ 733
AABCDEFGHIJ 732
@ABCDEFGHIJ 731
?ABCDEFGHIJ 730
>ABCDEFGHIJ 729
=ABCDEFGHIJ 728
<ABCDEFGHIJ 727
;ABCDEFGHIJ 726
:ABCDEFGHIJ 725
9ABCDEFGHIJ 724
8ABCDEFGHIJ 723
7ABCDEFGHIJ 722
6ABCDEFGHIJ 721
5ABCDEFGHIJ 720
4ABCDEFGHIJ 719
3ABCDEFGHIJ 718
2ABCDEFGHIJ 717
~ABCDEFGHI 716
}ABCDEFGHI 715
|ABCDEFGHI 714
{ABCDEFGHI 713
zABCDEFGHI 712
yABCDEFGHI 711
xABCDEFGHI 710
wABCDEFGHI 709
vABCDEFGHI 708
uABCDEFGHI 707
tABCDEFGHI 706
sABCDEFGHI 705
rABCDEFGHI 704
qABCDEFGHI 703
pABCDEFGHI 702
```

```
oABCDEFGHI 701
nABCDEFGHI 700
mABCDEFGHI 699
lABCDEFGHI 698
kABCDEFGHI 697
jABCDEFGHI 696
iABCDEFGHI 695
hABCDEFGHI 694
gABCDEFGHI 693
fABCDEFGHI 692
eABCDEFGHI 691
dABCDEFGHI 690
cABCDEFGHI 689
bABCDEFGHI 688
aABCDEFGHI 687
`ABCDEFGHI 686
_ABCDEFGHI 685
^ABCDEFGHI 684
]ABCDEFGHI 683
\ABCDEFGHI 682
[ABCDEFGHI 681
ZABCDEFGHI 680
YABCDEFGHI 679
XABCDEFGHI 678
WABCDEFGHI 677
VABCDEFGHI 676
UABCDEFGHI 675
TABCDEFGHI 674
SABCDEFGHI 673
RABCDEFGHI 672
QABCDEFGHI 671
PABCDEFGHI 670
OABCDEFGHI 669
NABCDEFGHI 668
MABCDEFGHI 667
LABCDEFGHI 666
KABCDEFGHI 665
JABCDEFGHI 664
IABCDEFGHI 663
HABCDEFGHI 662
GABCDEFGHI 661
FABCDEFGHI 660
EABCDEFGHI 659
DABCDEFGHI 658
```

```
CABCDEFGHI 657
BABCDEFGHI 656
AABCDEFGHI 655
@ABCDEFGHI 654
?ABCDEFGHI 653
>ABCDEFGHI 652
=ABCDEFGHI 651
<ABCDEFGHI 650
;ABCDEFGHI 649
:ABCDEFGHI 648
9ABCDEFGHI 647
8ABCDEFGHI 646
7ABCDEFGHI 645
6ABCDEFGHI 644
5ABCDEFGHI 643
4ABCDEFGHI 642
3ABCDEFGHI 641
2ABCDEFGHI 640
~ABCDEFGH 639
}ABCDEFGH 638
|ABCDEFGH 637
{ABCDEFGH 636
zABCDEFGH 635
yABCDEFGH 634
xABCDEFGH 633
wABCDEFGH 632
vABCDEFGH 631
uABCDEFGH 630
tABCDEFGH 629
sABCDEFGH 628
rABCDEFGH 627
qABCDEFGH 626
pABCDEFGH 625
oABCDEFGH 624
nABCDEFGH 623
mABCDEFGH 622
lABCDEFGH 621
kABCDEFGH 620
jABCDEFGH 619
iABCDEFGH 618
hABCDEFGH 617
gABCDEFGH 616
fABCDEFGH 615
eABCDEFGH 614
```

```
dABCDEFGH 613
cABCDEFGH 612
bABCDEFGH 611
aABCDEFGH 610
`ABCDEFGH 609
_ABCDEFGH 608
^ABCDEFGH 607
]ABCDEFGH 606
\ABCDEFGH 605
[ABCDEFGH 604
ZABCDEFGH 603
YABCDEFGH 602
XABCDEFGH 601
WABCDEFGH 600
VABCDEFGH 599
UABCDEFGH 598
TABCDEFGH 597
SABCDEFGH 596
RABCDEFGH 595
QABCDEFGH 594
PABCDEFGH 593
OABCDEFGH 592
NABCDEFGH 591
MABCDEFGH 590
LABCDEFGH 589
KABCDEFGH 588
JABCDEFGH 587
IABCDEFGH 586
HABCDEFGH 585
GABCDEFGH 584
FABCDEFGH 583
EABCDEFGH 582
DABCDEFGH 581
CABCDEFGH 580
BABCDEFGH 579
AABCDEFGH 578
@ABCDEFGH 577
?ABCDEFGH 576
>ABCDEFGH 575
=ABCDEFGH 574
<ABCDEFGH 573
;ABCDEFGH 572
:ABCDEFGH 571
9ABCDEFGH 570
```

```
8ABCDEFGH 569
7ABCDEFGH 568
6ABCDEFGH 567
5ABCDEFGH 566
4ABCDEFGH 565
3ABCDEFGH 564
2ABCDEFGH 563
~ABCDEFG 562
}ABCDEFG 561
|ABCDEFG 560
{ABCDEFG 559
zABCDEFG 558
yABCDEFG 557
xABCDEFG 556
wABCDEFG 555
vABCDEFG 554
uABCDEFG 553
tABCDEFG 552
sABCDEFG 551
rABCDEFG 550
qABCDEFG 549
pABCDEFG 548
oABCDEFG 547
nABCDEFG 546
mABCDEFG 545
lABCDEFG 544
kABCDEFG 543
jABCDEFG 542
iABCDEFG 541
hABCDEFG 540
gABCDEFG 539
fABCDEFG 538
eABCDEFG 537
dABCDEFG 536
cABCDEFG 535
bABCDEFG 534
aABCDEFG 533
`ABCDEFG 532
_ABCDEFG 531
^ABCDEFG 530
]ABCDEFG 529
\ABCDEFG 528
[ABCDEFG 527
ZABCDEFG 526
```

```
YABCDEFG 525
XABCDEFG 524
WABCDEFG 523
VABCDEFG 522
UABCDEFG 521
TABCDEFG 520
SABCDEFG 519
RABCDEFG 518
QABCDEFG 517
PABCDEFG 516
OABCDEFG 515
NABCDEFG 514
MABCDEFG 513
LABCDEFG 512
KABCDEFG 511
JABCDEFG 510
IABCDEFG 509
HABCDEFG 508
GABCDEFG 507
FABCDEFG 506
EABCDEFG 505
DABCDEFG 504
CABCDEFG 503
BABCDEFG 502
AABCDEFG 501
@ABCDEFG 500
?ABCDEFG 499
>ABCDEFG 498
=ABCDEFG 497
<ABCDEFG 496
;ABCDEFG 495
:ABCDEFG 494
9ABCDEFG 493
8ABCDEFG 492
7ABCDEFG 491
6ABCDEFG 490
5ABCDEFG 489
4ABCDEFG 488
3ABCDEFG 487
2ABCDEFG 486
~ABCDEF 485
}ABCDEF 484
|ABCDEF 483
{ABCDEF 482
```

```
zABCDEF 481
yABCDEF 480
xABCDEF 479
wABCDEF 478
vABCDEF 477
uABCDEF 476
tABCDEF 475
sABCDEF 474
rABCDEF 473
qABCDEF 472
pABCDEF 471
oABCDEF 470
nABCDEF 469
mABCDEF 468
lABCDEF 467
kABCDEF 466
jABCDEF 465
iABCDEF 464
hABCDEF 463
gABCDEF 462
fABCDEF 461
eABCDEF 460
dABCDEF 459
cABCDEF 458
bABCDEF 457
aABCDEF 456
`ABCDEF 455
_ABCDEF 454
^ABCDEF 453
]ABCDEF 452
\ABCDEF 451
[ABCDEF 450
ZABCDEF 449
YABCDEF 448
XABCDEF 447
WABCDEF 446
VABCDEF 445
UABCDEF 444
TABCDEF 443
SABCDEF 442
RABCDEF 441
QABCDEF 440
PABCDEF 439
OABCDEF 438
```

```
NABCDEF 437
MABCDEF 436
LABCDEF 435
KABCDEF 434
JABCDEF 433
IABCDEF 432
HABCDEF 431
GABCDEF 430
FABCDEF 429
EABCDEF 428
DABCDEF 427
CABCDEF 426
BABCDEF 425
AABCDEF 424
@ABCDEF 423
?ABCDEF 422
>ABCDEF 421
=ABCDEF 420
<ABCDEF 419
;ABCDEF 418
:ABCDEF 417
9ABCDEF 416
8ABCDEF 415
7ABCDEF 414
6ABCDEF 413
5ABCDEF 412
4ABCDEF 411
3ABCDEF 410
2ABCDEF 409
~ABCDE 408
}ABCDE 407
|ABCDE 406
{ABCDE 405
zABCDE 404
yABCDE 403
xABCDE 402
wABCDE 401
vABCDE 400
uABCDE 399
tABCDE 398
sABCDE 397
rABCDE 396
qABCDE 395
pABCDE 394
```

```
oABCDE 393
nABCDE 392
mABCDE 391
lABCDE 390
kABCDE 389
jABCDE 388
iABCDE 387
hABCDE 386
gABCDE 385
fABCDE 384
eABCDE 383
dABCDE 382
cABCDE 381
bABCDE 380
aABCDE 379
`ABCDE 378
_ABCDE 377
^ABCDE 376
]ABCDE 375
\ABCDE 374
[ABCDE 373
ZABCDE 372
YABCDE 371
XABCDE 370
WABCDE 369
VABCDE 368
UABCDE 367
TABCDE 366
SABCDE 365
RABCDE 364
QABCDE 363
PABCDE 362
OABCDE 361
NABCDE 360
MABCDE 359
LABCDE 358
KABCDE 357
JABCDE 356
IABCDE 355
HABCDE 354
GABCDE 353
FABCDE 352
EABCDE 351
DABCDE 350
```

```
CABCDE 349
BABCDE 348
AABCDE 347
@ABCDE 346
?ABCDE 345
>ABCDE 344
=ABCDE 343
<ABCDE 342
;ABCDE 341
:ABCDE 340
9ABCDE 339
8ABCDE 338
7ABCDE 337
6ABCDE 336
5ABCDE 335
4ABCDE 334
3ABCDE 333
2ABCDE 332
~ABCD 331
}ABCD 330
|ABCD 329
{ABCD 328
zABCD 327
yABCD 326
xABCD 325
wABCD 324
vABCD 323
uABCD 322
tABCD 321
sABCD 320
rABCD 319
qABCD 318
pABCD 317
oABCD 316
nABCD 315
mABCD 314
lABCD 313
kABCD 312
jABCD 311
iABCD 310
hABCD 309
gABCD 308
fABCD 307
eABCD 306
```

```
dABCD 305
cABCD 304
bABCD 303
aABCD 302
`ABCD 301
_ABCD 300
^ABCD 299
]ABCD 298
\ABCD 297
[ABCD 296
ZABCD 295
YABCD 294
XABCD 293
WABCD 292
VABCD 291
UABCD 290
TABCD 289
SABCD 288
RABCD 287
QABCD 286
PABCD 285
OABCD 284
NABCD 283
MABCD 282
LABCD 281
KABCD 280
JABCD 279
IABCD 278
HABCD 277
GABCD 276
FABCD 275
EABCD 274
DABCD 273
CABCD 272
BABCD 271
AABCD 270
@ABCD 269
?ABCD 268
>ABCD 267
=ABCD 266
<ABCD 265
;ABCD 264
:ABCD 263
9ABCD 262
```

```
8ABCD 261
7ABCD 260
6ABCD 259
5ABCD 258
4ABCD 257
3ABCD 256
2ABCD 255
~ABC 254
}ABC 253
|ABC 252
{ABC 251
zABC 250
yABC 249
xABC 248
wABC 247
vABC 246
uABC 245
tABC 244
sABC 243
rABC 242
qABC 241
pABC 240
oABC 239
nABC 238
mABC 237
lABC 236
kABC 235
jABC 234
iABC 233
hABC 232
gABC 231
fABC 230
eABC 229
dABC 228
cABC 227
bABC 226
aABC 225
`ABC 224
_ABC 223
^ABC 222
]ABC 221
\ABC 220
[ABC 219
ZABC 218
```

```
YABC 217
XABC 216
WABC 215
VABC 214
UABC 213
TABC 212
SABC 211
RABC 210
QABC 209
PABC 208
OABC 207
NABC 206
MABC 205
LABC 204
KABC 203
JABC 202
IABC 201
HABC 200
GABC 199
FABC 198
EABC 197
DABC 196
CABC 195
BABC 194
AABC 193
@ABC 192
?ABC 191
>ABC 190
=ABC 189
<ABC 188
;ABC 187
:ABC 186
9ABC 185
8ABC 184
7ABC 183
6ABC 182
5ABC 181
4ABC 180
3ABC 179
2ABC 178
~AB 177
}AB 176
|AB 175
{AB 174
```

```
zAB 173
yAB 172
xAB 171
wAB 170
vAB 169
uAB 168
tAB 167
sAB 166
rAB 165
qAB 164
pAB 163
oAB 162
nAB 161
mAB 160
lAB 159
kAB 158
jAB 157
iAB 156
hAB 155
gAB 154
fAB 153
eAB 152
dAB 151
cAB 150
bAB 149
aAB 148
`AB 147
_AB 146
^AB 145
]AB 144
\AB 143
[AB 142
ZAB 141
YAB 140
XAB 139
WAB 138
VAB 137
UAB 136
TAB 135
SAB 134
RAB 133
QAB 132
PAB 131
OAB 130
```

```
NAB 129
MAB 128
LAB 127
KAB 126
JAB 125
IAB 124
HAB 123
GAB 122
FAB 121
EAB 120
DAB 119
CAB 118
BAB 117
AAB 116
@AB 115
?AB 114
>AB 113
=AB 112
<AB 111
;AB 110
:AB 109
9AB 108
8AB 107
7AB 106
6AB 105
5AB 104
4AB 103
3AB 102
2AB 101
~A 100
}A 99
|A 98
{A 97
zA 96
yA 95
xA 94
wA 93
vA 92
uA 91
tA 90
sA 89
rA 88
qA 87
pA 86
```

```
oA 85
nA 84
mA 83
lA 82
kA 81
jA 80
iA 79
hA 78
gA 77
fA 76
eA 75
dA 74
cA 73
bA 72
aA 71
`A 70
_A 69
^A 68
]A 67
\A 66
[A 65
ZA 64
YA 63
XA 62
WA 61
VA 60
UA 59
TA 58
SA 57
RA 56
QA 55
PA 54
OA 53
NA 52
MA 51
LA 50
KA 49
JA 48
IA 47
HA 46
GA 45
FA 44
EA 43
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
===sort by file size===
EA 1024
~ABCDEFGHIJKLM 1024
aa 1024
bb 1024
cc 1024
dd 1024
ee 1024
ff 1024
gg 1024
hh 1024
ii 1024
jj 1024
kk 1024
ll 1024
mm 1024
nn 1024
oo 1024
pp 1024
qq 1024
```

```
}ABCDEFGHIJKLM 1023
|ABCDEFGHIJKLM 1022
{ABCDEFGHIJKLM 1021
zABCDEFGHIJKLM 1020
yABCDEFGHIJKLM 1019
xABCDEFGHIJKLM 1018
wABCDEFGHIJKLM 1017
vABCDEFGHIJKLM 1016
uABCDEFGHIJKLM 1015
tABCDEFGHIJKLM 1014
sABCDEFGHIJKLM 1013
rABCDEFGHIJKLM 1012
qABCDEFGHIJKLM 1011
pABCDEFGHIJKLM 1010
oABCDEFGHIJKLM 1009
nABCDEFGHIJKLM 1008
mABCDEFGHIJKLM 1007
lABCDEFGHIJKLM 1006
kABCDEFGHIJKLM 1005
jABCDEFGHIJKLM 1004
iABCDEFGHIJKLM 1003
hABCDEFGHIJKLM 1002
gABCDEFGHIJKLM 1001
fABCDEFGHIJKLM 1000
eABCDEFGHIJKLM 999
dABCDEFGHIJKLM 998
cABCDEFGHIJKLM 997
bABCDEFGHIJKLM 996
aABCDEFGHIJKLM 995
`ABCDEFGHIJKLM 994
_ABCDEFGHIJKLM 993
^ABCDEFGHIJKLM 992
]ABCDEFGHIJKLM 991
\ABCDEFGHIJKLM 990
[ABCDEFGHIJKLM 989
ZABCDEFGHIJKLM 988
YABCDEFGHIJKLM 987
XABCDEFGHIJKLM 986
WABCDEFGHIJKLM 985
VABCDEFGHIJKLM 984
UABCDEFGHIJKLM 983
TABCDEFGHIJKLM 982
SABCDEFGHIJKLM 981
RABCDEFGHIJKLM 980
```

```
QABCDEFGHIJKLM 979
PABCDEFGHIJKLM 978
OABCDEFGHIJKLM 977
NABCDEFGHIJKLM 976
MABCDEFGHIJKLM 975
LABCDEFGHIJKLM 974
KABCDEFGHIJKLM 973
JABCDEFGHIJKLM 972
IABCDEFGHIJKLM 971
HABCDEFGHIJKLM 970
GABCDEFGHIJKLM 969
FABCDEFGHIJKLM 968
EABCDEFGHIJKLM 967
DABCDEFGHIJKLM 966
CABCDEFGHIJKLM 965
BABCDEFGHIJKLM 964
AABCDEFGHIJKLM 963
@ABCDEFGHIJKLM 962
?ABCDEFGHIJKLM 961
>ABCDEFGHIJKLM 960
=ABCDEFGHIJKLM 959
<ABCDEFGHIJKLM 958
;ABCDEFGHIJKLM 957
:ABCDEFGHIJKLM 956
9ABCDEFGHIJKLM 955
8ABCDEFGHIJKLM 954
7ABCDEFGHIJKLM 953
6ABCDEFGHIJKLM 952
5ABCDEFGHIJKLM 951
4ABCDEFGHIJKLM 950
3ABCDEFGHIJKLM 949
2ABCDEFGHIJKLM 948
~ABCDEFGHIJKL 947
}ABCDEFGHIJKL 946
|ABCDEFGHIJKL 945
{ABCDEFGHIJKL 944
zABCDEFGHIJKL 943
yABCDEFGHIJKL 942
xABCDEFGHIJKL 941
wABCDEFGHIJKL 940
vABCDEFGHIJKL 939
uABCDEFGHIJKL 938
tABCDEFGHIJKL 937
sABCDEFGHIJKL 936
```

```
rABCDEFGHIJKL 935
qABCDEFGHIJKL 934
pABCDEFGHIJKL 933
oABCDEFGHIJKL 932
nABCDEFGHIJKL 931
mABCDEFGHIJKL 930
lABCDEFGHIJKL 929
kABCDEFGHIJKL 928
jABCDEFGHIJKL 927
iABCDEFGHIJKL 926
hABCDEFGHIJKL 925
gABCDEFGHIJKL 924
fABCDEFGHIJKL 923
eABCDEFGHIJKL 922
dABCDEFGHIJKL 921
cABCDEFGHIJKL 920
bABCDEFGHIJKL 919
aABCDEFGHIJKL 918
`ABCDEFGHIJKL 917
_ABCDEFGHIJKL 916
^ABCDEFGHIJKL 915
]ABCDEFGHIJKL 914
\ABCDEFGHIJKL 913
[ABCDEFGHIJKL 912
ZABCDEFGHIJKL 911
YABCDEFGHIJKL 910
XABCDEFGHIJKL 909
WABCDEFGHIJKL 908
VABCDEFGHIJKL 907
UABCDEFGHIJKL 906
TABCDEFGHIJKL 905
SABCDEFGHIJKL 904
RABCDEFGHIJKL 903
QABCDEFGHIJKL 902
PABCDEFGHIJKL 901
OABCDEFGHIJKL 900
NABCDEFGHIJKL 899
MABCDEFGHIJKL 898
LABCDEFGHIJKL 897
KABCDEFGHIJKL 896
JABCDEFGHIJKL 895
IABCDEFGHIJKL 894
HABCDEFGHIJKL 893
GABCDEFGHIJKL 892
```

```
FABCDEFGHIJKL 891
EABCDEFGHIJKL 890
DABCDEFGHIJKL 889
CABCDEFGHIJKL 888
BABCDEFGHIJKL 887
AABCDEFGHIJKL 886
@ABCDEFGHIJKL 885
?ABCDEFGHIJKL 884
>ABCDEFGHIJKL 883
=ABCDEFGHIJKL 882
<ABCDEFGHIJKL 881
;ABCDEFGHIJKL 880
:ABCDEFGHIJKL 879
9ABCDEFGHIJKL 878
8ABCDEFGHIJKL 877
7ABCDEFGHIJKL 876
6ABCDEFGHIJKL 875
5ABCDEFGHIJKL 874
4ABCDEFGHIJKL 873
3ABCDEFGHIJKL 872
2ABCDEFGHIJKL 871
~ABCDEFGHIJK 870
}ABCDEFGHIJK 869
|ABCDEFGHIJK 868
{ABCDEFGHIJK 867
zABCDEFGHIJK 866
yABCDEFGHIJK 865
xABCDEFGHIJK 864
wABCDEFGHIJK 863
vABCDEFGHIJK 862
uABCDEFGHIJK 861
tABCDEFGHIJK 860
sABCDEFGHIJK 859
rABCDEFGHIJK 858
qABCDEFGHIJK 857
pABCDEFGHIJK 856
oABCDEFGHIJK 855
nABCDEFGHIJK 854
mABCDEFGHIJK 853
lABCDEFGHIJK 852
kABCDEFGHIJK 851
jABCDEFGHIJK 850
iABCDEFGHIJK 849
hABCDEFGHIJK 848
```

```
gABCDEFGHIJK 847
fABCDEFGHIJK 846
eABCDEFGHIJK 845
dABCDEFGHIJK 844
cABCDEFGHIJK 843
bABCDEFGHIJK 842
aABCDEFGHIJK 841
`ABCDEFGHIJK 840
_ABCDEFGHIJK 839
^ABCDEFGHIJK 838
]ABCDEFGHIJK 837
\ABCDEFGHIJK 836
[ABCDEFGHIJK 835
ZABCDEFGHIJK 834
YABCDEFGHIJK 833
XABCDEFGHIJK 832
WABCDEFGHIJK 831
VABCDEFGHIJK 830
UABCDEFGHIJK 829
TABCDEFGHIJK 828
SABCDEFGHIJK 827
RABCDEFGHIJK 826
QABCDEFGHIJK 825
PABCDEFGHIJK 824
OABCDEFGHIJK 823
NABCDEFGHIJK 822
MABCDEFGHIJK 821
LABCDEFGHIJK 820
KABCDEFGHIJK 819
JABCDEFGHIJK 818
IABCDEFGHIJK 817
HABCDEFGHIJK 816
GABCDEFGHIJK 815
FABCDEFGHIJK 814
EABCDEFGHIJK 813
DABCDEFGHIJK 812
CABCDEFGHIJK 811
BABCDEFGHIJK 810
AABCDEFGHIJK 809
@ABCDEFGHIJK 808
?ABCDEFGHIJK 807
>ABCDEFGHIJK 806
=ABCDEFGHIJK 805
<ABCDEFGHIJK 804
```

```
;ABCDEFGHIJK 803
:ABCDEFGHIJK 802
9ABCDEFGHIJK 801
8ABCDEFGHIJK 800
7ABCDEFGHIJK 799
6ABCDEFGHIJK 798
5ABCDEFGHIJK 797
4ABCDEFGHIJK 796
3ABCDEFGHIJK 795
2ABCDEFGHIJK 794
~ABCDEFGHIJ 793
}ABCDEFGHIJ 792
|ABCDEFGHIJ 791
{ABCDEFGHIJ 790
zABCDEFGHIJ 789
yABCDEFGHIJ 788
xABCDEFGHIJ 787
wABCDEFGHIJ 786
vABCDEFGHIJ 785
uABCDEFGHIJ 784
tABCDEFGHIJ 783
sABCDEFGHIJ 782
rABCDEFGHIJ 781
qABCDEFGHIJ 780
pABCDEFGHIJ 779
oABCDEFGHIJ 778
nABCDEFGHIJ 777
mABCDEFGHIJ 776
lABCDEFGHIJ 775
kABCDEFGHIJ 774
jABCDEFGHIJ 773
iABCDEFGHIJ 772
hABCDEFGHIJ 771
gABCDEFGHIJ 770
fABCDEFGHIJ 769
eABCDEFGHIJ 768
dABCDEFGHIJ 767
cABCDEFGHIJ 766
bABCDEFGHIJ 765
aABCDEFGHIJ 764
`ABCDEFGHIJ 763
_ABCDEFGHIJ 762
^ABCDEFGHIJ 761
]ABCDEFGHIJ 760
```

```
\ABCDEFGHIJ 759
[ABCDEFGHIJ 758
ZABCDEFGHIJ 757
YABCDEFGHIJ 756
XABCDEFGHIJ 755
WABCDEFGHIJ 754
VABCDEFGHIJ 753
UABCDEFGHIJ 752
TABCDEFGHIJ 751
SABCDEFGHIJ 750
RABCDEFGHIJ 749
QABCDEFGHIJ 748
PABCDEFGHIJ 747
OABCDEFGHIJ 746
NABCDEFGHIJ 745
MABCDEFGHIJ 744
LABCDEFGHIJ 743
KABCDEFGHIJ 742
JABCDEFGHIJ 741
IABCDEFGHIJ 740
HABCDEFGHIJ 739
GABCDEFGHIJ 738
FABCDEFGHIJ 737
EABCDEFGHIJ 736
DABCDEFGHIJ 735
CABCDEFGHIJ 734
BABCDEFGHIJ 733
AABCDEFGHIJ 732
@ABCDEFGHIJ 731
?ABCDEFGHIJ 730
>ABCDEFGHIJ 729
=ABCDEFGHIJ 728
<ABCDEFGHIJ 727
;ABCDEFGHIJ 726
:ABCDEFGHIJ 725
9ABCDEFGHIJ 724
8ABCDEFGHIJ 723
7ABCDEFGHIJ 722
6ABCDEFGHIJ 721
5ABCDEFGHIJ 720
4ABCDEFGHIJ 719
3ABCDEFGHIJ 718
2ABCDEFGHIJ 717
~ABCDEFGHI 716
```

```
}ABCDEFGHI 715
|ABCDEFGHI 714
{ABCDEFGHI 713
zABCDEFGHI 712
yABCDEFGHI 711
xABCDEFGHI 710
wABCDEFGHI 709
vABCDEFGHI 708
uABCDEFGHI 707
tABCDEFGHI 706
sABCDEFGHI 705
rABCDEFGHI 704
qABCDEFGHI 703
pABCDEFGHI 702
oABCDEFGHI 701
nABCDEFGHI 700
mABCDEFGHI 699
lABCDEFGHI 698
kABCDEFGHI 697
jABCDEFGHI 696
iABCDEFGHI 695
hABCDEFGHI 694
gABCDEFGHI 693
fABCDEFGHI 692
eABCDEFGHI 691
dABCDEFGHI 690
cABCDEFGHI 689
bABCDEFGHI 688
aABCDEFGHI 687
`ABCDEFGHI 686
_ABCDEFGHI 685
^ABCDEFGHI 684
]ABCDEFGHI 683
\ABCDEFGHI 682
[ABCDEFGHI 681
ZABCDEFGHI 680
YABCDEFGHI 679
XABCDEFGHI 678
WABCDEFGHI 677
VABCDEFGHI 676
UABCDEFGHI 675
TABCDEFGHI 674
SABCDEFGHI 673
RABCDEFGHI 672
```

```
QABCDEFGHI 671
PABCDEFGHI 670
OABCDEFGHI 669
NABCDEFGHI 668
MABCDEFGHI 667
LABCDEFGHI 666
KABCDEFGHI 665
JABCDEFGHI 664
IABCDEFGHI 663
HABCDEFGHI 662
GABCDEFGHI 661
FABCDEFGHI 660
EABCDEFGHI 659
DABCDEFGHI 658
CABCDEFGHI 657
BABCDEFGHI 656
AABCDEFGHI 655
@ABCDEFGHI 654
?ABCDEFGHI 653
>ABCDEFGHI 652
=ABCDEFGHI 651
<ABCDEFGHI 650
;ABCDEFGHI 649
:ABCDEFGHI 648
9ABCDEFGHI 647
8ABCDEFGHI 646
7ABCDEFGHI 645
6ABCDEFGHI 644
5ABCDEFGHI 643
4ABCDEFGHI 642
3ABCDEFGHI 641
2ABCDEFGHI 640
~ABCDEFGH 639
}ABCDEFGH 638
|ABCDEFGH 637
{ABCDEFGH 636
zABCDEFGH 635
yABCDEFGH 634
xABCDEFGH 633
wABCDEFGH 632
vABCDEFGH 631
uABCDEFGH 630
tABCDEFGH 629
sABCDEFGH 628
```

```
rABCDEFGH 627
qABCDEFGH 626
pABCDEFGH 625
oABCDEFGH 624
nABCDEFGH 623
mABCDEFGH 622
lABCDEFGH 621
kABCDEFGH 620
jABCDEFGH 619
iABCDEFGH 618
hABCDEFGH 617
gABCDEFGH 616
fABCDEFGH 615
eABCDEFGH 614
dABCDEFGH 613
cABCDEFGH 612
bABCDEFGH 611
aABCDEFGH 610
`ABCDEFGH 609
_ABCDEFGH 608
^ABCDEFGH 607
]ABCDEFGH 606
\ABCDEFGH 605
[ABCDEFGH 604
ZABCDEFGH 603
YABCDEFGH 602
XABCDEFGH 601
WABCDEFGH 600
VABCDEFGH 599
UABCDEFGH 598
TABCDEFGH 597
SABCDEFGH 596
RABCDEFGH 595
QABCDEFGH 594
PABCDEFGH 593
OABCDEFGH 592
NABCDEFGH 591
MABCDEFGH 590
LABCDEFGH 589
KABCDEFGH 588
JABCDEFGH 587
IABCDEFGH 586
HABCDEFGH 585
GABCDEFGH 584
```

```
FABCDEFGH 583
EABCDEFGH 582
DABCDEFGH 581
CABCDEFGH 580
BABCDEFGH 579
AABCDEFGH 578
@ABCDEFGH 577
?ABCDEFGH 576
>ABCDEFGH 575
=ABCDEFGH 574
<ABCDEFGH 573
;ABCDEFGH 572
:ABCDEFGH 571
9ABCDEFGH 570
8ABCDEFGH 569
7ABCDEFGH 568
6ABCDEFGH 567
5ABCDEFGH 566
4ABCDEFGH 565
3ABCDEFGH 564
2ABCDEFGH 563
~ABCDEFG 562
}ABCDEFG 561
|ABCDEFG 560
{ABCDEFG 559
zABCDEFG 558
yABCDEFG 557
xABCDEFG 556
wABCDEFG 555
vABCDEFG 554
uABCDEFG 553
tABCDEFG 552
sABCDEFG 551
rABCDEFG 550
qABCDEFG 549
pABCDEFG 548
oABCDEFG 547
nABCDEFG 546
mABCDEFG 545
lABCDEFG 544
kABCDEFG 543
jABCDEFG 542
iABCDEFG 541
hABCDEFG 540
```

```
gABCDEFG 539
fABCDEFG 538
eABCDEFG 537
dABCDEFG 536
cABCDEFG 535
bABCDEFG 534
aABCDEFG 533
`ABCDEFG 532
_ABCDEFG 531
^ABCDEFG 530
]ABCDEFG 529
\ABCDEFG 528
[ABCDEFG 527
ZABCDEFG 526
YABCDEFG 525
XABCDEFG 524
WABCDEFG 523
VABCDEFG 522
UABCDEFG 521
TABCDEFG 520
SABCDEFG 519
RABCDEFG 518
QABCDEFG 517
PABCDEFG 516
OABCDEFG 515
NABCDEFG 514
MABCDEFG 513
LABCDEFG 512
KABCDEFG 511
JABCDEFG 510
IABCDEFG 509
HABCDEFG 508
GABCDEFG 507
FABCDEFG 506
EABCDEFG 505
DABCDEFG 504
CABCDEFG 503
BABCDEFG 502
AABCDEFG 501
@ABCDEFG 500
?ABCDEFG 499
>ABCDEFG 498
=ABCDEFG 497
<ABCDEFG 496
```

```
;ABCDEFG 495
:ABCDEFG 494
9ABCDEFG 493
8ABCDEFG 492
7ABCDEFG 491
6ABCDEFG 490
5ABCDEFG 489
4ABCDEFG 488
3ABCDEFG 487
2ABCDEFG 486
~ABCDEF 485
}ABCDEF 484
|ABCDEF 483
{ABCDEF 482
zABCDEF 481
yABCDEF 480
xABCDEF 479
wABCDEF 478
vABCDEF 477
uABCDEF 476
tABCDEF 475
sABCDEF 474
rABCDEF 473
qABCDEF 472
pABCDEF 471
oABCDEF 470
nABCDEF 469
mABCDEF 468
lABCDEF 467
kABCDEF 466
jABCDEF 465
iABCDEF 464
hABCDEF 463
gABCDEF 462
fABCDEF 461
eABCDEF 460
dABCDEF 459
cABCDEF 458
bABCDEF 457
aABCDEF 456
`ABCDEF 455
_ABCDEF 454
^ABCDEF 453
]ABCDEF 452
```

```
\ABCDEF 451
[ABCDEF 450
ZABCDEF 449
YABCDEF 448
XABCDEF 447
WABCDEF 446
VABCDEF 445
UABCDEF 444
TABCDEF 443
SABCDEF 442
RABCDEF 441
QABCDEF 440
PABCDEF 439
OABCDEF 438
NABCDEF 437
MABCDEF 436
LABCDEF 435
KABCDEF 434
JABCDEF 433
IABCDEF 432
HABCDEF 431
GABCDEF 430
FABCDEF 429
EABCDEF 428
DABCDEF 427
CABCDEF 426
BABCDEF 425
AABCDEF 424
@ABCDEF 423
?ABCDEF 422
>ABCDEF 421
=ABCDEF 420
<ABCDEF 419
;ABCDEF 418
:ABCDEF 417
9ABCDEF 416
8ABCDEF 415
7ABCDEF 414
6ABCDEF 413
5ABCDEF 412
4ABCDEF 411
3ABCDEF 410
2ABCDEF 409
~ABCDE 408
```

```
}ABCDE 407
|ABCDE 406
{ABCDE 405
zABCDE 404
yABCDE 403
xABCDE 402
wABCDE 401
vABCDE 400
uABCDE 399
tABCDE 398
sABCDE 397
rABCDE 396
qABCDE 395
pABCDE 394
oABCDE 393
nABCDE 392
mABCDE 391
lABCDE 390
kABCDE 389
jABCDE 388
iABCDE 387
hABCDE 386
gABCDE 385
fABCDE 384
eABCDE 383
dABCDE 382
cABCDE 381
bABCDE 380
aABCDE 379
`ABCDE 378
_ABCDE 377
^ABCDE 376
]ABCDE 375
\ABCDE 374
[ABCDE 373
ZABCDE 372
YABCDE 371
XABCDE 370
WABCDE 369
VABCDE 368
UABCDE 367
TABCDE 366
SABCDE 365
RABCDE 364
```

```
QABCDE 363
PABCDE 362
OABCDE 361
NABCDE 360
MABCDE 359
LABCDE 358
KABCDE 357
JABCDE 356
IABCDE 355
HABCDE 354
GABCDE 353
FABCDE 352
EABCDE 351
DABCDE 350
CABCDE 349
BABCDE 348
AABCDE 347
@ABCDE 346
?ABCDE 345
>ABCDE 344
=ABCDE 343
<ABCDE 342
;ABCDE 341
:ABCDE 340
9ABCDE 339
8ABCDE 338
7ABCDE 337
6ABCDE 336
5ABCDE 335
4ABCDE 334
3ABCDE 333
2ABCDE 332
~ABCD 331
}ABCD 330
|ABCD 329
{ABCD 328
zABCD 327
yABCD 326
xABCD 325
wABCD 324
vABCD 323
uABCD 322
tABCD 321
sABCD 320
```

```
rABCD 319
qABCD 318
pABCD 317
oABCD 316
nABCD 315
mABCD 314
lABCD 313
kABCD 312
jABCD 311
iABCD 310
hABCD 309
gABCD 308
fABCD 307
eABCD 306
dABCD 305
cABCD 304
bABCD 303
aABCD 302
`ABCD 301
_ABCD 300
^ABCD 299
]ABCD 298
\ABCD 297
[ABCD 296
ZABCD 295
YABCD 294
XABCD 293
WABCD 292
VABCD 291
UABCD 290
TABCD 289
SABCD 288
RABCD 287
QABCD 286
PABCD 285
OABCD 284
NABCD 283
MABCD 282
LABCD 281
KABCD 280
JABCD 279
IABCD 278
HABCD 277
GABCD 276
```

```
FABCD 275
EABCD 274
DABCD 273
CABCD 272
BABCD 271
AABCD 270
@ABCD 269
?ABCD 268
>ABCD 267
=ABCD 266
<ABCD 265
;ABCD 264
:ABCD 263
9ABCD 262
8ABCD 261
7ABCD 260
6ABCD 259
5ABCD 258
4ABCD 257
3ABCD 256
2ABCD 255
~ABC 254
}ABC 253
|ABC 252
{ABC 251
zABC 250
yABC 249
xABC 248
wABC 247
vABC 246
uABC 245
tABC 244
sABC 243
rABC 242
qABC 241
pABC 240
oABC 239
nABC 238
mABC 237
lABC 236
kABC 235
jABC 234
iABC 233
hABC 232
```

```
gABC 231
fABC 230
eABC 229
dABC 228
cABC 227
bABC 226
aABC 225
`ABC 224
_ABC 223
^ABC 222
]ABC 221
\ABC 220
[ABC 219
ZABC 218
YABC 217
XABC 216
WABC 215
VABC 214
UABC 213
TABC 212
SABC 211
RABC 210
QABC 209
PABC 208
OABC 207
NABC 206
MABC 205
LABC 204
KABC 203
JABC 202
IABC 201
HABC 200
GABC 199
FABC 198
EABC 197
DABC 196
CABC 195
BABC 194
AABC 193
@ABC 192
?ABC 191
>ABC 190
=ABC 189
<ABC 188
```

```
;ABC 187
:ABC 186
9ABC 185
8ABC 184
7ABC 183
6ABC 182
5ABC 181
4ABC 180
3ABC 179
2ABC 178
~AB 177
}AB 176
|AB 175
{AB 174
zAB 173
yAB 172
xAB 171
wAB 170
vAB 169
uAB 168
tAB 167
sAB 166
rAB 165
qAB 164
pAB 163
oAB 162
nAB 161
mAB 160
lAB 159
kAB 158
jAB 157
iAB 156
hAB 155
gAB 154
fAB 153
eAB 152
dAB 151
cAB 150
bAB 149
aAB 148
`AB 147
_AB 146
^AB 145
]AB 144
```

```
\AB 143
[AB 142
ZAB 141
YAB 140
XAB 139
WAB 138
VAB 137
UAB 136
TAB 135
SAB 134
RAB 133
QAB 132
PAB 131
OAB 130
NAB 129
MAB 128
LAB 127
KAB 126
JAB 125
IAB 124
HAB 123
GAB 122
FAB 121
EAB 120
DAB 119
CAB 118
BAB 117
AAB 116
@AB 115
?AB 114
>AB 113
=AB 112
<AB 111
;AB 110
:AB 109
9AB 108
8AB 107
7AB 106
6AB 105
5AB 104
4AB 103
3AB 102
2AB 101
~A 100
```

```
}A 99
|A 98
{A 97
zA 96
yA 95
xA 94
wA 93
vA 92
uA 91
tA 90
sA 89
rA 88
qA 87
pA 86
oA 85
nA 84
mA 83
lA 82
kA 81
jA 80
iA 79
hA 78
gA 77
fA 76
eA 75
dA 74
cA 73
bA 72
aA 71
`A 70
_A 69
^A 68
]A 67
\A 66
[A 65
ZA 64
YA 63
XA 62
WA 61
VA 60
UA 59
TA 58
SA 57
RA 56
```

```
QA 55
PA 54
OA 53
NA 52
MA 51
LA 50
KA 49
JA 48
IA 47
HA 46
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
```

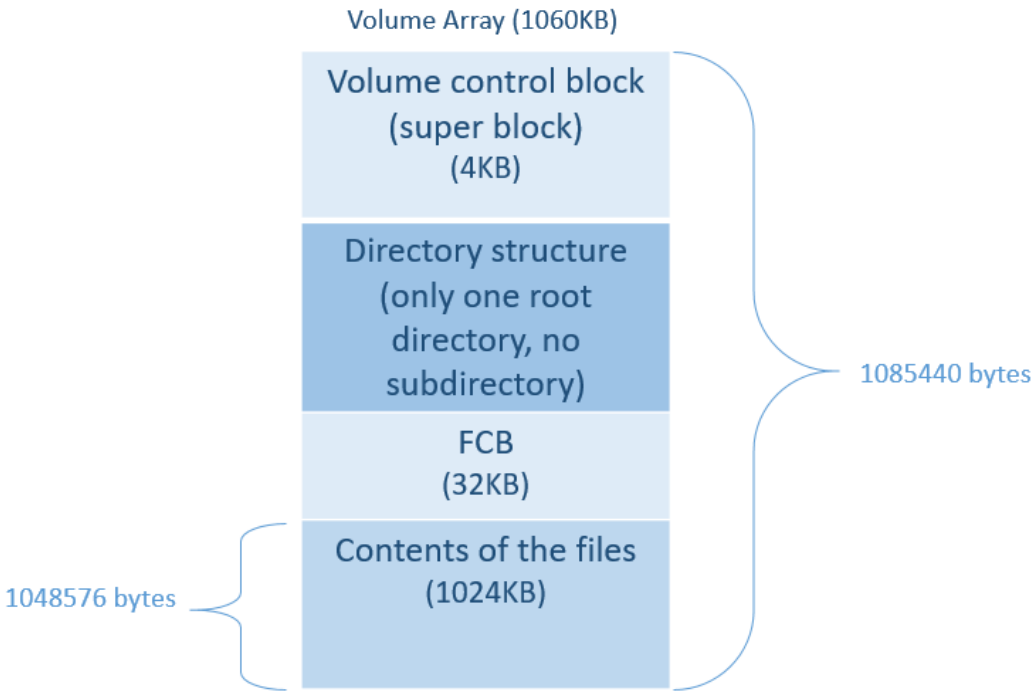
d. Bonus Test Case

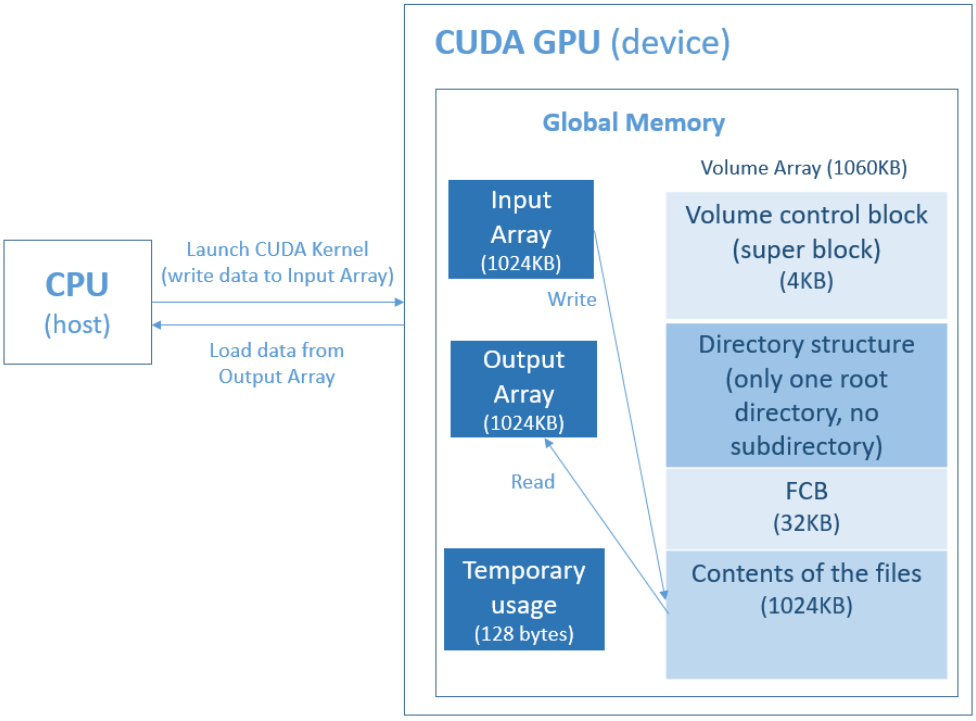
```
===sort by modified time===
t.txt
b.txt
```

```
===sort by file size===
t.txt 32
b.txt 32
===sort by modified time===
app d
t.txt
b.txt
===sort by file size===
t.txt 32
b.txt 32
app 0 d
===sort by file size===
===sort by file size===
a.txt 64
b.txt 32
soft 0 d
===sort by modified time===
soft d
b.txt
a.txt
/app/soft
===sort by file size===
B.txt 1024
C.txt 1024
D.txt 1024
A.txt 64
===sort by file size===
a.txt 64
b.txt 32
soft 24 d
/app
===sort by file size===
t.txt 32
b.txt 32
app 17 d
===sort by file size===
a.txt 64
b.txt 32
===sort by file size===
t.txt 32
b.txt 32
app 12 d
```

6. What Did I Learn from This Assignment?

a. How is a simple file system physically organized?





b. How to design the file-control block?

c. How to design a file system with directory structure?

   Use a tree structure to organize the files and directories. Also, think of a directory as a special file.

d. What is the bit-map and how to implement it?

e. How to deal with various errors? For example, insufficient storage in fs_write(), file not found in fs_open(G_WRITE), filename too long, and trying to delete a directory using fs_gsys(RM).