Jackson Stewart
329009382
CSCE 410

MP7: Basic File System

Assigned Tasks
Main: done
Bonus Options: not attempted

Modifications to provided header files

File.H: I added private file data structures for the file system reference, file ID, current position, and Inode.

File_system.H: I added GetFreeInode() and GetFreeBlock() helper functions.

Jackson Stewart
329009382
CSCE 410

MP7: Basic File System

File system implementation (file_system.C): Implements setup functions and creating/deleting files

The constructor and destructor just set up (and take down) the Inode and free blocks lists.

```
FileSystem::FileSystem() : disk(nullptr), size(MAX_INODES), inodes(nullptr), free_blocks(nullptr) {
    Console::puts("In file system constructor.\n");

    inodes = new Inode[MAX_INODES];
    if (!inodes) {
        Console::puts("Failed to allocate inodes.\n");
        assert(false);
    }

    for (int i = 0; i < MAX_INODES; i++) {
        inodes[i].id = -1;
        inodes[i].fs = this;
    }

    free_blocks = new unsigned char[SimpleDisk::BLOCK_SIZE];
    if (!free_blocks) {
        Console::puts("Failed to allocate free blocks.\n");
        assert(false);
    }

    memset(free_blocks, 0, SimpleDisk::BLOCK_SIZE);
}
```

```
FileSystem::~FileSystem() {
    Console::puts("unmounting file system\n");
    /* Make sure that the inode list and the free list are saved. */
    // assert(false);

    if (disk) {
        unsigned char inode_data[SimpleDisk::BLOCK_SIZE];
        unsigned char free_data[SimpleDisk::BLOCK_SIZE];

        memcpy(inode_data, inodes, sizeof(Inode) * MAX_INODES);

        disk->write(0, inode_data);

        memcpy(free_data, free_blocks, SimpleDisk::BLOCK_SIZE);

        disk->write(1, free_data);
    }

    if (inodes) {
        delete[] inodes;
        inodes = nullptr;
    }

    if (free_blocks) {
        delete[] free_blocks;
        free_blocks = nullptr;
    }
}
```

Jackson Stewart
329009382
CSCE 410

MP7: Basic File System

The mount function sets up a file system that already exists by reading inode and free list data from disk.

```
bool FileSystem::Mount(SimpleDisk* _disk) {
    Console::puts("mounting file system from disk\n");

    /* Here you read the inode list and the free list into memory */
    // assert(false);

    if (!_disk) {
        return false;
    }

    this->disk = _disk;

    unsigned char inode_data[SimpleDisk::BLOCK_SIZE];
    unsigned char free_data[SimpleDisk::BLOCK_SIZE];

    disk->read(0, inode_data);

    memcpy(inodes, inode_data, sizeof(Inode) * MAX_INODES);

    disk->read(1, free_data);

    memcpy(free_blocks, free_data, SimpleDisk::BLOCK_SIZE);

    return true;
}
```

LookupFile finds a file by its ID and returns its Inode:

```
Inode* FileSystem::LookupFile(int _file_id) {
    Console::puts("looking up file with id = ");
    Console::puti(_file_id);
    Console::puts("\n");
    /* Here you go through the inode list to find the file. */
    // assert(false);
    for (unsigned int i = 0; i < this->size; i++) {
        if (this->inodes[i].id == _file_id) {
            return &this->inodes[i];
        }
    }

    return nullptr;
}
```

Jackson Stewart
329009382
CSCE 410

MP7: Basic File System

Format resets the file system by initializing the disk with an empty inode and free list and marks that data as used.

```
bool FileSystem::Format(SimpleDisk* _disk, unsigned int _size) {  // static!
    Console::puts("formatting disk\n");
    /* Here you populate the disk with an initialized (probably empty) inode list
       and a free list. Make sure that blocks used for the inodes and for the free list
       are marked as used, otherwise they may get overwritten. */
    // assert(false);
    if (!_disk) {
        return false;
    }

    Inode local_inodes[MAX_INODES];
    unsigned char local_free_blocks[SimpleDisk::BLOCK_SIZE];

    for (int i = 0; i < MAX_INODES; i++) {
        local_inodes[i].id = -1;
    }

    memset(local_free_blocks, 0, SimpleDisk::BLOCK_SIZE);

    int inode_block_num = 0;
    int free_block_num = 1;

    local_free_blocks[inode_block_num / 8] |= 1 << (inode_block_num % 8);
    local_free_blocks[free_block_num / 8] |= 1 << (free_block_num % 8);

    _disk->write(inode_block_num, (unsigned char*)local_inodes);

    _disk->write(free_block_num, local_free_blocks);

    return true;
}
```

Jackson Stewart
329009382
CSCE 410

MP7: Basic File System

The CreateFile function creates a new file with the given ID on the disk, checking to make sure it doesn't already exist and updates inode and free list data accordingly.

```
bool FileSystem::CreateFile(int _file_id) {
    Console::puts("creating file with id:");
    Console::puti(_file_id);
    Console::puts("\n");
    /* Here you check if the file exists already. If so, throw an error.
       Then get yourself a free inode and initialize all the data needed for the
       new file. After this function there will be a new file on disk. */
    // assert(false);

    if (this->LookupFile(_file_id) != nullptr) {
        Console::puts("File already exists.\n");
        return false;
    }

    // get free inode
    Inode* inode = GetFreeInode();
    if (!inode) {
        Console::puts("Failed to get free inode.\n");
        return false;
    }

    inode->id = _file_id;
    inode->fs = this;

    int free_block = GetFreeBlock();
    if (free_block == -1) {
        Console::puts("Failed to get free block.\n");
        return false;
    }

    inode->block_no = free_block;

    return true;
}
```

Jackson Stewart
329009382
CSCE 410

MP7: Basic File System

The DeleteFile function deletes a file with the given ID by freeing its associated block and invalidating its Inode.

```
bool FileSystem::DeleteFile(int _file_id) {
    Console::puts("deleting file with id:");
    Console::puti(_file_id);
    Console::puts("\n");
    /* First, check if the file exists. If not, throw an error.
       Then free all blocks that belong to the file and delete/invalidate
       (depending on your implementation of the inode list) the inode. */

    Inode* inode = this->LookupFile(_file_id);
    if (!inode) {
        return false;
    }

    int block_to_free = inode->block_no;
    if (block_to_free != -1) {
        this->free_blocks[block_to_free] = 0;  // mark as free
    }

    inode->id = -1;
    inode->block_no = -1;

    return true;
}
```

Jackson Stewart
329009382
CSCE 410

MP7: Basic File System

File implementation (file.C): Implements individual file operations and management.

The constructor initializes the file by finding and reading its Inode.

```
File::File(FileSystem *_fs, int _id) : fs(_fs), file_id(_id), current_position(0) {
    Console::puts("Opening file.\n");
    // assert(false);

    inode = fs->LookupFile(_id);
    if (!inode) {
        Console::puts("Failed to find inode.\n");
        assert(false);
    }

    disk_block = new unsigned char[SimpleDisk::BLOCK_SIZE];
    fs->disk->read(inode->block_no, block_cache);
}
```

The destructor writes any cached data back to the disk and updates the Inode accordingly.

```
File::~File() {
    Console::puts("Closing file.\n");
    /* Make sure that you write any cached data to disk. */
    /* Also make sure that the inode in the inode list is updated. */

    fs->disk->write(inode->block_no, block_cache);
}
```

The Read function reads a given number of bytes from the disk into the provided buffer and updates the current position.

```
int File::Read(unsigned int _n, char *_buf) {
    Console::puts("reading from file\n");
    // assert(false);

    int bytes_read = 0;
    while (_n > 0 && current_position < SimpleDisk::BLOCK_SIZE) {
        _buf[bytes_read] = block_cache[current_position];
        bytes_read++;
        current_position++;
        _n--;
    }

    return bytes_read;
}
```

Jackson Stewart
329009382
CSCE 410

MP7: Basic File System

Write does the same thing as Read but in reverse, writing data from the buffer to the disk.

```
int File::Write(unsigned int _n, const char *_buf) {
    Console::puts("writing to file\n");
    // assert(false);

    int bytes_written = 0;
    while (_n > 0 && current_position < SimpleDisk::BLOCK_SIZE) {
        block_cache[current_position] = _buf[bytes_written];
        bytes_written++;
        current_position++;
        _n--;
    }

    return bytes_written;
}
```

Reset and EoF are simple functions related to the current position in the file.

```
void File::Reset() {
    Console::puts("resetting file\n");
    // assert(false);
    current_position = 0;
}

bool File::EoF() {
    Console::puts("checking for EoF\n");
    // assert(false);

    return current_position >= SimpleDisk::BLOCK_SIZE;
}
```