CSCE 611

Pranav Anand Taukari (UIN:433003781)

MP7 Design Document

Objective:

The objective of machine problem 7 is to implement a vanilla file system where the files support sequential access only and bonus question to allow files that are 64KB in size.

Implementation Details:

The file system maintains the metadata of the files and it is responsible for creating and deleting the files.

I am using Block 1 to maintain the metadata associated with the files and reserving Block 0 to maintain the list of free blocks.

Inode Variables:

1. id – This is used to store the file name
2. block_no – This variable is used to store the block no
3. inode_free – Used for checking if the inode is free or not
4. file_size - Used for storing the size of the file

File System Implementation:

It contains variables:

MAX_INODES : maximum number of inodes we can use in the file system

Free_block_count: maximum number of free blocks in the file system

Free_blocks : pointer to bitmap of free blocks. Uses identifier ‘f’ for denoting free block

and ‘u’ for denoting occupied block

Disk : pointer to disk

Inodes: pointer to array of inodes

FileSystem() :

The constructor is used for initializing the filesystem object

```
disk = NULL;
size = 0;
free_blocks = new unsigned char[free_block_count];
inode_counter= 0;
inodes = new Inode[MAX_INODES]; - initializes inode list
```

~FileSystem():

This destructor is used when we close the file and it is used to write the applied changes to disk

Mount (SimpleDisk *_disk) :

It loads the data structures like inode array and free blocks bitmap from disk to file system class. And associates this disk to our file system class.

Format(SimpleDisk *_disk, unsigned int _size):

This function is used to formats the disk. It is a static function and thus cant access non static class variables. So to overcome this I am creating a new free block array and inode list. It deletes the data structures on disk and initializes new empty free blocks bitmap and inodes array and writes them to the disk

LookupFile(int _file_id) :

Returns the inode associated with the given file id. Throws assertion error if the inode for the given file id is not found.

CreateFile(int _file_id) :

To create a new file I am searching for a free inode and free block. Once I have the index of these , I am initializing the inode list and marking the free block as used.

DeleteFile(int _file_id):

Deletes everything related to the file. Frees up the inode associated as well as the block(marks the block in free blocks bitmap as free). Post this I am writing NULL in the disk for the associated block number.

GetFreeBlocks():

This function return the index of the free block

File Implementation:

This contains all the functions and variables associated with one file like file reading, writing , updating, deleting.

block_cache[SimpleDisk::BLOCK_SIZE]; : buffer handler for file .It is basically a cached

copy of the block we are reading and writing to . max size is 512 bytes.

FileSystem *fs: file system associated with the file

int file_id; id of the file

unsigned int block_no; block number allocated the file

unsigned int inode_index; index of inode in inode array that is assigned to the file

unsigned int file_size; size of the file

unsigned int cur_pos; indicates where the next character will be read from or written to.

Functions Implemented:

File(FileSystem * _fs, int _id): The constructor of the file handle. Intialises the class variables with the variables from fs object.

~File():

Closes the file. Updates inode associated accordingly and saves it to the disk.

int Read(unsigned int _n, char * _buf):

Reads from the file from current position to _n characters and return the number of characters read.

int Write(unsigned int _n, const char * _buf):

Writes _n characters starting with current position. Increments file size up to 1 block size. Returns number of characters written

void Reset():

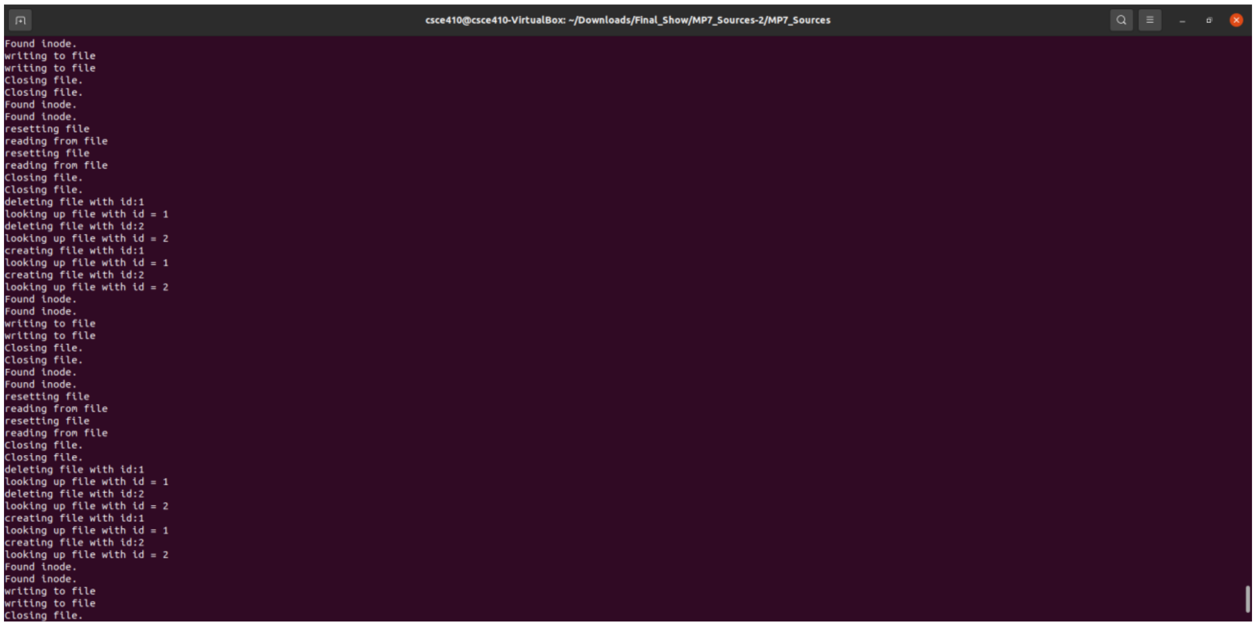
Sets the current position to beginning of the file

bool EoF():

Used to check if the pointer is at the end of file.

Output:

An infinite loop of read, write and delete. This proves the concept.



csce410@csce410-VirtualBox: ~/Downloads/Final_Show/MP7_Sources-2/MP7_Sources

```
Found inode.
writing to file
writing to file
Closing file.
Closing file.
Found inode.
Found inode.
resetting file
reading from file
resetting file
reading from file
Closing file.
Closing file.
deleting file with id:1
looking up file with id = 1
deleting file with id:2
looking up file with id = 2
creating file with id:1
looking up file with id = 1
creating file with id:2
looking up file with id = 2
Found inode.
Found inode.
writing to file
writing to file
Closing file.
Closing file.
Found inode.
Found inode.
resetting file
reading from file
resetting file
reading from file
Closing file.
Closing file.
deleting file with id:1
looking up file with id = 1
deleting file with id:2
looking up file with id = 2
creating file with id:1
looking up file with id = 1
creating file with id:2
looking up file with id = 2
Found inode.
Found inode.
writing to file
writing to file
Closing file.
```

OPTION 1: FILE SYSTEM FOR FILES THAT ARE UPTO 64KB LONG.

To implement this, we need to assign multiple blocks to one file. For this we maintain a block_list which maintains a sequence of block numbers for the file.

So we add long block_list[size] which will have a sequence of block numbers associated with the file.

Functions to be Changed:

1. Create File: We assign the sequence of blocks and mark them as free
2. Delete File: We mark the block numbers as free
3. Read: In this we initially calculate the size of the file from the block_list and read a sequence of blocks upto the end of the last block.
4. Write: In this we keep updating the size and once we reach the end of block and incase we have to write more we add a new block and write to it upto file size required. We can also define a new function, which can increase the file size if it is not enough, by taking a free block from the file system and appending it to the block_list.