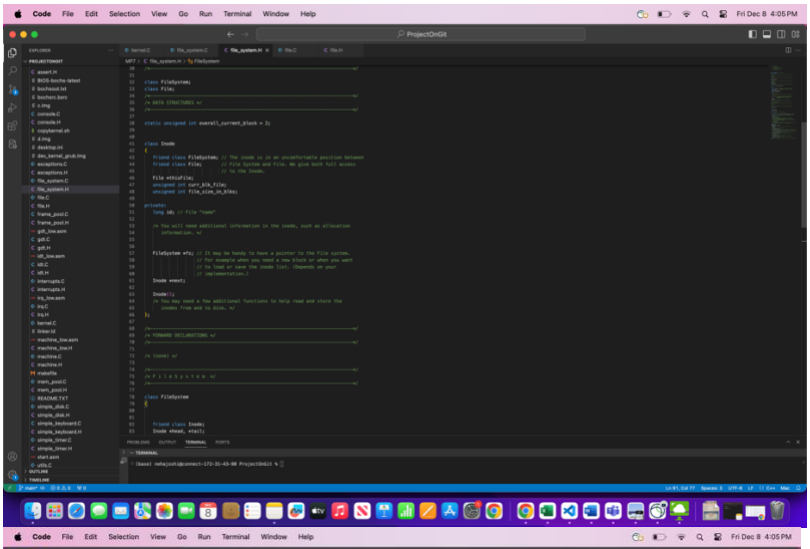
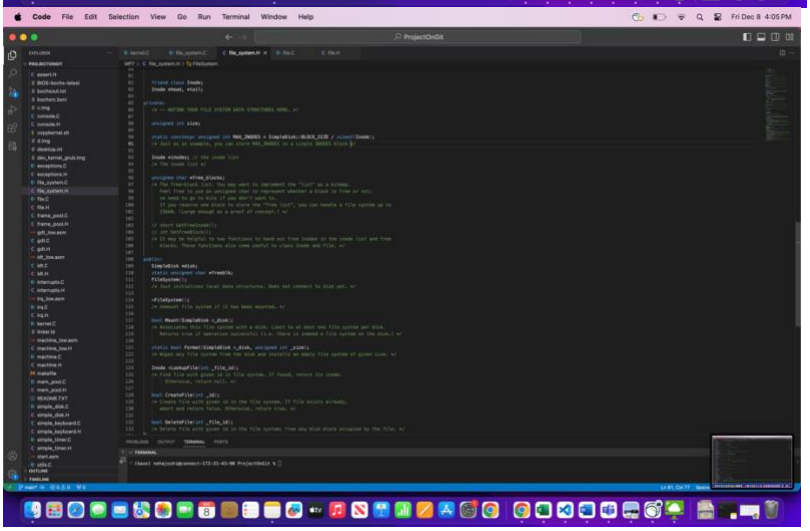
Design Document : MP7 - Simple File System

I have implemented the simple file system as mentioned in the handout. Along with the basic implementation- I have added Bonus1 as well as Bonus2 which included design and implementation of file system capable of handing files of size upto 64 kB.

Following are the function wise explanations (very comprehensive comments are written in the code as well):

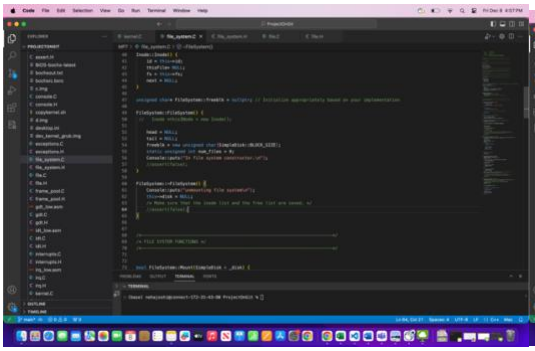
1. **file_system.H**
    a. Here we define the inode and its class. Inode will maintain the file block information.
    b. We also define the structure and functions of file_system.
    c. File system has pointer to inode
    d. We plan to maintain a linked list of inodes- 1 node per file and accordingly update meta data.

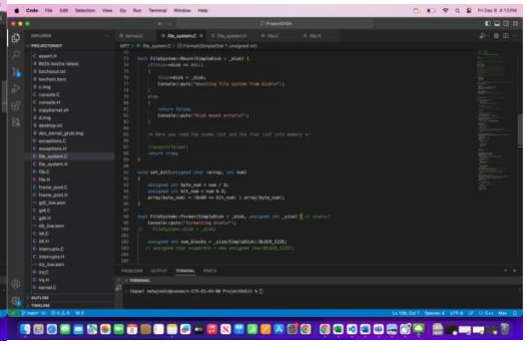


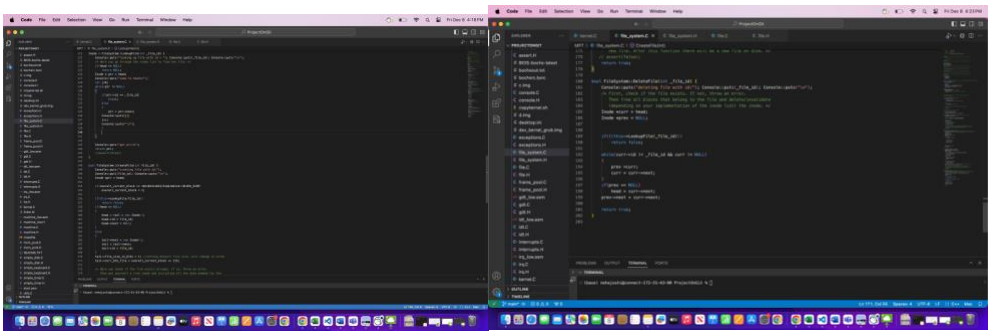


2. file_system.C

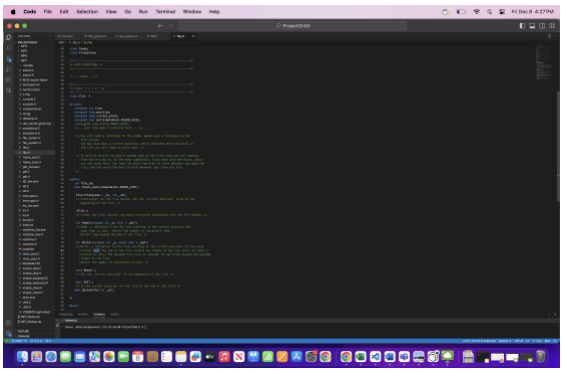
a. **Inode::Inode()** – this is the inode constructor where we initialize the inode variables.
b. **FileSystem::FileSystem()** – constructor for File system. Initializes the variables when a new file system is created.
c. **FileSystem::~FileSystem()** – destructor for file system
d. **bool FileSystem::Mount(SimpleDisk * _disk)** – mounts the disk to the current file system.
e. void set_bit(unsigned char *array, int num) – udpates the free block bitmap. This will be called whenever a new block is assigned.
f. **bool FileSystem::Format(SimpleDisk * _disk, unsigned int _size)** – sets the block 1 and 2 as used in the free block bitmap. Also sets other bits of free block bitmap to 0 - free.
g. **Inode * FileSystem::LookupFile(int _file_id)** – this function looks uup the file in inode using file id and returns the inode.
h. **bool FileSystem::CreateFile(int file_id)**- Adds a new node to inode linked list and sets the pointers accordingly. Also updates the inode fields like the block assigned /starting block assigned to the file. To handle 64kB files, we have also made a provision that for every new file we assign block after 128 blocks. We have also added a condition that after multiple adds and deleted if overall block crosses the disk size we bring it back to 2. (block 0 is super block and block 1 is inode storage space)
i. **bool FileSystem::DeleteFile(int _file_id)** – This simply deletes an inode and hence effectively file gets invalidated and deleted.





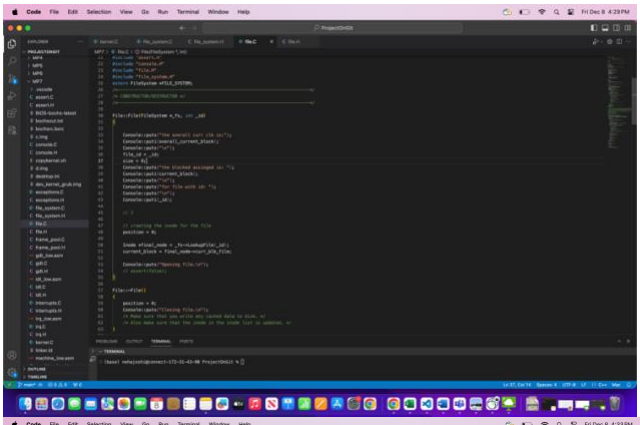


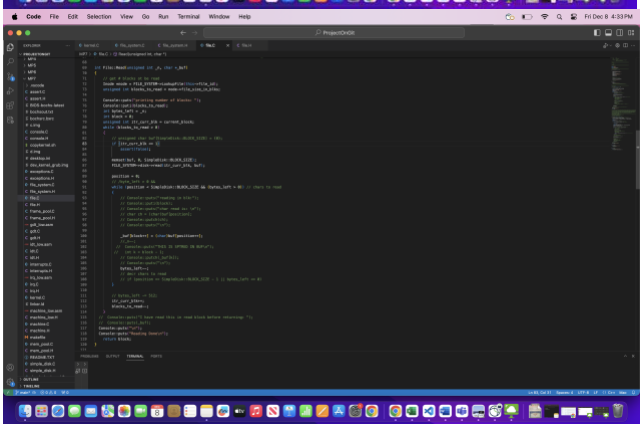
3. file.H
    a. Here we initialize all variables and functions needed in the file class

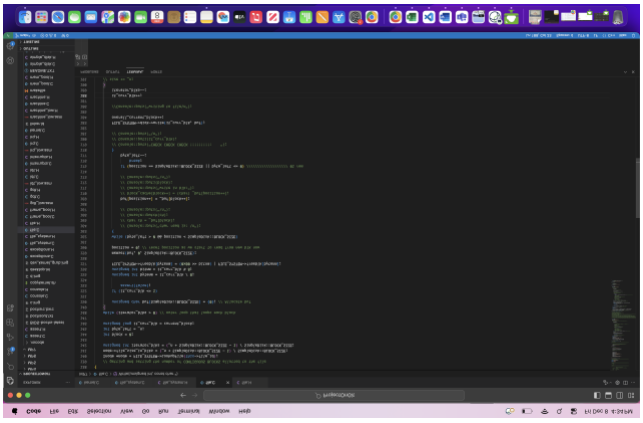


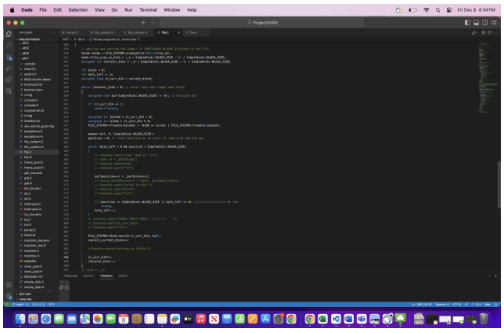
4. file.C
    a. File::File(FileSystem *_fs, int _id) – assigns current block of the file in context to its inode variable. Also makes reading position as 0.
    b. File::~File() – brings position to 0.
    c. int File::Read(unsigned int _n, char *_buf) –
        i. Reads the number of blocks to be rad from file from inode and iterates to read those number of blocks one by one.
        ii. In reach iteration, one block is read from the disk and then appended to the buffer. Finally the number of characters read are returned.
        iii. We put conditions on block number and position to verify If we cleanly read each block.
    d. int File::Write(unsigned int _n, const char *_buf) –
        i. We first calculate the number of blocks to be written using the parameters passed. We update this as a inode member of that file.
        ii. We now loop through the number of blocks to be written and write to disk block wise.
        iii. Again, we use the block size and number of block conditions to cleanly write and exit.
5. void File::Reset()
    a. make position to 0.

6. bool File::EoF()
    a. check if we have reached the end of file using block size.









Design for accommodating read and write of large files (upto 64kB):

1. Calculate the number of blocks to be written as per the write parameters and set it in inode. Itertate those many number of times to perform block wise read and write operations
2. Once done writing cleaning exit as we track the # bytes as well as blocks.

3. Initialize the start of block per file and for consecutive files give block number after 128 blocks so you can accommodate files of size upto 64kB.

Implementation:

I tried to write strings with size greater that block size (greater than 512 bytes) and was successful to do the same.
Also of course the main implementation works.

Following is the screenshot when I tried to write a 600 byte file into the disk :