Design Document MP7

Chinmay Ajit Sawkar

UIN 633002213

BONUS - OPTION 1 completed

In this machine problem we design a Vanilla file system. This is a simple file system in which we have only sequential file access, the files are identified directly by unsigned integers and there is no multi level directory support. This is implemented by using two classes File System and File.

The File Class:

The file class is used to implement sequential read/write operations on individual files. This class consist of following methods:

1. File()- this method is used to initialize the current position the file and other data structures of the file like block no, inode index. If block exist we read the block cache and store it in the buffer.
2. ~File()- this is used to close the file, we call the write function before closing the file and update the file length
3. Read() - this is used to read _n chars from the file. We then store the characters read to a buffer. The check is done whether the end of file is not reached every time we read. This function returns the number of characters read.
4. Write() - this function is used to write characters in the file. We write from the current position and update it when we are writing. The value in the buffer is basically being added to the block cache.
5. Reset() - This is used resect the current position in the file
6. EOF()- this is used to check if we have reached the end of the file while writing or reading

Class Inode: this is a friend class of File and File System classes and gives us the basic structure of the inode. This has the information regarding the block number, if the block is used or not, and flie length.

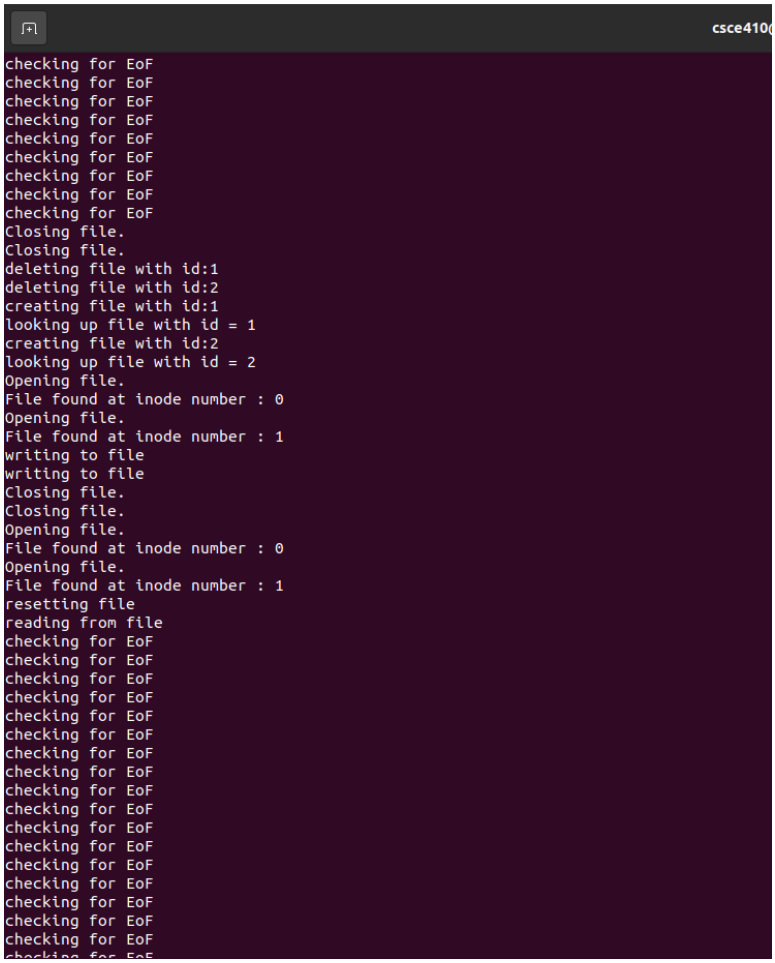
File System Class:

We have implemented a separate class File System so that we keep different aspects of file management in different classes. In the file class we have the functions like read/write operations and creating and deleting the file, mounting on disk and the backend operations like maintaining and updating inode list and free block list separately in the File System class. This class has following methods:

1. FileSystem(): This is the constructor method and is used to initialise the class data structure with the variables of the file system like free block count, size of file. This does not connect to the disk but just initializes the variables.
2. ~FileSystem(): This is the destructor function and is used to unmount the the file system on the disk and saves the data structures of file system to disk
3. Mount() : This function is used to mount the file system from the disk. Here we read the free blocks we read the inode list.
4. Format() : this is used to write the free list and the inode array to the disk. In this file system we use the zero th block for free block list and first block for inode array list.
5. Lookup(): this function is used to find the file with the file id given. If we find the file with the given id we return the inode object else we return null.
6. CreateFile() : this function is used to create the file with the given id. First we call the lookup() function to check if the file already exists. If not then we check for a free inode block and then also a free block in the free block list and then assign the free block to the file accordingly and update the data structures after this operation appropriately.
7. DeleteFile() : this is used to delete the file with the given id. The disk blocks that are occupied by the file on the disk are freed by writing NULL to these blocks using the Delete_Inode_State helper function and then updated in the free block list.

Testing

The MP was successfully tested with the given code in kernel.C with the correct test result as below:



Option 1 : **DESIGN extension to the basic file system to allow for files that are up to 64kB long.**

As the file size is now greater than the size of 1 block multiple blocks will be assigned to 1 file and thus we will have to keep track of the data blocks sequence which is being allocated to the file. This can be done using a different data structure like an array or linked list in the Inode class to store the sequence of blocks.

Function changes:
File System:

1. While creating the file a block list should be assigned to the file instead of just a block. It will be done easily using the array in Inode class.
2. While deleting the file we would have to delete all the blocks associated with the file.

3. We might have to add a function to allocate extra blocks when a file request.

File :

1. The read operation will be changed to read from a sequence of block instead of just 1 block
2. The write operation will be changed to write from a sequence of block instead of just 1 block
   and if there is an end of file while writing then the write function would have to request a new block from the file system.