# MP7: Simple File System

Dhanraj Murali
UIN: 734003894
CSCE611: Operating System

## Assigned Tasks:

**Main:** Completed.
**Bonus:**
Option 1 Completed.

## System Design:

1. Simple file system supporting sequential access only and simple name space.

## Code Description:

I. Two lists maintained, one each for inode list and freelist of blocks. These two lists are stored in block 0 and block 1 of the disks. The inode class is modified to maintain the block number where the file is stored, a boolean status bit indicating if the node is valid or if it is deleted.
II. Any read or write operation to the file is performed using temporary cache. Read or write is performed once everything is performed on the temporary cache and then fed onto the file.

### Files modified:

a. kernel.C
b. file.C
c. file.H
d. filesystem.C
e. filesystem.H

### Function Description:

**MP Basic Target:**

Inode's metadata consists of block number where the file is stored, status whether it is present or deleted, and length of the file. These are modified during creation and deletion of files as well as updated when it is written to. A counter for inodes is also maintained and whenever a new file is created the value is updated and when a file is deleted, the value is decremented.

Whenever a read or write operation is performed, the data is placed in the buffer called block_cache and once end of file is reached then only the content is fed on to the target file when the destructor is invoked. The block size is maintained as a variable defined as TEMP_SIZE which can be modified as per requirement. In this MP it is maintained as 512 bytes.

The GetFreeBlock function is used to get a free block whenever a new file is created. The function returns the block number to be used if at all there is available free blocks. This block number is updated in the metadata of the inode during creation. The delete function invalidates the file in the inode by modifying the status attribute of the inode to false in the metadata of the inode.

The EOF function is used to check if the End of File condition is reached or not. If the current position is greater than or equal to TEMP_SIZE which in this case is 512 then the function return true meaning end of file is reached.

In this vanilla implementation, there is a finite number of inodes that can be created which in turn restricts the number of files that can be created. In the create file function, the function checks if the ninode counter is equal to the MAX_INODES which is calculated by dividing the disk's block size by the size the inode object takes. Only if it is not equal then a file creation can be proceeded so that inode metadata can be stored, else the function returns false.

## **Bonus Option – 1:**
## **DESIGN of an extension to the basic file system to allow for files that are up to 64kB long**

Modification to I-node Class by introducing index-based structure with pointers to data blocks so that we can allocate the required number of blocks in order to attain a 64KB file size. The index structure should be created with direct, indirect blocks. The size attribute needs to be updated to store the size occupied in the last block and the number of blocks utilized attribute need to be introduced.

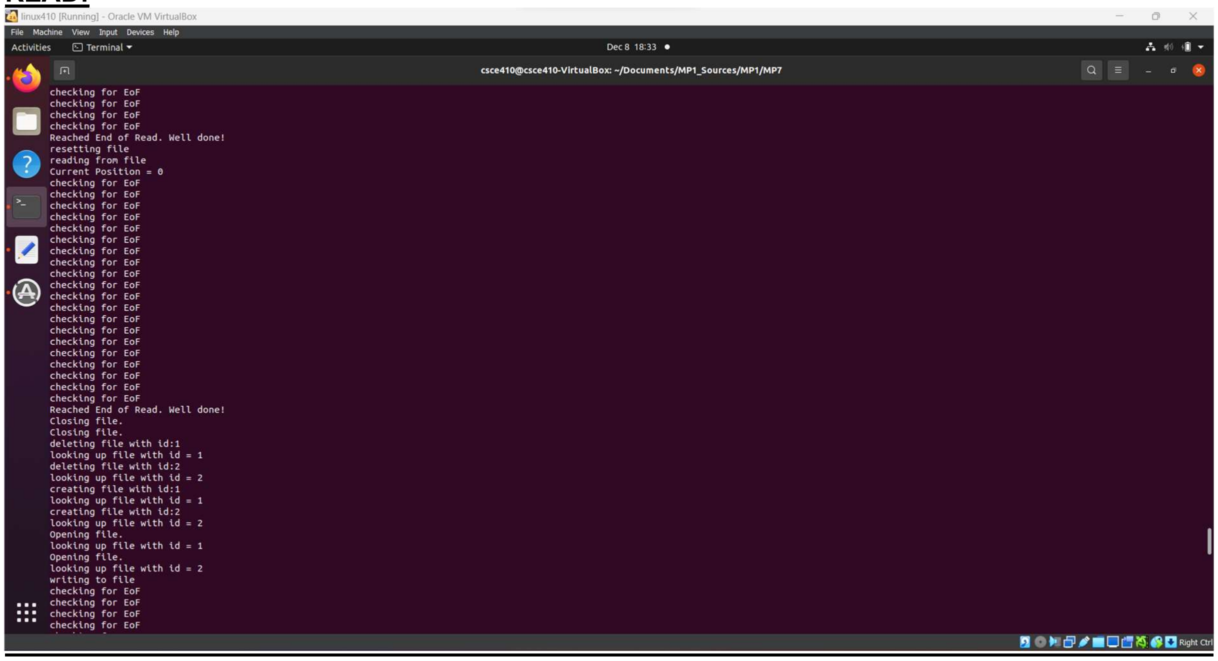
The create file function need not be modified however whenever a block is filled and the file requires a new block then a new function is to be designed that returns a free block and the block number using i-node object and also update the i-node with the used blocks.

A new attribute also needs to be maintained or existing block number attribute can be converted to an array and then every new block number of the block being used is appended to the array. During read operation also this array can be used to read the blocks sequentially.

The current position attribute needs to modified so as to point to two aspects of the file – the current block that is being catered and the current position in the current block. Based on this the reset function needs to update both the current position to 0 and current block to the first block number of the file which can be retrieved from the first position of the array.

The EOF function needs to be updated to make sure both the current position is reached to the maximum value and the next block of the target file has -1 as value indicating that there are no more blocks being used. For this to work, the used block array needs to be initialized with -1 values and whenever a new block is being added the corresponding position need to be overwritten with the new block value. Also the maximum size of this array needs to be 64KB/block size.

## READ:



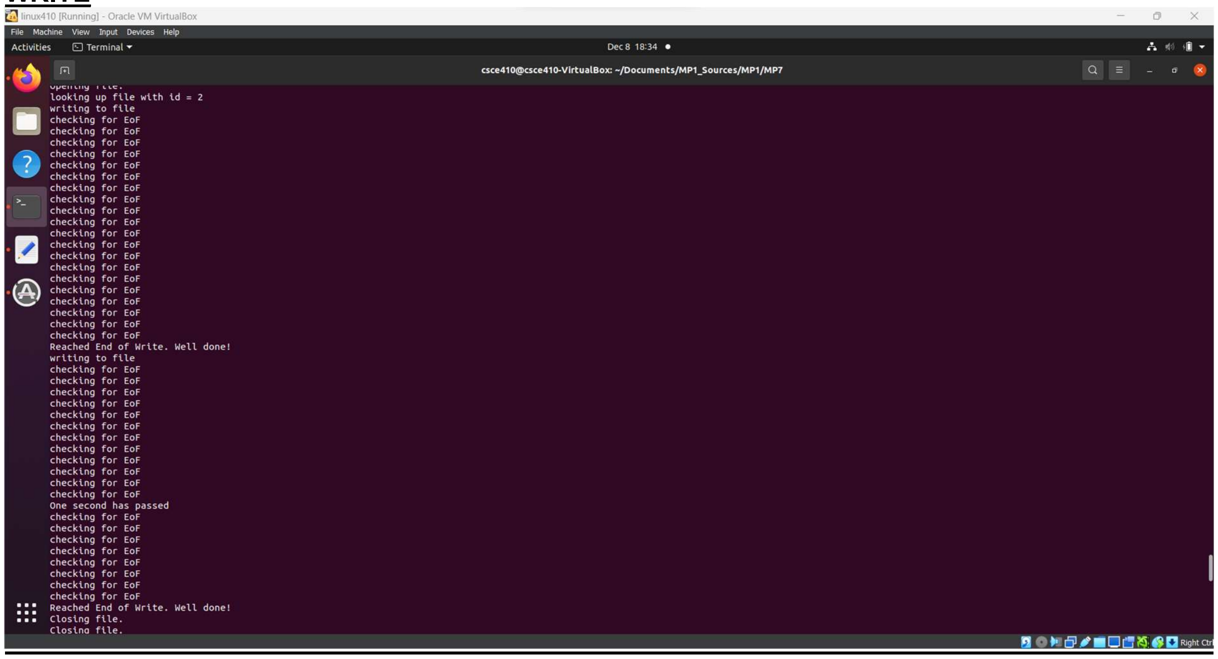
linux410 [Running] - Oracle VM VirtualBox

Dec 8 18:33

csce410@csce410-VirtualBox: ~/Documents/MP1_Sources/MP1/MP7

```
checking for EoF
checking for EoF
checking for EoF
checking for EoF
Reached End of Read. Well done!
resetting file
reading from file
Current Position = 0
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
Reached End of Read. Well done!
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
Opening file.
looking up file with id = 1
Opening file.
looking up file with id = 2
writing to file
checking for EoF
checking for EoF
checking for EoF
checking for EoF
```

## WRITE



linux410 [Running] - Oracle VM VirtualBox

Dec 8 18:34

csce410@csce410-VirtualBox: ~/Documents/MP1_Sources/MP1/MP7

```
looking up file with id = 2
writing to file
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
Reached End of Write. Well done!
writing to file
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
One second has passed
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
checking for EoF
Reached End of Write. Well done!
Closing file.
Closing file.
```