VANILLA FILE SYSTEM

OPERATING SYSTEMS
CS 611
MP6

SIRI NAMBURI
FALL 2021

The aim of this machine problem is to implement a simple file system where each file is of maximum size 1 block, that is 512 bytes.. The file system is implemented with the help of simpler versions of inode and free blocks array and this file system supports sequential access only in files. For implementation of the file system the following classes are used

1) FILE SYSTEM:

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

The free block bitmap is stored in the first block on the disk (block index number 0) and the inodes array is stored on the second block of the disk (block index number 1).

It contains the following functions:

- FileSystem(): initialises the data structures on class file system like inodes array etc.
- ~FileSystem() : saves the data structures in file system class to disk
- Mount(SimpleDisk *_disk) : 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): formats the disk, that is 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) : creates a new file with the given file id. Basically initializes a new inode for the file, maps it to the file, finds a new free block , allocates it to the file and marks the block as used in the free blocks bitmap.
- 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)

2) INODE

This is a class containing all the variables associated with inode like file number, allocated block number, file system, file size ,and a flag to see whether the inode is free or not.

3) FILE

This class contains all the variables and functions that are associated with a single file. Like opening a file, updating a file , reading a file etc.

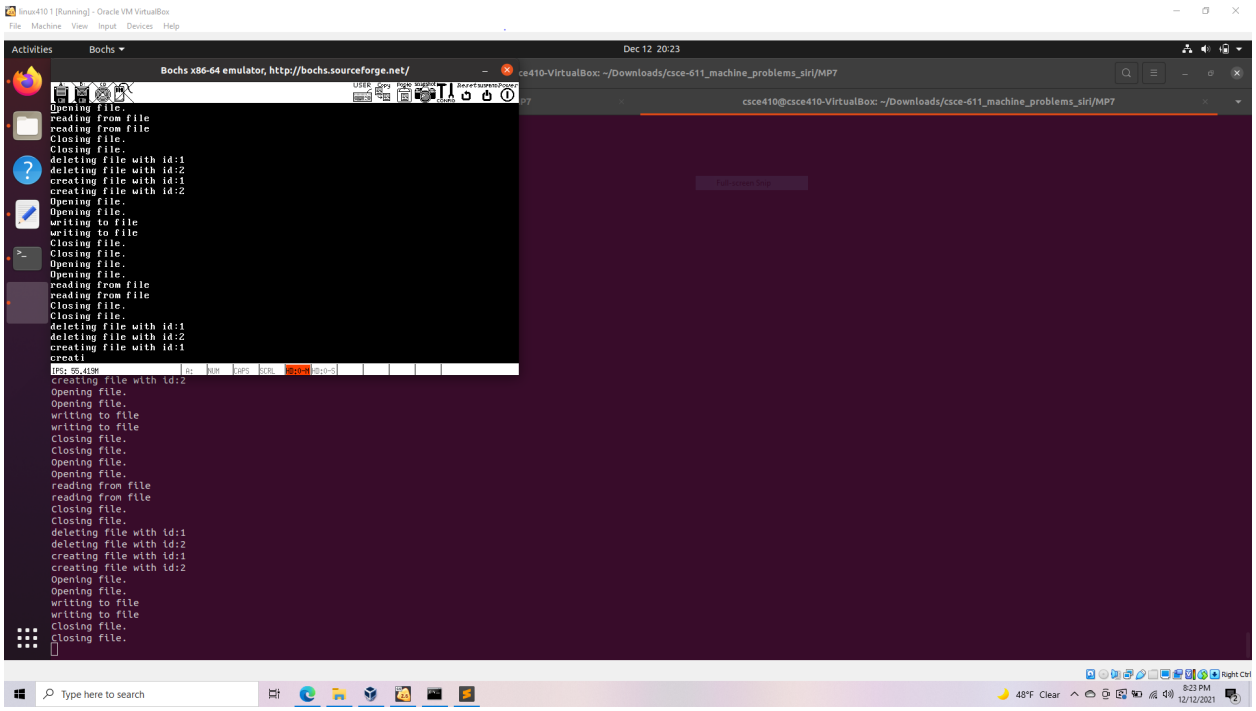
It contains the following variables:

- block_cache[SimpleDisk::BLOCK_SIZE]; : buffer handler for file . 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 current_position; indicates where the next character will be read from or written to

Functions present:

- File(FileSystem * _fs, int _id); constructor of the file handle. Intialises and updates the variables of a particular file into the class data structures from the data of the fs
- ~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. Reads _n characters and if _n characters are not there, then to the end of the file.
- int Write(unsigned int _n, const char * _buf); writes _n characters starting with current position. Increments file size to one block to 1 block if not enough. But never beyond that. Returns number of characters written
- void Reset(); sets the current position to beginning of the file
- bool EoF(); checks if the current position is at the end of the file and returns true in that case.



OPTION 1: IMPLEMENTATION OF FILE SYSTEM FOR FILES THAT ARE UPTO 64KB LONG.
In this case multiple blocks have to be assigned for one file. In this case , we can just maintain a pointer in inode for keeping the sequence of blocks associated with the file. Hence inode class have a extra array
Long blocks_list[BLOCK_SIZE]

Example: if the size of file is upto five blocks we can maintain a sequence of blocks like this :blocks_list = [2,3,4,5,7]

In the file system, in creating a new file, we assign a block list instead of a single block. And mark the blocks as free. In deleting a file, we delete the array in the inode associated, and mark all blocks in the array as free.

In the file class, the sequential reading and writing will have to be modified a bit. Instead of reading from a single block here we are reading from a sequence of blocks , thus the code has to be changed accordingly. Same goes for writing to a file. Eof of file is when the current position reaches to the end of the last block. 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.