CSCE - 611

MP6: Vanilla File System

DESIGN DOCUMENT

Submitted by: RAJENDRA SAHU (731008796)
github-repo : CSCE-410-611-fall-2021/rpsahu_CSCE611

INTRODUCTION:

This MP is focused on building a simple file system that supports only sequential access.

NO BONUS OPTIONS HAVE BEEN ATTEMPTED.

SOURCE CODES MODIFIED/ADDED:

1. *kernel.C: No change*
2. *file_system.H : Core Implementation for FileSystem & Inode class*
3. *file_system.C : Core Implementation for FileSystem & Inode class*
4. *file.H : Core implementation for File class*
5. *file.C: Core implementation for File class.*

CORE IMPLEMENTATION (Only the incremental additions/modifications have been mentioned, the rest of the code stays same as provided):

1. ***FileSystem constructor:***
   I don't initialize the state variables of FileSystem in this constructor to keep it simple. I have just allocated the inode and free list memory variable here.
2. ***FileSystem destructor:*** Deletes the dynamic allocated memory but also writes the latest state of free list and inode list to the disk.
3. ***FileSystem::mount():***It does 3 things.
   a. Initialize the disk attribute.
   b. Read the indode and free list into the memory variables
   c. Return false if the first two blocks are not occupied.
4. ***FileSystem::Format():*** it's a static function that writes initial invalid values to inode and set every free block free except first two in the free bitmap. These values are written to the disk

5. ***FileSystem::LookupFile():*** Goes through the inode list and looks for the inode that has the file id. Retunr that inode , return null if it doesn't find that inode.

6. ***FileSystem::CreateFile():*** It does the following things.
   - Lookup if the file id is already present
   - Find a free block for the file (one block long)
   - Find a free inode
   - Set the file properties in the inode
7. ***FileSystem::CreateFile():*** it first checks if the file exists or not, then invalidate the inode properties to the initial value of 0xFF and clear that particular free bitmap.
8. ***FileSystem::DiskOperation():*** A new helper function to disk transfers to and from disk when inside the FileSystem class or object. Calls the SimpleDisk Read() & Write().
9. ***FileSystem::GetFreeBlock():*** A new helper function that goes through the free list bitmap and return the first block number it finds free. If there are no free blocks return 0xFFFFFFFF.

10. ***FileSystem::GetFreeInode():*** A new helper function that goes through the indoe list and returns the first free inode ; return 0xFFFFFFF if no free inode.
11. ***File{} constructor:*** Initializes the state variables of the file class. Added inode and current position on top of the existing class variables. Also read the file block into the memory variable block_cache. This is where we open the file. We don't read the inode list since it would have been read while creating the file.
12. ***File{} Destrcutor:*** This is where the file is closed. The indoe list and the file block is written to the disk to store the latest state of the file.
13. ***File::Read():*** Function responsible read from the file(block cache variable). The reading policy is adhering to the handout viz don't read beyond the EoF or more than the given bytes.
14. ***File::Write():*** Function responsible for writing to the file. First we write to the block_cache and then we write that block_cache to disk. The wiritng policies in the handout are adhered.
15. ***File::EoF()***: Returns true if the current_postion is at the lock boundary else false.
16. ***File::Reset***: Reset the current_postion to zero
17. ***Inode class{} :*** Has file id, block number, file size and the file system hosting it as the attributes. The class don't have a constructor because the inode attributes are set manually in the FileSystem class.
18. ***Inode::inodes_to_from_disk :*** To transfer inode list to and from disk when inside the File class.

TESTING:

The kernel runs smoothly without hitting any assert. The kernel logic indefinitely calls exercise_file_system() which is the observed output.

GENERIC INSTRUCTIONS:

This code has been developed on the new code environment.
The TA is expected to unzip the file and the run *make.*
Please take note to run **sh ./copykernel.sh** when trying to copy the kernel to the floppy disk. The chmod properties are different in this code setup.

CODE MAINTENANCE:

The entire source code base has been pushed to the student's github repository. This have been done progressively over the course of development. This should help the TA in grading the assignment. Apart from the code base provided in the zip file submitted on canvas, the code can also be compiled from the github repo code base. solution to make the code complete.