Assignment Report : Memory Resident File System

Grp: 32
Suyash Damle 15CS10057
Arunansh Kaushik 15CS30004

Submitted files:

- A *myfs.h* file, which as a demonstration of the fact that the MRFS functions could be separated. (use *g++ myfs.h* to compile)
- A test files involving the use of this header file : "*myfs_test_case1.cpp*" and so on...
- A folder of check files that are uploaded in test case 1 to check *copy_pc2myfs()* function
- A makefile – used to compile all the codes

Specifics of Implementation:

- The MRFS has been implemented as a buffer of integers. This approach eases some int and bit reading / writing operations

The code has a number of low-level function implementations. These make the task of the higher – level functions easier:

- *add_block_to_file* – adds a block to the file and inserts the block idx in the right position in the file inode's direct or indirect block
- *get_file_block_list* - returns a list of the blocks currently in use by the file
- *get_file_pointer* – returns a pointer to the file's end – as a combination of block and byte index
- *get_free_inode* – returns a free inode index
- *get_free_block* – returns a free block index
- *find_file_by_name* – returns the inode of the file indicated by its name in the present directory
- *get_inode* – populates a struct from the MRFS corresponding to the given inode number and returns it
- *update_inode* – updates the inode in the MRFS with the one supplied as a struct by doing byte-level operations on the MRFS
- *create_inode* – creates an inode on the basis of the provided info

And several functions handling byte / bit level operations:

- *write_char_into_intArray*
- *write_bits_into_intArray*
- *read_char(/bits)_from_intArray*

The other functions as mentioned in the assignment statement are also written

Handling of race condition

The above use of low- level functions help in preventing the use of race condition, by providing small pieces of code that can be **locked to avoid race condition. Mutex locks** have been used on:
1) *find_free_block()* function – as it accesses a list of empty blocks in the super block
2) *find_free_inode()* functtion – same as above
3) *update_inode()* function – this function could lead to race condition – if 2 different processes try to update certain inode simultaneouly. Hence, it too is locked

- In case of the last test case, for checking the race condition and running the file system on 2 processes simultaneously, the **entire file system has been created on the shared memory**. The global variables are not changed after creation, and hence do not need to be in shared memory – though they too could have been put into it.