AOS - Assignment 6 : File System Report

- Murali K

Description of functions implemented :

fs_open(char *filename, int flags) :

To perform open(), I am first searching the root directory to find if a match occurs between the passed input parameter file name and names of the files present in the directory. Incase, a match is not found then, the file does not exist and I am returning error value along with printing a error message to let the user know. If a match/requested file has been found, then I am retrieving the inode number and fetching respective inode. Then I am making necessary changes in the file table entry. The mode is updated to the passed input parameter flag and state is updated to FSTATE_OPEN.

fs_close(int fd) :

While performing close, I am first verifying if the passed fd is in the valid range [0 to next_open_fd). If it is in the range then simply change the state to FSTATE_CLOSED and return, else print an error msg.

fs_create(char *filename, int mode) :

Before creating, I am first checking if the mode passed is O_CREAT. A check to confirm if the filename to be created is not already present in the root directory is done. Upon successful verification of above 2 constraints, I am making a file table entry to accommodate the request and updating the entry in root directory. Functions to store inode and set bitmask are provided already and I am making use of them. Finally, before returning I am updating the total inodes used.

fs_seek(int fd, int offset) :

Initially, I am verifying if the passed fd is in the valid range similar to the check done in fs_close(). If fd is valid then I am checking if the calculated offset value is non-negative. Incase, it is negative then I return before printing an error message. Offset is calculated by adding the passed offset to the file pointer. I am updating the seek value if calculated offset is valid(>=0). If it is negative I am resetting offset to zero.

fs_read(int fd, void *buf, int nbytes) :

In the read(), I perform initial checks to verify mode of file and state of file. If the mode is O_WRONLY (write only) then, since I cannot read the file, I am displaying an error message before returning error. Also, if the file state is FSTATE_CLOSED (closed) then I cannot read. Similarly, I am displaying an error message before returning error. If I pass these initial checks, I am obtaining the values of desired block and offset. The read request can be served if the block obtained is < INODEBLOCKS. I am looping till the requested bytes are processed. The provided function, bs_bread() is used in my code. Incase, a single block can accommodate the requested size, then I am directly reading the data and storing

it to the buffer. If a single block cannot accommodate out request, add the data to buffer and move to next offset until all the requested bytes are processed.

fs_write(int fd, void *buf, int nbytes) :

Similar to above functions, I am first verifying the mode and state of file. The write request cannot be processed if the mode is O_RDONLY (read only) and if the state is already FSTATE_CLOSED (closed). In these scenarios, I am printing an error message before returning error. As in fs_read(), I am obtaining the values of desired block, offset and checking if the write request can be served depending on if the block obtained is < INODEBLOCKS is True. I am attempting to loop till the requested bytes are processed. The provided function, bs_bwrite() is used in my code. If all the blocks are used up or if the requested size is more than maximum limit, I am displaying an error message. After obtaining the next available free block and checking if the requested size is available, I am processing the request by storing the inode and masking the index. Here, if we are successful in finding one free block with sufficient size, we store the data. Incase, if we need more than one block, I am storing data in the form of chunks. If we utilize all the free blocks and no more blocks are available to serve request (say big file), I am displaying an error message after attempting to save data as much as it can be accomodated.

Lessons Learned :

→ It is fascinating to understand how files and directories are maintained internally and all the steps, scenarios that need to be covered to implement basic file functions like read, write, open, close, seek, create.

→ By implementing this assignment, I have learned how to build a very basic file system and the structures that are needed.

→ How the above structures are accessed and keeping track of the entries, checking permissions, verifying availability of resources to perform operations successfully, is great to know.

→ There is much scope to address/improve the performance and efficiency of this basic file system implementation.