Assignment 6

File Systems

In this assignment I implemented a File System. Our header file fs.h consists of 5 structures that are needed to implement the file system properly. The structures are:

1. filetable: This structure contains information about the files like the state and the file pointer.
2. inode: This is to store all the inode information like the id, type, links, device, blocks etc.
3. dirent: This is used to store the directory entries with the information inode number and name.
4. directory: This is used to store the directory details.
5. fsystem: This is used to store the file system details i.e. about the inodes, blocks.

The functionalities that I have implemented in it include:

1. int fs_open (char *filename, int flags):

   In this function, we are iterating through all the directory entries and then setting the variable fileStatus to 1 once the file is found and we would be returning the file descriptor of that file.

2. int fs_close (int fd):
   In this function, we first need to check if the file is already closed, if it is then we would throw a system error then we would check for the validity of the file descriptor and then change the state of the file to CLOSED, if the file descriptor is not valid then we would print a statement of "File entry not found" and return a system error.

3. int fs_create (char *filename, int mode):
   In this function, first we would check for the existence of the file and if it does not exist then we would return a system error. If the file exists, then we would check if there are any inodes available. If the inodes are available too then we would allocate an inode to the file

and then create a new file table entry for this file. The index of this file entry table is returned at the end of the function.

4. int fs_seek (int fd, int offset):

   In this function, first we check if the file is closed, then we reset the offset to 0 in case the offset exceeds the boundary conditions. Seek allows the pointer to move in the file and it is used to write to a file. In the case of reading the file, we move the pointer to the position from where we have to start reading.

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

   In this function, we first check if the file is open or closed then we check for the validity of the file descriptor to make sure our file is valid and present in the filetable, then we need to check if the file has some content into it and then we would start reading the contents of the file. We are using the inode based on the file descriptor and then we obtain all the blocks associated with the file descriptor. We would read the nbytes starting from the seek pointer position into the process buffer and then we would update the seek pointer to the new position. We would return the number of bytes of data read from the file.

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

   In this function, we first check if the file is open or closed, then we check if the file is empty the we check for the validity of the file. Then we write the bytes from the process buffer to the file. The file pointer (seek pointer) is moved and its new position is updated. Then we return the size of the file at the end of the function.

Lessons learnt:

1. Understanding the implementation of file systems on Xinu.
2. Understood the concept of using memcpy to store the blocks of memory.
3. Understood how to read and write blocks of data to my file.