ADVANCED OPERATING SYSTEMS (FA15-BL-CSCI-P536-36187)

File Systems Assignment

Pralhad Sapre & Srivatsan Iyer

We have implemented the following functions

int fs_open(char *filename, int flags);

Handles opening the files using the flags of read, write and read/write. This function also returns an index into the file descriptor table which corresponds to the entry for this opened file.

int fs_close(int fd);

This function basically closes the file by setting the associated file descriptor to closed.

int fs_create(char *filename, int mode);

It creates a file using the specified filename in the only available mode of O_CREAT. It also implicitly does open the file returning a file descriptor to it.

int fs_seek(int fd, int offset);

It sets the position of the file pointer within the filetable structure. This is the point where all the reads or writes on the file will happen.

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

It reads the bytes of length nbytes into the buffer *buf from the current file pointer position. Reading may often span beyond a single block and these block may not necessarily be contiguous in memory.

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

It writes bytes of length nbytes from buffer *buf from the current file pointer position indicated by file descriptor fd. This is a complex function which often needs to allocate extra blocks of 512 bytes if the writing goes beyond the existing allocation for the file.

A big lesson learnt by us in this assignment is how Linux style file systems are implemented and managed. Referring to how Linux file systems work, we were able to replicate that paradigm for an in memory file system of Xinu.

Tasks breakup

Srivatsan implemented

- **fs_open()**
- **fs_read()**
- **fs_write()**

Pralhad implemented

- **fs_create()**
- **fs_seek()**
- **fs_close()**