**Implementing a Virtual File System**

**Background**

In lecture 9, we studied disk storage devices whose storage is divided into sectors of a track on a platter. The low-level disk controller translates this structure into a linear sequence of fixed-size blocks that can be addressed by their block numbers. We saw how file systems can be implemented on these block-oriented devices, and studied the architecture of a POSIX file and directory system.

In this assignment, you will have an opportunity to implement a simple version of this type of file system as a *virtual file system* that is supported by a file system driver, but is implemented as a user-space application. These virtual file systems use regular files on the host operating system as block-oriented devices.

Our fsx600 file system uses Fuse (File system in USEr space) to implement a file system that can be mounted and used just like a regular file system. The design of this file system has a superblock for the volume, inode and block bitmaps to track free inodes and blocks, inodes that represent files and directories, and a pool of blocks that can be allocated and used.The design of this file system is described in the lecture 9 slides and in this [PDF document](http://www.ccis.northeastern.edu/home/pgust/classes/cs5600/2019/Spring/resources/assignment-4-docs.pdf) that summarizes the design.

**Code**

The code for this assignment resides in two repositories. The"2019SPCS5600SV/assignment-4" repository is the code you will use to implement assignment. It contains the following files:

* fsx600.h - structure definitions
* homework.c - skeleton code for your Fuse file system
* blkdev.h - structure definitions for virtual block devices
* image.h - external declaration of a virtual block device
* image.c - implementaiton of a virtual device function
* misc.c - main program for virtual file system

The only file you will modify for this assignment is "homework.c" and all of your implementation will be in this file.

The "2019SPCS5600SV/assignent-4-util" repository has utility programs that you will use for development and testing. It contains the following files:

* fsx600.h - structure definitions (duplicate)
* mkfs-x6.c - program to generate empty file system image
* mktest.c - program to generate test file system image
* read-image.c - program to read and check file system images

The 'mkfs-x6' program is used to generate an empty disk image of a specified size (e.g. *./mkfs-x -size 50M big.img*).

The 'mktest' program creates a 1MB disk image that contains several files and directories (e.g. *./mktest test.img*).

When testing, be sure to create a clean disk image each time to ensure that your image has not become corrupted corrupted.

**Assignment**

This assignment is to implement the Fuse functions described in the attached design document to create a read/write virtual file system. The "assignment.c" code provides stubs for these functions. The design document and code comments describe some design simplifications that will make the code simpler to implement. You will also want to implement supporting functions, that factor out common functionality, as discussed in the design document and shown in the code.

Since this is a more complex assignment, you may work in teams of up to 4 people, and divide the implmentation, testing, and internal documentation work accordingly.

**Testing**

The main program for the assignment can be run in two modes:

* as an interactive program with a 'ftp' like command interpreter (e.g. *./assignment-4 -cmdline -image foo.img*).
* as a non-interactive program that mounts your file system at a location on disk (e.g. *./assignment-4 -image foo.img ~/fusefs*).

See the design documentation and the code for details on running the program in both modes. As part of this assignment, you should do at least basic "smoke-testing" of your code using the interactive mode, and then viewing the results using the "read-image" program. Note that it is possible to script the execution of the interactive program and to capture the output for comparison purposes. This is not required but may facilitate your testing process.

You may also test your implementation by mounting your file system as described earlier, and performing operations on files in the subdirectory using standard shell utilities and/or scripts. Be aware of some limitations of this implementation when performing operations on the mounted file system.