CSC345 Operating Systems
Class 31-32 File Systems Programming Project
April 18, 2016

Part 2: Implementing the file system
Completion Date: Monday, May 2

- **Canvas submission of deliverables by 11:00 a.m. on 5/2/16.**
- Demonstrations and assessment by appointment as assigned during the scheduled class times *(In order to permit a more relaxed demo session, I am seeking groups available on Monday between 1:00-2:00 p.m. Please self identify today.)*
- Remaining class sessions on 4/18, 4/21, 4/25, 4/28 are to be used for work on this project in the Linux lab. Attendance will be taken and participation will earn points; non-participation will result in lost points.

Please work within the same team as Part 1 of this project.

In this project, you are to design a simple file system that permits the user to *create*, *open*, *close*, *read*, and *write* files. The user must be able to create a hierarchical directory (tree) and be able to move through the tree through a *cd* (change directory) command. No soft or hard links need to be supported in the directory structure. You are to support *saving* your file system to the simulated disk (Part 1 of this project) and be able to *load* it at a later time. You must support a directory listing operation, *ls*, of your file system.

The simulated disk you created in Part 1 of this project supports disk access, but there is no organization to the disk. The simulated disk provides access to blocks (4096 bytes). You will now need to create structure for the disk in order to support an organized file system.

The file system must maintain a list of free disk blocks. Recall that we discussed approaches for this, including a bitmap, a linked list of available blocks, etc. This structure is to be stored on the simulated disk.

The inode contains the metadata for a file, and contains the mapping to the disk blocks of the file. For simplicity of this project, you may assume that only direct mapping is needed (no indirect pointers). You must configure your inode data structure to reflect this. Note that you must determine the max size of the files your simulation will support. Your simulated disk must hold multiple files of varying sizes. The size of some of the files must be larger than one block. You must also determine the maximum number of files supported by your file system. Remember that you must not run out of blocks, which are used for supporting structures, as well as for the inodes of the file systems, and for the data blocks of the files. Appropriate error messages are needed if thresholds are reached and the operation is not fulfilled. Do not corrupt your file system. You are to design a robust file system, albeit a simple one.

The file system that you create must support a data structure that maps the file names to the inode numbers. From an inode, you can access the mapping to the disk blocks of a

particular file identified by the inode. The file system must support a tree directory structure. Some blocks in the file system are data blocks and others are inodes. The structure of these two types of blocks is fundamentally different; however, for this project you may use the same data structure (a block is a block no matter its contents).

Once you determine the metadata needed within an inode, you need to develop the ability for the file system to allocate an inode to support creation of a file. You need to modify the directory structure that maps the user's filename to the inode number. (Where does this happen in the real world? When you create a file, it is placed in the current working directory, unless otherwise specified. You may assume that the file is placed in the current working directory for simplicity.) Finally, you must save this information to the simulated disk.

The file is more than the directory entry – you need to have the actual file! For this project, you may assume that the files are text files of varying sizes. To create a file, the file system must:

1. Get the file's inode from the simulated disk and load it into memory (a local copy is sufficient for this project)
2. Allocate the needed number of disk blocks (update your free list)
3. Update the inode for the file to reflect these assigned blocks by storing pointers to the data blocks in the inode
4. Store the data for the file
5. Flush these modifications to the simulated disk

Appending to a file is similar; however, the file must be open. This implies that a table of open files must be maintained. A single user is assumed in this project; you do not need to support multiple users. However, multiple files are to be supported. For simplicity, please assume sequential access for reading and writing; you do not need to support random access.

The file system must support the deletion of files. This is similar to creation in that the file's inode must be retrieved from the simulated disk and loaded into memory. Within the inode, the pointers to the disk blocks are found (remember that a varying number of data pointers will be used, depending on the size of the file) and removed. The disk blocks now are no longer used, to the free list must be updated. The modifications must be reflected in the state of the file system on the simulated disk (flushing the changes through to disk).

You will want to implement additional commands to illustrate the robustness of your system. For example, you may want to be able to view a summary of free blocks. Develop other useful meta-commands to give information about the file system that will be useful for debugging and for demonstrating your implementation.

- Your project must be original work that is designed and developed by your team's members. If you use online resources for reference, you must cite these appropriately. You are not permitted to employ code developed by other programmers. If you have questions regarding this point, please visit with me to discuss.
- Standard programming practices must be used, including coding documentation
- **You may not post your work on the File Systems Implementation project (Part 1 or Part 2) to a public repository** (such as git or other coding sites); posting in that manner would be a violation of the TCNJ Academic Integrity Policy

Information on deliverables will be forthcoming. In addition to your project implementation, you will need to turn in a design document and a report on your implementation and testing. You will spend time demonstrating and discussing your project with Dr. Knox. Each group will have about 15 minutes for this assessment.