

# CSCI.4210 Operating Systems

## Fall, 2009

### Programming Assignment 6

Let's write a simulation of a file system. Since this does not require any system calls, you can use whatever language you want. Make sure that you have a comment at the top telling the graders how to compile and run your program.

All data for all files, including the Master File Records and the freelist, will be stored on the disk. The disk consists of 256 blocks; each block has 64 bytes.

We will use a simplified implementation of the NTFS file system. You might want to reread section 11.8, although we will not be implementing many of these features. In our system, each file will have a 64 bit Master File Record (MFR).

A block can hold one of four structures:

- **A Master File Record** (one per file) This has an **R** (for regular file) in the first byte. The second byte will hold either a **B** (for big) or an **L** (for little). The third and fourth bytes are a short int containing the number of bytes in the file.

A little file is a file with less than 61 bytes and is stored completely in the MFR (NTFS does this, although the sizes are different). A big file is a file which has 61 or more bytes and is stored in separate blocks. To implement a big file, you will use runs. Each run consists of the starting block and the number of contiguous blocks. These should be stored as consecutive pairs of short ints in the remainder of the MFR block. There should be room for 15 pairs.

- **A Directory** A directory consists of a list of pairs, each pair consists of a name and the block number of the MFR (a short int). File names should be null terminated strings. The maximum name length is 7 characters, leaving at least one byte for the terminal null. In a directory block, the first byte is a **D**, the second byte is always an **L**. The third and fourth bytes are a short int containing the number of entries. One directory block can only contain 6 entries, (each entry has 8 bytes for the name, and 2 for the block number of the MFR). The first entry of each directory should be **..** and should point to the parent. The parent of root is itself. No directory can have more than 6 entries, and one always has to be the parent.
- **Data** up to 64 characters.
- **The free list** You can implement the free list any way that you want.

Your program will take two arguments. The first is a filename. This file will contain a number of commands. The program should open this file and read and execute the commands in order.

The second arguments will be either `-S` for silent or `-V` for verbose. In silent mode, nothing is written to standard output, although error messages should be written to standard error. In verbose mode, each command is displayed on standard output as it is executed, and all disk block allocations and deletions should be written as well.

Here are the possible commands.

- **export** *filename externalfile* reads the contents of a file called *filename* on your file system and copies the contents to a file on the host compute called *externalfile*.
- **import** *filename externalfile* creates a new file called *filename* in your filesystem; opens a file on the host operating system with the name *externalfile*, and copies the contents of the external file into the new file.
- **copy** *filename1 filename2* creates a new file and copies the contents of the old file to the new file
- **rename** *filename1 filename2* renames a file
- **mkdir** *directoryname* creates a new subdirectory in the current working directory
- **delete** *filename* deletes a file, returning its MFR and all of its data blocks to the free list and removing it from the directory
- **chdir** *directoryname* Change the current working directory.
- **append** *filename externalfile* Appends the contents of a file on the host computer called *externalfile* onto an existing file. If the file *filename* does not exist, this is an error.
- **rmdir** *directoryname* removes a directory from the file system, and deletes all files in that directory, in its subdirectories, in their subdirectories etc.
- **quit** Your program should write the contents of the file system, block by block, to a file on the host system called `dump.txt` and terminate. The contents of each block should be on a single block starting with the block number, then a space, then the contents on a single line. If the data happens to contain newlines, replace the newline character with a space so that the contents will be on a single line. You do not need to write the contents of the freelist block or unused blocks.

Filenames can be either relative (in the current directory) or absolute (the first char is a slash, so start at the root). They may be relative paths such as `subdir1/subdir2/file.txt`

You have to keep track of the current directory. When a program starts, the current directory is always the root directory.

The freelist should always be in block zero. The MFR of the root directory (/) should always be in block 1. Your program should initialize the root directory before reading any commands.

If a command cannot be executed because there is not enough room in the file system, display a suitable error message, write `dump.txt` and terminate.

Your program should do routine error checking and display appropriate error messages on standard error.

Note that files can grow, but they cannot shrink (this should make your life much easier)

The project is due at 11:59 on Friday, November 20.