CSCI.4210 Operating Systems
Fall, 2008
Programming Assignment 4

Let's write a simulation of a file system.

All data for all files (including the inodes and the freelist) will be stored on the disk. The disk consists of 256 blocks; each block has 32 bytes. Use this structure:

```
unsigned char disk[256][32];
```

Your program will first run a function `initialize()` which I will provide. This will open a file called `filesystem.txt` which reads data into `disk`. If this file does not exist, it will initialize the disk.

Your program will then open a file which will read a number of commands. The name of the file will be passed in as an argument to your program.

Here are the possible commands.

read *filename* reads the entire contents of a file (note that there is no concept of opening a file), and displays it on the terminal.

write *filename string* writes the string to a file. If the file already exists, it deletes the previous contents. If the file does not exist, it should be created

append *filename string* appends string onto the contents of a file. If the file does not exist, it should be created.

copy *oldfilename newfilename* creates a new file and copies the contents of the old file to the new file

rename *oldfilename newfilename* renames a file

delete *filename* deletes a file, returning its inode and all of its data blocks to the free list and removing it from the directory

list Lists all files and their size

quit Your program should write the data to a file called `filesystem.txt` (I will provide a function to do this.) and then terminate.

Strings will always be terminated by newlines, but the newline is not part of the string. Strings will not contain null characters or newlines.

An inode will take up one block. The first four bytes will contain the size (number of bytes) in the file (an integer). The remaining 28 bytes will be pointers to blocks (unsigned char) containing data. No file will require more than 28 blocks ($28 \times 32 = 896$ characters),

You will need a freelist. This should be a bitmap in block 0. Since there are 256 blocks in all, and one block has 256 bits, everything fits (What a coincidence!). I will provide two functions:

- `void setbit(int bitnum, int value)` which sets a bit. The first argument must be between 0 and 255, the second must be either 0 or 1.
- `int getbit(int bitnum)` returns the value of a bit. The argument must be between 0 and 255, and the return value will be either 0 or 1.

You should have a directory, and this should have its own inode. The inode for the directory will be in the second block of the disk (block 1). The inode pointers will point to directory blocks. Each directory block will have four entries, consisting of a name (up to six characters), a space for the terminal null, and a pointer to the inode of that file. No file name will have more than six characters and the directory will not have more than 112 (28×4) entries.

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

I would like this to be deterministic, so that every fully correct program would have the same output for the same input (it won't happen). To facilitate this, whenever you need to allocate a new block, use the lowest numbered free block.

The project is due at 11:59 on Tuesday, October 28.