EECS 211 – Spring Quarter, 2010
Program 7

Due Tuesday, June 1st, 2010 at 11:59PM

In this assignment and the next assignment we will create and store files on disks. In this assignment we will provide for storing "file descriptors" – records that contain the file name and related information. In the last assignment we will read and store the actual file contents. In preparation for storing files, we will need to provide for storing/retrieving blocks of data to/from a disk drive and for disk formatting, that is for initializing the disk with a file system containing no files yet.

Background:

Files are stored on disks in two separate sections. The first section, the file descriptor, contains information about the file, such as the file's name, the number of bytes in the file, whether or not the file is write protected, where the actual data of the file is stored on the disk, and many other bits of information about the file. The second section is the actual data itself. These two sections of a file are usually not stored together on the disk, i.e., are usually not stored in contiguous blocks. (In fact, in real file systems in UNIX or WINDOWS, even the data section itself is generally spread around the disk in blocks that may be far apart.) Very often, the file descriptor section for at least the files in the root directory is stored in a special place on the disk, usually in the first blocks of the disk. The descriptor section of files that occur in subdirectories and user folders, as well as the actual data, are then stored in available blocks throughout the rest of the disk drive.

In this assignment we will implement the first half of file storage on a disk drive and use a very simple scheme for storing the file descriptors. Each computer will have a fixed, maximum number of files that can be stored on that computer. (Of course, at any given time the computer may not have that many files.) The file descriptors will be stored in the first blocks on the drive. Our project will not implement sub-folders or directories; our simulated computers will have only a root directory. For this assignment the files will not contain any actual data; that will come in the last assignment.

Assignment:

(1) Add a new data member, **maximum_number_of_files**, to your **computer** class. The maximum number of files for a PC is 8; the maximum number of files for a server or printer is 16. Fix your constructors to have these values stored when a computer is created.

(2) Add the following two public function members to the class **diskDrive**:

- **void storeBlock(char *d, int blk)** – The argument **d** points to a block of 64 bytes which is to be stored on the disk. The integer **blk** represents a block

number. If **blk** does not represent a valid block number for that disk, print an error message and do not store the data.

- **void retrieveBlock(char *d, int blk)** – The reverse of the above function, i.e., copy block **blk** from the disk to **d**. If blk does not represent a valid block number for that disk, print an error message and do not store the data.

(3) Add a new **struct**, called **fileDescriptor**, to your definitions header file. A file_descriptor has three data members:

- A file name – an array of 8 characters.
- A file length – an integer.
- The block number of the first block on the disk containing the file data.

Names will not be strings but rather simply characters in sequence with blank fill to the right. We will not use the third data member in this assignment; it will be used in the last assignment. (Note that a file descriptor is 16 bytes long, so one disk block can hold exactly four file descriptors.)

(4) Add a public member function **formatDrive** to the **diskDrive** class. This function takes an integer argument that tells the maximum number of file entries that drive will hold. The function should mark all the blocks on the drive free. It should then fill all the file entries on that drive with the default non-file descriptor (name field is all zeroes, length is –1, first block is 0) and mark the corresponding blocks as used.

Change constructors in the **PC**, **server**, and **printer** classes so that the disk drive is formatted when the computer is created.

(5) Add the following functions to the protected section of the **computer** class:

- **int findFreeFileDescriptor()** – returns the integer index of the first unused file descriptor entry.
- **void getFileDescriptor(int n, fileDescriptor *fd)** – copies file descriptor n from the disk to fd.
- **void putFileDescriptor(int n, fileDescriptor *fd)** – copies fd to file descriptor n on the disk.

(6) Add a public member function **void createFile(char *n, int len)** to the **computer** class. The parameters are a file name and length. For now, this function only tries to create a file descriptor entry on its disk drive. If there are no spare file descriptors, print an error message.

(7) Add a public member function **void printDirectory()** to the **computer** class. This function prints one line for each file descriptor. If the file descriptor does not currently have a file (i.e., its length is –1) then print "not in use" for that file. Otherwise, print the

file name, length and first block number for the file. The output could look, for example, like:

File entry 0 – not in use
File entry 1 – file name = program8, length =241, first block on disk=0
…

Requirements and Specifications:

(1) File names are character sequences (not strings) of up to 8 characters. File names less than 8 characters are to be blank filled on the right (i.e., at the end of the name). You may assume that all file names in the test data will be legitimate, and you therefore do not have to check for more than 8 characters. (Note that if a file name is the full 8 characters long, the token you parse from the command line for this name will contain 9 characters because it is a string and has the string terminator.) The **create_file** function will need to convert the token containing the name into an 8-byte blank padded character array as well as convert the length token from string to integer.

(2) Remember that a computer cannot access bytes on the disk drive directly. It must determine the block number and read that block out into a buffer. If the computer wants to change some of the data, it has to make the change at the appropriate place in the buffer and then write that buffer back to the disk block.

On the other hand, note that a disk drive itself can access (both read and write) any part of its storage area. Therefore, the format function in the **diskDrive** class does not need to use buffering.

(3) You will be filling in two more cases in the switch in your main program – create_file and ls. For create_file, your code should find the computer in the network. If a computer with the indicated name does not exist, print an appropriate error message and abort the command. Otherwise, have the computer try to create the file. For the ls case, again search for the indicated computer in the network. If it does not exist, print an error message and abort the command. If it does, have that computer print its directory.

(4) Your project should process the command list in the file p7input.txt (http://www.eecs.northwestern.edu/~iraicu/teaching/EECS211/code/p7input.txt).

(5) The format of the create-file command line is:

```
create_file  computer-name   file-name  file-length
```

For example,

```
create_file   "larrys laptop"   program7    241
```

The file-length token tells the number of bytes that the file will contain.

Comments, suggestions, and hints:

(1) Remember that data can only be read or written to the actual disk drive in blocks of 64 bytes. Also remember that computers can hold up to 8 or 16 files, depending on the machine. So the root directory occupies the first two or four blocks of the disk drive. Therefore, functions in the **computer** class which deal with file descriptors (e.g., **createFile** or **PrintDirectory**) have to buffer out four file entries at a time. Moreover, when creating a new file descriptor the information for the new file (name, length, starting block) has to be copied to the buffer area, and then the buffer written back to the appropriate block on the drive.

(2) A suitable order for doing this assignment is:

1. Add the new data member to the **computer** class and the new **struct** for file descriptors. There are no function members in the file descriptor **struct**, so there is really no testing to be done for this part, just typing.
2. Implement the store/retrieveBlock functions and the format function in the **disk_drive** class and add calls to the format function in the constructors. You could use the debugger for this and examine the disk drive blocks in the watch window or simply have some cout statements that print raw bytes in hex format to the output window.
3. Implement the print directory function. If you do this now you won't be able to fully debug it because you won't have any real file descriptors. But at least it would be ready to help you do the next part.
4. Implement the remaining functions.

Test data:

```
add_network_node PC pc1 4096 Larry
add_network_node PC pc2 384 Julia
system_status
create_file pc1 f1 20
create_file pc1 f2 70
create_file pc1 f3 70
create_file pc2 help 200
create_file pc2 dear_mom 5397
create_file pc3 ouch 100
create_file pc1 f4 20
create_file pc1 f5 70
create_file pc1 f6 70
create_file pc1 f7 20
create_file pc1 f8 70
create_file pc1 f9 70
ls pc1
ls pr2
ls pc2
halt
```