O.S. Lab 7: File I/O Part II, Shell Part III

Introduction

In this project you will implement functions for deleting and writing files, and add several new commands to your shell. At the end of this project you will have a fully functional single-process operating system about as powerful as CP/M, an early PC operating system.

Like reading files, writing files requires that you understand the file system and how it keeps track of the names of the files on the disk and their locations. As such there is no new background required for this project. However, you may want to review the material on the disk directory and the disk map given previously.

What You Will Need

Download the linked files from the web page and add them to your existing files. These files include the executable program **lab7.c** (seen at right) and its data file **fpc02** that we will use for testing purposes in this lab.

```
void main()
{
  char buffer[12288]; int size;

  /* Step 1 - load/edit/print file */
  interrupt(33,3,"fpc02\0",buffer,&size);
  buffer[5] = '2'; buffer[6] = '0';
  buffer[7] = '1'; buffer[8] = '9';
  interrupt(33,0,buffer,0,0);
  interrupt(33,0,"\r\n\0",0,0);

  /* Step 2 - write revised file */
  interrupt(33,8,"fall19\0",buffer,size);

  /* Step 3 - delete original file */
  interrupt(33,7,"fpc02\0",0,0);

  /* Step 4 - terminate program and resume shell */
  interrupt(33,5,0,0,0);
}
```

When studying the code at right, note that 12288 bytes is the maximum file size. All that happens here is that *main()* reads the text file into the buffer, edits it slightly, prints out the new version and saves it while deleting the old one before terminating. After doing a **./compileOS.sh** to rebuild the disk and compile, compile and **loadFile** both the **lab7** executable and **fpc02** to add your test files to the disk image before executing **bochs**. Execute **lab7** from the shell prompt to test. Run **ddir** to make sure both the new **fall19** file was saved and **fpc02** deleted. Also remember to examine the contents of **fall19** and to test the "file not found" error (see below).

Updating Interrupt 33

The final functionality to be added to the o.s. includes two new functions. This code should be inserted into **kernel.c** before the **error()** function. Additionally, once these functions are complete, you will have to add new interrupt 33 calls to access them:

Function	AX=	BX=	CX=	DX=	Action
	…				
Delete a disk file.	**7**	Address of character array containing the file name.			Call **deleteFile(bx)**
Write a disk file.	**8**	Address of character array containing the file name.	Address of character array where file contents will be stored.	Total number of sectors to be written.	Call **writeFile(bx,cx,dx)**
		…			

We will now consider the writing of these described functions.

Step 1: Writing a File

First add the

void writeFile(char* name, char* buffer, int numberOfSectors)

function to the kernel that writes a file to the disk. The function should be called with a character array holding the file name, a character array holding the file contents, and the number of sectors to be written to the disk.

Writing a file means finding a free directory entry and setting it up, finding free space on the disk for the file, and setting the appropriate map bytes. Your function should do the following:

1. Load the disk directory (disk sector 257) and map (disk sector 256) to separate 512 byte character arrays (buffers).
2. Search through the directory, doing two things simultaneously:
    a. If you find the file name already exists, terminate with interrupt 33/15, function 1 (error one, "duplicate or invalid file name", see below).
    b. Otherwise find and note a free directory entry (one that begins with zero)
3. Copy the name to that directory entry. If the name is less than 6 bytes, fill in the remaining bytes with zeros.
4. To write the actual file to disk:
    a. Find an area of free space on the disk large enough to accommodate the file being written by searching through the map for consecutive zeros.
    b. For each sector to be used, set its map entry to 255.
    c. Add the starting sector number and length to the file's directory entry.

    d. Write the buffer holding the file to the correct sectors.
5. Write the map and directory sectors back to the disk.

If there are no free directory entries or not enough free sectors left, your **writeFile** function should terminate with interrupt 33/15, function 2 (error two, "insufficient disk space", see below). Don't forget to add **writeFile** to interrupt 33 as described above.

To test edit the file you read slightly and save the revised copy under a new name, as the **lab7** program does. Running a **hexdump** on the disk image should verify that this worked.

Step 2: Deleting a File

Now that you can write to the disk, you should be able to delete files. Deleting a file takes two steps. First, you need to change all the sectors reserved for the file in the disk map to free. Second, you need to set the first byte in the file's directory entry to zero.

Add a **void deleteFile(char* name)** function to the kernel. Your function should be called with a character array holding the name of the file. It should find the file in the directory and delete it if it exists. Your function should do the following:

1. Load the disk directory and map to 512 byte character arrays as before.
2. Search through the directory and try to find the file name. If you can't find it terminate with interrupt 33/15, function 0 (error zero, "file not found").
3. Set the first byte of the file name to zero.
4. Step through the sectors numbers listed as belonging to the file. For each sector, set the corresponding map byte to zero. For example, if sector 7 belongs to the file, set the *eighth* map byte to zero (index 7, since the map starts at sector 0).
5. Write the character arrays holding the directory and map back to their appropriate sectors.

Notice that this does not actually delete the file from the disk. It just makes the disk space used by the file available to be overwritten by something else. This is typically done in operating systems; it makes deletion fast and un-deletion possible. Test as above: create a file and then delete it, followed by a check of the **hexdump**.

Error Checking

Once you have the basic functionality in place and working you should remember to check each of the possible errors one at a time.

1. **File not found.** Create a disk image without the **fpc02** file and execute **lab7**. You should also test by creating the disk image then try and execute **lab7** a second time.

2. **Bad file name.** Try and save the revised file as **fpc02**.

3. **Disk full.** Cut-and-paste 40 copies of the "write file" step and try and save 40 different copies of your revised file. (Edit so attempting to save to 40 different names.) Your kernel should terminate with an error after the first 30 copies or so are saved. Double-check the final disk image with a **hexdump**.

In all cases your kernel should print out the correct error message and then resume execution of the shell. Note that this is just a partial list of potential test cases, you should think of others yourself.

To summarize, BlackDOS recognizes and reports these error messages:

Code (BX=)	Error message	Notes
0	File not found.	Carry-over from previous project
1	Bad file name.	
2	Disk full.	
Default	General error.	Carry-over from previous project
No code	Bad command.	Exists only in shell
No code	Bad or missing command interpreter.	Related only to shell

The first four are tied to the **error** function (interrupt 33/15) in the kernel, the last two reported directly from the shell.

Part II. Updating the Shell

Finally edit your remaining stub code in the shell to *implement* these commands:

- **copy** *file1 file2*
  Create *file2* and copy all bytes of *file1* to *file2* without deleting *file1*. Use only interrupt 33 for reading and writing files.

- **remv** *filename*
  Delete the named file (via interrupt 33/7) by clearing the appropriate map entries and marking it as deleted in the directory.

- **twet** *filename*
  Create a text file. Prompt the user for a line of text (shorter than 140 characters). Store it in a buffer and write this buffer to a file called *filename*.

For these commands issue a "duplicate or invalid file name" error (interrupt 33/15, error 2) if the file to be written or deleted has a name beginning with a capital letter, as discussed previously. (Most other error tests should have been built into the kernel.)

Conclusion

Congratulations! You have now created a fully functional command-line based single process operating system that is almost as powerful as Bill Gates' first MS-DOS version. When finished, submit a .tar or .zip file (no .rar files) containing all needed files (**bootload**, **bdos.txt**, **kasm.o**, **compileOS.sh**, **kernel.c**, **config**, **map**, **loadFile**, **shell.c**, **basm.o**, **Stenv**, **Help**, **Ddir** and **README**) to the drop box. Make sure that your disk image includes the **lab7** executable (original version) and **fpc02**. **README** should explain what you did and how the t.a. can verify it. Your .zip/.tar file name should be your name.

Last updated 8.5.2019 by T. O'Neil, based on material by M. Black and G. Braught. Previous revisions 1.8.2019, 1.26.2018, 2.2.2017, 1.25.2016, 1.27.2015, 2.21.2014, 2.22.2014, 11.3.2014.