

CSC 565 - Operating Systems
Spring, 2008

Project D - Writing Files and Improving the Shell

Objective

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

What you will need

You will need the same utilities and support files that you used in the last project, and you will need to have completed all the previous projects successfully. There are no new files you need to download for this project.

Step 1: Write Sector

The first step is to create a writeSector function in kernel.c to go with the readSector function. Writing sectors is provided by the same BIOS call as reading sectors, and is almost identical. The only difference is that AH should equal 3 instead of 2. Your write sector function should be added to interrupt 0x21, and should be handled as follows:

Write Sector

AX = 6

BX = address of a character array holding the sector to write

CX = the sector number

If you implemented readSector correctly, this step will be very simple.

Step 2: Delete File

Now that you can write to the disk, you can 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 0x0 .

You should 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 Directory and Map to 512 byte character arrays
2. Search through the directory and try to find the file name.
3. Set the first byte of the file name to 0x00.
4. Step through the sectors numbers listed as belonging to the file. For each sector, set the corresponding Map byte to 0x00. For example, if sector 7 belongs to the file, set the 7th Map byte to 0x00 (actually you should set the 8th, 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 it available to be overwritten by another file. This is typically done in operating systems; it makes deletion fast and undeletion possible.

Adding an interrupt

You should add delete file to interrupt 0x21:

Delete File

AX=7

BX=address of the character array holding the file name

Adding to the shell

You should make a “delete filename” command to the shell. Try loading *message.txt* onto floppy.img. When you type *delete messag*, the interrupt should be called and *messag* should be deleted. When you type *type messag*, nothing should be printed out.

You should open up floppy.img with hexedit before and after you call *delete messag*. You should see the appropriate Map entries changed to 0 and the file marked as deleted in the Directory.

Step 3 - Writing a file

You should now add one last function to the kernel *void writeFile(char* name, char* buffer, int numberOfSectors)* 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. You should then add *writeFile* to interrupt 0x21 as follows:

Write file:

AX = 8

BX = address of character array holding the file name

CX = address of character array holding the file to be written

DX = number of sectors

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 Map and Directory sectors into buffers
2. Find a free directory entry (one that begins with 0x00)
3. Copy the name to that directory entry. If the name is less than 6 bytes, fill in the remaining bytes with 0x00
4. For each sector making up the file:
 5. Find a free sector by searching through the Map for a 0x00
 6. Set that sector to 0xFF in the Map
 7. Add that sector number to the file's directory entry
 8. Write 512 bytes from the buffer holding the file to that sector
9. Fill in the remaining bytes in the directory entry to 0x00
10. Write the Map and Directory sectors back to the disk

If there are no free directory entries or no free sectors left, your writeFile function should just return.

Step 4 - Copying a file

Write a copy command for the shell. The copy command should have the syntax “copy filename1 filename2”. Without deleting filename1, the copy command should create a file with name filename2 and copy all the bytes of filename1 to filename2. Your copy command should use only the interrupt 0x21s for reading and writing files.

You can test this by loading *message.txt* onto floppy.img. At the shell prompt, type *copy messag m2*. Then type *type m2*. If the contents of message.txt print out, your copy function most likely works.

You should check the directory and map in floppy.img using hexedit after copying to verify that your writing function works correctly.

If your copy function works correctly, you should be able to copy *shell* to another file. Then try executing the duplicate shell. If you get the shell prompt, it works correctly.

Step 5 - Listing the directory contents

Write a shell command “dir”. This command should print out the files in the directory. Only extant (not deleted) files should be listed. Optionally, you might also print out the sizes of the files in sectors.

Step 6 - Creating a text file

Write a shell command “create filename”. This command should allow you to create a text file. The create command should repeatedly prompt you for a line of text until you enter an empty line. It should put each line in a buffer. It should then write this buffer to a file.

Test this step by calling *type filename* and see if what you typed is printed back to you.

You have now created a fully functional command-line based single process operating system! Your operating system has nearly the same functionality that CP/M did in the early 1980s, and is almost as powerful as MS-DOS version 1.

Submission

You should submit a .zip or .tar file (no .rar files please) containing all your files and a shell script for compiling on Blackboard Digital Dropbox. Be sure that all files have your name in comments at the top. Your .tar/.zip file name should be your name. You must include a README file that explains 1) what you did, and 2) how to verify it.