German University in Cairo
Faculty of Media Engineering and Technology
Dr. Aysha Alsafty
Eng. Nourhan Ehab
Eng. Eslam Osama

CSEN 602-Operating Systems, Spring 2017
Milestone 3: Writing Files and Enhancing the Shell

Due on Thursday 6/4/2017 by 11:59 pm

Milestone Objective

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

Before you start

You will need the same files you used in the last milestones, and you will also need to have completed the previous milestones successfully. Create a copy of your Milestone 2 folder and continue writing in `kernel.c` and `shell.c`.

Reminder: The File System

The main purpose of a file system is to keep a record of the names and sectors of files on the disk. The file system in this operating system is managed by two sectors at the beginning of the disk. The Disk Map sits at sector 1, and the Directory sits at sector 2.

The Map tells which sectors are available and which sectors are currently used by files. This makes it easy to find a free sector when writing a file. Each sector on the disk is represented by one byte in the Map. A byte entry of 0xFF means that the sector is used. A byte entry of 0x00 means that the sector is free.

The Directory lists the names and locations of the files. There are 16 file entries in the Directory and each entry contains 32 bytes (32 times 16 = 512, which is the storage capacity of a sector). The first six bytes of each directory entry is the file name. The remaining 26 bytes are sector numbers, which tell where the file is on the disk. If the first byte of the entry is 0x0, then there is no file at that entry. For example, a file entry of:

```
4B 45 52 4E 45 4C 03 04 05 06 00 00 00 00 00 00 00 00 00 .. .. ..
K  E  R  N  E  L
```

means that there is a valid file, with name "KERNEL", located at sectors 3, 4, 5, 6. (00 is not a valid sector number but a filler since every entry must be 32 bytes).

German University in Cairo
Faculty of Media Engineering and Technology
Dr. Aysha Alsafty
Eng. Nourhan Ehab
Eng. Eslam Osama

Step 1: Write Sector

The first step is to create a `writeSector` function in `kernel.c`. 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 when calling interrupt `0x13`[1]. Your `writeSector` function should be added to interrupt `0x21`. When interrupt `0x21` is called with `AX=6`, `writeSector` should be called. `BX` will contain the address of the buffer that will be written to the specified sector, and `CX` will contain the sector number where the buffer will be written. If you implemented `readSector` correctly, this step will be very simple. Write your own test code to test this part.

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 specifically 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 `0x07` belongs to the file, set the 8th Map byte to `0x00` (the 8th byte is at index 7 of the map since it starts by a byte representing whether sector 0 is free or full).
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 un-deletion possible.

You should add `deleteFile` as an interrupt `0x21` call. When interrupt `0x21` is called with `AX=7`, `deleteFile` should be called. `BX` will contain the name of the file to be deleted.

You should test your function by adding the following lines in your main function:

[1]Recall Step 3 in Milestone 2

```
char buffer[13312];
makeInterrupt21();
interrupt(0x21, 7, "messag\0", 0, 0); //delete messag
interrupt(0x21, 3, "messag\0", buffer, 0); // try to read messag
interrupt(0x21, 0, buffer, 0, 0); //print out the contents of buffer
```

then run the script file to compile `kernel.c` and load `message.txt` to `floppya.img`. If your delete function works, the message inside `message.txt` will not be printed out anymore. You can also verify that `deleteFile` works by opening `floppya.img` in `hexedit` and examining the directory entry for `messag`. The first byte should be set to `0x00`. Additionally, The corresponding bytes representing the sectors making up `messag` should be set to `0x00` in the map.

Step 3: Writing a file

You should now add one last function to the kernel
`void writeFile(char* name, char* buffer, int secNum)` 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` as an interrupt `0x21` call. When interrupt `0x21` is called with `AX=8`, `writeFile` should be called. `BX` will contain the name of the file to be written, `CX` will contain the address to the array holding the file contents and `DX` will contain the number of sectors to be written.

In order to write a file successfully, you should find a free directory entry and set it up, find free space on the disk for the file, and set 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:
   - Find a free sector by searching through the Map for a 0x00.
   - Set that sector to 0xFF in the Map.
   - Add that sector number to the file's directory entry.
   - Write 512 bytes from the buffer holding the file to that sector.
5. Fill in the remaining bytes in the directory entry to `0x00`.
6. Write the Map and Directory sectors back to the disk.

German University in Cairo
Faculty of Media Engineering and Technology
Dr. Aysha Alsafty
Eng. Nourhan Ehab
Eng. Eslam Osama

If there are no free directory entries or no free sectors left, your `writeFile` function should print an error message and return. You can test your function by writing the following in your main function:

```
int i=0;
char buffer1[13312];
char buffer2[13312];
buffer2[0]='h'; buffer2[1]='e'; buffer2[2]='l';  buffer2[3]='l';
buffer2[4]='o';
for(i=5; i<13312; i++) buffer2[i]=0x0;
makeInterrupt21();
interrupt(0x21,8, "testW\0", buffer2, 1); //write file testW
interrupt(0x21,3, "testW\0", buffer1, 0); //read file testW
interrupt(0x21,0, buffer1, 0, 0); // print out  contents of testW
```

If your function works, then "hello" will be printed out. You can also verify that `writeFile` works by opening `floppya.img` in `hexedit` and examining the directory sector. You should see an entry for `testW`.

Step 4-1: Enhancing the shell: delete command
You should add a `delete filename` command to the shell. Try loading `message.txt` onto `floppya.img` and set your kernel to load and execute the shell program just like you did in Milestone 3. When you type `delete messag`, the interrupt should be called and `messag` should be deleted. When you type `view messag`, nothing should be printed out. You should open up `floppya.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. Note that your shell should be user friendly. For example, if the user tries to delete or view a file that does not exist, he should be prompted that the file does not exist.

Step 4-2: Enhancing the shell: copy command
In this step, you will 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 interrupt `0x21` calls for reading and writing files. If the user tries to copy a file that does not exist, he should be prompted. You can test this by loading `message.txt` onto `floppya.img`. At the shell prompt, type `copy messag m2`. Then type `view m2`. If the contents of `message.txt` print out, your copy function works!! :). You should check the directory and map in `floppya.img` using `hexedit` after copying

to verify that your writing function works correctly.

Step 4-3: Enhancing the shell: dir command
In this step you will write a `dir` command to list the contents of the directory. This command should print out the files in the directory. Only existent (not deleted) files should be listed. You should also print out the sizes of the files in sectors.

Step 4-4: Enhancing the shell: create command
In this step, you will write a create command to create a text file. The create command should have the syntax `create filename`. 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 `view filename`, and see if what you typed is printed back to you.

Submission
For this milestone you are required to submit a zip containing all of your files. You should use this webform `https://podio.com/webforms/18198469/1222866` to submit your project. Late submissions will not be accepted.