O.S. Lab 6: BlackDOS Shell

Overview

Your main purpose now is to write a simple C-shell that will act as a command-interpreter interface between BlackDOS and a user making system calls. It should function like a simplified version of any popular UNIX shell (e.g. *csh*). This means using roughly the same syntax as UNIX shells and employing case sensitivity. At the end of the project, you will have a fully functional single-process operating system about as powerful as CP/M.

Modifications to Existing Pieces

Call your new file **Shell.c**. Add commands to **compileOS.sh** to compile it, assemble **blackdos.asm** and link it to the shell, compile **loadFile.c** and use it to load the shell executable onto the floppy disk. (See previous labs for reference.) In short the shell should be created and put on the disk automatically every time you remake your project from now on.

Next edit **kernel.c** in three places, two of them as seen at right. From now on the kernel simply sets up interrupt 33, then loads and executes the shell in the second segment. Also, instead of simply hanging up, programs terminating in BlackDOS resume execution of the shell. This includes programs terminating due to errors; remember to rewrite your *error()* function.

```
void main() {
  char buffer[512];
  makeInterrupt21();
  interrupt(33,2,buffer,258,0);
  interrupt(33,12,buffer[0]+1,buffer[1]+1,0);
  printLogo();
  interrupt(33,4,"Shell\0",2,0);
  interrupt(33,0,"Bad or missing command interpreter.\r\n\0",0,0);
  while (1) ;
}

/* more stuff here */

void stop() { launchProgram(8192); }

/* still more stuff here */
```

Updating Interrupt 33

One last time add an option to the interrupt-33 handler in **kernel.c**:

Function	AX=	BX=	CX=	DX=	Action
		…			
Reboot DOS.	**11**				Call **interrupt(25,0,0,0,0)**
		…			

This should be straight-forward and should be the last change you make to the kernel.

The Shell Itself

Your initial shell should first read the configuration values into local variables. Next, clear the screen, print a welcome message and then run in an infinite loop. On each iteration, it should print the right-facing-rat prompt

blackdos ~(__^>

The shell then reads in a line of text and tries to match that line to a command. If it is not a valid command, print the "bad command or file name" error message and prompt again.

All input/output in your shell are to be implemented using interrupt 33 calls. Do not rewrite or reuse any of the kernel functions. (You can use the **blackdos.h** file from the previous lab if you think that will make things easier.) This makes the OS modular: if you want to change the way things are printed to the screen, you only need to change the kernel, not the shell or any other user program.

Commands you are to recognize and implement, plus errors to catch:

- **boot** Reboot the system by calling 33/11 to reload and restart the operating system.
- **cls** Clear the screen by issuing interrupt 33/12.
- **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.
- **del** *filename*
  Delete the named file (via interrupt 33/7) by clearing the appropriate map entries and marking it as deleted in the directory.
- **dir** List disk directory contents, described next.
- **echo** *comment*
  Display *comment* on screen followed by a new line (multiple spaces/tabs may be reduced to a single space); if no argument simply issue a new prompt.
- **help** Display the user manual, described next.
- **lprint** *filename*

  Load *filename* into memory and print its contents (to the printer) using interrupt 33 calls.
- **run** *filename* Call interrupt 33/4 to load and execute *filename* at segment 4.

- **setenv fg** *color* and **setenv bg** *color*
  Change the values of the environment variables **fg** (foreground color) and **bg** (background color). After ensuring that the user has provided a legal value, both (1) change the local value and (2) read sector 258 from disk, edit it and write it back out. If the value provided is illegal issue a "bad environment value" error (see below).

- **tweet** *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*.

- **type** *filename*
  Load *filename* into memory and display its contents onscreen using interrupt 33 calls.

For **copy**, **del** and **tweet** 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 above. (Most other error tests should have been built into the kernel.)

Notes on Directory

The floppy used by the simulator is a 3½-inch disk with 1.44Mb of storage. As noted previously, by the way we designed the file system, a total of 255 sectors (and 255 × 512 = 130,560 bytes) is being tracked by the BlackDOS file system.

As in the last project load the directory sector (sector 257) into a 512-byte character array and parse it 32 bytes at a time. Print out names and sizes (number of sectors used) of each file. (Note that there is a maximum of 16 total files.) At the end, print out total space used by the files and total free space remaining, both in terms of number of sectors.

For purposes of this implementation, we will assume that all user-created files must begin with a lower-case letter. System files (such as **Shell**) begin with capital letters and have restricted access, i.e. the user cannot overwrite or delete them. Because this is a policy choice on our part and not necessarily required, we will implement this as part of the shell and not the kernel, which should be kept as small and general as possible.

Recall that deleted files (those whose file names begin with the number zero) do not appear in your directory or add to your file and sector counts. While here also construct **dir** so that file names beginning with capital letters (like **Shell**) are hidden (i.e. not listed but adding to file and sector counts).

Notes on the User Manual

When the user types **help** a simple manual describing how to use the o.s. and shell should be displayed on screen. The manual should contain enough detail for a UNIX beginner to use it. For an example of the sort of depth and type of description required, execute **man csh** or **man tcsh**. These shells have much more functionality than yours, so your manuals don't have to be quite so large. Keep in mind that this is an operator's manual and not a developer's manual.

Error messaging and file name policies

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.	Carry-over from previous project
2	Disk full.	Carry-over from previous project
Default	General error.	Carry-over from previous project
No code	**Bad command.**	**Exists only in shell**
No code	**Bad environment value.**	**Exists only in shell**
No code	**Bad or missing command interpreter.**	**Related only to shell**

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

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**, **osxterm.txt**, **kernel.asm**, **compileOS.sh**, **kernel.c**, **config**, **map**, **loadFile.c**, **shell.c**, **blackdos.asm** and **README**) to the drop box. Make sure that your disk image includes the **kitty** and **fib** files. **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 1.29.2018 by T. O'Neil, based on material by M. Black. Previous revisions 3.6.2017, 2.17.2016, 1.27.2015, 2.22.2014, 2.24.2014, 11.10.2014.