O.S. Lab 5: BlackDOS Shell Part I

Overview

In this lab you will begin 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.

Modifications to Existing Pieces

Call your new file **Shell.c**. Add commands to **compileOS.sh** to compile it, link it to **basm.o**, and use **dd** to add the shell executable onto the floppy disk at sector 30. (See previous labs for reference.) In short the shell should be created and put on the disk automatically.

Next edit **kernel.c** as seen at right. Eventually the kernel will simply set up interrupt 33, then load and execute the shell in the second segment. (We assume that your shell is smaller than 5120 bytes in size; if it's larger you will have to replace the parameter "10" in **runProgram** with something larger.)

```
void main() {
  char buffer[512];
  makeInterrupt21();
  interrupt(33,2,buffer,258,1);
  interrupt(33,12,buffer[0]+1,buffer[1]+1,0);
  printLogo();
  runProgram(30,10,2);
  interrupt(33,0,"Bad or missing command interpreter.\r\n\0",0,0);
  while (1) ;
}

/* more stuff here */
```

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 pig prompt (in honor of 2019, the Year of the Pig)

^(~(oo)~)^

The shell then reads in a line of text and tries to match that line to a command. (Remember that you don't have string comparison or substring functions provided for you, you have to write your own.) If it is not a valid command, print a "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.

Shell commands all contain four characters for simplicity. Those you are to recognize and implement now include:

- **boot**
  Reboot the system by calling **interrupt(25,0,0,0,0)** to reload and restart the operating system.

- **clrs** Clear the screen by issuing interrupt 33/12.

- **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.

For now you are only to write stub code that recognizes these commands; simply print a message to screen indicating that you have parsed the input line to isolate the command and its arguments:

- **copy** *file1 file2*
- **ddir**
- **exec** *filename*
- **help**
- **prnt** *filename*
- **remv** *filename*
- **senv**
- **show** *filename*
- **twet** *filename*

Conclusion

Obviously this is just a start, we have a lot of work to do yet. 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**, **shell.c**, **basm.o**, and **README**) to the drop box. **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 7.2.2019 by T. O'Neil, based on material by M. Black. Previous revisions 1.10.2019, 1.29.2018, 3.6.2017, 2.17.2016, 1.27.2015, 2.22.2014, 2.24.2014, 11.10.2014.