O.S. Lab 0: Introduction

Part I. Introduction

The term project for this class is a series of programs that result in a tiny yet functional multitasking operating system. The operating system is to be programmed primarily in C with a small amount of assembly functions. It contains no externally written functions or libraries; all the code is written exclusively for this project.

You are permitted to work in small groups of 2 – 3 on this term project if you wish. The needed system emulator and other software is maintained in our public labs. It is expected that you will use the public lab to complete your work. The completed operating system contains the following components:

- A boot loader
- System calls using software interrupts
- A CP/M-like file system, with system calls for creating, reading, and deleting files
- A program loader that can load and execute user programs
- A command-line shell with basic shell commands: dir, copy, delete, type, execute process

Motivation

The planned operating system is similar to CP/M and designed to fit on a 3½ inch floppy disk and to be bootable on any x86 PC. You should be asking yourself what you will gain from writing a museum piece. First of all, it should be clear that modern operating systems have become so big that to construct one in the space of one semester would be virtually impossible. All we could hope to do is study parts, and it makes sense to start with the basics.

More importantly, desktop computers are being rapidly supplanted by smaller devices with limited power consumption and resources. The operating systems for these devices have a lot in common with their text-driven counterparts for old PCs, such as restricted I/O paradigms, reduced resource options and limited functionality/responsibility. The starting point for a hand-held o.s. isn't that much different from where we begin this excursion, making this good practice for the types of devices you will use in the years to come.

Source Files

After compiling the operating system, you will create a floppy disk image file. An image is a byte-by-byte replica of the data stored on a floppy disk. You can then load the disk image onto a computer simulator.

Your final operating system will include the following provided source files:

- *bootload.asm*
  Assembly code to load the kernel from the disk and run it.

- *kernel.asm*
  Assembly functions called by the kernel. Provided to you since a few routines, such as calling interrupts and writing to registers, must be done in assembly.

- *lib.asm*
  This contains a single assembly function allowing the shell to make system calls.

- *map.img*
  This file contains an initial Disk Map sector image for the file system. You will use this when putting together your floppy disk image.

- *dir.img*
  This file contains an initial Directory sector image for the file system. You will also use this when making your floppy disk image.

- *loadFile.c*
  This is a Unix program that copies a file from Unix onto your disk image.

The files you will have to write yourself include:

- *kernel.c*
  The kernel code. The system calls, file handling, program loading, and process scheduling is done here.

- *shell.c*
  The shell code. This program prompts the user for shell commands and performs them by making system calls.

- *compileOS.sh*
  This is a Unix shell script you will write that will allow you to compile the operating system source files.

Tools

You will need the following utilities to complete this project:

● Bochs	An x86 processor simulator
● bcc (Bruce Evans' C Compiler)	A 16-bit C compiler
● as86, ld86	A 16-bit assembler and linker that comes with bcc
● gcc	The standard 32-bit GNU C compiler
● nasm	The Netwide Assembler
● dd	A standard low-level copying utility
● pico, nano, or vi	A simple text-based editor

All of these utilities except the first two are available for Linux but not for Windows.

To complete this assignment, you will need a Linux account. The department lab machines have all of the above utilities already installed. You will need to obtain an account on the department server (talk to your professor). Once you have an account, you should use an SSH utility (such as PuTTY) to connect to the server and log in to your account. You should be able to complete the project using your account for storage.

Bochs

Bochs is a simulator of an x86 computer. It allows you to simulate an operating system without potentially wrecking a real computer in the process. It is sufficiently elaborate to run both Linux and Windows 95!

To run Bochs you will need a Bochs configuration file. The configuration file tells things like what drives, peripherals, video, and memory the simulated computer has. The second thing you will need is a disk image. A disk image is a single file containing every single byte stored on a simulated floppy disk. In this project you are writing an operating system that will run off of a 3½ inch floppy disk. To get Bochs to recognize the disk, you should name the file **floppya.img** and store it in the same directory as the configuration file.

Running Bochs from the CAS 241/254 Labs

The recommended way to complete the work on this project is to reboot one of the PCs in our public labs to Linux and work from there. From the log-in screen, type <Ctrl><Alt><Delete>, then select *Restart* from the red menu bar in the lower-right-hand corner. After the machine reboots, use the down-arrow key to select the first "Ubuntu" option and hit <Enter>. After Linux starts, log in with your UANet i.d. and password as you did on the Windows side. From the Linux Start menu (left-click in the lower-left-hand corner) select “Utilities” then "Xterm". You now have the UNIX-like command line environment to work in. Navigate to your working directory for this project.

To test Bochs, type **firefox &** at the prompt in your Linux window. Proceed to the lab web site

http://www.cs.uakron.edu/~toneil/teaching/bdos20/

and download **test.img** and **osxterm.txt** to your working directory by clicking on each file and selecting the “Save file as” option. Quit firefox and type **mv test.img floppya.img** to rename the file. Type **bochs –f osxterm.txt** at the command prompt to start bochs running. If you see the message “Bochs works!” appear in the Bochs window, you are ready to proceed. From the top command bar select **Simulate** then **Stop**. Click **OK** and close the Bochs window.

Feel encouraged to download and play with the other sample disk images available on the web page. Just repeat the above procedure for the other files. When completely done, left-click again in the lower-left-hand corner of the desktop and select “Leave” then “Restart” to quit. This should restart the lab machine to Windows.

Starting with version 2.6 Bochs forces a breakpoint before starting BIOS, which is annoying. (More about BIOS later.) This forces a display of all register values before anything useful happens. Simply click “Continue” then “Close” to get rid of this and see your execution window.

(There are other possibilities but you're on your own. These are all issues I know of and I will not do tech support on your home-made Bochs implementations.)

Conclusion

This exercise was intended to give you an overview of the BlackDOS project and familiarize you with the environment you will be working in. There is nothing to submit for grade. Just get ready to move onto Lab 1.

Last updated 1.9.2018 by T. O'Neil, based on material by M. Black and G. Braught. Previous revisions 12.21.2016, 2.11.2016, 1.16.2016, 1.14.2015, 2.21.2014.