CSC 565 - Operating Systems

Build an Operating System from Scratch Project

Overview

The Build an Operating System from Scratch project is a series of projects that result in a tiny yet functional operating system similar to CP/M, but with multitasking. The operating system is intended to fit on a 3 1/2 inch floppy disk and to be bootable on any normal x86 PC. 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.

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, kill process
- Multiple processes and multitasking using timer interrupts. A scheduler chooses processes using round-robin.

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 either load the disk image onto a real floppy and run the operating system, or load it onto a computer simulator.

A disk image with the completed operating system on it is provided to you on Blackboard with the name *floppya.img*. You are strongly encouraged to open it with Bochs (instructions are in the first project) and play around with it to get a feel for what is expected of you.

Your final operating system will consist of the following source files:

- *bootload.asm*

  Assembly code to load the kernel from the disk and run it. This is provided to you.
- *kernel.c*

  The kernel code. The system calls, file handling, program loading, and process scheduling is done here. You will write this entirely yourself.

- *kernel.asm*

  Assembly functions called by the kernel. Since a few routines, such as calling interrupts and writing to registers, must be done in assembly, you are provided with this file.
- *shell.c*

  The shell code. This program prompts the user for shell commands and performs them by making system calls. You will write this entirely yourself.
- *lib.asm*

  This is provided to you. It contains a single assembly function allowing the shell to make system calls.

The operating system also will have the following support files:

- *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.
- *compileOS.sh*

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

  This is a Unix program provided to you that copies a file from Unix onto your disk image.

The Projects

The overall operating system project is divided into five pieces. The five projects build on each other so you must complete them in order. A sixth project asks you to extend the operating system in a way of your choosing.

- Project A

  This is a warmup project to get you familiar with the computer simulator. You will make a kernel that just prints out “Hello World” when the computer is started up.
- Project B

  Here you will make a software interrupt handler to handle system calls to the kernel. You will write system calls to print a string, read a line from the keyboard, and read a sector from the disk. All your work will be in kernel.c.
- Project C

  You will make two additional system calls to read a file from the disk and execute a program file. You will then write a shell, and make shell commands “type” and “execute” to print out a file and execute a program file.

- Project D

  You will make three more system calls to write a sector to the disk, delete a file, and create a file. You will then add shell commands to delete files, copy files, create text files, and list the directory.

- Project E

  You will add a process table to handle where programs are loaded in memory. You will create a timer interrupt handler that will switch programs on timer interrupts to create multitasking. You will then add a shell command to kill processes.

- Project F

  You are given several options to improve on the operating system from the previous projects. A user interface, virtual memory, or a FAT file system are possible projects.