Chapter 1

Introduction

This textbook is for the CS5600 class in the Northeastern College of Computer and Information Science, based on the design of CS5600 starting in Fall of 2008. The goal of this class is not to teach you how to write an operating system—that is an obscure skill, practiced by far fewer people than you might think. Nor is it to learn how to use an operating system—depending on the type of use, that would be system administration, programming, or just using a computer. Instead the goal is to teach you how computers work, by describing the interacting parts underneath the user and programming interfaces.

```
root@cs5600-vbox:/# ls -l
total 68
drwxr-xr-x   2 root root  4096 Sep 20  2013 bin
drwxr-xr-x   3 root root  4096 Sep  2 12:35 boot
drwxr-xr-x  14 root root  3960 Apr  1  2015 dev
drwxr-xr-x 104 root root  4096 Apr  8  2015 etc
drwxr-xr-x   4 root root  4096 Aug 24  2013 home
lrwxrwxrwx   1 root root    24 Sep  2 12:36 initrd.img -> boot/initrd.img-3.2.0-
51
drwxr-xr-x  17 root root  4096 Sep 20  2013 lib
drwx------   2 root root 16384 Aug 18  2013 lost+found
drwxr-xr-x   3 root root  4096 Oct  4  2013 media
drwxr-xr-x   3 root root  4096 Sep  2 12:30 mnt
dr-xr-xr-x 110 root root     0 Apr  1  2015 proc
drwx------   2 root root  4096 Sep  2 12:32 root
drwxr-xr-x  17 root root   640 Sep 13  2016 run
drwxr-xr-x   2 root root  4096 Feb 25  2014 sbin
drwxr-xr-x   2 root root  4096 Mar  5  2012 selinux
drwxr-xr-x  13 root root     0 Apr  1  2015 sys
drwxrwxrwt   8 root root  4096 Jan 28  2016 tmp
drwxr-xr-x  10 root root  4096 Jan 25  2014 usr
drwxr-xr-x  12 root root  4096 Nov 23  2014 var
lrwxrwxrwx   1 root root    21 Sep  2 12:32 vmlinuz -> boot/vmlinuz-3.2.0-51
root@cs5600-vbox:/# _
```

Figure 1.1: Linux text console with simple command.

For an example of what this means, consider running a simple command such as `ls` on a Linux system. In Figure 1.1 we see the screen of a system booted in text mode, using the simple character display that the BIOS uses. In responding to the keystrokes typed by the user, we can identify not only the basic actions being performed ("run the `ls -l` command with output

to the console") but a large number of interacting actions and components as well:

- the keyboard control hardware (assuming an old-fashioned PS/2 keyboard) interrupts the processor, causing it to run a portion of the keyboard input driver.
- The driver reads data from the keyboard and calls scheduling functions to wake the shell process, which was sleeping waiting for input.
- the shell process spawns a copy of itself, by invoking a system call which copies some of the shell process state and shares other parts of it between the *parent* and *child* processes using the virtual memory system.
- The new process invokes the `exec` system call, causing the operating system to map the `/bin/ls` binary into the process address space.
- As `ls` starts up, the dynamic loader loads additional shared libraries into the process address space; these as well as the `ls` code itself is loaded into memory on demand as the CPU accesses them.
- `ls` invokes system calls to read the list of files in the current directory.
- The file system code receives requests to read files containing the executable and libraries, as well as the directory listing request from the `ls` program itself, and in turn requests data from the disk (via the block device system) to fulfill these requests.
- Since the example was actually running in a virtual machine, not a physical machine[1] the hardware interactions described above were actually emulated by another software system (i.e. VirtualBox) which translated them into requests to the underlying operating system, which in turn interacted with the real keyboard and screen.

The remainder of this book, and the corresponding class, is concerned with the detailed analysis of the interactions involved in performing this simple operation. The major sections of this text concern:

OS organization: Memory organization and OS interface to decouple applications from hardware and OS details, context switching, and system calls. This section describes and uses a simple computer, described more fully in the appendices.

Synchronization: Beginning with practical problems arising from multiple simultaneous actions, we describe methods such as semaphores and monitors to control simultaneous actions, as well as methods to reason about the operation and performance of parallel operations.

[1] It makes screenshots far easier.

Virtual memory: At the hardware level, how is address translation implemented via the MMU, TLB, and page table? In the OS, how are page faults used to implement copy-on-write, demand loading, and paged virtual memory?

Block devices: These are devices such as disks, RAID arrays, and SSDs, used for storing files and similar information. Topics covered include performance and interfaces, I/O operation at a hardware level, and methods of structuring I/O systems for reliability (RAID), manageability (logical volume management) and efficiency (deduplication).

File systems: What is a file system and what are its operations? How do we implement these, and how do we lay files out on disk?

Security: What are the goals of security mechanisms in an operating system? How can we specify and implement policies to control access and operations?

The objective of the class is to be able to identify the steps involved in this and other computer operations. In learning this we will touch on hardware, device drivers, scheduling, virtual memory, and networking. We focus on *behavior*—i.e. the sequence of events which occurs in response to an input, and results in an output. This behavior cuts across layers and subsystems, as an event at the hardware level may trigger actions within a device driver, then in the core of the operating system, within a user process, etc. Rather than looking at the operating system in a structured way we are going to follow these sequences of behavior and see where they lead.