Persistence

Andrew Quinn

Learning Objectives:

1. What is the overall structure that an operating system uses to manage persistence?
2. How does an operating system interact with storage devices?
3. What should our mental model of a hard drive be?

Announcements:

1. Have you started working on Project 3?

We are moving on to the final unit for this course: persistence! How can we ensure that a system does not lose data when the power is turned off?

We'll split the discussion of persistence into two parts. First, we will talk about how an operating system interacts with storage devices at a low level. We'll talk a bit about how storage devices work, focusing on the hard disk drive, and will discuss RAID.

Then, we will talk about how an operating system uses storage devices to provide the high-level interfaces to which you are accustomed, called the file system. We will talk about the internal data-structures that these file systems use, how they ensure crash-consistency, and how they are often designed specifically around certain classes of hardware.

The boundary between the two halves is the *block layer*. Blocks are fixed-size units of storage (often 4KiB). The file system issues reads/writes to blocks, without having to know how the system actually implements these operations. Meanwhile, the code responsible for block reads/writes responds to them, without having to know why they occur.

I/O devices

1. Our first model is that a CPU is attached through a bus to peripheral devices. Such a bus can be SCSI, SATA, USB, etc.
2. Each device has a few registers that the CPU can program to instruct the device to do something. E.g., a Status, Command, and Data register.
3. An operating system then might issue commands to a device by specifying:

```
While (status == BUSY) {}
Write data to DATA
Write command to command
While (status == BUSY)
```

4. This protocol is an example of programmed I/O, a class of I/O approaches in which the CPU is involved moving data to the I/O device.
5. This protocol is also an example of *polling*, since the CPU repeatedly checks the status register. It has a major downside in that while polling, the CPU cannot be doing something else!

6. Interrupts are a solution to the polling problem. The approach requires that the hardware device issue interrupts to the CPU when they finish useful work. Then, instead of polling on the status, the CPU can context switch to a useful routine instead of polling.
7. Are interrupts *always* better than polling? When would they not be? What if there is an extremely large sequence of incoming requests (e.g., lots of very small reads)? What if each read needs to be processed *very* quickly?
8. An issue with programmed I/O is that it spends considerable time copying memory to the device. Direct Memory Access (DMA) is a solution for this challenge. A DMA engine is a separate hardware device that can perform the memory copy, allowing the CPU to do other important things.
9. Two mechanisms that CPUs use to interact with devices: special (privileged) instructions and memory-mapped Input-Output (MMIO).
10. Device drivers within an operating system encapsulates all of this logic. They often account for *most* of the logic in an operating system.

HARD DISK DRIVES

Hard disk drives were the dominant storage technology for decades. Even though we have emerging storage technology today (solid state drives, non-volatile memory express (NVMe) drives, and even archival storage devices through DNA and glass), hard disks remain an important technology. We'll describe some of there details here to get a sense for how they work.

1. Logically, each disk is exposed to the operating system as a giant array of disk sectors, typically 512 bytes. Sectors are the smallest unit of access in a disk; disks ensure that a sector read/write is atomic.
2. A disk is made up of a series of platters: circular hard surfaces that are capable of maintaining a charge without power.
3. Platters are stacked vertically and bound together around a spindle, which spins at a constant-rate of revolutions per minute (RPMs). Today's typical RPM rates include 7,200–15,000 RPM.
4. Platters are made up of a series of tracks, concentric circles on the platter, each of which stores a series of sectors.
5. Disks read data using a disk head. The head is attached to a disk arm, which allows it to move between the tracks.
6. There are three components of reading a sector. (1) Moving the disk head to the right track; the time for this to complete is called the seek time. (2) waiting for the right sector on that track to move to below the disk head; the time for this to complete is called the rotational delay. (3) the data has to actually be read from the track. This usually takes negligible time.
7. Disk often include a cache, and thus make caching decisions internally.
8. Because of the cost of arm movement and rotational delay, sequential accesses on a disk are *much faster* than random accesses.
9. Disk Scheduling, or choosing what I/O to perform next, is an important problem for

disks. Disk scheduling is both an OS and device problem.

10. Two disk scheduling approaches are Shortest-seek time first and the elevator algorithm.