Section 9: I/O, Devices

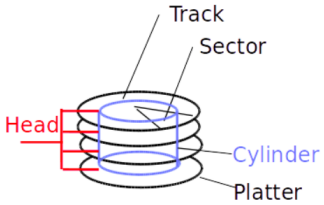
CS 162

October 29, 2021

Contents

1 Vocabulary

- **I/O** In the context of operating systems, input/output (I/O) consists of the processes by which the operating system receives and transmits data to connected devices.
- **Controller** The operating system performs the actual I/O operations by communicating with a device controller, which contains addressable memory and registers for communicating the the CPU, and an interface for communicating with the underlying hardware. Communication may be done via programmed I/O, transferring data through registers, or Direct Memory Access, which allows the controller to write directly to memory.
- **Interrupt** One method of notifying the operating system of a pending I/O operation is to send a interrupt, causing an interrupt handler for that event to be run. This requires a lot of overhead, but is suitable for handling sporadic, infrequent events.
- **Polling** Another method of notifying the operating system of a pending I/O operating is simply to have the operating system check regularly if there are any input events. This requires less overhead, and is suitable for regular events, such as mouse input.
- **Response Time** Response time measures the time between a requested I/O operating and its completion, and is an important metric for determining the performance of an I/O device.
- **Throughput** Another important metric is throughput, which measures the rate at which operations are performed over time.
- **Asynchronous I/O** For I/O operations, we can have the requesting process sleep until the operation is complete, or have the call return immediately and have the process continue execution and later notify the process when the operation is complete.
- **Memory-Mapped I/O** Memory-mapped I/O (not to be confused with memory-mapped file I/O) uses the same address bus to address both memory and I/O devices – the memory and registers of the I/O devices are mapped to (associated with) address values. So when an address is accessed by the CPU, it may refer to a portion of physical RAM, but it can also refer to memory of the I/O device. Thus, the CPU instructions used to access the memory can also be used for accessing devices.
- **Device Driver** - Device-specific code in the kernel that interacts directly with the device hardware. They support a standard, internal interface so the same kernel I/O system can interact easily with different hardware. The top half of a device driver is used by the kernel to start I/O operations. The bottom half of a device driver services interrupts produced by the device. You should know that Linux has different definitions for "top half" and "bottom half", which are essentially the reverse of these definitions (top half in Linux is the interrupt service routine, whereas the bottom half is the kernel-level bookkeeping).
- **Hard Disk Drive (HDD)** - A storage device that stores data on magnetic disks. Each disk consists of multiple **platters** of data. Each platter includes multiple concentric **tracks** that are further divided into **sectors**. Data is accessed (for reading or writing) one sector at a time. The **head** of the disk can transfer data from a sector when positioned over it.



- **Seek Time** - The time it takes for an HDD to reposition its disk head over the desired track.
- **Rotational Latency** - The time it takes for the desired sector to rotate under the disk head.
- **Transfer Rate** - The rate at which data is transferred under the disk head.
- **Checksum** - A mathematical function which maps a (typically large) input to a fixed size output. Checksums are meant to detect changes to the underlying data and should change if changes occur to the underlying data. Common checksum algorithms include CRC32, MD5, SHA-1, and SHA-256.
- **Replication** - Replication or duplication is a common technique for preserving data in the face of disk failure or corruption. If a disk fails, data can be read from the replica. If a sector is corrupted, it will be detected in the checksum. The data can then be read from another replica.
- **Queueing Theory** Here are some useful symbols: (both the symbols used in lecture and in the book are listed)
  - μ is the average service rate (jobs per second)
  - T_{ser} or S is the average service time, so $T_{ser} = \frac{1}{\mu}$
  - λ is the average arrival rate (jobs per second)
  - U or u or ρ is the utilization (fraction from 0 to 1), so $U = \frac{\lambda}{\mu} = \lambda S$
  - T_q or W is the average queueing time (aka waiting time) which is how much time a task needs to wait before getting serviced (it does not include the time needed to actually perform the task)
  - L_q or Q is the average length of the queue, and it's equal to λT_q (this is Little's law)

2 Input/Output

2.1 Warmup

1. (True/False) If a particular IO device implements a blocking interface, then you will need multiple threads to have concurrent operations which use that device.

2. (True/False) For I/O devices which receive new data very frequently, it is more efficient to interrupt the CPU than to have the CPU poll the device.

3. (True/False) With SSDs, writing data is straightforward and fast, whereas reading data is complex and slow.

4. (True/False) User applications have to deal with the notion of file blocks, whereas operating systems deal with the finer grained notion of disk sectors.

2.2 I/O Devices

What is a block device? What is a character device? Why might one interface be more appropriate than the other?

Describe the difference between port-mapped I/O and memory-mapped I/O.

Explain what is meant by "top half" and "bottom half" in the context of device drivers.

3 Storage Devices

What are the major components of disk latency? Explain each one.

In class we said that the operating system deals with bad or corrupted sectors. Some disk controllers magically hide failing sectors and re-map to "back-up" locations on disk when a sector fails.

If you had to choose where to lay out these "back-up" sectors on disk - where would you put them? Why?

How do you think that the disk controller can check whether a sector has gone bad?

Can you think of any drawbacks of hiding errors like this from the operating system?

4 I/O Performance

This question will explore the performance consequences of using traditional disks for storage. Assume we have a hard drive with the following specifications:

- An average seek time of 8 ms
- A rotational speed of 7200 revolutions per minute (RPM)
- A controller that can transfer data at a maximum rate of 50 MiB/s

We will ignore the effects of queueing delay for this problem.

1. What is the expected throughput of the hard drive when reading 4 KiB sectors from a random location on disk?

2. What is the expected throughput of the hard drive when reading 4 KiB sectors from the same track on disk (i.e., the read/write head is already positioned over the correct track when the operation starts)?

3. What is the expected throughput of the hard drive when reading the very next 4 KiB sector (i.e. the read/write head is immediately over the proper track and sector at the start of the operation)?

4. What are some ways the Unix Fast File System (FFS) was designed to deal with the discrepancy in performance we just saw?