Section 9: I/O, Devices, Queueing Theory

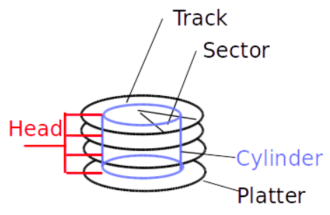
CS 162

October 30, 2020

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.

   True. Only with non-blocking IO can you have concurrency without multiple threads.

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.

   False. It is more efficient to poll, since the CPU will get overwhelmed with interrupts.

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

   False, it is the opposite. SSD's have complex and slower writes because their memory can't be easily mutated.

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.

   False, blocks are also an OS concept and are not exposed to users.

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?

Both of these are types of interfaces to I/O devices. A block device accesses large chunks of data (called blocks) at a time. A character device accesses individual bytes at a time. A block device interface might be more appropriate for a hard drive, while a character device might be more appropriate for a keyboard or printer.

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

Port-mapped I/O uses special IN and OUT instructions in the Intel x86 architecture, in a separate address space from main memory. On the other hand, memory-mapped I/O uses load and store instructions in physical address space.

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

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).

3 Storage Devices

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

```
Queueing time - How long it spends in the OS queue
Controller - How long it takes to send the message to the controller
Seek - How long the disk head has to move
Rotational - How long the disk rotates for
Transfer - The delay of copying the bytes into memory
```

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?

Should spread them out evenly, so when you replace an arbitrary sector your find one that is close by.

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

Using a checksum - this can be efficiently checked in hardware during disk access.

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

Excessive sector failures are warning signs that a disk is beginning to fail.

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?

   The time to read the sector can be broken down into three parts: seek delay, rotational delay, and data transfer delay. We are already given the expected seek delay: 8 ms.

   We can assume that, on average, the hard disk must complete 1/2 revolution before the sector we are interested in reading moves under the read/write head.

   Given that the disk makes 7200 revolutions per minute, the time to complete a revolution is $60sec/7200Revolution \approx 8.33$ ms per revolution.

   The time to complete 1/2 revolution, the expected rotational delay, is ~ 4.17 ms.

   If the controller can transfer 50 MiB per second, it will take:

   $$4 \times 2^{10} bytes \times \frac{1 sec.}{50 \times 2^{20} bytes} \approx 0.078125 ms$$

   to transfer 4 KiB of data.

   In total, it takes $8ms + 4.17ms + 0.078125ms \approx 12.248ms$ to read the 4 KiB sector, yielding a throughput of $4KiB/12.248ms \approx 326.6KiB/s$

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)?

   Now, we can ignore seek delay and only need to account for rotational delay and data transfer delay.

   We already know that the expected rotational delay is 4.17 ms and we know that the expected data transfer delay is 0.078125 ms.

   Therefore, it takes a total of $4.17ms + 0.078125ms \approx 4.24ms$ to read the 4 KiB sector, yielding a throughput of $4KiB/4.24ms \approx 943.4KiB/s$.

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)?

   Now, we can ignore both rotational and seek delays. The throughput of the hard disk in this case is limited only by the controller, meaning we can take full advantage of its 50 MiB/s transfer rate.

Note that this is roughly a 156$\times$ improvement over the random read scenario!

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

- Attempt to keep contents of a file contiguous on disk (first-fit block allocation)
- Break disk into a set of *block groups* — sets of adjacent tracks, each with its own free space bitmap, inodes, and data blocks
- Keep a file's header information (inode) in same block group as its data blocks
- Keep files in the same directory in the same block group

5 Queueing Theory

Explain intuitively why response time is nonlinear with utilization. Draw a plot of utilization (x axis) vs response time (y axis) and label the endpoints on the x axis.

Even with high utilization (99%), some of the time (1%), the server is idle, which is a waste. All this wasted time adds up, and in the steady state, the queue becomes very long.
Graph should be linear-ish close to $u = 0$ and grow asymptotically toward ∞ at $u = 1$.

If 50 jobs arrive at a system every second and the average response time for any particular job is 100ms, how many jobs are in the system (either queued or being serviced) on average at a particular moment? Which law describes this relationship?

$50\ jobs/s \times 0.1s = 5\ jobs$ (5 jobs at any time). This is Little's law - arrival rate $\times$ average response time = average length of the queue.

Is it better to have N queues, each of which is serviced at the rate of 1 job per second, or 1 queue that is serviced at the rate of N jobs per second? Give reasons to justify your answer.

One queue that can process N jobs per second is faster (i.e the second option). Better response time ($\frac{1}{N}$ sec vs 1 sec) and better utilization (no load-balancing problems), which gives you lower queueing delays on average.

What is the average queueing time for a work queue with 1 server, average arrival rate of λ, average service time S, and squared coefficient of variation of service time $\mathbf{C}$?

(Recall from definitions that we sometimes swap between symbols - S is another name for T_{ser}.)
$T_q = T_{ser}(\frac{u}{1-u})(\frac{C+1}{2})$ where $u = \lambda T_{ser}$

What does it mean if $\mathbf{C} = 0$? What does it mean if $\mathbf{C} = 1$?

If $\mathbf{C} = 0$, then your service rate is regular and deterministic, which means that tasks are completed at a constant rate.
If $\mathbf{C} = 1$, then your service rate can be modeled as a Poisson distribution, and the interval between jobs being serviced can be modeled as a exponential distribution.