Lambda School - Computer Architecture 2

Sarah

October 2, 2017

Abstract

Computer Architecture 2 discusses computer peripherals including the major three peripherals that differentiate current computers from the Commodore: the network card, the video card, and the hard drive.

1 Question 1

1.1 The question:

The minimum seek time for an HDD is 9msec, and the maximum seek time is 90msec. The block size of this HDD is 4KB. How long on average does it take to read 100MB of data?

1.2 Thoughts

NOTE: I wrote this prior to the Q&A session where you said that you included rotational latency in your seek times, however, since I had already given thought and research to this question, I have left it in its original form to ensure that I was able to research for the other questions. The logic would not change, simply the calculations. For your average 9ms * 25,000 blocks / 1000 ms/sec / 60 sec/min = 3.75 minutes.

The minimum seek time for an HDD is 9msec, and the maximum seek time is 90msec. The block size of this HDD is 4KB. How long on average does it take to read 100MB of data? (Look up average seek times of real hard disks, not just min and Max. Average isn't necessarily half way. Also compare the size of the blocks on modern hard drives plus how many blocks can be read in a single seek).

Using the equation Access time = seek time + rotational latency + transfer time per block and a 7200rpm hard drive with half a rotation to find the correct location and a 80Mb/s transfer rate, I calculated the access time per block at both extremes [?]. With a 9ms seek time, finding and reading one block would be 13.21 ms, which for 100 MB would be around 5.5 minutes. For the larger extreme, using the same assumptions, the read time for one block is 94.2ms. Again calculating for 25,000 individual blocks (100MB) is 39.25 minutes.

However, more than one block can be read with one seek. So long as the data is not fragmented, with either the 9ms or 90 ms seek time, the average read time for 100MB is 1.73 minutes.

This problem addresses the importance of avoiding memory fragmentation. This seems to mostly be a problem encountered with C, C++, and embedded

systems, however, as the question is specifically about hard drives, embedded systems are outside the scope of this essay. Avoiding dynamic memory allocation, will help avoid memory fragmentation issues. Higher level languages contain 'garbage collection' to help mitigate the memory fragmentation issue [?].

Additionally, with age bearings in the hard drive will slow down or seize, among other mechanical issues that slow down the rotational latency and the seek time. There are also the drivers that will slow down the time to actually return the data, whether the CPU is available to handle the information or has moved on to another application and can not handle the interrupt at that moment. Adding a GPU will mean that the hard drive has more time to wait for its interrupt to be handled. Also to factor in is the state of the hard drive. If it is not spinning, there is additional time for it to spin. This is all assuming that there is no catastrophic failure leading to the data never being returned. There are so many factors affecting the access time that it is not entirely reasonable to be able to calculate the access time. It would need to be physically timed with the application, and acknowledge that this time will be different for each machine and pending the state of the machine at the time. Optimizing at this level is not terribly practical. Given this amount of uncertainty, I can say definitively that the data will be returned in the range of 9ms to never...

2 Question 2

2.1 The question:

Describe a TCP/IP packet in detail. Describe the header, how many bytes it is, and which components it contains. What data can come after the header?

2.2 Thoughts

A TCP/IP packet starts with a header, which contains between 5 and 15 32-bit words of structured information. The first 32 bits identify the source and destination ports of the packet. Following this are 32 bits of a sequence number, which start with a SYN flag. The flag is set if the number is a one and clear if the number is a 0 [?]. The SYN flag is part of the SYN, SYN-ACK, ACK handshake, where the first computer sends SYN, which opens a connection, the second computer receives it and returns SYN-ACK to inform the first computer that the synchronize was acknowledged. Finally, the first computer receives SYN-ACK and returns an ACK informing the second computer that the acknowledgement was accepted, and the second computer receives the ACK finalizing the handshake [?]. Following the Acknowledgement number is 4 bits called Data offset which is the offset between the beginning of the packet to the beginning of the actual data in the packet; it is simply the size of the header measured in 32-bit words. The next set of bits are set to zero in order to align the header into sets of 4 bits [?].

The next nine bits are nine flags, each one bit long. The first is NS (nonce sum) which is the first of three congestion flags. This set of flags was created to solve the issue that had existed prior to its creation: the only feedback for problems was lost packets, which lead to latency due to re-transmission of packets,

which could be a problem in many applications. The following two bits work together to handle network congestion. First, the sender and receiver both must be capable of using these flags as not everyone has adopted their use yet. This is resolved in the handshake with both ECE and CWR flags set high with the SYN package and the receiver returns the ACK with the ECE flag pulled high. This only establishes that the both ports can use this as a way of communication. Then later, should the router determine that there is a possibility of dropped packets, it pulls high the CE (Congestion Experienced) flag, and if the router receives this packet it will return a packet with the ECE (Echo Congestion Experienced) flag high [?]. In some way that I do not fully understand thusly, this makes it faster and has less lost packets.... The urgent flag is exactly what it sounds like - it flags the packet with important information. This flag is followed by an ACK flag which will be high so long as it is not the first transmission. Following this is the PSH flag. Usually a packet uses a buffer to wait to be sent until it is filled. This is a good practice unless the data should be sent in real time. Pulling this flag high will denote that the the information should not wait in the buffer to be sent or pulled in to the receiving application [?]. The RST flag resets the connection if high. The SYN flag is only high for the first transmission for the purposes of the handshake. Finally, the FIN flag is high to indicate the last packet from the sender [?].

After the flags section is a 16 bit section that indicates the size of packet that the receiving machine is willing to allow [?]. The checksum is 16 bits of error checking to ensure that the data has not been mangled between sender and receiver [?]. This brings us to the options section which is optional and between 0 and 320 bytes in a number divisible by 32.

And now, after all that header, we finally reach the data!

3 Question 3

3.1 The question:

How does the network protocol guarantee that a TCP/IP packet is complete after transmission?

3.2 Thoughts

The network protocol guarantees that a packet is complete and accurate by using the checksum found in the header and the cyclic redundancy check. The checksum is a calculation based on the bits of the header and text. Interestingly, the checksum is part of the header that it is using as part of the calculation. To solve this, the calculation places all zeros in the spot for the checksum during the calculation [?]. The checksum at a basic level adds two words at a time, each word being two bytes long in 1's complement, which means that in all binary the 1's are switched to 0's and the 0's to 1's. The checksum is not able to detect all possible corruption such as if pairs of words are reversed such as words C and D being before words A and B as addition is commutative. There are also issues where even numbers of errors that are non-consecutive can be missed with this algorithm. This is why the checksum is combined with other measures. The cyclic redundancy check works very similarly to

the checksum but has a different algorithm involving polynomial division [?]. These two algorithms working in coordination with the handshake process and the congestion experienced procedures, where available ensure complete and accurate transmission.

4 Question 4

4.1 The question:

What is the difference between TCP and IP?

4.2 Thoughts

Transmission Control Protocol is part of the Internet Protocol. IP functions at a lower level than the TCP, with TCP acting as an intermediary between the application and the IP. The app simply send data to be sent out to the TCP and awaits returned data. The TCP breaks the data into appropriately sized packets, hands it over to the IP, and then deals with all the issues with the returned data. If packets are damaged or missing, the TCP requests retransmission. The TCP also reorders packets appropriately and when complete, accurate, and ordered returns this data to the app [?]. IP is part of OSI level 3. It focuses on taking the packets from TCP and moving them from gateway to gateway, rerouting should issues arise and ultimately delivering the packet to the destination computer. The Internet protocol also concerns itself with the addressing of the computers, routers, phones... all the items that use the Internet so that the packets can be directed appropriately. Unofficial and unprofessional side note: I dislike how I have this written as though the TCP and IP are actively doing things. They are protocols - sets of rules about the processes I have described, nothing more. All the resources about TCP and IP seem to describe these protocols in an active sense so in my essay, I, too, wrote in an active way, but it is misleading. Much like how people talk about evolution as some sort of active process that "selects" attributes, when in reality evolution just describes the fact that some plants and animals die before having hoards of offspring to spread their genes.

5 Question 5

5.1 The question:

Why is 3d performance so much higher with a graphics card than without? Modern CPUs are extremely fast, what is limiting their performance?

5.2 Thoughts

A CPU runs at a faster rate than a GPU. For example, an Intel i7-7800X Skylake (arbitrarily chosen from NewEgg for no compelling reason) has a base clock of 3.5GHz whereas the GTX 1080 from Nvidia only has a base clock speed of 1708MHz. That is quite a difference in speed, yet 3D graphics are much more feasible with a GPU than without. There are a plethora of reasons for this. The first two being core counts and thread counts. Sticking with the

same Intel and Nvidia examples from before, the CPU has six cores and 12 threads, while the GPU has 2560 cores. If we multiplied the number of CPU's to allow for the same number of cores as the GPU, the CPU may outperform the GPU, however, that would be very cost prohibitive, quite large, and would need serious cooling solutions. (This sounds like an interesting experiment, but I do not have the finances or technical expertise to run that experiment, so maybe is the most definitive answer I can give at the moment on that). The reason so many cores can be placed on the GPU and have not been replicated on the CPU is that each GPU core does not have the full functionality of a CPU core. These are special devices that were customized for the task at hand which is graphical computation, mainly matrix math. Other functions have been able to use similar calculations and have been able to utilize the speed and efficiency of the GPU such as bitcoin mining, protein folding from Fold At Home, physics calculations and simulations, and more. There have been many streamlining techniques that can optimize the efficiency of the cores on graphics cards that sacrifice their flexibility, however for executing the same functions repeatedly, there is no need for the additional functionality of CPU's. A CPU will also have more cache with lower bandwidth to main memory, while a GPU has very little cache but with much, much higher bandwidth to graphics memory. CPU's also need to handle interrupts from many inputs, all the peripherals attached to the machine. A GPU does not have these additional tasks to contend with. There is no need to pause calculation to handle a mouse click or a keyboard press unless it affects the visuals that it is rendering.

References

[1] https://superuser.com/questions/595967/calculating-hard-disk-block-model-reading-time.

[2] Walls, Colin. "Dynamic Memory Allocation And Fragmentation in C and C++." *Design and Reuse*, 28 Sept. 2017, https://www.design-reuse.com/articles/25090/dynamic-memory-allocation-fragmentation-c.html.

[3] "TCP 3-Way Handshake (SYN, SYN-ACK, ACK)." *InetDaemon.com*, 28 Sept. 2017, http://www.inetdaemon.com/tutorials/internet/tcp/3-way_handshake.shtml.

[4] "Transmission Control Protocol." *Wikipedia*, 28 Sept. 2017, https://en.wikipedia.org/wiki/Transmission_Control_Protocol.

[5] "TCP Headers and UDP Headers Explained." *lifewire*, 28 Sept. 2017, https://www.lifewire.com/tcp-headers-and-udp-headers-explained-817970.

[6] "TCP Flags Contingued: CWR + ECE." *catchpoint*, 28 Sept. 2017, blog.catchpoint.com/2015/10/30/tcp=flags-crw-ece/.

[7] stretch. "TCP Flags: PSH and URG." *PacketLife.net*, 28 Sept. 2017, pakcetlife.net/block/2011/mar/2/tcp-flags-psh-and-urg/.

[8] Fortunato, Tony. "Network Analysis: TCP Window Size." *Network Computing*, 28 Sept. 2017, https://www.networkcomputing.com/careers/netowrk-analysis-tcp-wondow-size/97729416.

[9] "The TCP/IP Checksum." *Lock Less*, 29 Sept. 2017, https://locklessinc.com/articles/tcp_checksum/.

[10] Tyson, Jeff. "How Encryption Works." *howstuffworks*, 29 Sept. 2017, computer.howstuffworks.com/encryption7.htm

[11] Amitash. "Difference Between TCP and IP." *DifferenceBetween.net*, 29 Sept. 2017, http://www.differencebetween.net/technology/internet/difference-between-tcp-and-ip/.