# Section 10: Intro to I/O, Device Drivers, File Systems, FAT, and Queuing Theory

October 30, 2018

## Contents

# 1 Warmup

## 1.1 Short questions

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 IO 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. SSDs have complex and slower writes because their memory cant 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.

5. How does the OS catch a null pointer exception? Trace every action that happens in the OS based on what you have learned in cs162 so far.

   A process generates a virtual address of 0. The hardware tries to look up the VPN (also 0 here) in the TLB, and suffers a TLB miss. The page table is consulted, and the entry for VPN 0 is found to be marked invalid. Thus, we have an invalid access, which transfers control to the OS, which likely terminates the process (on UNIX systems, processes are sent a signal which allows them to react to such a fault; if uncaught, however, the process is killed).

6. 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.

7. 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).

# 2 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 with 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.
- **Simple File System** - The disk is treated as a big array. At the beginning of the disk is the Table of Content (TOC) field, followed by data field. Files are stored in data field contiguously, but there can be unused space between files. In the TOC field, there are limited chunks of file description entries, with each entry describing the name, start location and size of a file.

  **Pros and Cons**

  The main advantage of this implementation is simplicity. Whenever there is a new file created, a continuous space on disk is allocated for that file, which makes I/O (read and write) operations much faster.

  However, this implementation also has many disadvantages. First of all, it has external fragmentation problem. Because only continuous space can be utilized, it may come to the situation that there is enough free space in sum, but none of the continuous space is large enough to hold the whole file. Second, once a file is created, it cannot be easily extended because the space after this file may already be occupied by another file. Third, there is no hierarchy of directories and no notion of file type.
- **External Fragmentation** - External fragmentation is the phenomenon in which free storage becomes divided into many small pieces over time. It occurs when an application allocates and deallocates regions of storage of varying sizes, and the allocation algorithm responds by leaving the allocated and deallocated regions interspersed. The result is that although free storage is available, it is effectively unusable because it is divided into pieces that are too small to satisfy the demands of the application.
- **Internal Fragmentation** - Internal fragmentation is the space wasted inside of allocated memory blocks because of the restriction on the minimum allowed size of allocated blocks.

- **FAT** - In FAT, the disk space is still viewed as an array. The very first field of the disk is the boot sector, which contains essential information to boot the computer. A super block, which is fixed sized and contains the metadata of the file system, sits just after the boot sector. It is immediately followed by a **file allocation table** (FAT). The last section of the disk space is the data section, consisting of small blocks with size of 4 KiB.

  In FAT, a file is viewed as a linked list of data blocks. Instead of having a "next block pointer" in each data block to make up the linked list, FAT stores these pointers in the entries of the file allocation table, so that the data blocks can contain 100% data. There is a 1-to-1 correspondence between FAT entries and data blocks. Each FAT entry stores a data block index. Their meaning is interpreted as:

  If $N > 0$, N is the index of next block

  If $N = 0$, it means that this is the end of a file

  If $N = -1$, it means this block is free

  Thus, a file can be stored in a non-continuous pattern in FAT. The maximum internal fragmentation equals to 4095 bytes (4K bytes - 1 byte).

  Directory in the FAT is a file that contains directory entries. The format of directory entries look as follows:

  Name — Attributes — Index of 1st block — Size

  **Pros and Cons**

  Now we have a review of the pros and cons about FAT. Readers will find most of the following features have been already talked about above. So we only give a very simple list of these features.

  Pros: no external fragmentation, can grow file size, has hierarchy of directories

  Cons: no pre-allocation, disk space allocation is not contiguous (accordingly read and write operations will slow), assume File Allocation Table fits in RAM. Otherwise lseek and extending a file would take intolerably long time due to frequent memory operation.

- **Queuing Theory** Here are some useful symbols: (both the symbols used in lecture and in the book are listed)
  - $\mu$ is the average service rate (jobs per second)
  - $T_{ser}$ or $S$ is the average service time, so $T_{ser} = \frac{1}{\mu}$
  - $\lambda$ is the average arrival rate (jobs per second)
  - $U$ or $u$ or $\rho$ is the utilization (fraction from 0 to 1), so $U = \frac{\lambda}{\mu} = \lambda S$
  - $T_q$ or $W$ is the average queuing 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)
  - $T_{sys}$ or $R$ is the response time, and it's equal to $T_q + T_{ser}$ or $W + S$
  - $L_q$ or $Q$ is the average length of the queue, and it's equal to $\lambda T_q$ (this is Little's law)

# 3 Problems

## 3.1 Disabling Interrupts

We looked at disabling CPU interrupts as a simple way to create a critical section in the kernel. Name a drawback of this approach when it comes to I/O devices.

You can't receive interrupts from devices or timers within a critical section now. For instance, what if you accidentally have an infinite loop in the kernel critical section?

## 3.2 Disks

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

```
Queuing 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.

### 3.3 FAT

What does it mean to format a FAT file system? Approximately how many bytes of data need to be written in order to format a 2GiB flash drive (with 4KiB blocks and a FAT entry size of 4 bytes) using the FAT file system?

Formatting a FAT file system means resetting the file allocation table (mark all blocks as free). The actual data can be zero-ed out for additional security, but it is not required. Formatting a 2GiB FAT volume will require resetting $2^{19}$ FAT entries, which will involve approximately $2^{21}$ bytes (2 MiB).

Your friend (who has never taken an Operating Systems class) wants to format their external hard drive with the FAT32 file system. The external hard drive will be used to share home videos with your friend's family. Give one reason why FAT32 might be the right choice. Then, give one reason why your friend should consider other options.

FAT32 is supported by many different operating systems, which will make it a good choice for compatibility if it needs to be used by many users. However, FAT32 has a 4GiB file size limit, which may prevent your friend from sharing large video files with it.

Explain how an operating system reads a file like "`D:\My Files\Video.mp4`" from a FAT volume (from a software point of view).

First, the operating system must know that the FAT volume is mounted as "`D:\`". It looks at the first data block on the FAT volume, which contains the root directory, and searches for a subdirectory named "`My Files`". If necessary, the root directory listing might occupy many blocks, and the operating system will follow the pointers in the file allocation table to scan through the entire root directory. Once the subdirectory entry for "`My Files`" is found, the operating system searches the subdirectory's listing for a file named "`Video.mp4`". Once it knows the block number for the file, it can begin reading the file sequentially by following the pointers in the file allocation table.

Compare bitmap-based allocation of blocks on disk with a free block list.

Bitmap based block allocation is a fixed size proportional to the size of the disk. This means wasted space when the disk is full. A free block list shrinks as space is used up, so when the disk is full, the size of the free block list is tiny. However, contiguous allocation is easier to perform with a bitmap. Most modern file systems use a free block bitmap, not a free block list.

### 3.4 Queuing 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 $\infty$ 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?

$50 \times 0.1 = 5$ (5 jobs at any time). This is Little's law.

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 server that can process $N$ jobs per millisecond is faster. Better response time ($\frac{1}{N}$ sec vs 1 sec) and better utilization (no load-balancing problems), which gives you lower queuing delays on average.

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

$T_q = T_{ser}(\frac{u}{1-u})(\frac{C+1}{2})$ where $u = \lambda S$

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

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