Section 10: Device Drivers, FAT, Queuing Theory, Memory Mapped Files

CS162

Oct 31st, 2017

Contents

1 Warmup: I/O and Device Drivers

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.

Why might you choose to use DMA instead of memory mapped I/O? Give a specific example where one is more appropriate than the other.

DMA is more appropriate when you need to transfer large amounts of data to/from main memory without occupying the CPU, especially when the operation could potentially take a long time to finish (accessing a disk sector, for example). The DMA controller will send an interrupt to the CPU when the DMA operation completes, so the CPU does not need to waste cycles polling the device. While memory mapped I/O can be used to transfer device data into main memory, it must involve the CPU. Memory mapped I/O is useful for accessing devices directly from the CPU (writing to the frame buffer or programming the interrupt vector, for example).

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

- **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)
  - 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 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 λT_q (this is Little's law)

3 Problems

3.1 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.

3.2 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 ∞ 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 λ, 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.

3.3 Tying it all together

Assume that you have a disk with the following parameters:

- 1TB in size
- 6000RPM
- Data transfer rate of 4MB/s (4×10^6 bytes/sec)
- Average seek time of 3ms
- I/O controller with 1ms of controller delay
- Block size of 4000 bytes

What is the average rotational delay?

$\frac{1}{2} \times \frac{60\text{sec/minute}}{6000\text{RPM}} = 5\text{ms}$

What is the average time it takes to read 1 random block? Assume no queuing delay.

$\frac{4{,}000\text{bytes}}{4{,}000{,}000\text{bytes/sec}} = 1\text{ms}$, and $1 + 3 + 5 + 1 = 10\text{ms}$

Will the actual measured average time to read a block from disk (excluding queuing delay) tend to be lower, equal, or higher than this? Why?

It will be lower, because the operating system will use a disk scheduling algorithm to improve locality. This model assumes the disk is always seeking to a random location.

Assume that the average I/O operations per second demanded is 50 IOPS. Assume a squared coefficient of variation of $\mathbf{C} = 1.5$. What is the average queuing time and the average queue length?

$$T_q = T_{ser}(\frac{u}{1-u})(\frac{\mathbf{C}+1}{2})$$
$$u = \lambda T_{ser}$$
$$u = 50\text{IOPS} \times 0.01\text{sec}$$
$$u = 0.5$$
$$T_q = 10\text{ms}(\frac{0.5}{1-0.5})(\frac{1.5+1}{2})$$
$$T_q = 12.5\text{ms}$$
$$L_q = T_q\lambda$$
$$L_q = 0.0125 \times 50$$
$$L_q = 0.625 \text{ operations}$$

3.4 Memory Mapped Files

Memory-mapped files, either as the fundamental way to access files or as an additional feature, is supported by most OS-s. Consider the Unix mmap system call that has the form (somewhat simplified):

void ⋆ mmap(void ⋆ addr, sizet_len, int prot, int flags, int filedes, off_t offset);

where data is being mapped from the currently open file *filedes* starting at the position in the file described by *offset*; *len* is the size of the part of the file that is being mapped into the process address space; prot describes whether the mapped part of the is readable or writable. The return value, *addr*, is the address in the process address space of the mapped file region.

When the process subsequently references a memory location that has been mapped from the file for the first time, what operations happen in the operating system? What happens upon subsequent accesses?

First access - Page Fault. OS commits the page, i.e maps the page to a physical page. Subsequent accesses, the page of the file is there in memory.

Assume that you have a program that will read sequentially through a very large file, computing some summary operation on all the bytes in the file. Compare the efficiency of performing this task when you are using conventional read system calls versus using mmap.

Sequential reading with normal read library may be worse because it may read the file in larger number of chunks, and incurs multiple copy overheads for all reads. Actually the answer is more complex, and depends very much on implementation. You can think of various ways to tune either mmap() or read() performance, depending on what you code uses. To get a sense of the complexities involved, read this answer taken from Stack Overflow : http://stackoverflow.com/questions/9817233/why-mmap-is-faster-than-sequential-io

Assume that you have a program that will read randomly from a very large file. Compare the efficiency of performing this task when you are using conventional read and lseek system calls versus using mmap.

mmap is generally better. Truly random access - similar performance.