162 Dis 10 Notes

Thursday, April 4, 2019 1:21 AM

Section 10: Device Drivers, FAT, Queuing Theory

CS162

April 5, 2019

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?

Block Devices: blocks at a time; Character Devices: Bytes (or stream of them) at a time

+ lots of data transfer e.g. hard drive
+ easier to implement on slow mem
- may be slower on solid state
- eventually needs to be word addr to use

+ small amts of data, e.g. keyboard, sound card
+ flexibility in handling I/O
- not buffered

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: hand off transfer to controller → let CPU know when done
- good for long, slow transfers - things that use it: Disk Drives, GPUs, Network Cards

MM: same addr space for CPU & I/O devices
- reserves space & CPU/device communicate
- easy to implement, flexible, good for embedded systems

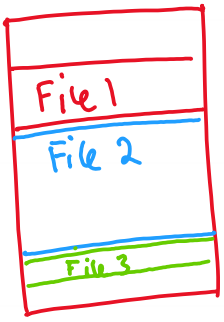
Explain what is meant by "top half" and "bottom half" in the context of device drivers.

Interrupt handler split in half (and so are device drivers)

top half used to start I/O ops ← "can be done later"
bottom half used to service interrupts produced ← "crucial"

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

easy to implement

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.

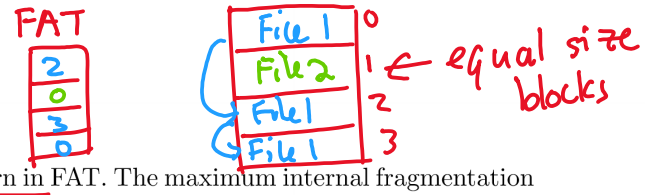
However, this implementation also has many disadvantages. First of all, it has external fragmentation problem. Because only continuous space can be utilized, it may come to the situation that there is enough free space in sum, but none of the continuous space is large enough to hold the whole file. Second, 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?

Reset the device - all blocks freed ← maybe 0 it out to protect data

$2\,GiB = 2^{31}\,B \Rightarrow \frac{2^{31}}{2^{12}} = 2^{19}$ FAT entries

$\Rightarrow$ free 2^{19} entries $\Rightarrow$ work $2^{21}\,B$

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.

+ supported by many OS → compatibility

- FAT 32 → 32 bit addressed $\Rightarrow 2^{32}B = 4\,GiB$ max file size.
  videos probably larger !!

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

1) OS knows D: is some volume, FAT format
2) First block has root dir → looks for subdir in there
   - could follow lots of pointers
3) find subdir → look for file
4) if found, read that data (pointer following)

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

Bitmap

- always same size, wastes space
+ easier for contiguous allocation
+ easier to implement

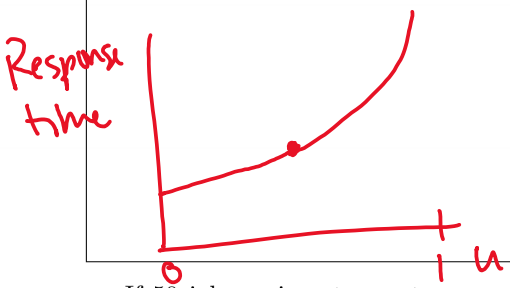
Block List

+ variable in size, shrinks w/ more data ⇒ can use more space

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.

more things to handle → longer to respond, things pile up
- there is some idle time → wasted work → builds up queue



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?

N = jobs λ = arrival rate L = response time

Littles Law: $N = \lambda \cdot L$

$N = 50/s \cdot 0.1s = \boxed{5 \text{ jobs}}$

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.

1 queue: better response time $\frac{1}{N}$ & easier to utilize
(with N queues, you need to assign to each queue → load balancing issue)

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 $\mathbf{C}$?

$T_q = T_{ser} \cdot \left(\frac{u}{1-u}\right)$ ← memoryless

$T_q = T_{ser}\left(\frac{u}{1-u}\right) \cdot \left(\frac{1+c}{2}\right)$ ← general service time

use this, plug in $u = \lambda S$

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

$C = 0 \Rightarrow$ constant rate

$C = 1 \Rightarrow$ Poisson dist of arrivals

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?

$\frac{1}{2} \frac{60000 \text{ms/min}}{6000 \text{ RPM}} = 5\text{ms}$

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

$\frac{4000 \text{ B}}{4000000 \text{ B/s}} = 1\text{ms}$

queue + controller + seek + rotate + transfer
0 + 1 + 3 + 5 + 1 = 10ms

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?

lower - disk scheduling alg improves seek time.

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?

See solns