Section 10: I/O and Basic File Systems

November 6-8, 2019

Contents

1 Clock Algorithm

1.1 Clock Page Table Entry

Suppose that we have a 32-bit virtual address split as follows:

10 Bits Table ID	10 Bits Page ID	12 Bits Offset

Assume that the physical address is 32-bit as well. Show the format of a page table entry (PTE) complete with bits required to support the clock algorithm.

20 Bits PPN	8 Bits Other	1 Bit Dirty	1 Bit Use	1 Bit Writable	1 Bit Valid

1.2 Clock Algorithm Step-through

For this problem, assume that physical memory can hold at most four pages. What pages remain in memory at the end of the following sequence of page table operations and what are the use bits set to for each of these pages?

Page	A	B	C	A	C	D	B	D	A	E	F

E: 1, F: 1, C: 0, D: 0
Recall that the clock hand only advances on page faults. No page replacement occurs until $t = 10$, when all pages are full. At $t = 10$, all pages have the use bit set. The clock hand does a full sweep, setting all use bits to 0, and selects page 1 (currently holding A) to be paged out. The clock hand advances and now points to page 2 (currently holding B). At $t = 11$, we check page 2's use bit, and since it is not set, select page 2 to be paged out. F is brought in to page 2. The clock hand advances and now points to page 3. We reach the end of the input and end.
Note: The table shows the clock hand position before page faults occur.

Page	A	B	C	A	C	D	B	D	A	E	F
1	A: 1	A: 1	A: 1	A: 1	A: 1	A: 1	A: 1	A: 1	A: 1	E: 1	E: 1
2		B: 1	B: 1	B: 1	B: 1	B: 1	B: 1	B: 1	B: 1	B: 0	F: 1
3			C: 1	C: 1	C: 1	C: 1	C: 1	C: 1	C: 1	C: 0	C: 0
4						D: 1	D: 1	D: 1	D: 1	D: 0	D: 0
Clock	1	2	3	4	4	4	1	1	1	1	2

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 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.
- **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.
- **Unix File System (Fast File System)** - The Unix File System is a file system used by many Unix and Unix-like operating systems. Many modern operating systems use file systems that are based off of the Unix File System.
- **inode** - An inode is the data structure that describes the metadata of a file or directory. Each inode contains several metadata fields, including the owner, file size, modification time, file mode, and reference count. Each inode also contains several data block pointers, which help the file system locate the file's data blocks.

Each inode typically has 12 direct block pointers, 1 singly indirect block pointer, 1 doubly indirect block pointer, and 1 triply indirect block pointer. Every direct block pointer directly points to a data block. The singly indirect block pointer points to a block of pointers, each of which points to a data block. The doubly indirect block pointer contains another level of indirection, and the triply indirect block pointer contains yet another level of indirection.

3 Input/Output

3.1 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.

3.2 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.

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).

3.3 Storage Devices

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.

4 File Allocation Table

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.

5 Inode-Based File System

1. What are the advantages of an inode-based file system design compared to FAT?

   Fast random access to files. Support for hard links.

2. What is the difference between a hard link and a soft link?

   Hard links point to the same inode, while soft links simply list a directory entry. Hard links use reference counting. Soft links do not and may have problems with dangling references if the referenced file is moved or deleted. Soft links can span file systems, while hard links are limited to the same file system. Fast random access to files. Support for hard links.

3. Why do we have direct blocks? Why not just have indirect blocks?

   Faster for small files.

4. Consider a file system with 2048 byte blocks and 32-bit disk and file block pointers. Each file has 12 direct pointers, a singly-indirect pointer, a doubly-indirect pointer, and a triply-indirect pointer.

   (a) How large of a disk can this file system support?

   2^{32} blocks x 2^{11} bytes/block $= 2^{43} =$ 8 Terabytes.

   (b) What is the maximum file size?

   There are 512 pointers per block (i.e. 512 4-byte pointers in 2048 byte block), so: blockSize $\times$ (numDirect + numIndirect + numDoublyIndirect + numTriplyIndirect)

   $$\begin{aligned}2048 \times (12 + 512 + 512^2 + 512^3) &= 2^{11} \times (2^2 \times 3 + 2^9 + 2^{9\times 2} + 2^{9\times 3}) \\ &= 2^{13} \times 3 + 2^{20} + 2^{29} + 2^{38} \\ &= 24K + 513M + 256G\end{aligned}$$

5. Rather than writing updated files to disk immediately, many UNIX systems use a delayed *write-behind policy* in which dirty disk blocks are flushed to disk once every x seconds. List two advantages and one disadvantage of such a scheme.

   Advantage 1: The disk scheduling algorithm (i.e. SCAN) has more dirty blocks to work with at any one time and can thus do a better job of scheduling the disk arm.
   Advantage 2: Temporary files may be written and deleted before data is written to disk.
   Disadvantage: File data may be lost if the computer crashes before data is written to disk.

6. List the set of disk blocks that must be read into memory in order to read the file `/home/cs162/test.txt` in its entirety from a UNIX BSD 4.2 file system (10 direct pointers, a singly-indirect pointer, a doubly-indirect pointer, and a triply-indirect pointer). Assume the file is 15,234 bytes long and that disk blocks are 1024 bytes long. Assume that the directories in question all fit into a single disk block each. Note that this is not always true in reality.

1. Read in file header for root (always at fixed spot on disk).
2. Read in first data block for root (/).
3. Read in file header for home.
4. Read in data block for home.
5. Read in file header for cs162.
6. Read in data block for cs162.
7. Read in file header for test.txt.
8. Read in data block for test.txt.

9. - 17. Read in second through 10th data blocks for test.txt.

18. Read in indirect block pointed to by 11th entry in test.txt's file header.

19. - 23. Read in 11th – 15th test.txt data blocks. The 15th data block is partially full.