Section 9: Intro to I/O and File Systems

October 16, 2015

Contents

1 Warmup

1.1 True/False

Write True or False for each question and justify your answer.

1. 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. 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. 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. 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. Directories in UNIX are basically files with pointers to inodes in them.

   True, this is basically how directories work.

2 Vocabulary

- **Second Chance Algorithm** A modified form of FIFO that is used to approximate LRU. Each page also has a reference of a use bit to keep track of whether that page has been accessed. It works by looking at the front of the queue as FIFO does, but instead of immediately paging out that page, it checks to see if its referenced bit is set. If it is not set, the page is swapped out. Otherwise, the referenced bit is cleared, the page is inserted at the back of the queue and the process is repeated until a page is swapped out.
- **Clock Algorithm** Clock is a more efficient version of FIFO than Second-chance because pages don't have to be constantly pushed to the back of the list. The clock algorithm keeps a circular list of pages in memory, with the "hand" pointing to the last examined page in the list. When a page fault occurs and no empty frames exist, then the use bit is inspected at the hand's location. If U is 0, the new page is put in place of the page the "hand" points to, otherwise the U bit is cleared. Then, the clock hand is incremented and the process is repeated until a page is replaced.
- **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.
- **inode** In UNIX based operating systems, an inode is a data structure used to represent a filesystem object. It contains attributes about the file, as well as the locations of the disk blocks where the file contents reside. For large files, it also contains pointers to multi-level disk block locations.

3 Problems

3.1 Clock Algorithm

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

10 Bits	10 Bits	12 Bits
Table ID	Page ID	Offset

Show the format of a PTE complete with bits required to support the clock algorithm.

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

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 is paged in
- Page B is paged in
- Page C is paged in
- Page A is accessed
- Page C is accessed
- Page D is paged in
- Page B is accessed
- Page D is accessed
- Page A is accessed
- Page E is paged in
- Page F is paged in

E: 0, F: 0, C: 0, D: 0

3.2 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?

3.3 File Systems

Indirect blocks look a lot like another data structure we looked at earlier this semester. What is the other data structure? Can you compare the tradeoffs you make in either case?

Its very similar to the indirection used in multi level page tables. This also lets you save space (i.e. your inode is smaller in the general case) at the cost of access delay (you have to seek several times). The idea is nearly identical with page tables.

How many times is the disk accessed when you type ls /home/cs162? What is each access for? Assume you already know the block number of the root inode and that ls and all of its corresponding libraries are already loaded in memory.

```
6 times
- Once to read root inode
- Once to read root directory listing
- Once to read home inode
- Once to read home directory listing
- Once to read cs162 inode
- Once to read cs162 directory listing
```

Assuming the total response time to read an inode or block from disk is 5ms, and all inodes and directories consume one block, how long does this take?

6*5 = 30ms

Now say the following is true:
- The root directory listing is cached in memory.
- The disk controller is using a track buffer.
- I-nodes are always stored on the same cylinder as the blocks they refer to.
Assume that data in the track buffer or in memory is free (e.g. 0ms) to access. Now much time would it take to read the contents of /home/cs162?

Now it takes 10ms. You dont have to do the first two seeks, and the 3-4 and 5-6 seeks incur only one seek delay due to track buffering.

3.4 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.