Section 9: Intro to I/O and File Systems

March 19, 2019

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).

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 File** A memory-mapped file is a segment of virtual memory which has been assigned a direct byte-for-byte correlation with some portion of a file or file-like resource. This resource is typically a file that is physically present on-disk, but can also be a device, shared memory object, or other resource that the operating system can reference through a file descriptor. Once present, this correlation between the file and the memory space permits applications to treat the mapped portion as if it were primary memory.
- **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.

3 Problems

3.1 Clock Algorithm

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

10 Bits Table ID	10 Bits Page ID	12 Bits Offset

Show the format of a 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

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

E: 1, F: 1, 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 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.

3.4 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.