

ITEE



What's New:

There are fewer questions this time again, since some of you may want to focus on your assignment. Should you get behind on tutorials, these questions can be worked on a week late or in lectures as time allows.

Learning objectives for this week:

- Understand the major concepts we've covered, including:
 - different types of I/O hardware
 - implications of speed differences
 - solutions to performance problems
- Apply these concepts to reasoning about I/O hardware and the interface of I/O to operating systems
- Apply these concepts to choosing an appropriate implementation strategy
- Understand issues in improving performance

Solutions this time are reasonably complete, but some alternatives to a few of the questions are proposed so you can work through different numbers to make sure you have the concepts and techniques straight.

1. Concepts

- a. Why is performance a hard problem when dealing with I/O?

There are huge speed differences – the CPU is clocked in the GHz range (1 GHz is 1ns per clock tick – 10^{-9} – and each multiple of GHz makes the time per clock tick smaller) – and most I/O devices have latencies in the ms (10^{-3}) range. Bridging these speed gaps without having the average time per instruction closer to the slowest part of the system than the fastest part of the system is a huge challenge.

- b. Why is polling not a desirable strategy?

Polling requires that the CPU waste time checking the status of an I/O device when the status may not change over a very large number of CPU cycles. A trade-off has to be made between polling often enough to deal with I/O with minimal delay, and polling infrequently enough not to waste CPU time unnecessarily.

- c. Explain why disks have largely replaced slower media like tape.

Purely in terms of speed, if you can use a disk instead of a tape, you will. What has driven tape out of mainstream I/O to only being used as backup is a dramatic drop in the price of disk, in relation to demand. Terabytes of disk are sufficiently affordable to replace tape in large-scale installations.

2. Low-Level Implementation

- a. Explain why DMA can sometimes still interfere with CPU operation.

A DMA operation, though it does not directly use the CPU, uses the same memory as the CPU. If DMA uses a part of memory which is not cached, the CPU will not see DMA activity unless it overlaps with a cache miss. In that case, the cache miss will contend with the DMA operation for the bus and DRAM, and may be slowed down. If DMA uses memory which can be cached, the effect would be more serious, because DMA activity which modifies memory (a disk read, for example, which would modify a buffer in RAM) would result in the cached copy becoming invalid. This invalidated cached data would have to be flushed from the cache, which would be noticeable at the CPU, where the caches (especially first-level) are tightly integrated with CPU operation. For this reason, DMA operations usually work on memory which can't be cached, or which is in kernel space, and has to be copied to the user process after DMA completes.

- b. Give three differences between a networked I/O operation and a disk I/O operation, and explain whether the differences are significant.

Here are some differences (many others are possible):

- *a networked I/O operation could time out – while this could happen with a disk, it's much easier to manage timeouts in the hardware and do retries up to some reasonable limit on local devices because information is moving on a reliable medium*
- *networked operations involve a protocol stack – the OS can't just hand the operation to a simple driver: networks have layered protocols, which involve several layers of software before hitting the hardware, which makes for more overheads, so significant speed loss is possible*
- *the network is more general – because the network is not specific to one kind of device, a networked disk operation is more unpredictable in terms of timing, probability of failure and effects of external events in general; this difference is quite significant in what an OS can expect of a disk operation*

- c. Which of the following can best be described as *blocking*, *nonblocking*, or *asynchronous* (as defined in the textbook pp 471–472, lecture 10 slide 14)

None of these is a nonblocking call, which is one which returns immediately with whatever results it can immediately determine; an example of a nonblocking call would be one which requests the next character from the keyboard, and returns a value whether one was available or not, with some way of showing that no value was available.

- i. You send an I/O request to a device, and continue doing other work. You later check whether a variable has been set specifying the I/O has completed.
Asynchronous – check against the discussion in the book.
- ii. You launch a thread to do I/O, and that specific thread can block, but others continue.
This is also a blocking call, but one which doesn't halt the whole process.
- iii. You do an I/O operation, and wait for it to complete.
Blocking – see the book if you don't get it.

3. Disk Scheduling

- a. For each of the following disk scheduling algorithms, work through the head movements for the queue 100, 200, 300, 10, 12, 44
- i. FCFS

Missing from the question: where the head was to start with. Let's start it at 100 (you could start it anywhere to vary the question) and keep this consistent for other algorithms.

Easy – in the order given; total head movement is $(200-100) + (300 - 200) + (300 - 10) + (12 - 10) + (44 - 12) = 524$.

ii. SSTF

Here we are doing the shortest seek time first, so we start at 100, then the next shortest seek is 44, then 12, then 10, then 200, then 300, for a total: $(100-44)+(44-12)+(12-10)+(200-10)+(300-200) = 380$

iii. SCAN

Missing from question: direction we are going to start, and the highest cylinder number. Assume we are starting from 100 (as before), and scanning towards high numbers (for a variation in the question, go the other way, or start somewhere else), highest cylinder number 300: total = $(200-100)+(300-200)+(300-44)+(44-12)+(12-10) = 490$

iv. LOOK

Same as SCAN except we don't go right to the end of the disk, but stop at the last request. In this case, the "forward" scan does go right to the end (cylinder 300), so there is no difference, but if the initial direction had been reversed, we would have stopped in the LOOK algorithm at 44 before turning back, whereas SCAN would have gone back to 0.

v. C-LOOK

Here we go back to the lowest-numbered request after hitting the highest-numbered request; total = $(200-100)+(300-200)+(300-10)+(12-10)+(44-12) = 524$

- b. Which – both in this case, and in general – is likely to result in less overall head movement? Consider what would happen if the queue could grow as requests were being serviced.

In this case, SSTF was best, and in general, is likely to be close to optimal. As new requests arrive, if they are closely clustered, the nearest ones can be taken advantage of, which could look better than the examples we've used without new arrivals. **Further thoughts, not asked in the question ...** However, it doesn't take into account the fact that reversing head direction is slower than keeping moving in a fixed direction, which the SCAN and LOOK variations do take into account. It can also result in starvation, and so in practice, a LOOK or SCAN variation is more likely to be used.

- c. If it is a lot quicker to move the head a long distance at once, than to move that same distance in smaller steps, which of the algorithms looked at before might be better than you first thought?

See last part of (b): also, C-LOOK is probably better in practice than it looks from this example, because the (300-10) move to get back to the first request after the initial scan is faster than it looks.

4. Long-Term Trends

Disk farms appear to have largely replaced tape drives for large-scale storage. Discuss why this is the case, and what may change in future, which may cause disks to be replaced by another technology. In the light of this discussion, is the whole section in the book (14.8) about tertiary storage irrelevant?

The cost per bit of disks has become increasingly competitive with tape and given the high latency of tape, especially the fact that drives are expensive and the latency in minutes of swapping tapes can't be avoided, makes tapes very uncompetitive if the price is anywhere close. The discussion in the book is useful because should design trade-offs change as the result of some new technology (e.g., extremely cheap recordable DVDs) replacing tapes, but with some of the same performance issues, the idea of large tape libraries and related technologies could come back in another form.