I/O Systems

Practice Exercises

12.1 State three advantages of placing functionality in a device controller, rather than in the kernel. State three disadvantages.

Answer:
Three advantages:

a. Bugs are less likely to cause an operating-system crash.

b. Performance can be improved by utilizing dedicated hardware and hard-coded algorithms.

c. The kernel is simplified by moving algorithms out of it.

Three disadvantages:

a. Bugs are harder to fix—a new firmware version or new hardware is needed.

b. Improving algorithms requires a hardware update rather than just a kernel or device-driver update.

c. Embedded algorithms could conflict with an application's use of the device, causing decreased performance.

12.2 The example of handshaking in Section 12.2 used two bits: a busy bit and a command-ready bit. Is it possible to implement this handshaking with only one bit? If it is, describe the protocol. If it is not, explain why one bit is insufficient.

Answer:
It is possible, using the following algorithm. Let's assume we simply use the busy bit (or the command-ready bit; this answer is the same regardless). When the bit is off, the controller is idle. The host writes to data-out and sets the bit to signal that an operation is ready (the equivalent of setting the command-ready bit). When the controller is finished, it clears the busy bit. The host then initiates the next operation.

This solution requires that both the host and the controller have read and write access to the same bit, which can complicate circuitry and increase the cost of the controller.

12.3 Why might a system use interrupt-driven I/O to manage a single serial port and polling I/O to manage a front-end processor, such as a terminal concentrator?

Answer:
Polling can be more efficient than interrupt-driven I/O. This is the case when the I/O is frequent and of short duration. Even though a single serial port will perform I/O relatively infrequently and should thus use interrupts, a collection of serial ports such as those in a terminal concentrator can produce a lot of short I/O operations, and interrupting for each one could create a heavy load on the system. A well-timed polling loop could alleviate that load without wasting many resources through looping with no I/O needed.

12.4 Polling for an I/O completion can waste a large number of CPU cycles if the processor iterates a busy-waiting loop many times before the I/O completes. But if the I/O device is ready for service, polling can be much more efficient than catching and dispatching an interrupt. Describe a hybrid strategy that combines polling, sleeping, and interrupts for I/O device service. For each of these three strategies (pure polling, pure interrupts, hybrid), describe a computing environment in which that strategy is more efficient than either of the others.

Answer: A hybrid approach could switch between polling and interrupts depending on the length of the I/O operation wait. For example, we could poll and loop N times, and if the device is still busy at N+1, we could set an interrupt and sleep. This approach would avoid long busy-waiting cycles and would be best for very long or very short busy times. It would be inefficient if the I/O completes at N+T (where T is a small number of cycles) due to the overhead of polling plus setting up and catching interrupts.

Pure polling is best with very short wait times. Interrupts are best with known long wait times.

12.5 How does DMA increase system concurrency? How does it complicate hardware design?

Answer:
DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory buses. Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and the DMA controller to share use of the memory bus.

12.6 Why is it important to scale up system-bus and device speeds as CPU speed increases?

Answer:
Consider a system that performs 50% I/O and 50% computes. Doubling the CPU performance on this system would increase total system performance by only 50%. Doubling both system aspects would increase performance by 100%. Generally, it is important to remove the current system bottleneck, and to increase overall system performance, rather than blindly increasing the performance of individual system components.

12.7 Distinguish between a driver end and a stream module in a STREAMS operation.

Answer:
The driver end controls a physical device that could be involved in a STREAMS operation. The stream module modifies the flow of data between the stream head (the user interface) and the driver.

Exercises

12.8 What are the advantages and disadvantages of supporting memory-mapped I/O to device-control registers?

Answer:
The advantage of supporting memory-mapped I/O to device-control registers is that it eliminates the need for special I/O instructions from the instruction set and therefore does not require the enforcement of protection rules that prevent user programs from executing these I/O instructions. The disadvantage is that the resulting flexibility needs to be handled with care; the memory translation units need to ensure that the memory addresses associated with the device-control registers are not accessible by user programs in order to ensure protection.

12.9 In most multiprogrammed systems, user programs access memory through virtual addresses, while the operating system uses raw physical addresses to access memory. What are the implications of this design for the initiation of I/O operations by the user program and their execution by the operating system?

Answer:
The user program typically specifies a buffer for data to be transmitted to or from a device. This buffer exists in user space and is specified by a virtual address. The kernel needs to issue the I/O operation and needs to copy data between the user buffer and its own kernel buffer before or after the I/O operation. In order to access the user buffer, the kernel needs to translate the virtual address provided by the user program to the corresponding physical address within the context of the user program's virtual address space. This translation is typically performed in software and therefore incurs overhead. Also, if the user buffer is not currently present in physical memory, the corresponding pages need to be obtained from the swap space. This operation might require careful handling and might delay the data copy operation.

12.10 Describe three circumstances under which blocking I/O should be used. Describe three circumstances under which nonblocking I/O should be used. Why not just implement nonblocking I/O and have processes busy-wait until their devices are ready?

Answer:
Generally, blocking I/O is appropriate when the process will be waiting only for one specific event. Examples include a disk, tape, or keyboard read by an application program. Nonblocking I/O is useful when I/O may come from more than one source and the order of the I/O arrival is not predetermined. Examples include network daemons listening to more than one network socket, window managers that accept mouse movement as well as keyboard input, and I/O-management programs, such as a copy command that copies data between I/O devices. In the last case, the program could optimize its performance by buffering the input and output and using nonblocking I/O to keep both devices fully occupied.

Nonblocking I/O is more complicated for programmers, because of the asynchronous rendezvous that is needed when an I/O occurs. Also, busy waiting is less efficient than interrupt-driven I/O, so the overall system performance would decrease.

12.11 Some DMA controllers support direct virtual memory access, where the targets of I/O operations are specified as virtual addresses and a translation from virtual to physical address is performed during the DMA. How does this design complicate the design of the DMA controller? What are the advantages of providing such functionality?

Answer:
Direct virtual memory access allows a device to perform a transfer from two memory-mapped devices without the intervention of the CPU or the use of main memory as a staging ground; the device simply issues memory operations to the memory-mapped addresses of a target device, and the ensuing virtual address translation guarantees that the data are transferred to the appropriate device. This functionality, however, comes at the cost of having to support virtual address translation on addresses accessed by a DMA controller and requires the addition of an address-translation unit to the DMA controller. The address translation results in both hardware and software costs and might also result in coherence problems between the data structures maintained by the CPU for address translation and corresponding structures used by the DMA controller. These coherence issues would also need to be dealt with and would result in a further increase in system complexity.

12.12 Write (in pseudocode) an implementation of virtual clocks, including the queueing and management of timer requests for the kernel and applications. Assume that the hardware provides three timer channels.

Answer:
Each channel would run the following algorithm:

```
// A list of interrupts sorted in earliest-time-first
    order
List interruptList

// the list that associates a request with an entry
    in interruptList
List requestList

// an interrupt-based timer
Timer timer

while (true) {
    // Get the next earliest time in the list
    timer.setTime = interruptList.next();

    // An interrupt will occur at time timer.setTime

    //now wait for the timer interrupt
    i.e. for the timer to expire

   notify( requestList.next() );
}
```