CS 4500 - Operating Systems
Module 8: Input/Output

Stanley Wileman

Department of Computer Science
University of Nebraska at Omaha
Omaha, NE 68182-0500, USA

August 2, 2017
In This Module...

- Principles of Input / Output Hardware
- Overview of Input / Output Hardware
- Details of Common Input / Output Devices
  - Disks
  - RAM Disks
  - Clocks
  - Terminals
Much I/O hardware maps software requests to physical actions external to a processor.

Some I/O devices are virtual - that is, they don’t physically exist.

Understanding I/O requires understanding

- the operational characteristics of I/O devices (how they are used), and
- the software interface to the I/O devices from the operating system.
I/O Devices can be characterized in several ways:

- **What is the purpose of the device?**
  - data storage (e.g. disk drive, tape, CD-ROM, flash drive)
  - data transfer (e.g. serial port, modem, network interface, USB hub, Ethernet switch, \(I^2C\) and \(SPI\) bus)
  - environmental interface (e.g. keyboard, monitor, mouse, A/D converter)

- **How is the device accessed?**
  - block by block
  - one character or byte at a time
A **device driver** is the software component of the operating system that manages detailed interactions with the device.

Each device driver provides a common interface to the kernel of the operating system.

To some extent, then, the drivers **normalize** the I/O software used by the operating system.

Example: different disks have different sector sizes, number of cylinders, tracks, etc. But a disk device driver might provide a generic function like “read block,” given a buffer address, a device number, and a logical block number that will work with all of the different disks.
Device Controllers

- A **device controller** consists of the electronic interface between a device and the computer system. A device controller is sometimes called a **device adapter**.
- The controller physically connects the device to the computer system’s bus.
- Many modern devices have a device controller as an integral component.
- Example: SCSI devices, IDE disk drives
- For these devices a simplified interface (frequently called a **host adapter**) is used to connect it to the system’s bus.
Typical Device Driver Functions

- At autoconfiguration, to determine availability and to initialize
- To perform I/O resulting from standard library I/O calls
- To handle interrupts from the device to say data is ready
- To handle special requests such as an ioctl
- On some buses, to reset and reinitialize the device
- To perform user-level requests to the sysconfig utility and kernel configuration requests
Each device controller has two interfaces, one to the system bus and one to the device.

Some device controllers can handle multiple devices of the same type. For example, most PC floppy disk controllers can handle two drives.

Many device interfaces are standardized, including those for floppy disks, IDE and SCSI hard disks, and network interfaces. Likewise, many bus interfaces are standardized.

The operating system (device driver) always interacts with the controller, not directly with the device.
A Real PCs I/O Device, Bus, Port, and Memory Layout

From http://duartes.org/gustavo/blog/post/motherboard-chipsets-memory-map
Mainframe Data Channels

- Used by mainframes, a data channel is a specialized processor that deals exclusively with I/O operations.
- Channel programs, written using an instruction set different from the main processors, are executed exclusively by the I/O channels.
- Device controllers attach to the I/O channel’s bus, and not to the same bus as the memory and processors.
- There are two major kinds of channels: one for block structured devices (called a selector channel), and one for character devices (called a multiplexor channel).
A Possible Mainframe Configuration

- CPU
- CPU
- Memory

- I/O Channel #1
- I/O Channel #2

- Controller #1
- Controller #2
- Etc.

- Device
- Device
- Device
- Device
Bus Adapters

- A bus is used to connect various components of a computer system.
- It is often desirable to provide more than one type of bus in a system, each bus designed for the typical communication needs of a set of devices.
- Device controllers are designed to be connected to a specific bus (e.g. Multibus, ISA, PCI, USB, ATA, SCSI)
- A bus adapter can be used to interconnect the separate busses in a system.
- Example: Sequent uses a proprietary bus for its parallel systems, but uses Multibus device controllers for terminal attachment to the system.
A Sequent Configuration

- CPU
- CPU
- CPU
- CPU
- Memory

- Multibus Adapter
- Controller #1
- Terminal Controller
- Device
- Device
- MULTIBUS (STANDARD)

- Term
- Term
A bridge is typically a device (often a single integrated circuit) that functions as a bus adapter and a router.

Many PCs employ two bridges:
- The northbridge handles fast communication (with RAM and video).
- The southbridge deals with slower devices (disks, USB ports, PCI devices, etc.)

As integrated circuit technology develops, many of the bridge functions are included on the processor chip, even including video controller/GPU features.
Disk Drives

- Floppy and most modern hard disk drive motors can be turned on and off.
- The serial data stream to/from a disk drive is created/interpreted by the disk controller.
- The data in each disk sector is preceded by a preamble containing identifying information (like the track number, cylinder number, and sector number).
- The speed of the disk is constant (during operation), requiring the controller to keep up with the disk.
The display itself is functionally similar to those found in televisions, with the screen’s pixels (the smallest individual picture elements) being continually and rapidly updated (e.g. the entire screen might be redrawn 120 times each second).

The controller must “pump” data to the display to keep up with the constant scan rate.

The video data comes from “video RAM” (directly or indirectly). This RAM maintains a copy of the displayed data, and can be directly accessed by the processor.

Commands to the controller modify various video parameters, primarily at system initialization time. Manipulating the display controller registers is done relatively infrequently.
Direct Memory Access (DMA)

- In older computer systems, the processor had to keep up with the constant data rate required by the device, essentially eliminating any possibility of multiprocessing (multitasking).
- DMA devices manage data transfers between the controller and the system’s primary memory independent of the CPU (once the transfer has been set up).
- DMA is a support device that connects to the system bus.
DMA transfers are usually performed only for block devices that transfer many bytes for each request. The DMA device can be used in both directions.
Interleaving Disk Sectors

- After a sector has been read from the disk, it must be transferred to primary memory (usually using DMA).
- The disk continues to rotate after a sector has been read, thus
- The next sequential sector has “passed by” before the controller is ready to read again.
- To eliminate the delay while waiting for the next sequentially numbered sector, some systems logically number the sectors so they are not contiguous!
- Instead, the sectors are numbered to allow some processing time for a sector before the next sequential sector is ready for reading (or writing).
Assume the disk rotates at 1 rev/sec (unrealistic, but easy for the math), and processing the data from a sector requires 1/8 sec. The green arrow (at the top) shows the initial head position. The red arrow shows the position of the head after processing sector 1. How long will it take to process all sectors in the order 1, 2, 3, 4 in each case?
Interleaving Example - Calculations

- Non-interleaved
  - 1/4 sec to read and 1/8 sec to process sector 1; $t = 3/8$ sec
  - 7/8 sec delay to reach beginning of sector 2; $t = 10/8$ sec
  - 1/4 sec to read and 1/8 sec to process sector 2; $t = 13/8$ sec
  - 7/8 sec delay to reach beginning of sector 3; $t = 20/8$ sec
  - 1/4 sec to read and 1/8 sec to process sector 3; $t = 23/8$ sec
  - 7/8 sec delay to reach beginning of sector 4; $t = 30/8$ sec
  - 1/4 sec to read and 1/8 sec to process sector 4; $t = 33/8$ sec

- Interleaved
  - 1/4 sec to read and 1/8 sec to process sector 1; $t = 3/8$ sec
  - 1/8 sec delay to reach beginning of sector 2; $t = 4/8$ sec
  - 1/4 sec to read and 1/8 sec to process sector 2; $t = 7/8$ sec
  - 3/8 sec delay to reach beginning of sector 3; $t = 10/8$ sec
  - 1/4 sec to read and 1/8 sec to process sector 3; $t = 13/8$ sec
  - 1/8 sec delay to reach beginning of sector 4; $t = 14/8$ sec
  - 1/4 sec to read and 1/8 sec to process sector 4; $t = 17/8$ sec

- The interleaved disk's data can be processed in about half the time required for the non-interleaved disk.
Principles of I/O Software

- Build I/O software in layers, so each layer can deal with one type of complexity.
- The lowest layers deal with device and controller details (e.g. device drivers).
- The upper layers provide a clean, generic interface to the user software.
Goals of the I/O Software - 1

- **Device independence**: most devices should be capable of being used for an arbitrary purpose without modifying user software (e.g. I/O redirection, pipes vs. files)

- **Uniform naming conventions**: device characteristics should not influence file naming requirements unnecessarily

- **Error handling**: most transient errors should be handled by the lower layers. Only catastrophic errors should be reported to the user. Error logs should be maintained.
Concurrent: users should be able to wait for I/O completion, or start an I/O operation and continue computing (or start another I/O operation).

Shared devices (printers, disks, some communication devices): these should be managed so each user thinks they are the only one using the device. Dedicated devices (e.g. terminals) must be protected from misuse.
The I/O software in a system is built in layers, with each layer having explicit responsibilities. From the bottom (lowest) layer up, these components are as follows:

- Interrupt Handlers (the lower “half” of a device driver)
- Device drivers (the upper “half”)
- Device-independent OS software
- User-level software
Interrupts (short asynchronous messages from a device controller to a processor) are generated when an I/O operation completes or an important device event occurs; these happen at imprecise times which is why interrupts are asynchronous.

An interrupt report identifies its source, but additional information must be explicitly obtained from the device controller by the processor. For example, termination status, device status, and small amounts of data (for keyboards, serial ports, etc.) must be obtained from or transmitted to the controller.

Since interrupt handlers must not block, they often perform an up (or V) operation on the semaphore on which a process blocked in the upper half of the device driver is waiting.
An interrupt handler is often characterized as belonging to the related device driver, but they have different entry mechanisms.

**UNIX** device drivers often have the following functions:
- Initialization
- Open and close
- Character read and write
- Block strategy routine
- General purpose I/O control routine

We will next consider the the tasks associated with each of these functions.
Device Initialization

- This routine is usually called once during the startup of an operating system.
- It may probe the system to determine specific I/O device configuration information.
- It will initialize counters, local variables, and allocate any needed resources.
- It permits drivers to be included for devices that may optionally be present.
Device Open and Close Routines

- These routines are invoked by the kernel on behalf of a user process that issues the open or close system calls for a device.
- For simple devices, these routines may have no required actions.
- For other devices, these routines guarantee the device is placed in a consistent/specified state.
- These are **NOT** the same as the open/close routines for disk files!
For character devices, these routines accomplish the transfer of a single byte, or a sequence of bytes between a buffer and the user process.

If the device is busy, the routines will queue the request until the device becomes available, when an interrupt will trigger its processing.
The kernel passes a structure containing a request for I/O on a block structured device.

If the device is idle, the requested action is started immediately.

Otherwise, a request is queued until it can be processed (indicated when the device generates an interrupt).

The requests are not necessarily performed in FIFO order. They may be reordered for efficiency. This is what gives this routine its name.
General Purpose I/O Control

- This routine is used to handle device requests that do not fit the traditional open/close/read/write model.
- For example, setting the data rate on a serial port would be handled by this routine.
- This routine is invoked by the kernel when a user process issues the `ioctl` system call.
Q. What happens if an I/O request is issued, but the corresponding interrupt never occurs?

A. Obviously this would “lock up” the device driver, since it would never be able to proceed to the next request.

Good driver design will include setting a timer when an I/O request starts, so a missing interrupt can be identified when the timer goes off.
Device-Independent I/O Software

The following activities are usually handled in the kernel, often without regard to the device type.

- Uniform interfacing for device drivers
- Device naming
- Device protection
- Storage allocation on block devices
- Allocating and releasing dedicated devices
- Error reporting and logging
- Device-independent (logical) block sizes
- Buffering
There may be many different applications and libraries that focus on input/output, either on specific devices or on generic devices. Here’s a short summary of the types of things these applications and libraries might address.

- System call interface to application programs. This represents the usual open, close, read, write, and ioctl system calls for devices.
- I/O libraries (e.g. formatting functions, language-specific I/O “statements”).
- User-level buffering—reading a large amount of data with one system call, but then performing the extraction of data in user space with no kernel involvement (e.g. getchar()).
- Utility programs (e.g. UNIX lpr/lpd to deal with line printer queueing and control, and mt to deal with magnetic tape drives, dd to do copies between arbitrary device types)
Advantages of Disks Over Random-Access Memory (RAM)

- Persistence: disks retain their contents after power is removed
- Price per bit is lower; typical 2016 prices:
  - RAM costs about $5 per GB
  - SATA disk costs about $25 - $50 per TB
  - (SSD costs about $30 - $50 per 100 GB)
- Disk storage capacity is much larger
128GB SSD and 500 GB HDD Price Trends

Source: DRAMeXchange, Mar., 2016
Cost Projection: Magnetic Disk vs. NAND Flash

Source: Wikibon 2014. 4-Year Cost/TB Magnetic Disk includes Power, Maintenance, Space & Disk Data Reduction. 4-year Cost/TB SSD includes Power, Maintenance, Space, SSD Data Reduction & Data Sharing.
Physical Organization

- One or more “platters,” each with two surfaces (some may utilize only one of the two surfaces)
- Each surface has multiple concentric tracks
- Each track has some number of sectors

- Most modern drives have integrated controllers (e.g. SCSI, IDE, SATA)

- Overlapped seeks are possible with many systems (positioning the heads on one disk while transferring data on a different disk).
Disk Drive Terminology

- Sector
- Tracks
- Heads
- Platters
- Cylinder: set of tracks which can be accessed without moving the arm

Figure from wwwranish.com
Legacy / Advanced Sector Formats

Legacy 512Byte Sectors

4k Advanced Format (with data density equal to current legacy sectors)

4096 Bytes (4KB)

Key: □ = Sync/DAM □ = Data □ = ECC

Sector Gap

7-11% saving
Steps in Reading or Writing a Sector

Assuming the disk drive and controller are ready to accept a command, these are the steps required in a typical system to read or write the contents of a disk sector.

- Issue a _seek_ command to move the heads to the appropriate cylinder and wait for that to complete.
- Issue a _read_ or _write_ command (the seek and read/write can often be combined in one command)
- Wait for command completion (usually indicated by an interrupt).
- Assess the results (examine status register)
Three Separate Times for Disk I/O Operations

The total time required to perform a disk read or write operation on a moving head disk can be divided into three parts:

- **Seek time**: the time required for the heads to move to the proper cylinder. This time is not fixed, and is related to the distance between the position of the heads at the beginning of the operation and the position of the sector(s) to be accessed.

- **Rotational latency**: the time required for the disk to rotate so the beginning of the desired sector is under the heads. This time is directly related to the rotational speed of the disk. If the target disk locations are random, then the expected rotational latency will be half the time required for a complete disk rotation.

- **Transfer time**: the time required to transfer the data. This time is directly related to the rotational speed of the disk, the number of sectors being transferred, and the number of sectors per track. If there are $N$ sectors per track, the time required to transfer one sector is $1/N$ of the time required for a complete disk rotation.

Note that these individual times cannot usually be observed directly.
Modern Disk Drive Organization

- Modern disk drives have several common organizations:
  - disks with rotating media – these are the most “traditional” disks; all data is recorded on the surface of one of the disk “platters”.
  - solid state disks – as suggested by their name, these drives have no moving parts, and instead store data in flash memory.
  - hybrid drives – these drives contain both rotating media and solid state media; they attempt to intelligently control which sectors from the rotating media should be copied to the solid-state portion of the device for faster access.

- Integrated controllers on modern disks may also use techniques designed to speed data transfers. Some of these techniques include
  - track caching – copying all of the sectors on a track to local memory on the controller when the first of any sectors is read from a track; subsequent sector reads can then be satisfied quickly from the cached data.
  - rotational sensing – provide techniques to allow the host system to sense the sector currently under the heads. This can permit the device driver to appropriately reorder the queue of disk I/O requests.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>1.44 MB Floppy</th>
<th>6.23 GB Drive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cylinders</td>
<td>80</td>
<td>24,704</td>
</tr>
<tr>
<td>Tracks/cyl</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Sectors/track</td>
<td>18</td>
<td>63</td>
</tr>
<tr>
<td>Sectors/disk</td>
<td>2,880</td>
<td>12,450,816</td>
</tr>
<tr>
<td>Bytes/sector</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Bytes/disk</td>
<td>1,474,560</td>
<td>6,374,817,792</td>
</tr>
</tbody>
</table>
### Example Disk Parameters - 2

<table>
<thead>
<tr>
<th>Parameter</th>
<th>1.44 MB Floppy</th>
<th>6.23 GB Drive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adj. Cyl Seek</td>
<td>3 msec</td>
<td>2 msec</td>
</tr>
<tr>
<td>Avg. Seek</td>
<td>94 msec</td>
<td>9.5 msec</td>
</tr>
<tr>
<td>Rotation Time</td>
<td>200 msec</td>
<td>11 msec</td>
</tr>
<tr>
<td>Transfer Time</td>
<td>11 msec</td>
<td>176 µsec</td>
</tr>
<tr>
<td>Bytes/sector</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Transfer rate</td>
<td>45 KB/sec</td>
<td>2.77 MB/sec</td>
</tr>
</tbody>
</table>
Modern disk drives store data much more “densely” that older drives. 63 sectors per track is the logical limit only because legacy controllers have only 6 bits in the register that specifies the sector number.

The actual number of sectors per track (and number of cylinders and heads) is probably different from the logical values used by software. In fact, the number of number of sectors per track probably varies from track to track!

Thanks to clever controllers, we usually don’t have to worry about these logical versus physical differences.
Modern disks controllers have significant RAM (8 to 128 MB is common) used to buffer requests.

A read request may copy an entire track into RAM so additional requests for other sectors from that track can be processed without another physical read.

Likewise, a write request may save the data in RAM and write it to disk after reporting completion of the write!
Q. Given a collection of requests to read / write a disk drive, in what order should the requests be performed?

A. It depends on the sophistication of the operating system and the needs of the users.

We will consider several scheduling algorithms:

- First Come, First Served (FCFS)
- Shortest Seek Time First (SSTF)
- “Elevator” Algorithms:
  - SCAN
  - C-SCAN
  - LOOK
  - C-LOOK
First Come, First Served (FCFS)

- As implied by the name, this algorithm keeps a queue of pending disk I/O requests.
- New requests are always added to the rear of the queue.
- When the disk becomes available (that is, it completes the processing of a previous request), the request at the head of the queue is selected for processing.
- The main advantage of this algorithm is its simplicity.
- Another often overlooked advantage is that it guarantees there will never be indefinite postponement of an I/O request.
- This is the only appropriate algorithm for single-process systems without concurrent I/O available to a single device (e.g. MS-DOS).
- It may also be appropriate if most of the I/O requests are likely to be in the same area of the disk.
In this algorithm, the next disk I/O request selected for processing is for the cylinder that is closest to the current cylinder.

For example, if the heads are on cylinder 20, and there are pending requests for cylinder 25, 17, and 30, the request for cylinder 17 would be selected since it is closest to cylinder 20.

If there are multiple “closest” requests then tie-breakers are used. For example, the algorithm might choose the request for the closest cylinder that has been waiting the longest.

This algorithm is appropriate for systems with multiple processes and potentially multiple pending disk I/O requests.

There is the potential for indefinite postponement, which could result in poor performance for some requests. The next slide illustrates such a scenario.

In cases of indefinite postponement, aging could possibly be used to minimize the problem.
Assume we have 3 processes named A, B, and C using data on cylinders 1, 5, and 20 respectively. Assume it takes 2 msec to seek between adjacent cylinders, and 1 msec to complete reading once the desired cylinder is reached. Each process spends 1 msec processing the data before starting another read operation. All processes are initially ready, and the disk heads are at cylinder 1.

- **T = 0.** A issues a read request and blocks. No seek is required. B and C each issue a read request and become blocked.
- **T = 1.** A’s data is ready and it starts computation. B’s request is started, and will complete at \( T = 10 \) \((1 + 2*(5-1) + 1)\).
- **T = 2.** A’s computation is done and it issues another read for cylinder 1.
- **T = 10.** B’s data is ready and it starts computation. A’s disk request is for a cylinder closer than C’s request, so A’s request is started, and will complete at \( T = 19 \) \((10 + 2*(5-1) + 1)\).
- **T = 11.** B’s computation is done and it issues another read for cylinder 5.
- ... Poor process C. It has to wait, and wait, and wait.
Elevator Algorithms

- This name for this class of algorithms was chosen because they cause the disk heads to move in a manner similar to that of elevator cars.
- In general, once an elevator begins moving in a particular direction (up or down), it will continue in that direction until it has visited each of the floors for which there is a pending service request in that direction.
- The elevator may reverse direction before reaching the highest (or lowest) floor if there are no further pending requests in that direction.
- Or, to anticipate requests, the elevator cars may always travel between the lowest and highest floors on each trip.
- We will consider four elevator algorithm variants, SCAN, C-SCAN, LOOK and C-LOOK.
The "SCAN" algorithm moves the disk heads in one direction only until all cylinders in that direction have been processed.

It then reverses direction and moves the heads in the other direction until all the cylinders in that direction have been processed.

This algorithm may produce a high variation in response times for requests near the ends of head travel. For example, consider the time between successive “visits” to cylinders 2, 50 and 98 on a disk with cylinders numbered 1 . . . 99.

SCAN always covers the entire range of cylinders on the disk, even though there may be no requests for the lowest and highest cylinder numbers.
The “C-SCAN” algorithm is similar to the SCAN algorithm, with two major conceptual differences:

- It assumes the highest-numbered cylinder is logically adjacent to the lowest-numbered cylinder.
- The heads only move in one direction.

Of course, physical disks aren’t constructed as the first difference suggests. But because of the second difference, the only downward movement of the heads is *directly* from the highest numbered cylinder to the lowest numbered cylinder, and doing a large (that is, long distance) seek usually takes less time than than doing many short seek operations.

This guarantees that if we continually have requests for the same set of cylinders, the variations in time between successive visits to those cylinders will be very small. This is the primary advantage of C-SCAN over SCAN, and is valuable if the processes are being used interactively.
Implementation for SCAN and C-SCAN

- Implementation of the SCAN and C-SCAN algorithms is done by using a set of queues – in effect, there is a queue for each cylinder.
- In the SCAN algorithm there are two sets of queues. The entries on one set are processed as the heads are moving upward – a so-called “upsweep”. The other queues are processed during a “downsweep”.
- When all of the requests in a particular queue have been processed, the heads are moved in the appropriate direction to the cylinder for which the corresponding queue is not empty.
- When a new request is made, it is placed in the set of queues for the current sweep if it is for a cylinder number beyond that to which the heads are positioned. For example, if the heads are on cylinder $K$ and we’re doing an upsweep, then a request for any cylinder number larger than $K$ will be placed in the appropriate upsweep queue. Otherwise it will be placed on the appropriate downsweep queue.
- For the C-SCAN algorithm, no requests are processed while the heads are moving from the highest numbered cylinder down to the lowest numbered cylinder. Thus only one set of queues is required for the C-SCAN algorithm’s implementation.
A Potential Problem with SCAN C-SCAN Implementations

Q. When should new requests for the current cylinder – that is, the cylinder to which the heads are positioned – no longer be placed on the current queue?

A. After the heads reach cylinder $K$, no new requests for cylinder $K$ should be added to the current queue.
   - In the SCAN algorithm such requests should be added to the queue for the sweep in the direction opposite to that of the current sweep.
   - In the C-SCAN algorithm such requests should be held in a temporary queue. Once all the current requests for cylinder $K$ have been processed, the requests in the temporary queue are moved to the normal queue, but not processed until the next upsweep.

If we didn’t have this rule, then we risk the potential of indefinite postponement. For example, consider a “real” elevator on floor $K$ with a continuous stream of people, one at a time, shouting “hold the elevator!” Apart from the physical impossibility of handling all the people (which isn’t problem for disk drives), the elevator would be forever stuck on the same floor!
One simple modification of the SCAN and C-SCAN algorithms is to move the heads only as far as the highest numbered or lowest numbered request. That is, the algorithm “looks ahead” to see if there are any requests requiring further movement in the current direction.

While this appears to be obvious, requests that would arrive from cylinders between the last referenced cylinder and the highest numbered cylinder while the heads are moving to the highest numbered cylinder won’t be processed by the LOOK algorithms.

The C-LOOK algorithm is similar to C-SCAN, but the highest numbered cylinder for which there is a request is considered adjacent to the lowest numbered cylinder for which there is a request.
Idle Disks

- It is hopefully obvious that if all the disk I/O request queues are empty, no seek operations are required.
- It is appropriate on most systems to remove power from the motor that is rotating the platters after a significant period of time with no activity.
- The goal is to conserve power, and is most important when batteries are used as the power source for the computer system.
- The biggest disadvantage of doing this is that disk I/O operations cannot later be resumed until the motor has been turned on again, and the platters have reached the appropriate rotation rate.
Batch Processing of Disk I/O Requests

- We have observed that in some of the algorithms we had to be cautious to avoid actions that could lead to indefinite postponement, or to come up with approaches to dealing with the potential for this problem, like *aging*.

- Another scheduling approach that eliminates the potential of indefinite postponement is called “batching.” Instead of saving a request on a queue when it arrives, add it to an ordered group called a “batch.”

- When all the requests in the current batch have been processed, then start processing the next batch of requests, and begin saving new requests in another batch.

- It’s obvious that each of the requests in the current batch will be processed before any of the requests in the next batch will be considered.
RAID Disks

- RAID = Redundant Array of Inexpensive (or Independent) Disks.
- The idea, first discussed in the late 1980s, is to combine multiple inexpensive disk drives used with personal computers with suitable hardware and software to challenge – or beat – the reliability of the best, much more expensive disks for mainframe computer systems.
- There are many RAID “levels”, each representing a different way of organizing the data. Here are brief descriptions of a few of the RAID levels. There are more.
  - RAID 0 distributes data in “stripes” across multiple drives. This typically increases the opportunity for concurrent operations (at least seek operations) to improve throughput. There’s no reliability improvement, however.
  - RAID 1 duplicates data on a set of “mirrored” disks. Obviously if one drive fails, another drive in the mirrored set still has a good copy of the data.
  - RAID 2 distributes individual bits of data to different drives, and the disk rotation is synchronized so that each sequential bit is on a different drive. Hamming-code parity is computed for groups of bits and stored on still another drive.
Types of I/O Errors and Their Handling

- Programming error – the device driver code is incorrect, either because of a typical type of error, or because the developer did not completely understand the controller and device behaviors.
- Transient error – these errors may disappear if the operation is repeated. More common with earlier disk systems where media is exposed to dust, etc.
- Permanent error – these are the worst errors. For example, if a disk is dropped, the potential exists for heads to “crash” into the surface of the media. Mobile devices are obviously prone to this problem.
- Seek error – hopefully transient. Moving disk heads to a specific cylinder is a mechanical operation, but still requires precision. If the controller detects heads are on the wrong cylinder, it can seek to a mechanically known-good location and seek again.
- Controller error – many device controllers are small computer systems (microcontrollers), and execute code stored in ROM or RAM. If unforeseen situations arise, the controller may need to be rebooted. Code for older controllers was downloaded from the main processor!
RAM Disks (NOT SSD Drives)

- Basically, a RAM disk is just a region of primary memory made to function like a small disk. It does not exhibit persistence (like SSD drives).
- The interface to the user and system is the same as for any other disk; only a different device driver is used.
- RAM disks are used when fast, temporary storage is needed. Example uses are for temporary file systems during booting (used in most Linux systems), or for temporary files for editors.
The most common clock/calendar chip used in PCs was the Motorola MC146818. On modern systems, the RTC is integrated into the southbridge chip.

This device has registers to record the time and the date. It is powered by a lithium battery (or a supercapacitor) when the main power to the system is off.

This device is usually referenced only when system boots to obtain the current date and time.

Afterwards, the time and date is recorded in memory and periodically updated using interrupts from a programmable interval timer.
Programmable Interval Timers

- The original device in a PC was the Intel 8253 (three separate timers); its functionality is now in the southbridge chip.
- One timer register (say A) holds a value that is used to load an auto-decremented register (B).
- The B register is decremented each time a fixed-frequency oscillator goes through one cycle.
- When the B register reaches 0, an interrupt is generated and possibly, B is automatically reloaded from A. This can cause a sequence of periodic interrupts.
- The oscillator frequency can be controlled by software.
Typical Uses for Clocks and Timers

- Maintaining the time of day
- Measuring the time used by a running process, and enforcing the quantum for running processes
- Accounting for processor utilization
- Allowing processes to delay, or limit the time spent in certain functions (e.g. login)
- Providing “watchdog” timers for the system (e.g. detection of missing interrupts)
- Profiling, monitoring, generating statistics
As noted earlier, most systems read the clock/calendar chip when they boot to obtain the current date and time of day.

Thereafter, these items are updated as a result of periodic interrupts from a programmable interval timer.

The time of day can be represented in different ways by different systems:

- A single 64-bit integer could keep track of the total number of clock “ticks” (e.g. one-milliseconds each) that have occurred today.
- An additional integer could keep track of seconds as well, with that value being updated once after every 1000 one-millisecond ticks.
- The system could just keep track of when it was booted, and how long it’s been since it was booted. Windows systems use this approach.
Q. How does the system handle multiple requests for timers, when it may have only one?

A. Most systems keep a queue of scheduled events – that is, times when some action is to be taken.

Keeping an unordered set of such events would not be efficient, since we’d then have to execute a $O(N)$ algorithm each time the clock ticks.

Instead, an queue ordered by event time is maintained. Each queue entry has

- identification of the event that is to occur
- the time remaining until the event is to occur – not the absolute time, but the delta time – that is, how long from now until the event
- identification of the following event (in time)

With this approach, we execute a $O(1)$ algorithm at each clock tick: decrement remaining time; if zero, schedule the event and remove the entry; repeat the test.

Of course, adding a new event to the queue is $O(N)$. 
Q. Suppose the system time is incorrect. For example, suppose it’s really 12:01 PM, but the system says it’s 11:59 AM. How do you change it?

If you just change the software clock directly to 12:01 PM, what will happen to an event that’s scheduled for noon?

A. Cleverly, UNIX (and perhaps other systems) have the capability of just speeding up the clock ticks until the time is correct.

Thus the system time doesn’t “jump” directly from one time to another, but passes through every time between the two times.

As a result, all scheduled events are still processed.

The purpose of considering this is to emphasize how careful system implementors must be with the various system resources.
A “terminal” has two major required components:
- a display, and
- an input device, usually a keyboard

There may also be other related components, each treated as separate devices:
- a mouse
- a pen
- a fingerprint reader
There are three major terminal categories to consider:

- Memory-mapped terminals
  - Character only terminals
  - Graphic terminals
- RS-232 terminals
  - Glass Teletype®
  - Graphic terminals
- X terminals

Although some of these are now infrequently used, it is useful to understand their behavior, and the complexities associated with their device drivers, since modern terminals have similar characteristics.
Memory-mapped Terminals

- Video RAM, which maintains information related to the image that should appear on the display, is effectively between the system bus (with the processor and primary memory) and the video controller (which is often called the video adapter).

- Actually, the video RAM is usually considered to be part of the video adapter, but conceptually it’s between the system bus and the video controller.

- Once configured (at system boot time), the video controller is often never accessed again (or, at most, it’s accessed infrequently to do things like change the video brightness, or turn the display off entirely).

- The video controller continually looks at the contents of video RAM and repeatedly sends the appropriate signals to the display.
Video RAM can contain either a bit-by-bit mapping of the image that is to appear on the display, or it can contain character codes and other information to specify blinking and color for text.

Each “picture element” is called a “pixel.” The display size is usually specified as the number of pixels in one row of the display and the number of rows (e.g. 2560 by 1440).

If video RAM is used to hold character codes, then the video controller is placed in a mode that maps from these codes to the appropriate pixels for the character as it continually updates the display.

If video RAM contains a graphic image, then the individual pixels are mapped directly to the display.
Keyboards

- Keyboards are actually completely separate devices from displays.
- On a typical keyboard, there are two planes of wires separated by a small distance. One plane has horizontal wires, and the other has vertical wires. Keys are positioned so when they are pressed, they cause two wires in separate planes to connect.
- The keyboard’s controller identifies the key code corresponding to the key that was pressed, and reports it to the device driver along with an interrupt request.
- The same behavior happens when a previously pressed key is released.
- Note that the keyboard just reports a key code, and not a character code. The mapping from key code to character code is usually the responsibility of the device driver. This is because keyboards for various nationalities will often have significant differences in what’s printed on the key tops.
- Since there are separate key up and key down actions, the driver is capable of differentiating between the a key by itself, or the a key and a shift key (like SHIFT, ALT, and/or CONTROL).
“RS-232” refers to a standard for serial communication.

Early integrated terminals, with a display and a keyboard, were often connected to a computer system’s serial port using RS-232.

These terminals had a small microprocessor (or in very early systems, discrete logic), a serial port, a keyboard, and a display.

In operation, data was sent from the remote computer system to the terminal over the RS-232 channel, and the terminal took the appropriate action (usually to display a character on the display). When a key was pressed on the terminal’s keyboard, the corresponding character code was communicated to the remote computer system over the RS-232 channel.

RS-232 terminals are now obsolete; inexpensive microprocessor systems with displays and keyboards have replaced them.
One “holdover” from the RS-232 terminal days is the concept of “escape sequences.”

The very earliest terminals emulated printing devices, like teletypewriters. These devices could do little more than print a character, and occasionally do a line feed and carriage return to get to the next output line. Terminals with displays that behaved like this were often called “Glass TTYs”.

More modern terminals with displays could perform many more actions, like erasing displayed characters, change color, blink characters, ul, clear or scroll the entire display, and so forth. These actions were typically request by using what’s often called an “escape sequence,” primarily because the first character in each such character sequence was the ASCII escape character (hexadecimal 1B).
Without going into too much detail, it is important to understand something about RS-232 communication.

In general, RS-232 must send a start bit, 7 or 8 data bits, an optional parity bit, and a stop bit for each character. Thus between 9 and 11 bits must be sent for each character.

The data rates associated with RS-232 communication are typically between 9600 and 115,200 bits per second.

The higher RS-232 data rates are also typically associated with smaller distances.

Compare those rates with typical local area network speeds of 10, 100, or 1000 megabits per second.
X Terminals

The X Window System is essentially a protocol that governs the exchanges between a client (that wants to display something) and a server (that manages the display, the keyboard, and the mouse or other pointing device).

An X terminal is/was just a dedicated memory-mapped display device and a microprocessor system running an X server. These are essentially obsolete. But the X protocol is not obsolete.

However, there are numerous similar network devices in common use. For example, most tablet devices, with much more powerful processors than X terminals, can function as a networked terminal.

And open-source X server software is available for modern microcomputers, enabling them to function as X terminals.
Terminal Modes

- Because of their significance – humans use them to communicate with the computer – terminals are equipped with multiple software “modes” in which they operate. These are complex and arcane, so we’ll just consider an overview.

- Using UNIX and POSIX terms, there are two major terminal modes:
  - cooked or canonical mode
    - Only complete lines, usually ending in an end of line character, result in input being made available to a process. Even if you type 100 characters, until you press the end of line key, none of them will be read by a process.
    - Editing (e.g. backspace and control-U) and special characters that generate a UNIX signal (e.g. control-C) are handled by the input software.
    - Input characters are appropriately echoed (like tabs, backspace, and ends of lines).
  - raw or non-canonical mode
    - Every input character, regardless of what it is, is made available for input by a process. This includes things like backspace, control-C, and any other character code that an input device can generate.
    - None of the editing characters are processed.
    - By default, none of the input characters are echoed (redisplayed).
Task: Collect the input. Usually each keypress causes at least one interrupt, regardless of the terminal type. As noted earlier, PC keyboards generate one interrupt for a keypress, and another for key release.

Task: Map key codes to character codes. Recall that key codes identify the physical key pressed on a keyboard, not the character code to which it corresponds. In most systems, this is configurable. And there isn’t just one set of character codes. ASCII, EBCDIC, Unicode, and numerous others can be used.
Task: process the collected characters.

- Deal with different keyboard arrangements and character sets.
- In cooked mode, process editing characters (the `ioctl` system call is used to modify the editing characters).
- Buffer characters not yet requested by a process associated with the terminal. There are two main approaches:
  - Use a common pool of small-sized buffers for all terminals (UNIX calls these "clists")
  - Use a statically allocated buffer for each terminal (has potential for overflow)
- Task: Echo input characters (in cooked mode)
  - Where should each character be displayed?
  - When should each character be displayed?
  - How should tab characters be displayed?

- Task: Handle end-of-line characters (in cooked mode)

- Task: Handle special characters (in cooked mode)
  - Which characters are “special?”
  - How should changes in the special character set be made? (POSIX tcsetattr, tcgetattr functions, UNIX ioctl)
The output side of a terminal handler is usually much simpler than the input size.

There are two main tasks to be performed:

- Get characters from the user (which may be presented in a queue, since they may be generated faster than they can be displayed), and display them.
- Process any special output editing sequences.

Although an RS-232 terminal or a terminal emulator (like “putty”) will deal with escape sequences, when these are sent to a local display device they must be processed locally.

And the remote processing of escape sequences (for RS-232 terminals or terminal emulators) must still be done, so knowledge of the escape sequences and their interpretation is still required.
Task: Display Characters

- Identify the correct location for the next output character
- Handle end-of-line and special characters (e.g. bell and tab)
- Handle scrolling (or not scrolling) at the end of a line
- Handle cursor positioning
- Handle special display attributes (e.g. blinking, color)
Task: Process escape sequences for a local display

- ANSI has defined a standard for escape sequences, but the standard is not always followed.
- In particular, there are numerous escape sequences defined in the heyday of RS-232 terminals for different terminal types.
- Most systems have the capability of emulating one or more terminal types, but the DEC VT-100 is the most popular. A subset of the escape sequences for this terminal can be found in the file vt100.txt in the class directory.
In the next module we’ll begin our study of memory management.

We will study a sequence of memory management techniques, relating them to the various problems related to the use of memory.

We’ll also examine a simple model that shows the effectiveness of multiprogramming.