CTBTCSE – SIV P4 – Operating System

“Unit-4 Storage Management”

Dr. Parag Shukla
Assistant Professor,
School of Cyber Security and Digital Forensics
National Forensic Sciences University

STORAGE MANAGEMENT

Presentation Outline

- **Mass Storage Structure**
- **Disk Structure, Disk Management**
- **Swap Space Management,**
- **RAID Structure**
- **I/O System Overview**

Mass Storage Structure

Overview of Mass Storage Structure

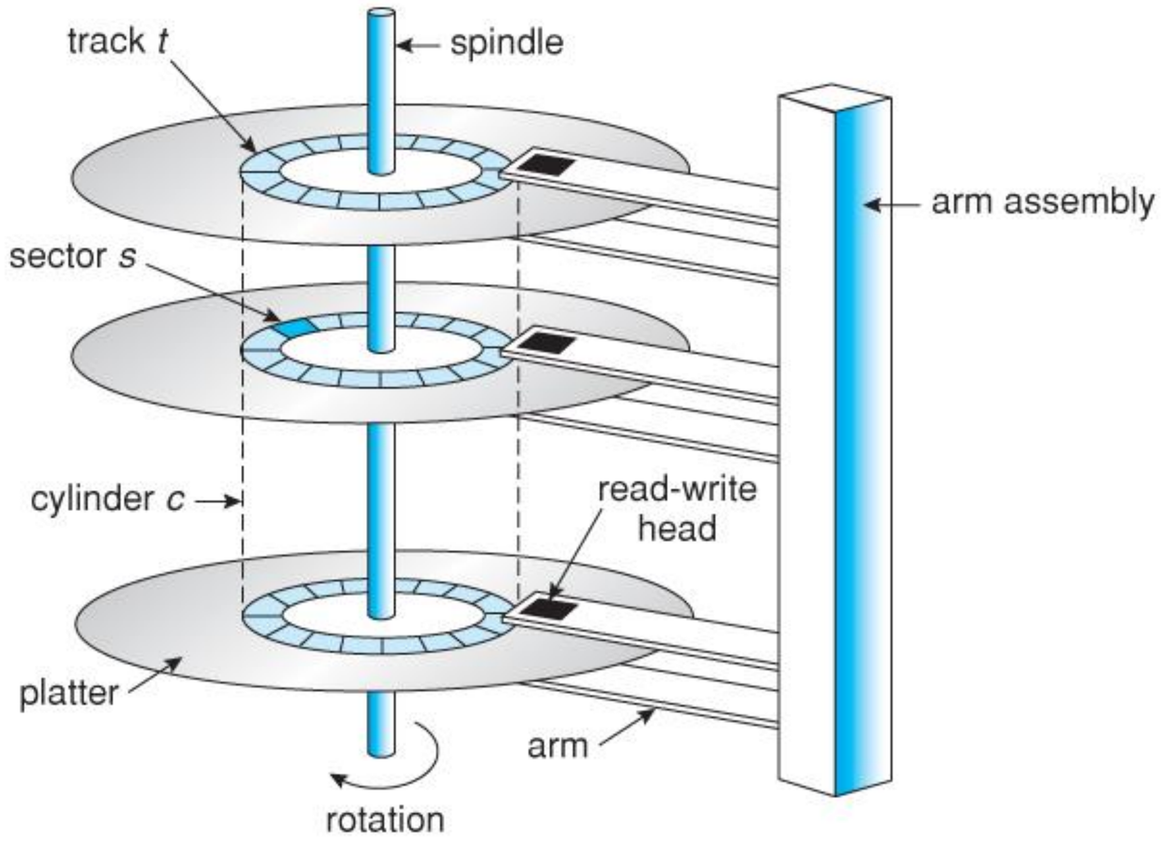
1. Magnetic Disks
2. Solid State Disks
3. Magnetic Tapes

Magnetic Disks

- Traditional magnetic disks have the following basic structure:
  - One or more ***platters*** in the form of disks covered with magnetic media. ***Hard disk*** platters are made of rigid metal, while "***floppy***" disks are made of more flexible plastic.
  - Each platter has two working ***surfaces.*** Older hard disk drives would sometimes not use the very top or bottom surface of a stack of platters, as these surfaces were more susceptible to potential damage.
  - Each working surface is divided into a number of concentric rings called ***tracks.*** The collection of all tracks that are the same distance from the edge of the platter, (i.e. all tracks immediately above one another in the following diagram) is called a ***cylinder***.
  - Each track is further divided into ***sectors,*** traditionally containing 512 bytes of data each, although some modern disks occasionally use larger sector sizes. (Sectors also include a header and a trailer, including checksum information among other things.
  - Larger sector sizes reduce the fraction of the disk consumed by headers and trailers, but increase internal fragmentation and the amount of disk that must be marked bad in the case of errors.)

Mass Storage Structure – Magnetic Disk

- The data on a hard drive is read by read-write ***heads.*** The standard configuration (shown below) uses one head per surface, each on a separate ***arm***, and controlled by a common ***arm assembly*** which moves all heads simultaneously from one cylinder to another. (Other configurations, including independent read-write heads, may speed up disk access, but involve serious technical difficulties.)
- The storage capacity of a traditional disk drive is equal to the number of heads (i.e. the number of working surfaces), times the number of tracks per surface, times the number of sectors per track, times the number of bytes per sector. A particular physical block of data is specified by providing the head-sector-cylinder number at which it is located.



Moving-head disk mechanism

Mass Storage Structure

- **Solid-State Disks - New**
- As technologies improve and economics change, old technologies are often used in different ways. One example of this is the increasing used of ***solid state disks, or SSDs.***
- SSDs use memory technology as a small fast hard disk. Specific implementations may use either flash memory or DRAM chips protected by a battery to sustain the information through power cycles.
- Because SSDs have no moving parts they are much faster than traditional hard drives, and certain problems such as the scheduling of disk accesses simply do not apply.
- However SSDs also have their weaknesses: They are more expensive than hard drives, generally not as large, and may have shorter life spans.
- SSDs are especially useful as a high-speed cache of hard-disk information that must be accessed quickly. One example is to store filesystem meta-data, e.g. directory and inode information, that must be accessed quickly and often. Another variation is a boot disk containing the OS and some application executables, but no vital user data. SSDs are also used in laptops to make them smaller, faster, and lighter.
- Because SSDs are so much faster than traditional hard disks, the throughput of the bus can become a limiting factor, causing some SSDs to be connected directly to the system PCI bus for example.

Mass Storage Structure

Magnetic Tapes

- Magnetic tapes were once used for common secondary storage before the days of hard disk drives, but today are used primarily for backups.
- Accessing a particular spot on a magnetic tape can be slow, but once reading or writing commences, access speeds are comparable to disk drives.
- Capacities of tape drives can range from 20 to 200 GB, and compression can double that capacity.

Disk Structure

Disk Structure

- The traditional head-sector-cylinder, HSC numbers are mapped to linear block addresses by numbering the first sector on the first head on the outermost track as sector 0.
- Numbering proceeds with the rest of the sectors on that same track, and then the rest of the tracks on the same cylinder before proceeding through the rest of the cylinders to the center of the disk. In modern practice these linear block addresses are used in place of the HSC numbers for a variety of reasons:
  - The linear length of tracks near the outer edge of the disk is much longer than for those tracks located near the center, and therefore it is possible to squeeze many more sectors onto outer tracks than onto inner ones.
  - All disks have some bad sectors, and therefore disks maintain a few spare sectors that can be used in place of the bad ones. The mapping of spare sectors to bad sectors in managed internally to the disk controller.
  - Modern hard drives can have thousands of cylinders, and hundreds of sectors per track on their outermost tracks. These numbers exceed the range of HSC numbers for many (older) operating systems, and therefore disks can be configured for any convenient combination of HSC values that falls within the total number of sectors physically on the drive.

Disk Structure

- There is a limit to how closely packed individual bits can be placed on a physical media, but that limit is growing increasingly more packed as technological advances are made.
- Modern disks pack many more sectors into outer cylinders than inner ones, using one of two approaches:
  - With ***Constant Linear Velocity, CLV,*** the density of bits is uniform from cylinder to cylinder. Because there are more sectors in outer cylinders, the disk spins slower when reading those cylinders, causing the rate of bits passing under the read-write head to remain constant. This is the approach used by modern CDs and DVDs.
  - With ***Constant Angular Velocity, CAV,*** the disk rotates at a constant angular speed, with the bit density decreasing on outer cylinders. (These disks would have a constant number of sectors per track on all cylinders.)

Disk Management

- **Disk Formatting**
- Before a disk can be used, it has to be ***low-level formatted***, which means laying down all of the headers and trailers marking the beginning and ends of each sector. Included in the header and trailer are the linear sector numbers, and ***error-correcting codes, ECC,*** which allow damaged sectors to not only be detected, but in many cases for the damaged data to be recovered (depending on the extent of the damage.) Sector sizes are traditionally 512 bytes, but may be larger, particularly in larger drives.
- ECC calculation is performed with every disk read or write, and if damage is detected but the data is recoverable, then a ***soft error*** has occurred. Soft errors are generally handled by the on-board disk controller, and never seen by the OS. (See below.)
- Once the disk is low-level formatted, the next step is to partition the drive into one or more separate partitions. This step must be completed even if the disk is to be used as a single large partition, so that the partition table can be written to the beginning of the disk.
- After partitioning, then the filesystems must be ***logically formatted,*** which involves laying down the master directory information (FAT table or inode structure), initializing free lists, and creating at least the root directory of the filesystem. (Disk partitions which are to be used as raw devices are not logically formatted. This saves the overhead and disk space of the filesystem structure, but requires that the application program manage its own disk storage requirements.)

Disk Management

Boot Block

- Computer ROM contains a ***bootstrap*** program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller, load that sector into memory, and transfer control over to it. (The ROM bootstrap program may look in floppy and/or CD drives before accessing the hard drive, and is smart enough to recognize whether it has found valid boot code or not.)
- The first sector on the hard drive is known as the ***Master Boot Record, MBR,*** and contains a very small amount of code in addition to the ***partition table.*** The partition table documents how the disk is partitioned into logical disks, and indicates specifically which partition is the ***active*** or ***boot*** partition.
- The boot program then looks to the active partition to find an operating system, possibly loading up a slightly larger / more advanced boot program along the way.
- In a ***dual-boot*** (or larger multi-boot) system, the user may be given a choice of which operating system to boot, with a default action to be taken in the event of no response within some time frame.
- Once the kernel is found by the boot program, it is loaded into memory and then control is transferred over to the OS. The kernel will normally continue the boot process by initializing all important kernel data structures, launching important system services (e.g. network daemons, sched, init, etc.), and finally providing one or more login prompts.
- Boot options at this stage may include ***single-user*** a.k.a. ***maintenance*** or ***safe*** modes, in which very few system services are started - These modes are designed for system administrators to repair problems or otherwise maintain the system.

Bad Blocks

Bad Blocks

- No disk can be manufactured to 100% perfection, and all physical objects wear out over time. For these reasons all disks are shipped with a few bad blocks, and additional blocks can be expected to go bad slowly over time. If a large number of blocks go bad then the entire disk will need to be replaced, but a few here and there can be handled through other means.
- In the old days, bad blocks had to be checked for manually. Formatting of the disk or running certain disk-analysis tools would identify bad blocks, and attempt to read the data off of them one last time through repeated tries. Then the bad blocks would be mapped out and taken out of future service. Sometimes the data could be recovered, and sometimes it was lost forever. (Disk analysis tools could be either destructive or non-destructive.)
- Modern disk controllers make much better use of the error-correcting codes, so that bad blocks can be detected earlier and the data usually recovered. (Recall that blocks are tested with every write as well as with every read, so often errors can be detected before the write operation is complete, and the data simply written to a different sector instead.)

Bad Blocks

- Note that re-mapping of sectors from their normal linear progression can throw off the disk scheduling optimization of the OS, especially if the replacement sector is physically far away from the sector it is replacing. For this reason most disks normally keep a few spare sectors on each cylinder, as well as at least one spare cylinder. Whenever possible a bad sector will be mapped to another sector on the same cylinder, or at least a cylinder as close as possible. ***Sector slipping*** may also be performed, in which all sectors between the bad sector and the replacement sector are moved down by one, so that the linear progression of sector numbers can be maintained.
- If the data on a bad block cannot be recovered, then a ***hard error*** has occurred., which requires replacing the file(s) from backups, or rebuilding them from scratch.

Swap Space Management

- Modern systems typically swap out pages as needed, rather than swapping out entire processes. Hence the swapping system is part of the virtual memory management system.
- Managing swap space is obviously an important task for modern OSes.

Swap-Space Use

- The amount of swap space needed by an OS varies greatly according to how it is used. Some systems require an amount equal to physical RAM; some want a multiple of that; some want an amount equal to the amount by which virtual memory exceeds physical RAM, and some systems use little or none at all!
- Some systems support multiple swap spaces on separate disks in order to speed up the virtual memory system.

- **Swap-Space Location**
- Swap space can be physically located in one of two locations:

- As a large file which is part of the regular filesystem. This is easy to implement, but inefficient. Not only must the swap space be accessed through the directory system, the file is also subject to fragmentation issues. Caching the block location helps in finding the physical blocks, but that is not a complete fix.
- As a raw partition, possibly on a separate or little-used disk. This allows the OS more control over swap space management, which is usually faster and more efficient. Fragmentation of swap space is generally not a big issue, as the space is re-initialized every time the system is rebooted. The downside of keeping swap space on a raw partition is that it can only be grown by repartitioning the hard drive.

Swap Space Management

- **Swap-Space Management: An Example**
- Historically OSes swapped out entire processes as needed. Modern systems swap out only individual pages, and only as needed.
- (For example process code blocks and other blocks that have not been changed since they were originally loaded are normally just freed from the virtual memory system rather than copying them to swap space, because it is faster to go find them again in the filesystem and read them back in from there than to write them out to swap space and then read them back.)
- In the mapping system shown below for Linux systems, a map of swap space is kept in memory, where each entry corresponds to a 4K block in the swap space. Zeros indicate free slots and non-zeros refer to how many processes have a mapping to that particular block (>1 for shared pages only.)

RAID Structure

- The general idea behind RAID is to employ a group of hard drives together with some form of duplication, either to increase reliability or to speed up operations, (or sometimes both.)
- ***RAID*** originally stood for ***Redundant Array of Inexpensive Disks,*** and was designed to use a bunch of cheap small disks in place of one or two larger more expensive ones. Today RAID systems employ large possibly expensive disks as their components, switching the definition to ***Independent*** disks.

- **Improvement of Reliability via Redundancy**
- The more disks a system has, the greater the likelihood that one of them will go bad at any given time. Hence increasing disks on a system actually ***decreases*** the **Mean Time To Failure, MTTF** of the system.

- **Improvement in Performance via Parallelism**
- There is also a performance benefit to mirroring, particularly with respect to reads. Since every block of data is duplicated on multiple disks, read operations can be satisfied from any available copy, and multiple disks can be reading different data blocks simultaneously in parallel. (Writes could possibly be sped up as well through careful scheduling algorithms, but it would be complicated in practice.)

RAID Levels

- Mirroring provides reliability but is expensive; Striping improves performance, but does not improve reliability. Accordingly there are a number of different schemes that combine the principals of mirroring and striping in different ways, in order to balance reliability versus performance versus cost.
- These are described by different ***RAID levels***, as follows: (In the diagram that follows, "C" indicates a copy, and "P" indicates parity, i.e. checksum bits.)
- ***Raid Level 0 -*** This level includes striping only, with no mirroring.
- ***Raid Level 1 -*** This level includes mirroring only, no striping.
- ***Raid Level 2 -*** This level stores error-correcting codes on additional disks, allowing for any damaged data to be reconstructed by subtraction from the remaining undamaged data. Note that this scheme requires only three extra disks to protect 4 disks worth of data, as opposed to full mirroring. (The number of disks required is a function of the error-correcting algorithms, and the means by which the particular bad bit(s) is(are) identified.)
- ***Raid Level 3 -*** This level is similar to level 2, except that it takes advantage of the fact that each disk is still doing its own error-detection, so that when an error occurs, there is no question about which disk in the array has the bad data. As a result a single parity bit is all that is needed to recover the lost data from an array of disks. Level 3 also includes striping, which improves performance. The downside with the parity approach is that every disk must take part in every disk access, and the parity bits must be constantly calculated and checked, reducing performance. Hardware-level parity calculations and NVRAM cache can help with both of those issues. In practice level 3 is greatly preferred over level 2.

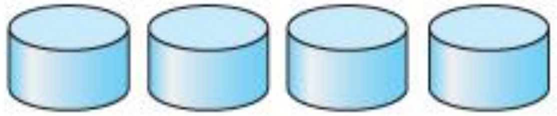
RAID Levels

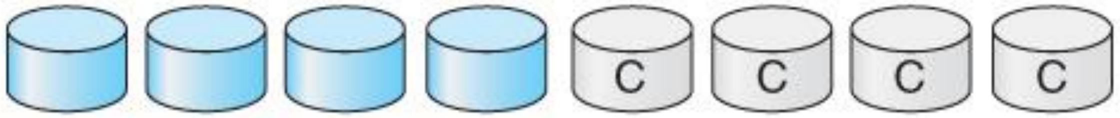
Raid Level 4 - This level is similar to level 3, employing block-level striping instead of bit-level striping. The benefits are that multiple blocks can be read independently, and changes to a block only require writing two blocks (data and parity) rather than involving all disks. Note that new disks can be added seamlessly to the system provided they are initialized to all zeros, as this does not affect the parity results.

Raid Level 5 - This level is similar to level 4, except the parity blocks are distributed over all disks, thereby more evenly balancing the load on the system. For any given block on the disk(s), one of the disks will hold the parity information for that block and the other N-1 disks will hold the data. Note that the same disk cannot hold both data and parity for the same block, as both would be lost in the event of a disk crash.

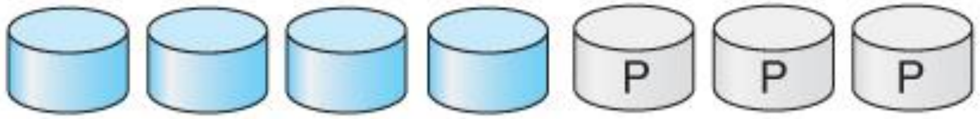
Raid Level 6 - This level extends raid level 5 by storing multiple bits of error-recovery codes, for each bit position of data, rather than a single parity bit. In the example shown below 2 bits of ECC are stored for every 4 bits of data, allowing data recovery in the face of up to two simultaneous disk failures. Note that this still involves only 50% increase in storage needs, as opposed to 100% for simple mirroring which could only tolerate a single disk failure.

RAID Levels

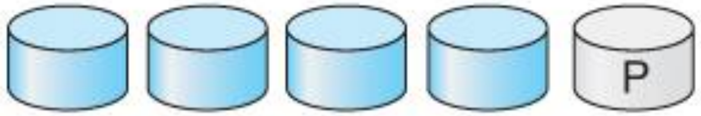

(a) RAID 0: non-redundant striping.



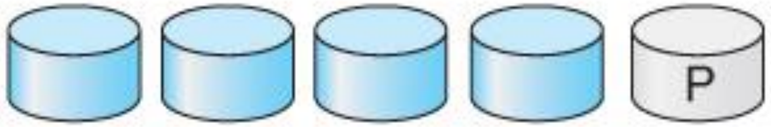
(b) RAID 1: mirrored disks.



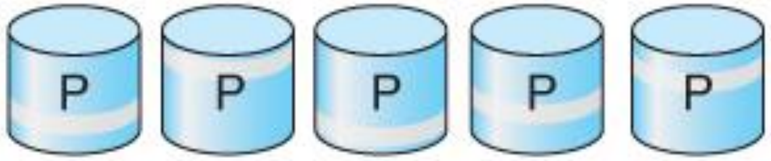
(c) RAID 2: memory-style error-correcting codes.



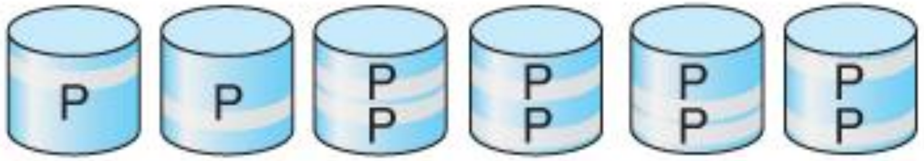
(d) RAID 3: bit-interleaved parity.



(e) RAID 4: block-interleaved parity.



(f) RAID 5: block-interleaved distributed parity.



(g) RAID 6: P + Q redundancy.

I/O Systems

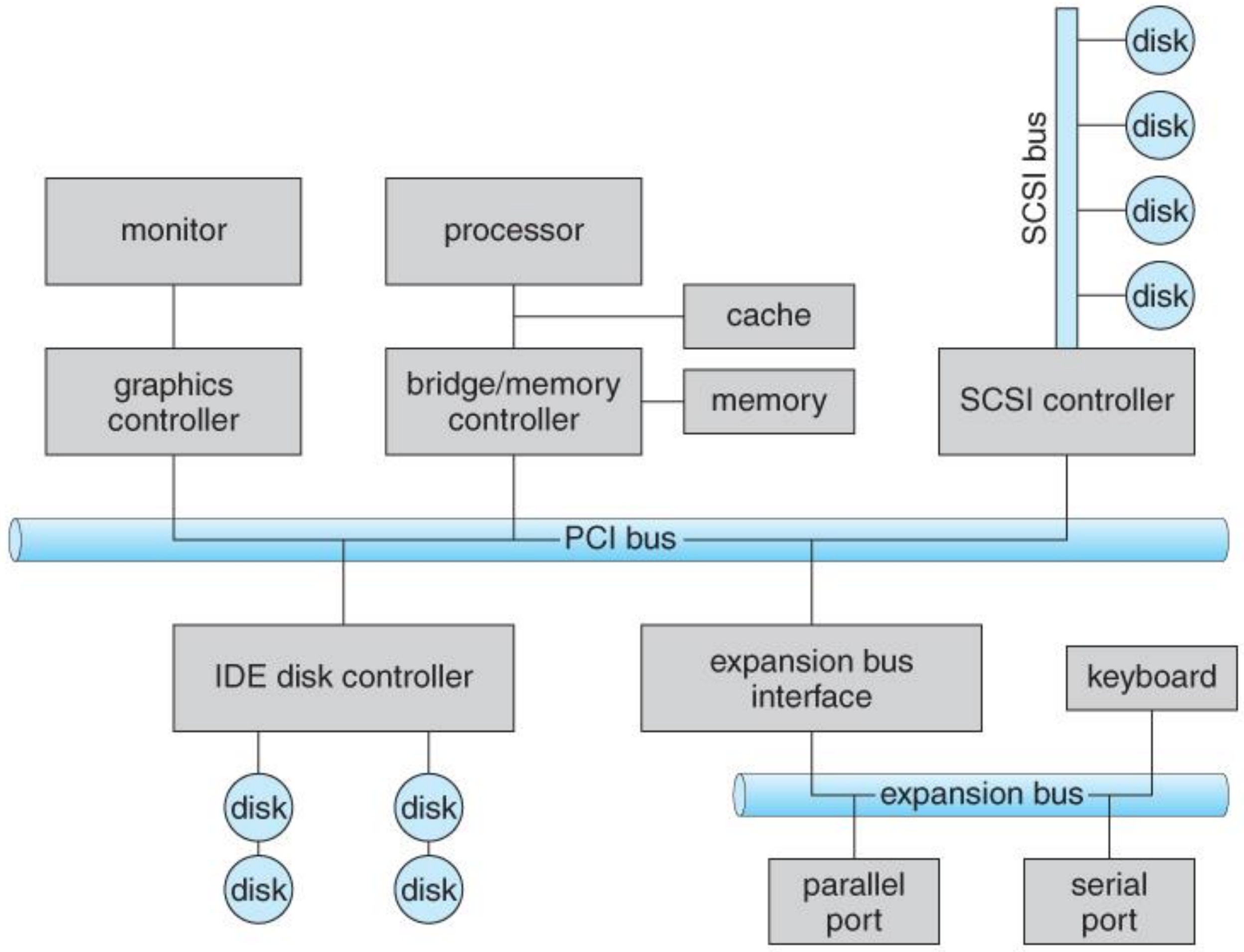
Overview

- Management of I/O devices is a very important part of the operating system - so important and so varied that entire I/O subsystems are devoted to its operation.
- (Consider the range of devices on a modern computer, from mice, keyboards, disk drives, display adapters, USB devices, network connections, audio I/O, printers, special devices for the handicapped, and many special-purpose peripherals.)
- I/O Subsystems must contend with two (conflicting?) trends: (1) The gravitation towards standard interfaces for a wide range of devices, making it easier to add newly developed devices to existing systems, and (2) the development of entirely new types of devices, for which the existing standard interfaces are not always easy to apply.
- ***Device drivers*** are modules that can be plugged into an OS to handle a particular device or category of similar devices.

I/O Hardware

- I/O devices can be roughly categorized as storage, communications, user-interface, and other
- Devices communicate with the computer via signals sent over wires or through the air.
- Devices connect with the computer via ***ports***, e.g. a serial or parallel port.
- A common set of wires connecting multiple devices is termed a ***bus.***
  - Buses include rigid protocols for the types of messages that can be sent across the bus and the procedures for resolving contention issues.
  - Figure below illustrates three of the four bus types commonly found in a modern PC:
    - The ***PCI bus*** connects high-speed high-bandwidth devices to the memory subsystem (and the CPU.)
    - The ***expansion bus*** connects slower low-bandwidth devices, which typically deliver data one character at a time (with buffering.)
    - The ***SCSI bus*** connects a number of SCSI devices to a common SCSI controller.
    - A ***daisy-chain bus,*** (not shown) is when a string of devices is connected to each other like beads on a chain, and only one of the devices is directly connected to the host.

I/O Hardware

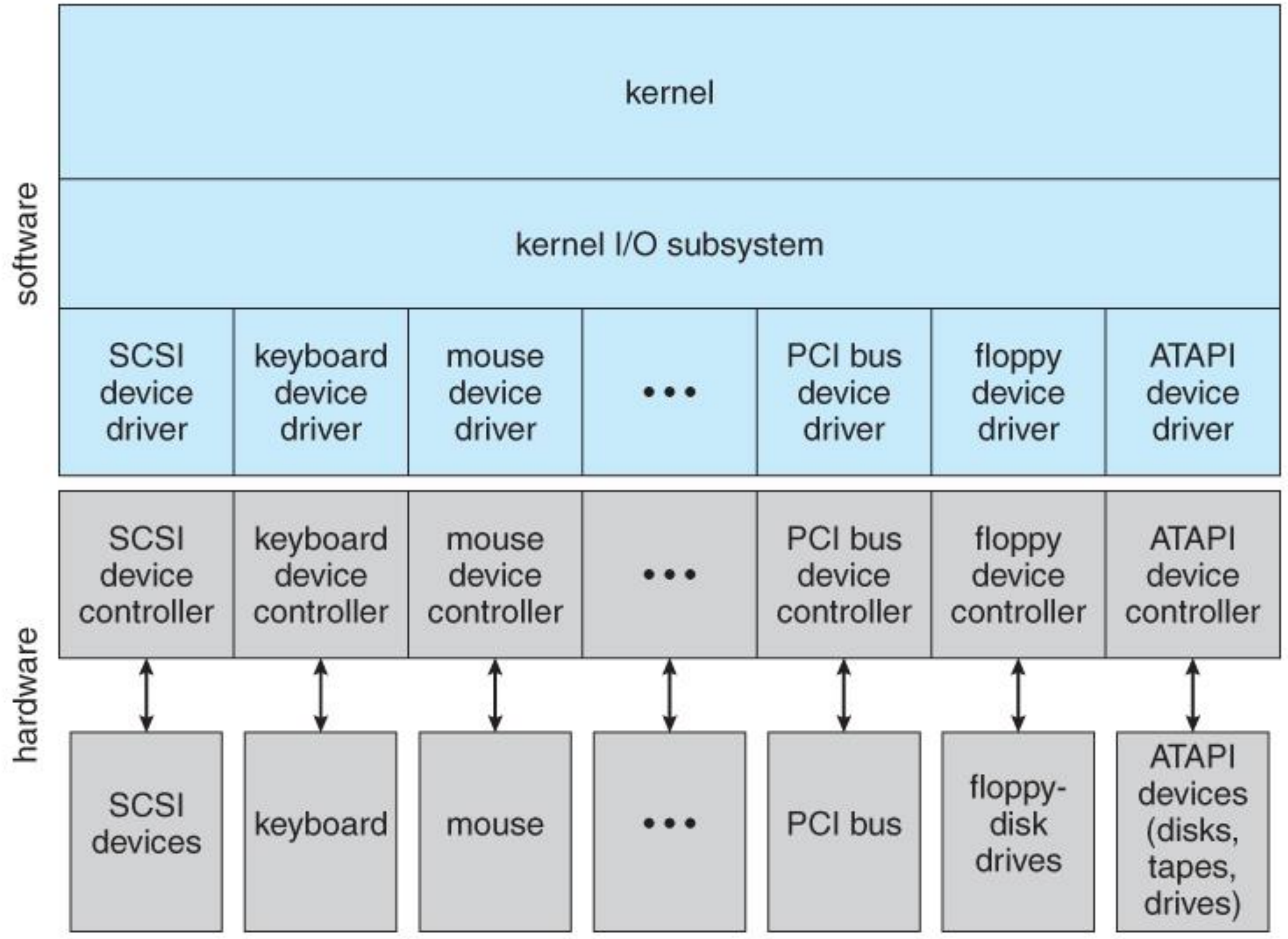


A typical PC bus structure

- One way of communicating with devices is through ***registers*** associated with each port.
- Registers may be one to four bytes in size, and may typically include (a subset of) the following four:
  - The ***data-in register*** is read by the host to get input from the device.
  - The ***data-out register*** is written by the host to send output.
  - The ***status register*** has bits read by the host to ascertain the status of the device, such as idle, ready for input, busy, error, transaction complete, etc.
  - The ***control register*** has bits written by the host to issue commands or to change settings of the device such as parity checking, word length, or full- versus half-duplex operation.

Application I/O Interface

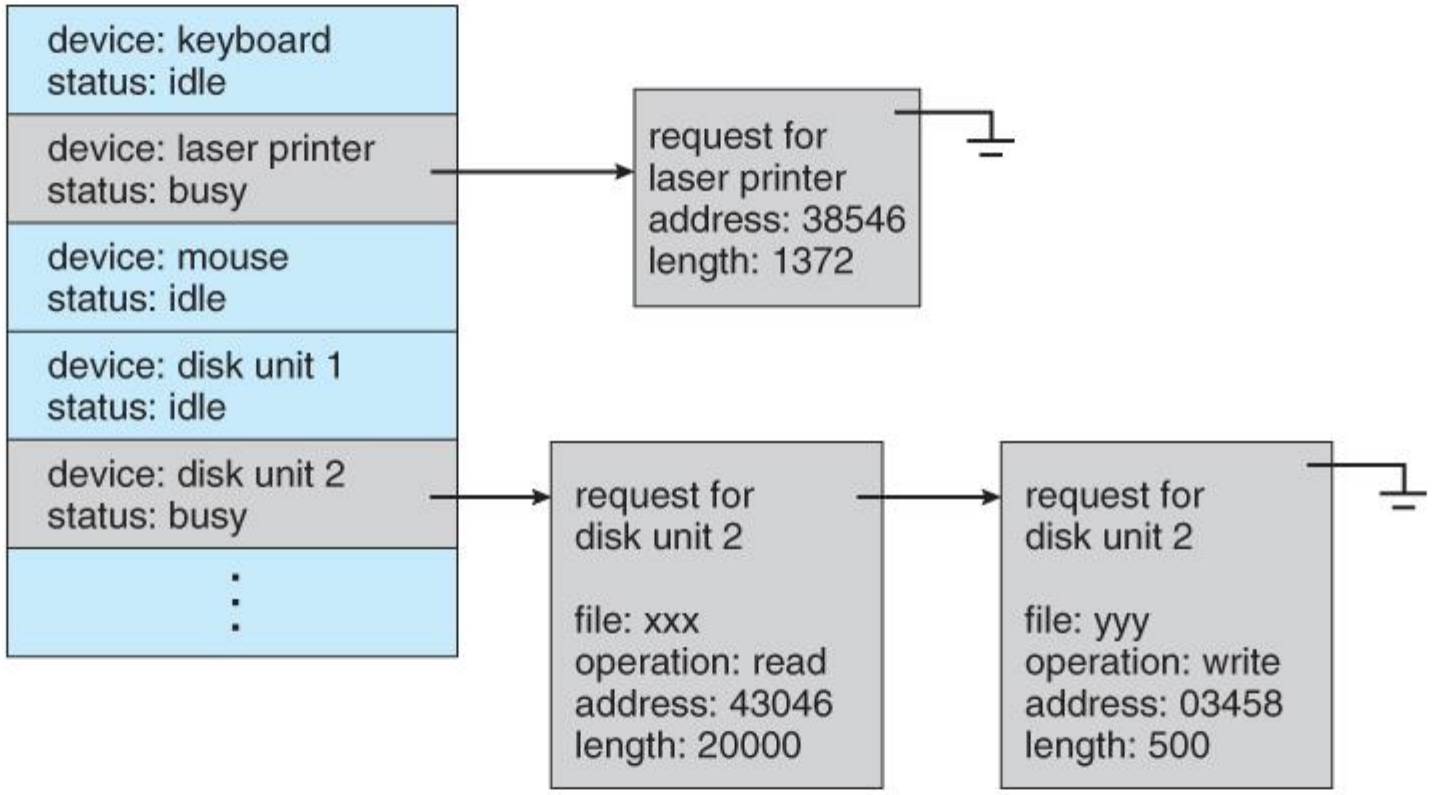
- User application access to a wide variety of different devices is accomplished through layering, and through encapsulating all of the device-specific code into ***device drivers***, while application layers are presented with a common interface for all (or at least large general categories of) devices.



A kernel I/O structure

Kernel I/O Subsystem

- **I/O Scheduling**
  - Scheduling I/O requests can greatly improve overall efficiency. Priorities can also play a part in request scheduling.
  - Buffering and caching can also help, and can allow for more flexible scheduling options.
  - On systems with many devices, separate request queues are often kept for each device
- Buffering
- Caching
- Error Handling
- I/O Protection

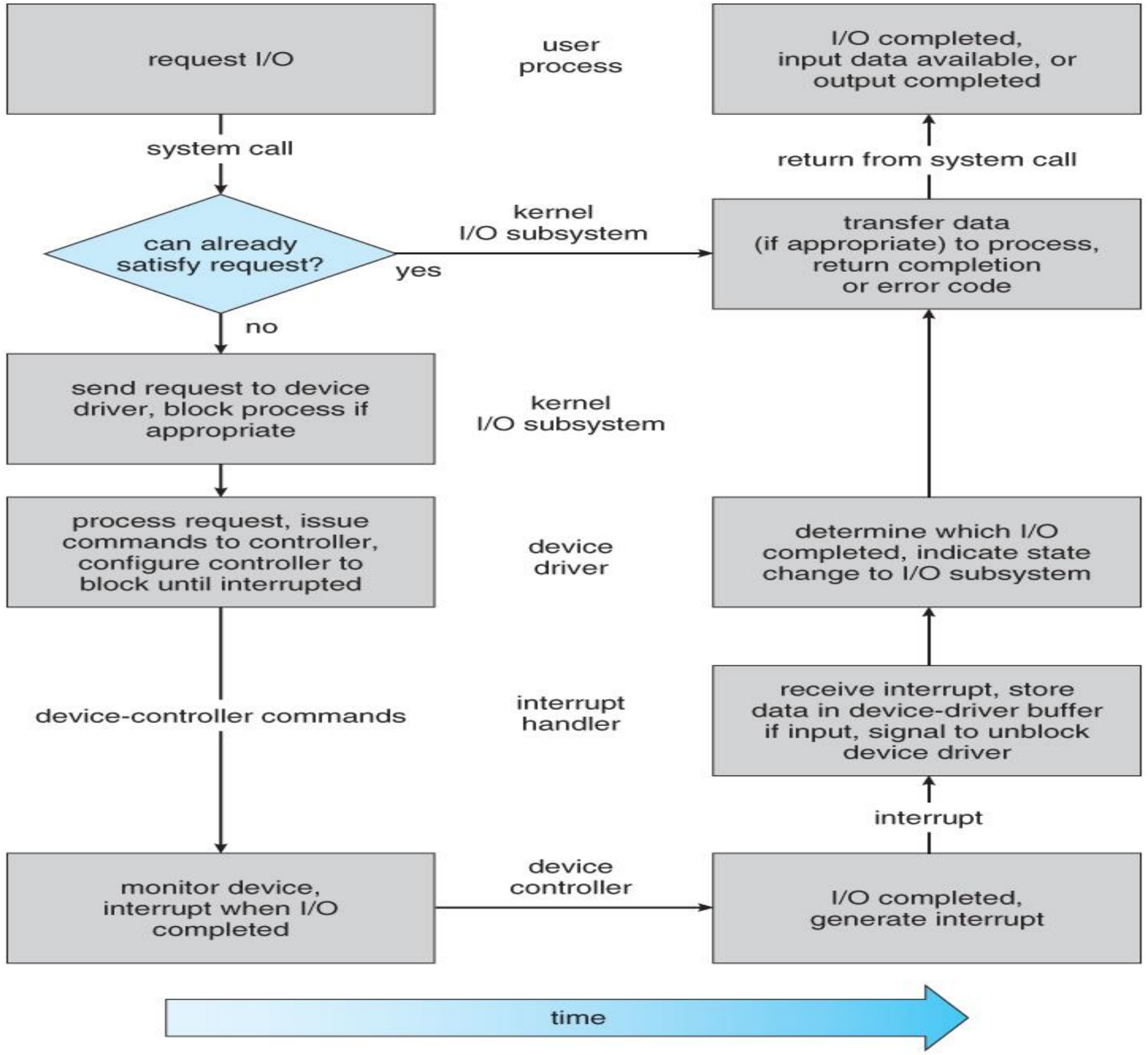

device: keyboard
status: idle
device: laser printer
status: busy
device: mouse
status: idle
device: disk unit 1
status: idle
device: disk unit 2
status: busy
request for laser printer address: 38546 length: 1372
request for disk unit 2 file: xxx operation: read address: 43046 length: 20000
request for disk unit 2 file: yyy operation: write address: 03458 length: 500

Device-status table

Transforming I/O Requests to Hardware Operations

- Users request data using file names, which must ultimately be mapped to specific blocks of data from a specific device managed by a specific device driver.
- DOS uses the colon separator to specify a particular device (e.g. C:, LPT:, etc.)
- UNIX uses a ***mount table*** to map filename prefixes (e.g. /usr) to specific mounted devices. Where multiple entries in the mount table match different prefixes of the filename the one that matches the longest prefix is chosen. (e.g. /usr/home instead of /usr where both exist in the mount table and both match the desired file.)
- UNIX uses special ***device files,*** usually located in /dev, to represent and access physical devices directly.
  - Each device file has a major and minor number associated with it, stored and displayed where the file size would normally go.
  - The major number is an index into a table of device drivers, and indicates which device driver handles this device. (E.g. the disk drive handler.)
  - The minor number is a parameter passed to the device driver, and indicates which specific device is to be accessed, out of the many which may be handled by a particular device driver. (e.g. a particular disk drive or partition.)
- A series of lookup tables and mappings makes the access of different devices flexible, and somewhat transparent to users.
- Following Figure illustrates the steps taken to process a (blocking) read request:

Transforming I/O Requests to Hardware Operations

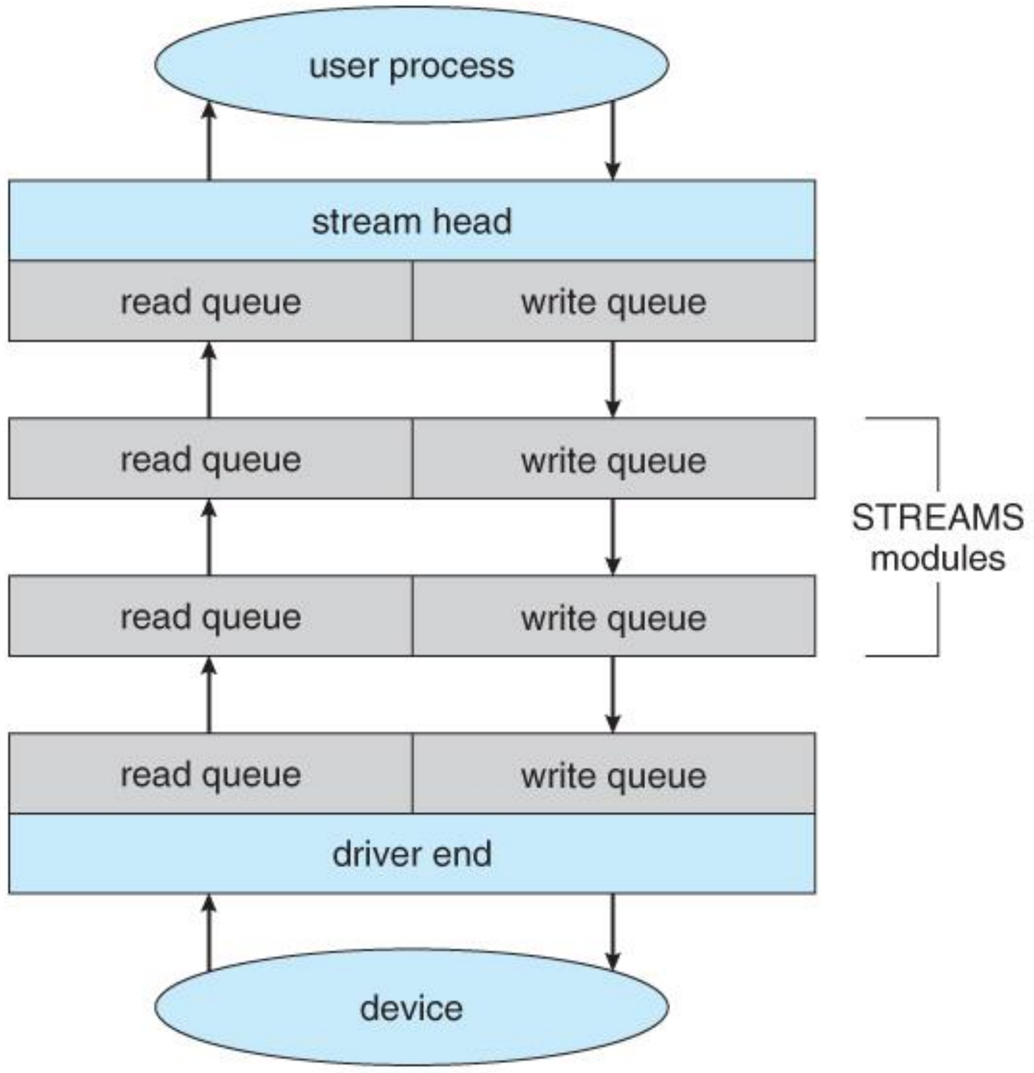

request I/O
user process
I/O completed, input data available, or output completed
system call
return from system call
can already satisfy request?
kernel I/O subsystem
yes
transfer data (if appropriate) to process, return completion or error code
no
send request to device driver, block process if appropriate
kernel I/O subsystem
process request, issue commands to controller, configure controller to block until interrupted
device driver
determine which I/O completed, indicate state change to I/O subsystem
device-controller commands
interrupt handler
receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
interrupt
monitor device, interrupt when I/O completed
device controller
I/O completed, generate interrupt
time

The life cycle of an I/O request

Streams

- The ***streams*** mechanism in UNIX provides a bi-directional pipeline between a user process and a device driver, onto which additional modules can be added.
- The user process interacts with the ***stream head.***
- The device driver interacts with the ***device end.***
- Zero or more ***stream modules*** can be pushed onto the stream, using ioctl( ). These modules may filter and/or modify the data as it passes through the stream.
- Each module has a ***read queue*** and a ***write queue.***
- ***Flow control*** can be optionally supported, in which case each module will buffer data until the adjacent module is ready to receive it. Without flow control, data is passed along as soon as it is ready.
- User processes communicate with the stream head using either read( ) and write( ) (or putmsg( ) and getmsg( ) for message passing.)
- Streams I/O is asynchronous (non-blocking), except for the interface between the user process and the stream head.
- The device driver **must** respond to interrupts from its device - If the adjacent module is not prepared to accept data and the device driver's buffers are all full, then data is typically dropped.
- Streams are widely used in UNIX, and are the preferred approach for device drivers. For example, UNIX implements sockets using streams

Streams



The STREAMS structure

धन्यवाद