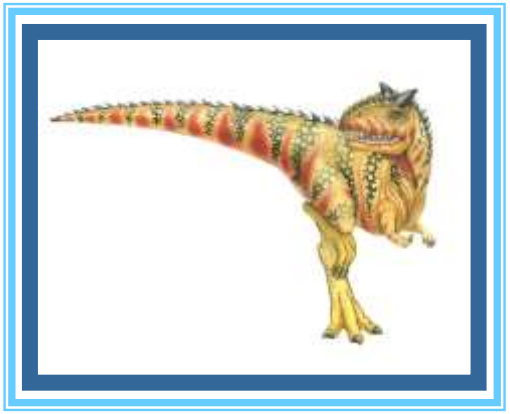
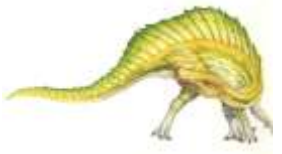
UNIT VI: I/O Systems



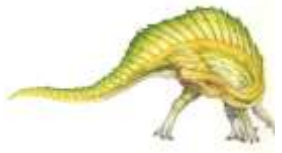
UNIT VI- I/O Systems

- I/O Hardware
- Polling
- Interrupts and DMA
- Protection:Goals of protection
- Principles of protection
- Access matrix
- Implementation of Access matrix
- Access control list
- Capability List
- Program Threats



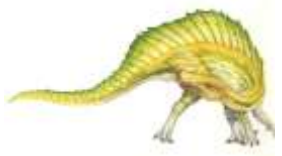
Overview

- I/O management is a major component of operating system design and operation
- The role of the operating system in computer I/O is to manage and control I/O operations and I/O devices.
  - Important aspect of computer operation
  - I/O devices vary greatly
  - Various methods to control them
  - Performance management
  - New types of devices frequent
- Ports, busses, device controllers connect to various devices

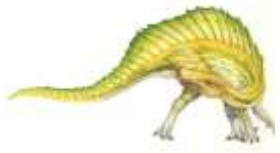


I/O Hardware

- Computers operate a great many kinds of devices.
- Most fit into the general categories of
  - ✓ **storage devices** (disks, tapes),
  - ✓ **transmission devices** (network connections,Bluetooth), and
  - ✓ **human-interface devices** (screen, keyboard, mouse, audio in and out).
- A device communicates with a computer system by sending signals over a **cable** or even through the **air.**
- The device communicates with the machine via a connection point, or **port**—for example, a serial port.
- If devices share a common set of wires, the connection is called a **bus.**



- A **bus** is a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires.
- When device *A* has a cable that plugs into device *B*, and device *B* has a cable that plugs into device *C*, and device *C* plugs into a port on the computer, this arrangement is called a **daisy chain**. A daisy chain usually operates as a **bus**.
- A typical PC bus structure appears in Figure 13.1. In the figure,
- a **PCI bus** (the common PC system bus) connects the processor–memory subsystem to fast devices, and
- an **expansion bus** connects relatively slow devices, such as the keyboard and serial and USB ports.



A Typical PC Bus Structure

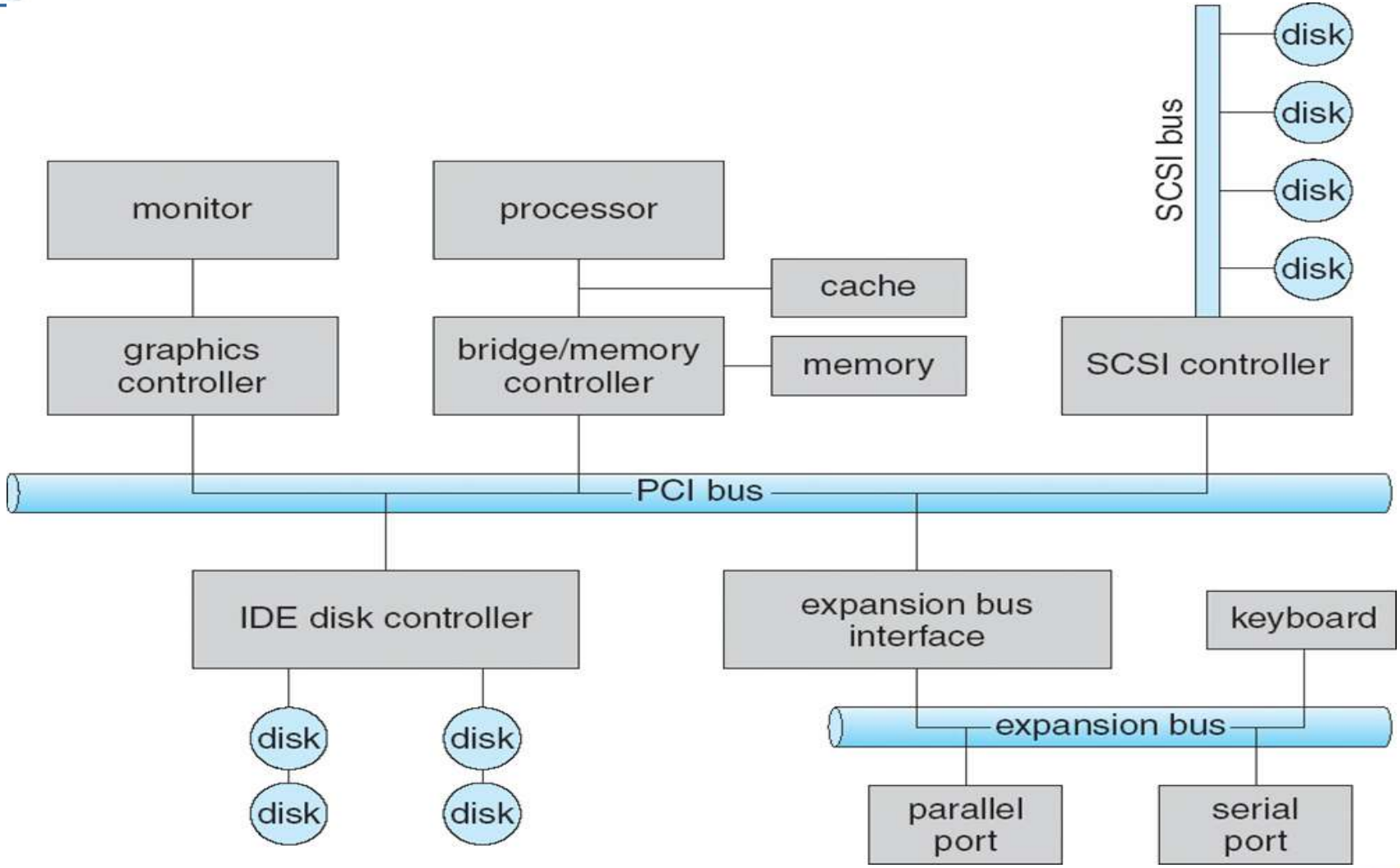
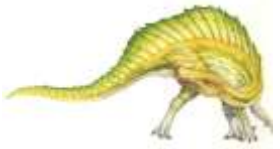
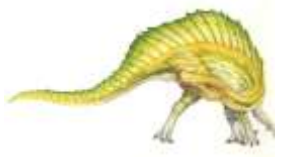


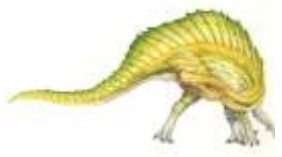
Figure 13.1 A typical PC bus structure.



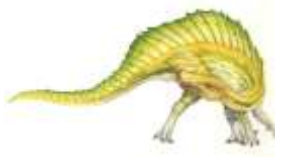
- In the upper-right portion of the figure, four disks are connected together on a **Small Computer System Interface (SCSI)** bus plugged into a SCSI controller.
- Other common buses used to interconnect main parts of a computer include **PCI Express (PCIe)**, with throughput of up to 16 GB per second, and **HyperTransport**, with throughput of up to 25 GB per second.
- A **controller** is a collection of electronics that can operate a port, a bus, or a device.
- A **serial-port** controller is a simple device controller.
- It is a single chip (or portion of a chip) in the computer that controls the signals on the wires of a serial port.



- An I/O port typically consists of four registers, called the status, control,
- data-in, and data-out registers.
  - ✓ The **data-in register** is read by the host to get input.
  - ✓ The **data-out register** is written by the host to send output.
  - ✓ The **status register** contains bits that can be read by the host. These bits indicate states, such as whether the current command has completed, whether a byte is available to be read from the data-in register, and whether a device error has occurred.
- • The **control register** can be written by the host to start a command or to change the mode of a device.



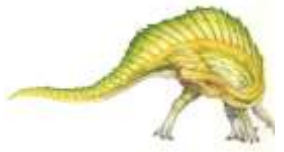
- I/O instructions control devices
- The data registers are typically 1 to 4 bytes in size.
- Some controllers have FIFO chips that can hold several bytes of input or output data to expand the capacity of the controller beyond the size of the data register.
- Devices have addresses, used by
  - **Direct I/O instructions-**that specify the transfer of a byte or word to an I/O port address
  - **Memory-mapped I/O-**the device-control registers are mapped into the address space of the processor.



Device I/O Port Locations on PCs (partial)

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Figure 13.2 Device I/O port locations on PCs (partial).



Polling

What is Polling?

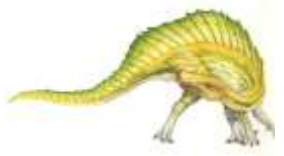
Polling is not a hardware mechanism, its a protocol in which CPU steadily checks whether the device needs attention. Whenever device tells process unit that it desires hardware processing, in polling process unit keeps asking the I/O device whether or not it desires CPU processing. The CPU ceaselessly check every and each device hooked up thereto for sleuthing whether or not any device desires hardware attention. Each device features a command-ready bit that indicates the standing of that device, i.e., whether or not it's some command to be read by hardware or not. If command bit is ready one, then it's some command to be read else if the bit is zero, then it's no commands.

Advantages

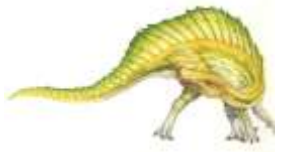
- No context switching
- Reduced interrupt overhead

Disadvantages

- Inefficiency
- Increased Power Consumption



- For each byte of I/O
  1. Read busy bit from status register until 0
  2. Host sets read or write bit and if write copies data into data-out register
  3. Host sets command-ready bit
  4. Controller sets busy bit, executes transfer
  5. Controller clears busy bit, error bit, command-ready bit when transfer done
- Step 1 is **busy-wait** cycle to wait for I/O from device
  - Reasonable if device is fast
  - But inefficient if device slow
  - CPU switches to other tasks?
    - But if miss a cycle data overwritten / lost



Interrupts

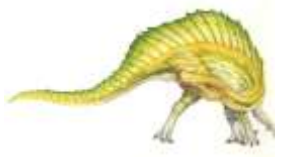
- **What is Interrupt?**
- Interrupt is a hardware mechanism in which, the device notices the CPU that it requires its attention.
- Interruptions can take place at any time. So when the CPU gets an interrupt signal through the indication interrupt-request line, the CPU stops the current process and responds to the interrupt by passing the control to the interrupt handler which services the device.

Advantages

- ✓ Sleep Modes
- ✓ Parallel Processing

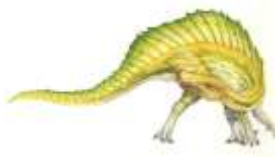
Disadvantages

- ✓ Context Switching
- ✓ Interrupt Latency



- The CPU hardware has a wire called the **interrupt-request line** that the CPU senses after executing every instruction.
- When the CPU detects that a controller has asserted a signal on the interrupt-request line, the CPU performs a state save and jumps to the **interrupt-handler routine** at a fixed address in memory.
- The interrupt handler determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a return from interrupt instruction to return the CPU to the execution state prior to the interrupt.

■ Figure 13.3 summarizes the interrupt-driven I/O cycle.



Interrupt-Driven I/O Cycle

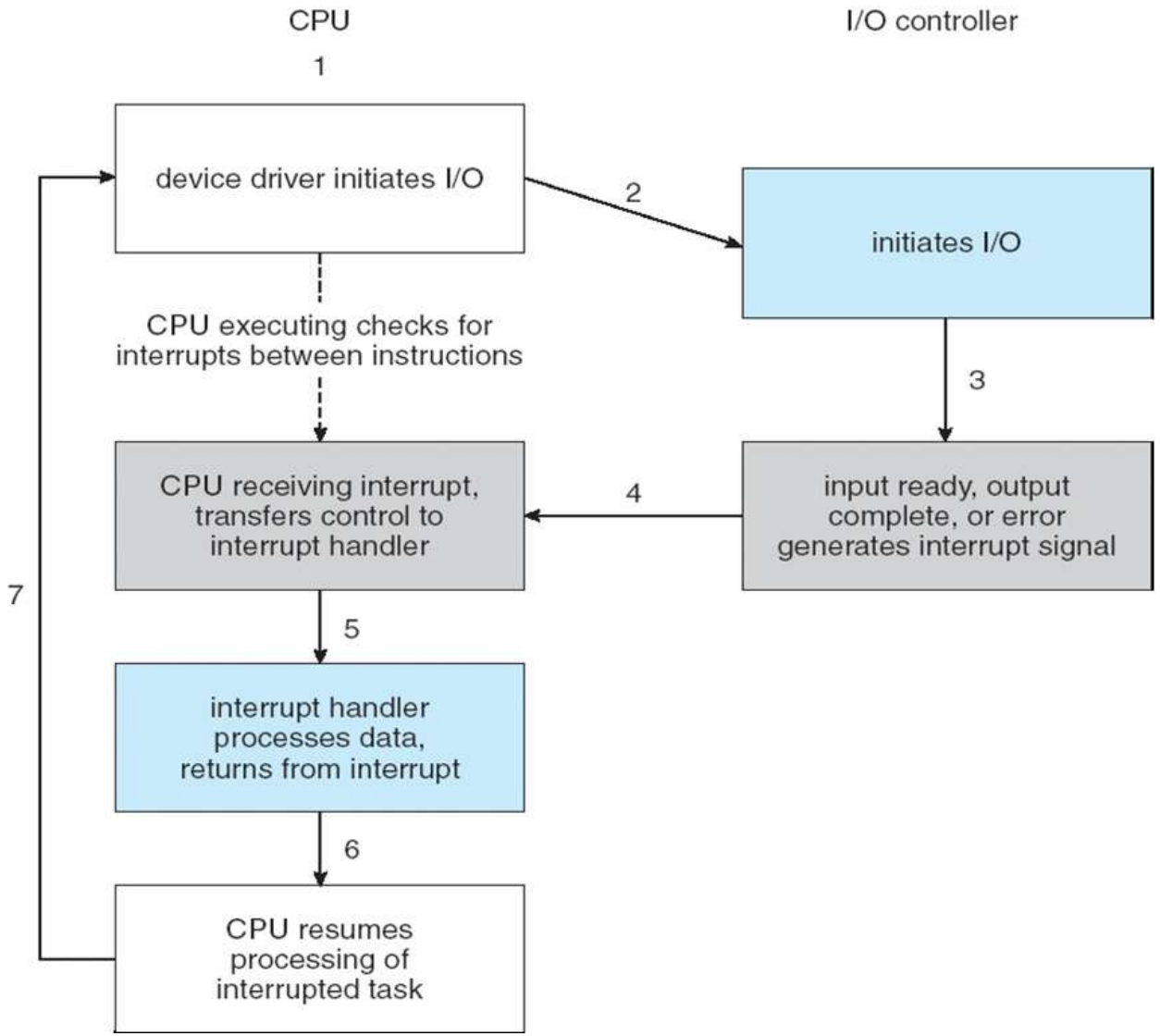


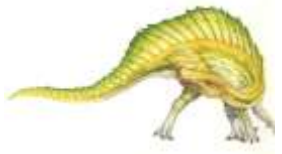
Figure 13.3 Interrupt-driven I/O cycle.

- In a modern operating system, however, we need more sophisticated interrupt-handling features.

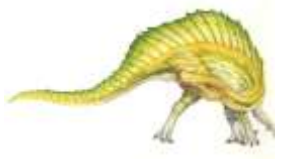
  1. We need the ability to defer interrupt handling during critical processing.

  2. We need an efficient way to dispatch to the proper interrupt handler for a device without first polling all the devices to see which one raised the interrupt.

  3. We need multilevel interrupts, so that the operating system can distinguish between high- and low-priority interrupts and can respond with the appropriate degree of urgency.



- Most CPUs have two interrupt request lines. One is the **nonmaskable interrupt**, which is reserved for events such as unrecoverable memory errors.
- The second interrupt line is **maskable**: it can be turned off by the CPU before the execution of critical instruction sequences that must not be interrupted.
- The **maskable interrupt** is used by device controllers to request service.
- The interrupt mechanism accepts an **address**—a number that selects a specific interrupt-handling routine from a small set.
- In most architectures, this address is an offset in a table called the **interrupt vector table.**



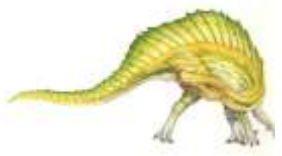
Intel Pentium Processor Event-Vector Table

- Figure 13.4 illustrates the design of the interrupt vector for the Intel Pentium processor.
- The events from 0 to 31, which are **nonmaskable,** are used to signal various error conditions.
- The events from 32 to 255, which are maskable, are used for purposes such as device-generated interrupts.

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

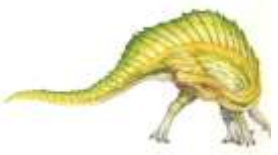
Figure 13.4 Intel Pentium processor event-vector table.

- Interrupt mechanism also used for **exceptions**
  - Terminate process, crash system due to hardware error
- Page fault executes when memory access error
- System call executes via **trap** to trigger kernel to execute request
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it
- Used for time-sensitive processing, frequent, must be fast



Difference Between Interrupt and Polling

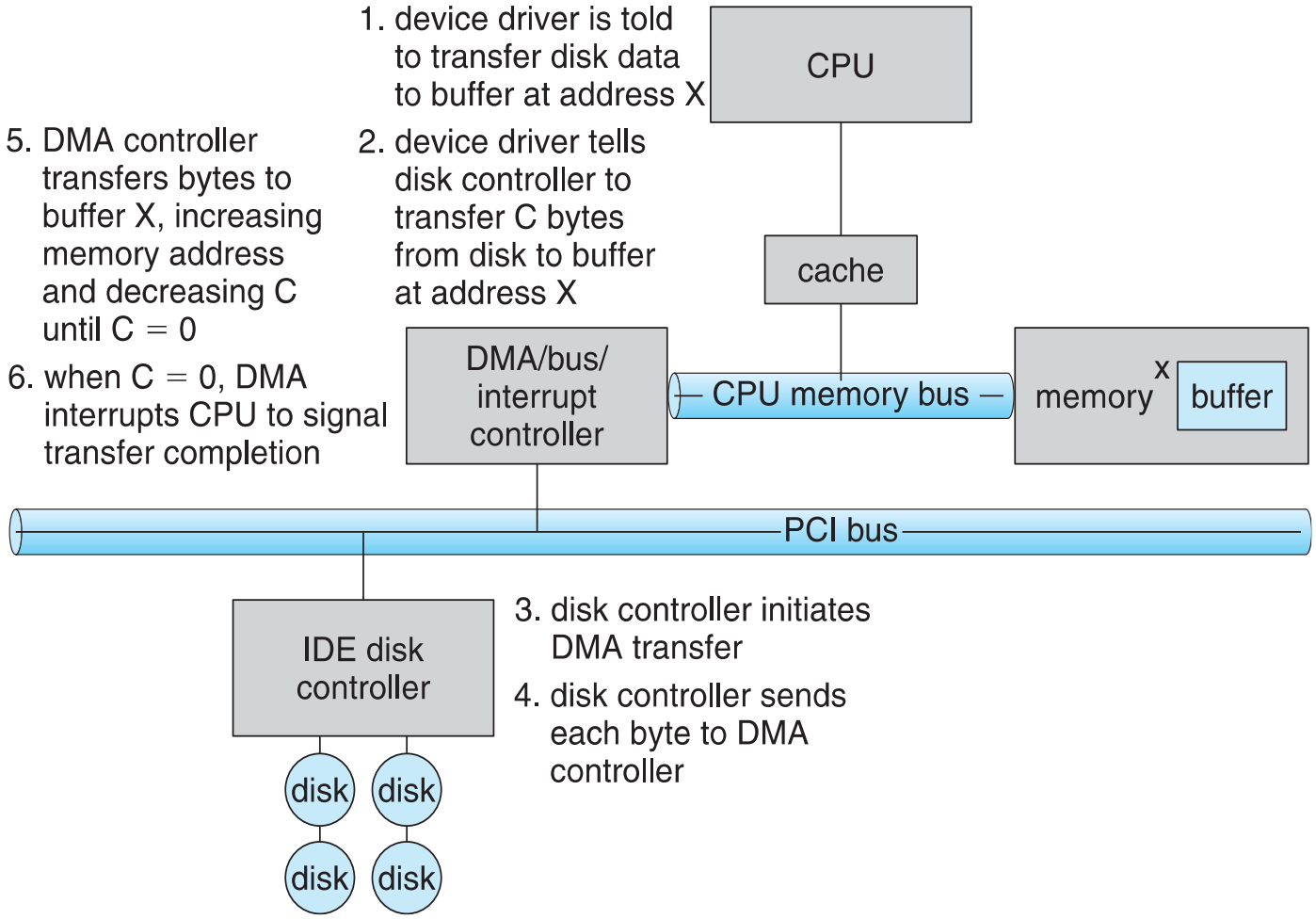
Interrupt	Polling
In interrupt, the device notices the CPU that it requires its attention.	Whereas, in polling, CPU steadily checks whether the device needs attention.
An interrupt is not a protocol, its a hardware mechanism.	Whereas it isn't a hardware mechanism, its a protocol.
In interrupt, the device is serviced by interrupt handler.	While in polling, the device is serviced by CPU.
Interrupt can take place at any time.	Whereas CPU steadily ballots the device at regular or proper interval.
In interrupt, interrupt request line is used as indication for indicating that device requires servicing.	While in polling, Command ready bit is used as indication for indicating that device requires servicing.
In interrupts, processor is simply disturbed once any device interrupts it.	On the opposite hand, in polling, processor waste countless processor cycles by repeatedly checking the command-ready little bit of each device.

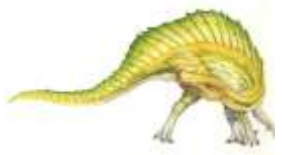


Direct Memory Access

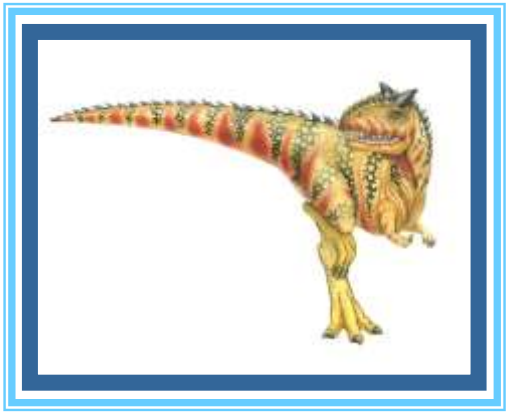
- Used to avoid **programmed I/O** (one byte at a time) for large data movement
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory
- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Writes location of command block to DMA controller
  - Bus mastering of DMA controller – grabs bus from CPU
    - **Cycle stealing** from CPU but still much more efficient
  - When done, interrupts to signal completion
- Version that is aware of virtual addresses can be even more efficient - **DVMA**

Six Step Process to Perform DMA Transfer





Protection



❖Goals of Protection

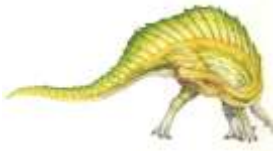
•In one protection model, computer consists of a collection of objects, hardware or software

•Each object has a unique name and can be accessed through a well-defined set of operations

•Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so

Security Violation Categories

- **Breach of confidentiality**
  - Unauthorized reading of data
- **Breach of integrity**
  - Unauthorized modification of data
- **Breach of availability**
  - Unauthorized destruction of data
- **Theft of service**
  - Unauthorized use of resources
- **Denial of service (DOS)**
  - Prevention of legitimate use



❖Principles of Protection

Guiding principle – **principle of least privilege**

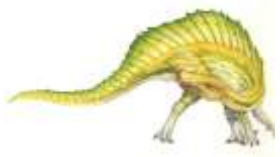
- Programs, users and systems should be given just enough privileges to perform their tasks
- Properly set permissions can limit damage if entity has a bug, gets abused

✓Can be **static** (during life of system, during life of process)

✓Or **dynamic** (changed by process as needed) – domain switching, privilege escalation

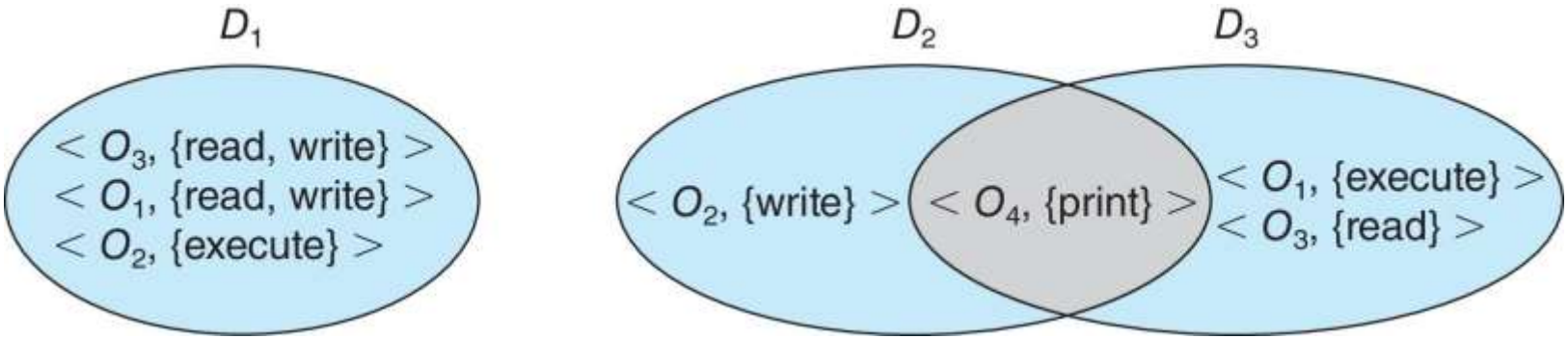
Compartmentalization a derivative concept regarding access to data

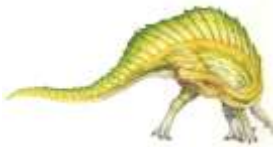
□ Process of protecting each individual system component through the use of specific permissions and access restrictions



Domain Structure

- Access-right = <*object-name*, *rights-set*>
- where *rights-set* is a subset of all valid operations that can be performed on the object
- Domain = set of access-rights

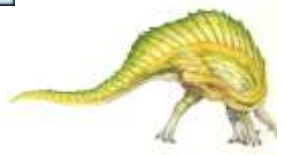




Access Matrix

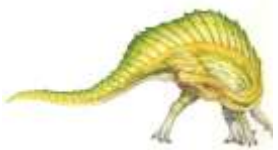
- View protection as a matrix (access matrix) Rows represent domains
- Columns represent objects
- **Access(i, j)** is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	



Use of Access Matrix

- I f a process in Domain D_i tries to do "op" on object O_j, then "op " must be in the access matrix
- User who creates object can define access column for that object
- Can be expanded to dynamic protection
- Operations to add, delete access rights Special access rights:
  - *owner of* O_i
  - *copy op from* O_i *to* O_j *(denoted by "*")*
  - *control* – D_i *can modify* D_j *access rights*
  - *transfer – switch from domain* D_i *to* D_j *Copy* and *Owner* applicable to an object *Control* applicable to domain object



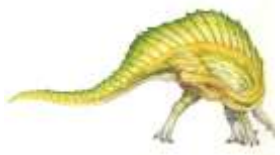
Use of Access Matrix (Cont.)

But doesn't solve the general confinement problem

- Access matrix design separates mechanism from policy
- Mechanism
  - Operating system provides access-matrix + rules
  - I f ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced

Policy

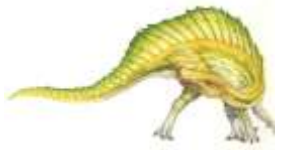
- User dictates policy
- Who can access what object and in what mode



Access Matrix

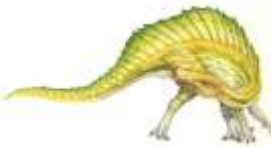
- The model of protection can be viewed an **access matrix**.
- The rows of the access matrix represent domains, and the columns represent objects.
- Each entry in the matrix consists of a set of access rights.
- The entry access(*i*,*j*) defines the set of operations that a process executing in domain *Di* can invoke on object *Oj* .
- Process executing in domain Di can access only those objects specified in row i, and then only as allowed by the access-matrix entries.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	



Access Matrix...

- The access matrix can implement policy decisions concerning protection.
- The policy decisions involve which rights should be included in the (*i*, *j*)*th* entry.
- This policy is decided by the operating system.
- The users decide the contents of the access-matrix entries.
- When a user creates a new object *Oj* , the column ***Oj* is added to the access matrix with the appropriate initialization entries, as dictated by the creator.**
- The user decide to enter some rights, in some entries in column *j*, and other rights in other entries, as needed.



Access Matrix of Figure A with Domains as Objects

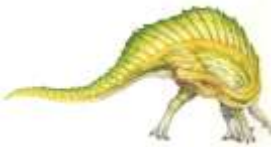
domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Methods of Implementing The Access Matrix

- Global Table
- Access Lists for Objects
- Capability Lists for Domains
- A Lock–Key Mechanism

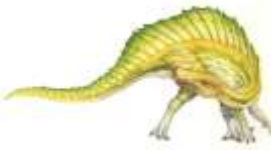
1. Global Table

- The simplest implementation of the access matrix is a global table.
- It consisting of a set of ordered triples <domain, object, rights-set>.
- Whenever an operation *M* is executed on an object *Oj* within domain *Di*, the global table is searched for a
- triple <*Di* , *Oj* , *Rk*>, with $M \in Rk$.
- If this triple is found, the operation is allowed to continue; otherwise, an exception (or error) condition is raised.



Drawbacks of Global Table

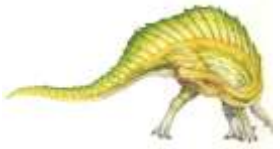
- The table is usually large and thus cannot be kept in main memory, so additional I/O is needed.
- Virtual memory techniques are often used for managing this table.
- It is difficult to take advantage of special groupings of objects or domains.
- For example, if everyone can read a particular object, this object must have a separate entry in every domain.



2. Access Lists for Objects

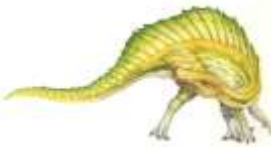
- Each column in the access matrix can be implemented as an access list for one object, the empty entries can be discarded.
- The resulting list for each object consists of ordered pairs <domain, rights-set>,
- which define all domains with a nonempty set of access rights for that object.
- This approach can be extended easily to define a list, and a ***default*** set of access rights.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	



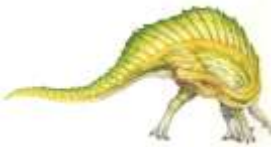
Access Lists for Objects...

- When an operation *M* on an object *Oj* is attempted in domain *Di*, we search the access list for object *Oj*, looking for an entry <*Di* , *Rk*> with $M \in Rk$.
- If the entry is found, we allow the operation; if it is not, we check the default set.
- If *M* is in the default set, we allow the access.
- Otherwise, access is denied, and an exception condition occurs.
- For efficiency, we may check the default set first and then search the access list.



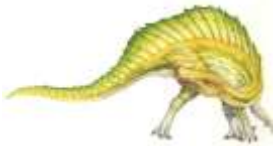
3. Capability Lists for Domains

- A **capability list** for a domain is a list of objects together with the operations allowed on those objects.
- An object is represented by its physical name or address, called a **capability**.
- To execute operation *M* on object *Oj*, the process executes the operation *M*, specifying the capability (or pointer) for object *Oj* as a parameter.
- The **possession** of the capability means that access is allowed.



4. A Lock–Key Mechanism

- The **lock–key scheme** is a compromise between access lists and capability lists.
- Each object has a list of unique bit patterns, called **locks**.
- Each domain has a list of unique bit patterns, called **keys**.
- A process executing in a domain can access an object, only if that domain has a key, that matches one of the locks of the object.
- As with capability lists, the list of keys for a domain must be managed by the operating system on behalf of the domain.
- Users are not allowed to examine or modify the list of keys (or locks) directly.



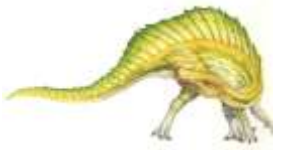
Access Matrix with *Copy* Rights

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)



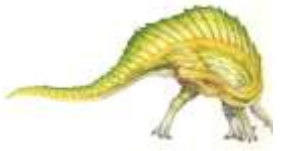
Access Matrix With *Owner* Rights

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

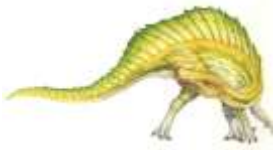
object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)



Modified Access Matrix of Figure B

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			



Implementation of Access Matrix

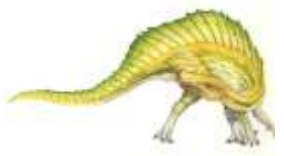
- Generally, a sparse matrix
- Option 1 – Global table
  - Store ordered triples **<domain, object, rights-set>** in table

A requested operation M on object O_j within domain D_i -> search table for $< D_i, O_j, R_k >$

- with M E R_k

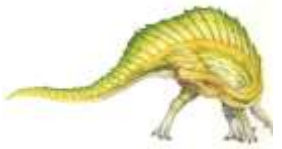
But table could be large -> won't fit in main memory

Difficult to group objects (consider an object that all domains can read)



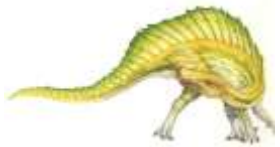
Implementation of Access Matrix (Cont.)

- Option 2 – Access lists for objects
  - Each column implemented as an access list for one object
  - Resulting per-object list consists of ordered pairs `<domain, rights-set>` defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> I f M E default set, also allow access



Revocation of Access Rights (Cont.)

- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
  - **Reacquisition** – periodic delete, with require and denial if revoked
  - **Back-pointers** – set of pointers from each object to all capabilities of that object (Multics)
  - **Indirection** – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
  - **Keys** – unique bits associated with capability, generated when capability created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key
    - Policy decision of who can create and modify keys – object owner or others?



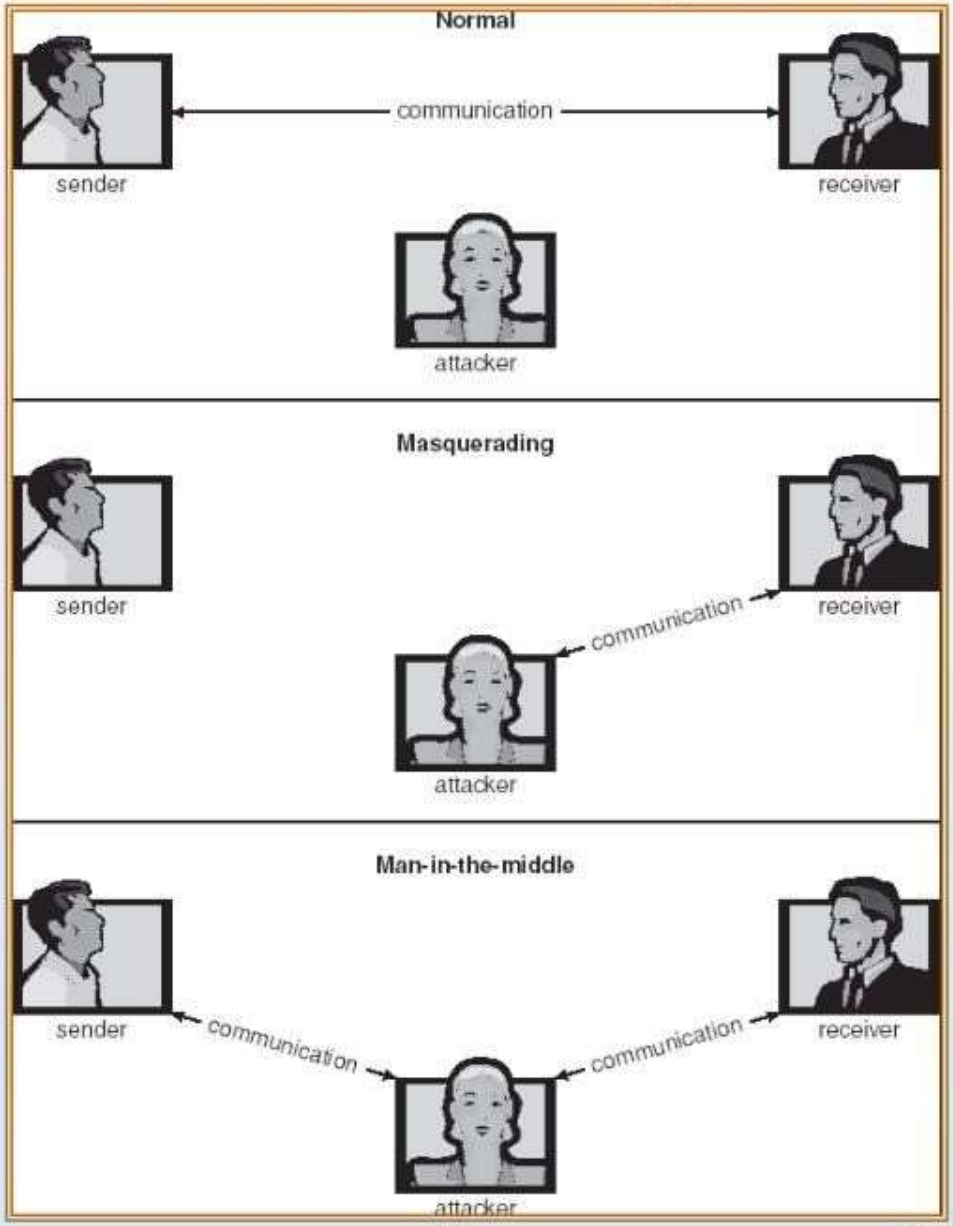
Security Violations

- **Methods**(see next slide)
  - **Masquerading (breach authentication)**
    - One participant in a communication pretends to be someone else
    - Could be another host or person.
    - Goal is to gain access that they would not normally be allowed
    - Or could try to escalate their privileges
  - **Replay attack**
    - Replay a captured exchange of data
    - Sometimes the replay is the attack: ex: repeat of a request to transfer money
    - **Message modification**: replace some data in the replay to obtain access for unauthorized user.
  - **Man-in-the-middle attack**
    - Attacker sits in the data flow of a communication
    - Masquerades as the sender to the receiver and vice versa
  - **Session hijacking**
    - An active comunication session is intercepted.

Security Violations

- **Categories** (cont)
  - **Theft of service**
    - Unauthorized use of resources
    - Ex: using someones computer as a porn server
  - **Denial of service**
    - Preventing legitimate use of the system
    - DOS is sometimes accidental
    - The original internet worm became a DOS when a bug failed to delay its rapid spread.

Standard Security Attacks

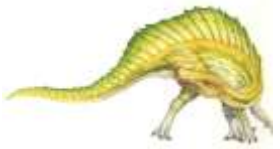


Security Measure Levels

- Security must occur at **four levels** to be effective:
  - **Physical**
    - Physically secure the computers
  - **Human**
    - Avoid **social engineering**
      - **Phishing:** a legitimate-looking e-mail or web page misleads a user into entering confidential information
      - **dumpster diving:** looking through trash, finding phone books, etc
  - **Operating System**
    - The system must protect itself from accidental or purposeful security breaches.
    - Ex: a runaway process
    - Stack overflow
  - **Network**
    - Intercepting data traveling between comptuers
- ***Security is as week as the weakest chain***

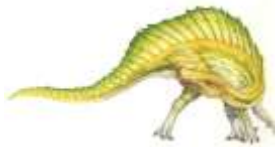
Revocation of Access Rights

- Various options to remove the access right of a domain to an object
  - **Immediate vs.**
  - **delayed Selective vs.**
  - **general Partial vs.**
  - **total Temporary vs.**
  - **permanent**
- Access List – Delete access rights from access list
  - **Simple** – search access list and remove entry
  - **Immediate**, **general** or **selective**, **total** or **partial**, **permanent** or **temporary**



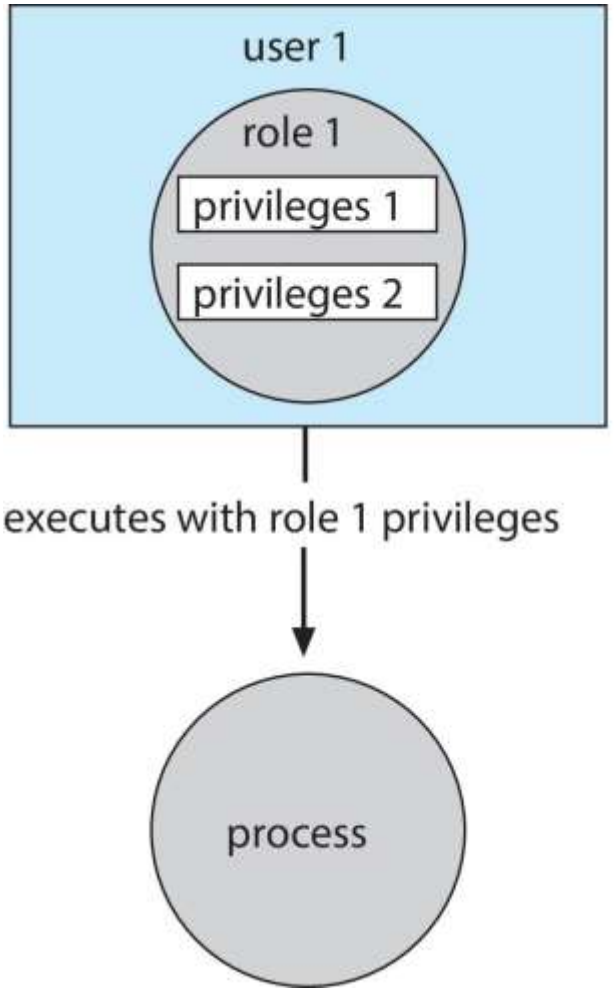
Revocation of Access Rights (Cont.)

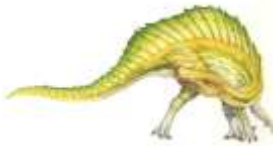
- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
  - **Reacquisition** – periodic delete, with require and denial if revoked
  - **Back-pointers** – set of pointers from each object to all capabilities of that object (Multics)
  - **Indirection** – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
  - **Keys** – unique bits associated with capability, generated when capability created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key
    - Policy decision of who can create and modify keys – object owner or others?



Role-based Access Control

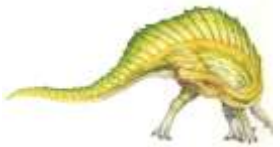
- Protection can be applied to non-file resources
- Oracle Solaris 10 provides **role-based access control** (**RBAC**) to implement least privilege
  - ***Privilege*** is right to execute system call or use an option within a system call
  - Can be assigned to processes
  - Users assigned ***roles*** granting access to privileges and programs
  - Enable role via password to gain its privileges
  - Similar to access matrix





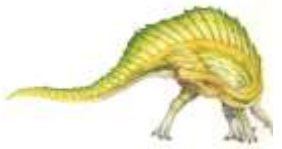
Mandatory Access Control (MAC)

- Operating systems traditionally had discretionary access control (DAC) to limit access to files and other objects (for example UNIX file permissions and Windows access control lists (ACLs))
  - Discretionary is a weakness – users / admins need to do something to increase protection
- Stronger form is mandatory access control, which even root user can't circumvent
  - Makes resources inaccessible except to their intended owners
  - Modern systems implement both MAC and DAC, with MAC usually a more secure, optional configuration (Trusted Solaris, TrustedBSD (used in macOS), SELinux), Windows Vista MAC)
- At its heart, labels assigned to objects and subjects (including processes)
  - When a subject requests access to an object, policy checked to determine whether or not a given label-holding subject is allowed to perform the action on the object



Capability-Based Systems

- Hydra and CAP were first capability-based systems
- Now included in Linux, Android and others, based on POSIX.1 e (that never became a standard)
  - Essentially slices up root powers into distinct areas, each represented by a bitmap bit
  - Fine grain control over privileged operations can be achieved by setting or masking the bitmap
  - Three sets of bitmaps – permitted, effective, and inheritable
    - Can apply per process or per thread Once revoked, cannot be
    - reacquired
    - Process or thread starts with all privs, voluntarily decreases set during execution
    - Essentially a direct implementation of the principle of least privilege
- An improvement over root having all privileges but inflexible (adding new privilege difficult, etc)

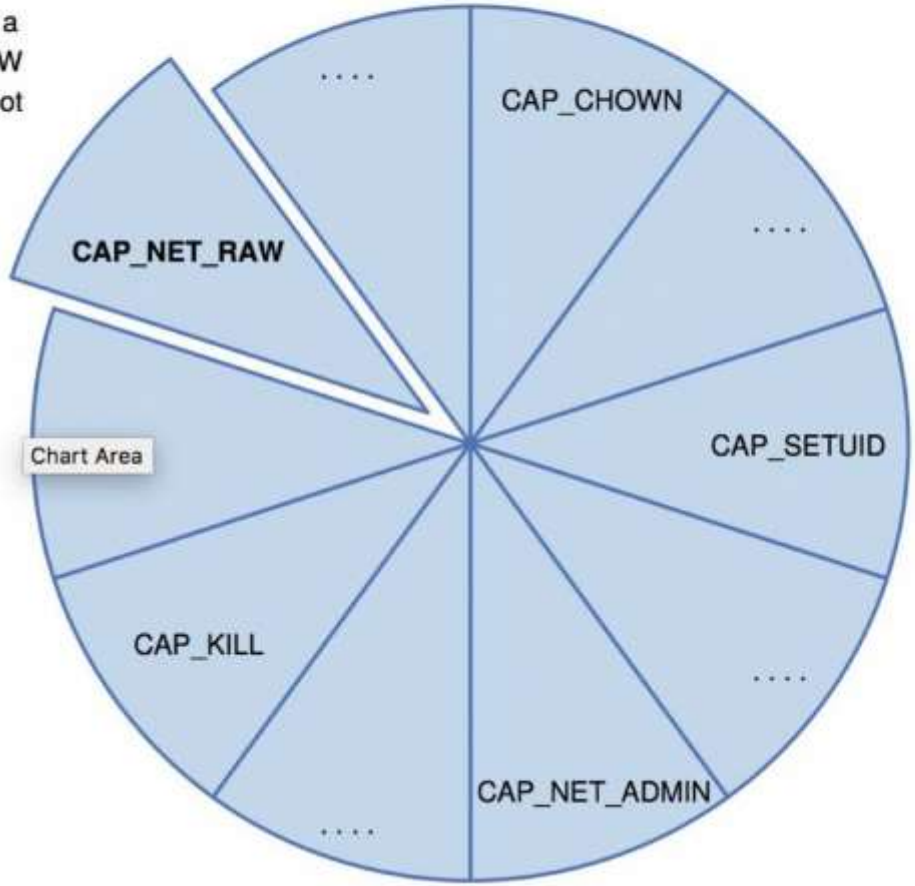


Capabilities in POSIX.1e

In the old model, even a simple `ping` utility would have required root privileges, because it opens a raw (ICMP) network socket

Capabilities can be thought of as "slicing up the powers of root" so that individual applications can "cut and choose" only those privileges they actually require

With capabilities, ping can run as a normal user, with CAP_NET_RAW set, allowing it to use ICMP but not other extra privileges



Program Threats

- The most common goal of crackers:
  - Write a program that creates a breach of security
  - Or cause a normal process to change its behavior and create a breach
- Example:
  - Useful to log into a system without authorization
  - More useful to leave behind a **back-door** daemon that provides info or allows easy access even if the original exploit is blocked.

Program Threats

- **Trojan Horse**
  - Code segment that misuses its environment
  - Exploits mechanisms for allowing programs written by users to be executed by other users
  - **Spyware, pop-up browser windows, covert channels**
- **Examples**
  - Text-editor program may include code to search the file to be edited for certain keywords
    - These are then saved in a hidden file accessible to the creator of the text editor.

Program Threats

- **Trojan Horse** more examples:
  - Long search paths (the PATH environmental variable)
  - If not every path is secure, could execute wrong program
  - Example: use of the "." in the path (search current directory)
    - If you go to a friend's directory and execute a command, the command may be run from her directory
    - This would give the program her permissions
    - Could delete her files, etc.

Program Threats

- Trojan Horse (cont)
- More examples:
  - Example: a program that emulates a login program
    - User logs in at a terminal and notices that he as apparently mistyped his password
    - He tries again and is successful
    - What really happened? His account name/password were stolen by the login emulator that was left running by the thief.
    - The emulator stored his information, printed out a login error message, and exited.
    - User then got the real prompt.
  - Protection:
    - Use non-trappable key sequence (ctrl-alt-delete)
    - Have the OS print a usage message at the end of an interactive session instead of just a new login prompt

Program Threats

- Trojan Horse (cont)
- More examples:
  - **Spyware**.
    - Sometimes accompanies a program that the user has chosen to install
    - Sometimes with freeware/shareware sometimes with commercial software
    - Goals:
      - Download ads to display on the user's system
      - Create pop-up windows when certain sites are visited
      - Capture info from the user's system and return it to a central site.
    - **Cover Channel attack**
      - Surreptitious communication occurs
      - A spyware daemon is loaded
      - It contacts a central site and is given a message and a list of recipient addresses
      - It delivers the spam message to those users from the infected machine
      - 80% of spam was delivered this way in 2004!

Program Threats

- **Trap Door**
  - The designer of a program or system might leave a hole in the software that only she is capable of using.
  - Used in the movie *War Games*.
  - Example: program recognizes a specific user identifier or password that circumvents normal security procedures
  - Example: programmer for a bank might include rounding errors in their code and have the resulting half-cent deposited in their account
  - Could be included in a compiler
    - Compiler generates standard object code
    - Also includes the trap door.
    - Hard to find: searching the source code will not reveal the trap door; it is only in the compiler!

Program Threats

- **Logic Bomb**
  - Program that initiates a security incident under certain circumstances
  - Under normal circumstances there is no security hole.
  - When a predefined set of parameters were met, the security hole is created.
  - Example: programmer writes code that checks to see if she is still employed
    - If not, a daemon is spawned to allow remote access or to damage the site.

Program Threats

- **Stack and Buffer Overflow**
  - Most common technique for attacker from outside the system to gain unauthorized access to the target system.
  - Exploits a bug in a program
    - overflow either the stack or memory buffers
    - Usually programmer neglected to code bounds checking on an input field
    - Attacker sends more data than the program expects

Program Threats

- **Stack and Buffer Overflow**
  - Using trial & error (or examining code if open source), attacker writes a program to do the following:
  1. Overflow an input field, command-line argument, or input buffer for a program (like a network daemon) until it writes into the stack
  2. Overwrite the current return address and insert the address of the exploit code loaded in step 3.
  3. Write a simple set of code for the next space in the stack that includes the commands that the attacker wishes to execute
     1. Example to spawn a shell
  4. Result is a root shell or other privileged command execution.

Program Threats (Cont.)

- **Viruses**
  - Code fragment embedded in legitimate program
  - Very specific to CPU architecture, operating system, applications
  - Self-replicating and designed to "infect" other programs
  - Used to modify or destroy files and programs
  - Windows more susceptible than UNIX
    - In UNIX executable programs are protected from being written on by the OS
  - Usually borne via email or as a macro
    - Spam is the most common vector
    - Can spread via downloaded files

Viruses (cont)

- **Virus Categories (cont)**
  - **Macro**
    - Most common are MS Office macro viruses
  - **Source code**
    - Looks for source code and modifies it to include the virus and to help spread the virus.
  - **Polymorphic**
    - Virus changes each time it is installed to avoid detection by anit-virus software.
    - Changes do not affect the virus's functionality, just its signature.
    - A signature is a series of bytes that make up the code; can be used to identify the virus.
  - **Encrypted**
    - Virus is encrypted and includes decryption code along with the encrypted virus to avoid detection.
    - Virus first decrypts itself, then executes.

Viruses (cont)

- **Virus Categories (cont)**
  - **Stealth**
    - Modifies parts of the system that could be used to detect it.
    - Eg, could modify the read system call so that if the file it has modified is read, the original form of the code is returned rather than the infected code.
  - **Tunneling**
    - Virus attempts to bypass detection by an antivirus scanner by installing itself in the interrupt-handler chain or a device driver
  - **Multipartite**
    - Infects multiple parts of a system including boot sectors, memory, and files
    - Makes it difficult to detect
  - **Armored**
    - Coded to make itself hard for antivirus researchers to unravel and understand.
    - Can also be compressed to avoid detection
    - Also virus droppers and other full files that are part of the virus infestation are hidden via file attributes or unviewable file names.

System and Network Threats

- Internet worm
  - November 2, 1988. Robert Tappan Morris Jr., a first-year Cornell graduate student unleashed the first worm on the internet.
  - Exploited UNIX networking features (remote access) and bugs in *finger* and *sendmail* programs
    - Targeted Sun's Sun 3 workstations and VAX computers running variants of Version 4 BSD UNIX
    - Within a few hours it had consumed system resources to the point of bringing down the infected machines.
  - How it spread
    - Morris chose for the initial infection an Internet host left open for and accessible to outside users
    - From there the worm exploited flaws in the UNIX OS security routines
    - Also took advantage of UNIX utilities that simplify resource sharing in LANs.
    - Thereby gained access to other connected sites.

Symmetric Encryption

- Same key used to encrypt and decrypt
  - $E(k)$ can be derived from $D(k)$, and vice versa
- DES is most commonly used symmetric block-encryption algorithm (created by US Govt)
  - Encrypts a block of data at a time
- Triple-DES considered more secure
- Advanced Encryption Standard (**AES**), **twofish** up and coming
- RC4 is most common symmetric stream cipher, but known to have vulnerabilities
  - Encrypts/decrypts a stream of bytes (i.e wireless transmission)
  - Key is a input to psuedo-random-bit generator
    - Generates an infinite **keystream**

Asymmetric Encryption

- Public-key encryption based on each user having two keys:
  - public key – published key used to encrypt data
  - private key – key known only to individual user used to decrypt data
- Must be an encryption scheme that can be made public without making it easy to figure out the decryption scheme
  - Most common is RSA block cipher
  - Efficient algorithm for testing whether or not a number is prime
  - No efficient algorithm is know for finding the prime factors of a number

End of Unit VI