operating systems

UNIT - 5

I/O MANAGEMENT & SECURITY

feedback/corrections: vibha@pesu.pes.edu

VIBHA MASTI

I/O MANAGEMENT

- Role of OS: manage and control I/O operations and I/O devices
- I/O devices vary greatly ; no easy way to manage all (keyboards, hard disks etc.)
- I/O subsystem of kernel manages control over different I/O devices and their interaction with the rest of the system
- Separate from rest of kernel
- I/O subsystem uses device driver modules
- Device driver: abstraction of access to I/O hardware via software for kernel to interact with, irrespective of hardware device (akin to system calls for apps to interact with OS)

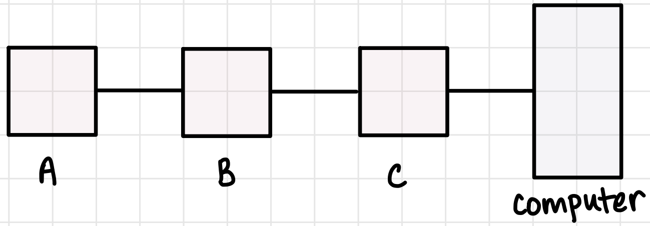
I/O hardware

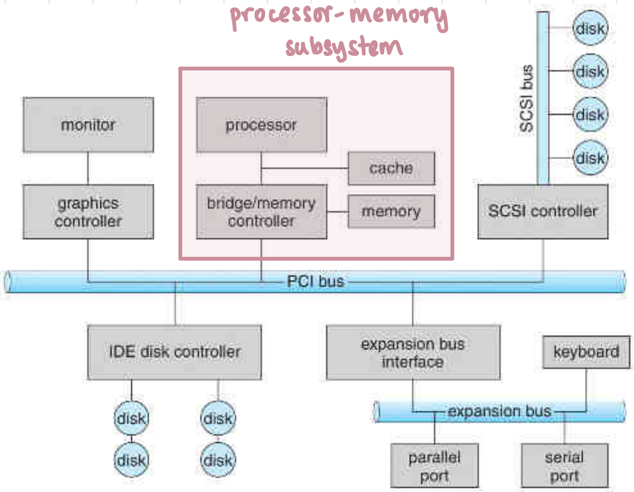
- Three major categories: storage devices, transmission devices, human interface devices

- Port: connection point where device connects to computer and sends signals via

- Bus: set of wires & a protocol specifying set of messages that can be sent across the wires

- Daisy chain: Device A plugged to device B plugged to device C plugged to computer (operates on a bus)





- PCI bus connects processor-memory subsystem to fast devices and expansion bus connects slower devices (USB and serial ports, keyboards)

- Four disks connected via Small Computer System Interface (SCSI) bus, plugged into SCSI controller

- Other common buses to interconnect computer parts:
  - PCIe: PCI Express, throughput upto 16 GB per second
  - HyperTransport: throughput upto 25 GB per second

CONTROLLER

- hardware that operates a bus/port/device

- Serial-Port controller is a simple device controller, which is a single chip in the computer, that controls the signals on the wires of a serial port

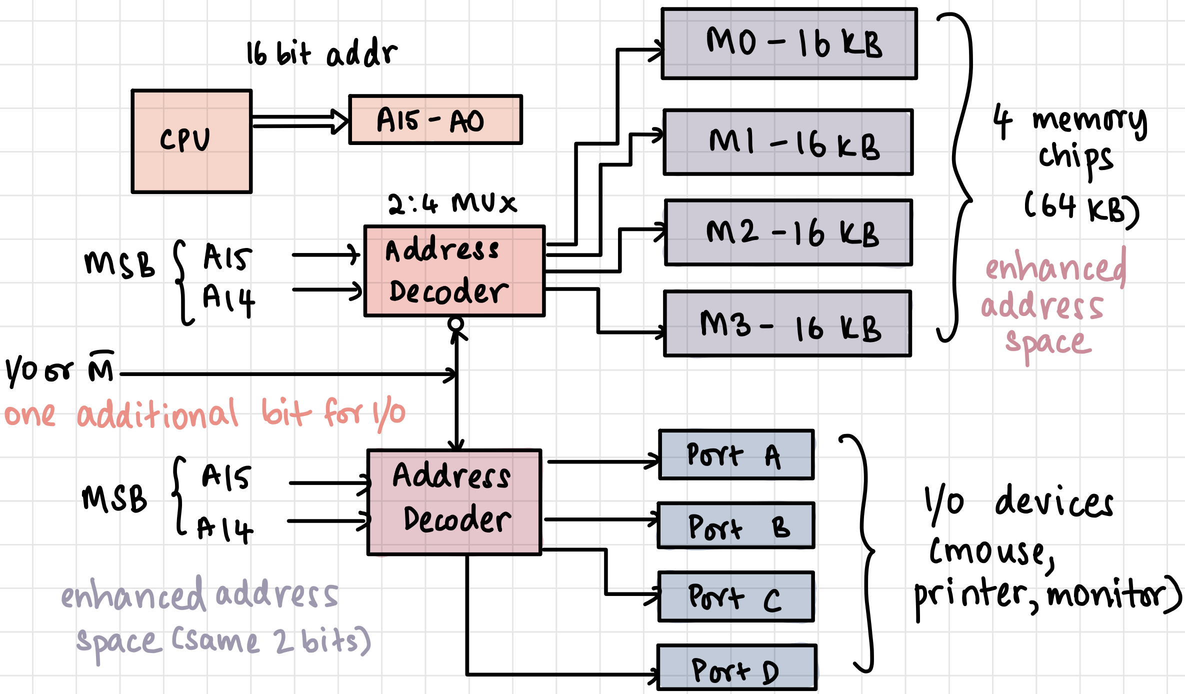
- SCSI controller more complex; controller separate circuit board called host adapter and consists of a processor, microcode and private memory

- Buit-in controllers are present in disk drives as a circuit board attached to one side. They implement disk side of a protocol such as SCSI or Serial Advanced Technology Attachment (SATA) and contain processor & microcode

Communication b/w I/O Devices & CPU

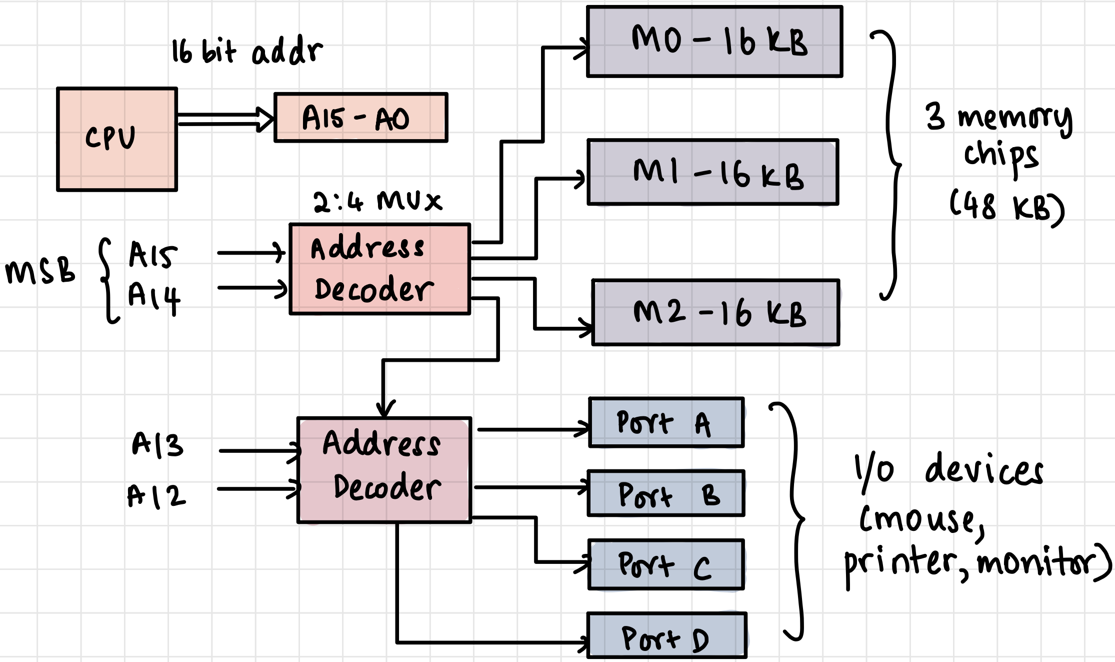
- CPU writes control information to registers on I/O device controllers

(1) I/O Mapped I/O
- Special I/O instructions (IN/OUT)
- Flow of bytes/words in/out of I/O port addresses



(2) Memory Mapped I/O

- Registers viewed as extension of addressible memory space
- Device registers mapped into physical address space
- No specific I/O intructions (IN/OUT)



- Some devices use both I/O mapped and Memory mapped I/O (graphics controllers)

- Memory mapped I/O runs the risk of accidental modification by incorrect pointers; can be solved by protected memory

Registers at I/O Ports

1. Data-in Register: read by host to get input

2. Data-out Register: written into by host to send output

3. Status Register: bits that can be read by the host (whether current command completed, whether byte available to be read from in data-in register, whether device error occured)

4. Control Register: bits than can be written by the host to change the mode of the device or to start a command (enable)

- Typically 1-4 bytes in size

- Some controllers expand capacity with FIFO chips (buffer)

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)

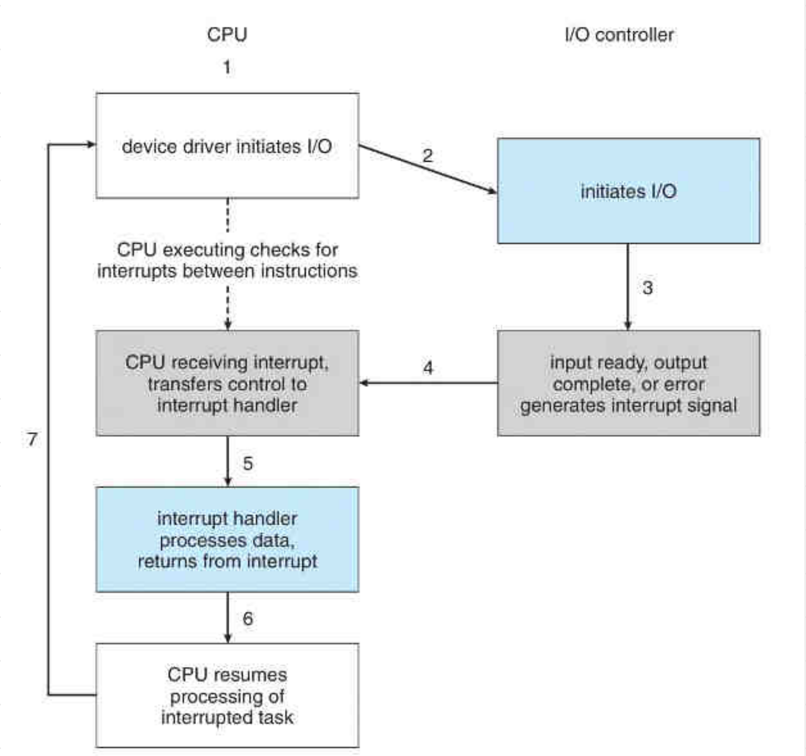
Device I/O port location on PCs

POLLING

- Method of performing I/O used by host (also called busy-waiting)
- Device controller's status register's busy bit indicates the state of the I/O device as either busy (1) or idle (0).
- Host sets the command-ready bit in the DMA controller's command register when a command is ready for the device controller to execute
- Handshaking between host & device controller for writing output:
  1. Host keeps reading status register's busy bit until it is clear
  2. Host sets write bit in command register & writes bit to data-out register
  3. Host sets command-ready bit in command register
  4. When controller notices command-ready bit is set, it sets the busy bit
  5. Controller reads command register and sees the write command
  6. Controller reads data-out register to get the byte and performs I/O on the device
  7. Controller clears the command-ready bit, error bit in the status register and clears the busy bit
- Handshake loop performed on every bite of data to be written
- Step 1: host is busy-waiting or polling the status register's busy bit
- Many cycles wasted even though basic polling operation is efficient and requires only 3 cycles
- Better if hardware controller notifies CPU when it is ready for service (command)

interrupts

- Hardware controllers notify CPU when they are ready for service via interrupts
- CPU sences interrupt service line (ISL) wire after executing every instruction
- If ISL asserted, CPU performs save state, executes interrupt service routine (ISR) from fixed memory location and then performs a state restore, returning the CPU back to its pre-interrupt state
- Device controller raises an interrupt, CPU catches it and dispatches it to the interrupt handler, and the handler clears the interrupt by servicing the device



- Required features of interrupt-handlers
  1. Ability to defer interrupt handling during critical processing
  2. Way to dispatch correct interrupt handler for a device without polling all devices
  3. Multilevel interrupts based on priority (high, low)

- Provided by CPU and interrupt-controller hardware

- Two interrupt service lines: non-maskable — cannot be turned off by CPU before running critical code, and maskable

- Interrupt mechanism accepts address (offset in IVT) that selects interrupt routine

- Interrupt vector table contains memory addresses of specialised interrupt service routines

- Interrupt chaining: entry of IVT points to head of another (multi-level indexing)

- Interrupt priority levels for high & low priority interrupts

- Interrupt mechanism also used for handling exceptions, virtual memory paging, system calls (trap or software interrupt)

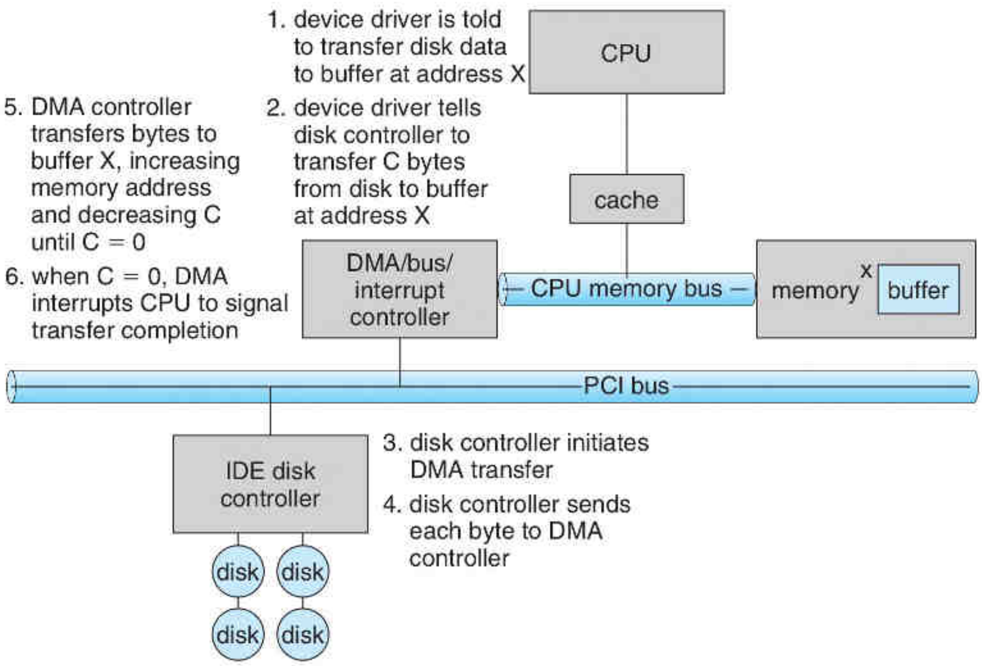
- Traps given lower priority than device interrupts

- When system call made, interrupt hardware saves the state of user code, switches to kernel mode, and dispatches to the kernel routine that implements the requested service

DIRECT memory ACCESS

- Expensive for CPU to watch status bits and feed data into a device controller's registers (programmed I/O — PIO — one byte at a time)

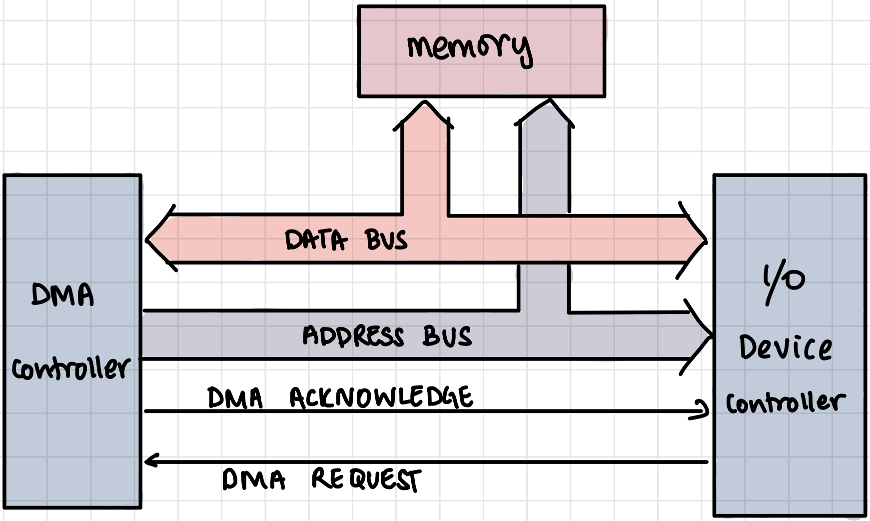
- Special purpose processor — DMA controller



- DMA transfer
  1. Host writes DMA command block into memory (contains pointers to source and destination of transfer and a count of number of bytes to be transferred)
  2. CPU writes address of this command block to the DMA controller
  3. DMA controller operates the memory bus directly and performs transfers without CPU

HANDSHAKING BETWEEN DMA CONTROLLER & DEVICE CONTROLLER

- Performed via DMA Acknowledge and DMA Request wires



- Steps:

1. Device controller places signal on DMA Request wire (DRQ) when device is ready to transfer a word of data

2. DMA controller sends a hold request to the CPU, asking it to stall for a few cycles (HLD). CPU acknowledges this request (HLDA)

3. DMA controller siezes memory bus, places desired address on the memory-address wires and places a signal on the DMA Acknowlege wire (DACK)

4. Device controller transfers word of data bytewise to memory via memory bus and removes DMA Request signal

5. When transfer complete, DMA controller interrupts the CPU signalling end of transfer

- When DMA controller seizes control of memory bus, CPU cannot access main memory (can still access caches)
- Process called cycle stealing & overall performance due to using DMA controller is improved

Protected Kernels

- Processes cannot directly issue device commands, protecting data from access-control violations and preventing system crash
- OS exports functions that privileged processes can call to access device

Non-Protected Kernels

- Processes can access device controllers directly
- High performance, low system security

DMA DATA TRANSFER MODES

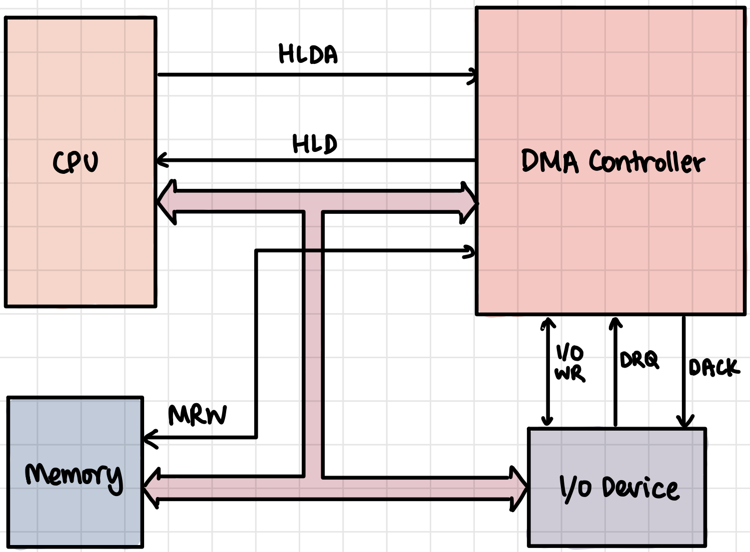
1. Burst or Block Transfer Mode
   - fastest DMA mode
   - transfers all N bytes of data in a single burst
   - for N cycles, processor disconnected from system bus
   - DMA sends HLD signal to CPU to request for memory bus and waits for HLDA signal
   - after HLDA, DMA siezes control of memory bus and transfers N bytes byte-by-byte
   - after completion, disables HLD and releases memory bus

2. Cycle Stealing or Single Byte Transfer Mode

- slower than burst mode
- after HLDA, DMA siezes control of system bus and only transfers a single byte (executes one DMA cycle)
- after single cycle, disables HLD signal and CPU regains system bus
- DMA controller needs to request system bus control for next byte
- DMA controller steals clock cycles from CPU to transfer every byte

3. Transparent or Hidden Mode

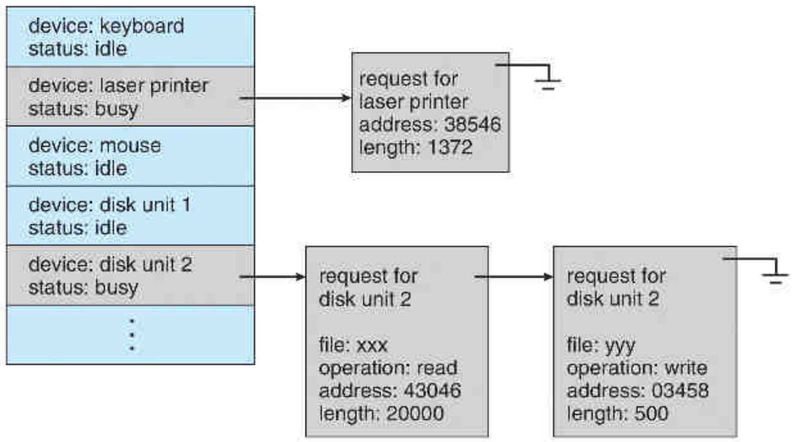
- slowest mode
- DMA controller siezes memory bus when CPU does not require bus
- processor speed unaffected



KERNEL-I/O SUBSYSTEM

1. I/O Scheduling

- Order of execution of I/O requests
- eg: optimise disk reads order
- Wait queue of requests for each device maintained by OS
- When app issues a blocking I/O system call, request added to queue for that device
- I/O scheduler reorders queue entries to improve efficiency
- Unit 4 disk scheduling (FCFS, SSTF, SCAN, C-SCAN, LOOK etc.)
- OS might attach wait queue to device status table, managed by kernel
- Table entry: device type, address, state (request details if busy)



2. Buffering

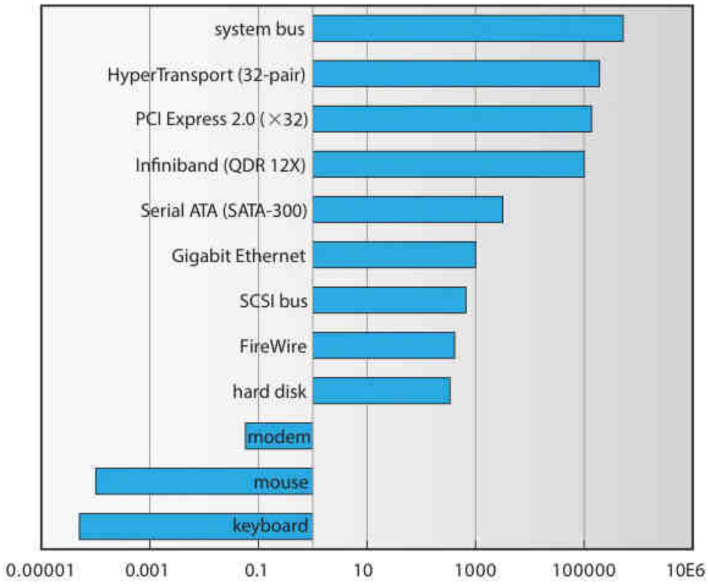
- Buffer: memory area storing data to be transferred between two devices / a device and an app

- Done for three reasons
  (i) Speed mismatch between producer & consumer (devices / apps)
  (ii) Device transfer size mismatch
  (iii) To maintain copy semantics

- Copy semantics for application I/O: version of data copied onto a disk is guaranteed to be the version of application data at the time of application system call regardless of subsequent changes
  - App data copied onto kernel buffer before control returned to app

- Double buffering: two buffers allocated so that one buffer may fill up with data while the other buffer can write onto disk
  - Decouples producer & consumer
  - Once buffer full, it writes to disk while incoming data fills other buffer

- Device transfer rates (log) for Sun Enterprise 6000



3. Caching

- Cache: region of fast memory holding copies of data

- Buffer may hold the only copy of data whereas cache always holds copy

- Sometimes same region of memory used for both buffer & cache
  - buffers in main memory for disk I/O
  - also used as cache

- Disk writes accumulated in buffer cache for several seconds

4. Spooling
   - Spool: buffer that holds output of a device that can only service one job at a time

   - Eg: printer can only print for one file at a time ; each app's output spooled to separate disk file

   - Spooling system queues spool files to printer one at a time

5. Device Reservation
   - Exclusive device access allowed
   - Process can allocate & deallocate idle devices
   - Some OSes limit number of open file handlers to one
   - Upto applications to avoid deadlock

ERROR HANDLING

- OS can usually recover from transient failures
  - disk read() fail results in read() retry
  - network send() errors result in resend()

- If permanent failure of important component occurs, OS cannot recover

- I/O system calls return one bit of information regarding status of call (success / failure)

- UNIX: integer errno returned, each corresponding to error code

- Failure of SCSI (Small Computer System Interface) device reported by SCSI protocol in three levels of detail
  - Sense key: nature of failure
  - Additional sense code: category of failure
  - Additional sense code qualifier: more detail

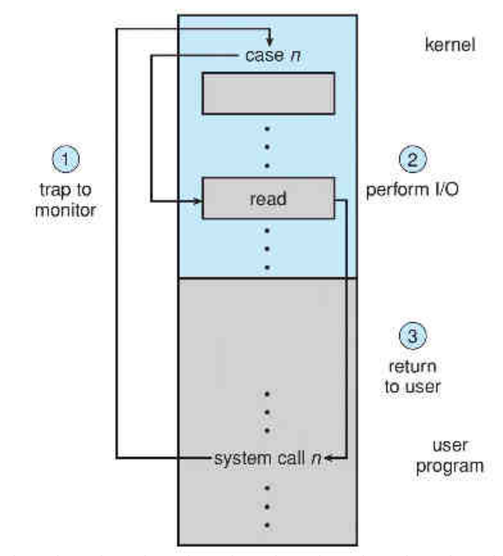
I/O PROTECTION

- To prevent illegal I/O from being performed, all I/O instructions are privileged instructions

- I/O must be done via system calls that request OS to perform I/O

- Memory-mapped and I/O mapped I/O locations must be protected from users by memory protection system

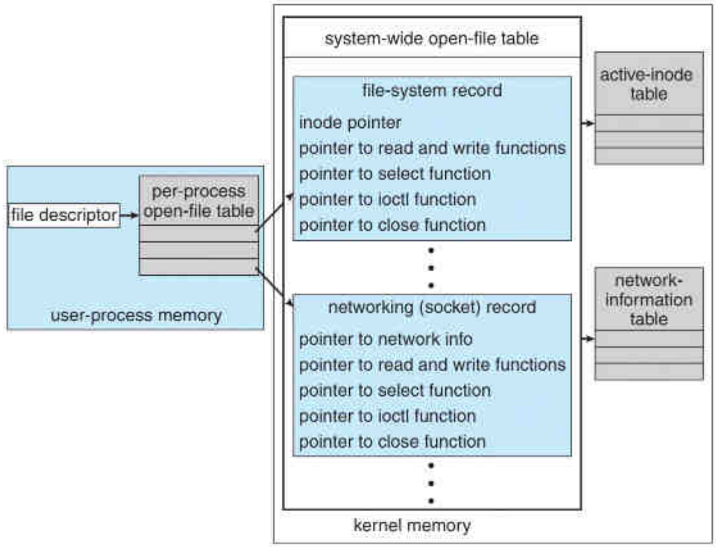
- System call for I/O



Kernel Data Structures

- Kernel keeps state information about use of I/O components (eg: open file table structure, network connection tables etc.)

- UNIX: objected oriented technique; pointers to appropriate routines for entries in open file table



- Windows: message passing technique (extensively object oriented)

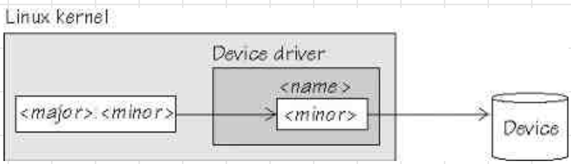
Transferring I/O Requests to Hardware Operations

- For disk access, mapping between file names and physical location on disk

- MS-DOS and Windows: first part of filename, preceding the colon, is a string identifying specific hardware device
  - Eg: C:\Users\PESU\Desktop\test.txt
  - Colon separates device namespace from file namespace

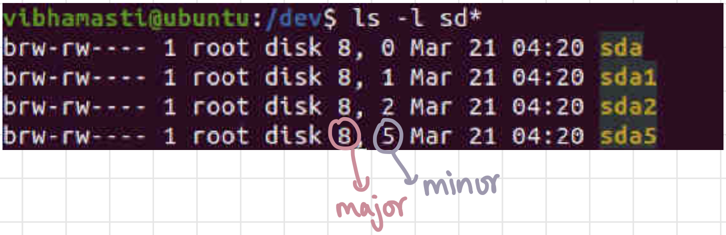
- C: represents primary hard disk, mapped to specific port address through device table

- UNIX: device names represented in regular file system namespace
  - no part of path name contains device name
  - Mount table maps prefixes of path names to device names
  - To resolve path name, OS looks for longest matching prefix on mount table
  - The name obtained from the mount table is looked up on the file system directory structures and a <major, minor> pair of numbers is returned
  - Major: number identifies device driver to handle I/O
  - Minor: passed to device driver to index into a device table to obtain address of device controller



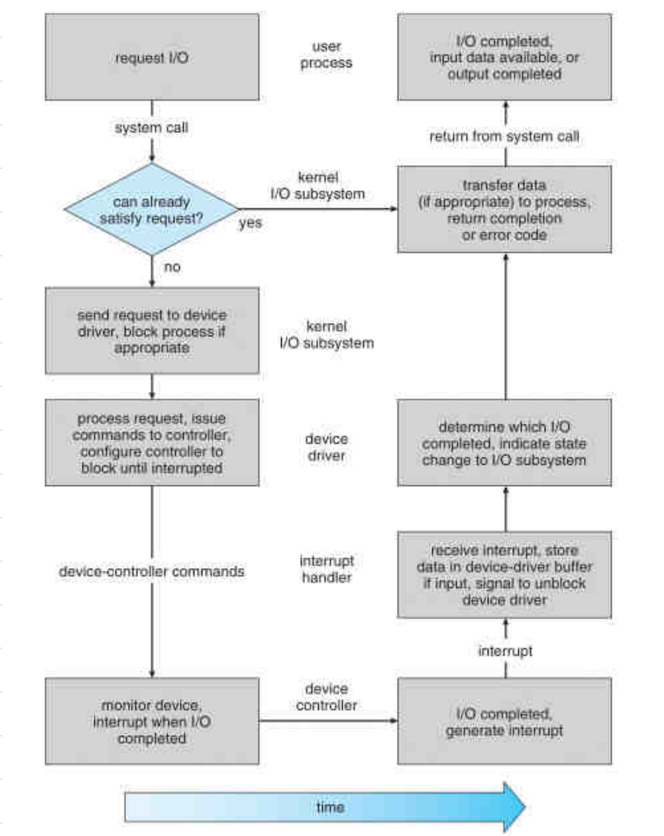
Source: IBM

- Linux: check devices by reading /proc/devices
- All devices stored in /dev directory with major & minor numbers



```
vibhamasti@ubuntu:/dev$ ls -l sd*
brw-rw---- 1 root disk 8, 0 Mar 21 04:20 sda
brw-rw---- 1 root disk 8, 1 Mar 21 04:20 sda1
brw-rw---- 1 root disk 8, 2 Mar 21 04:20 sda2
brw-rw---- 1 root disk 8, 5 Mar 21 04:20 sda5
```

LIFE CYCLE of BLOCKING READ REQUEST



1. Process calls blocking read() system call to fd of an open file

2. If data present in buffer cache, data returned to process and I/O completed

3. If not, process moved from run queue to wait queue of device and waits for I/O subsystem to send request to device driver (to perform I/O)

4. Device driver allocates kernel buffer space to receive disk data and schedules I/O ; driver writes into device control registers

5. Device controller operates device hardware to perform data transfer

6. If using DMA controller, interrupt sent to CPU when transfer is complete

7. Correct interrupt handler from IVT handles interrupt and returns from interrupt

8. Device driver receives signal, checks to see which I/O request completed and signals kernel I/O subsystem that it has completed

9. Kernel transfers data, return codes to address space of requesting process and moves process back to ready queue

10. Process unblocked and execution resumes from after system call when scheduler assigns job to CPU

PROTECTION

- Mechanism for controlling access of programs, processes & users on files, memory segments and CPU of a system

Goals of Protection

- Prevent violation of access restriction by a user
- Processes only use those system resources that they are allowed to
- Mechanisms to implement policies that guard resources

Principles of Protection

- Guiding principle: Principle of Least Privilege — programs, users & systems given minimum privileges required to perform their tasks

- Minimum damage if misused

- Implementation possibilities
  - Apps with fine-grained access controls
  - Audit trails to track protection & security activities
  - Role-based access control (RBAC) for users
  - Access control lists where access can be toggled

- Grain of access
  - Fine-grained access more secure, more tedious
  - Rough-grained privilege management easier

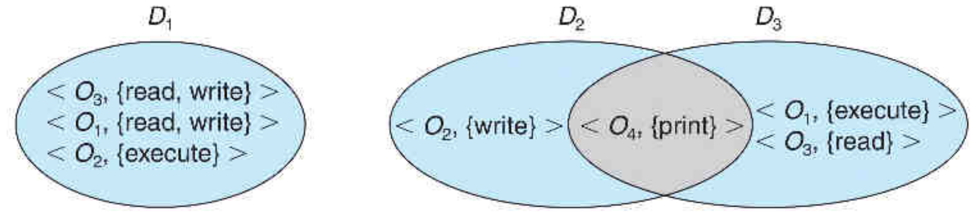
Domain of Protection

- Process must only access resources it is authorised to access as well as resources it requires to complete the task (need-to-know principle)

- If a process invokes a procedure, the procedure must only have access to its local variables and the arguments passed to it, and not the process' variables

- Every process operates within a protection domain that specifies the resources the process can access

- Each domain is a collection of access rights, which are ordered pairs of the form <object-name, rights-set>
  - Eg: domain D has access right <file F, {read, write}>



- Processes can be associated with domains statically or dynamically (more complicated than static)

- Static association may lead to violation of need-to-know principle

- Dynamic requires domain switching; domains can be realised in different ways
  1. User — domain switch when user changes
  2. Process — domain switch when process sends message to another process and waits
  3. Procedure — domain switch when procedure call made

- Standard dual mode (monitor-user mode) of OS execution
  - Monitor mode: privileged acess
  - User mode: non-privileged access
  - insufficient modes

1. Domains in UNIX

- Each user is a domain

- Each file has associated with it an owner and a setuid bit

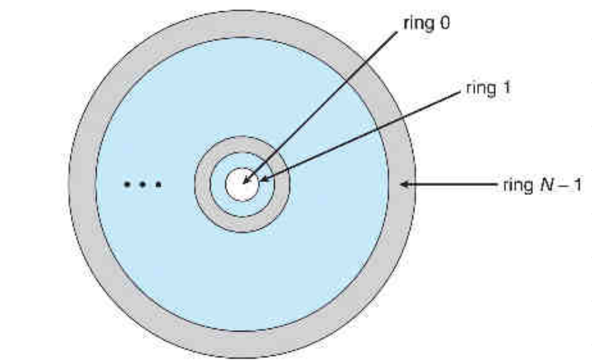
- Domain switching: temporarily switching user ID when executing a file; if setuid is set, the user temporarily changes to be the owner of the file until the process exits

- Alternate approach: place privileged programs in special directory; OS changes user when executing files here

- TOPS-20 OS: does not allow user ID change; user must send request to privileged daemon running as root

2. Domains in MULTICS

- Protection domains organised hierarchically into a ring structure



- Domains: $D_0, D_1, \ldots, D_{N-1}$ for N rings where D_0 has the most privilege and D_{N-1} has the least

- Rings numbered from 0 to 7 (N=8)

- If N=2, monitor-user mode where monitor = D_0

- MULTICS has segmented space where each segment is one file and is associated with one of the domain rings
  - Each seg also has 3 access bits for reading, writing, execution
  - Ring field of segment includes
    (i) Access bracket (b_1, b_2) such that $b_1 < b_2$
    (ii) Limit b_3 such that $b_3 > b_2$
    (iii) List of gates

- Each executing process has current-ring-number counter set to i; process can access segments associated with rings $k \geq i$ where the type of access is determined by the rwx bits

- Domain switching: calling process in different ring

- If process calls procedure/segment with access bracket (b_1, b_2), then the call is allowed if $b_1 < i < b_2$ and current ring number remains i
- Otherwise, a trap to the OS occurs
  - if $i < b_1$ then call allowed to occur as transfer to be made to ring with lower privileges, provided parameters that are passed to a lower numbered ring are copied first
  - if $i > b_2$, call only allowed if $b_3 \geq i$ and call has been directed to a designated entry point / gate
- Ring structure does not enforce need-to-know

ACCESS MATRIX

- Protection viewed abstractly as matrix where rows → domains and columns → objects
- Each entry Access (i,j) contains 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	

- Domain switching: add domains as objects (columns); switching from domain Di to Dj is allowed if switch ∈ Access (Di, Dj)

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
D_3		read	execute					
D_4	read write		read write		switch			

- Changing entries of access matrix in a controlled manner: three operations — copy, owner, control

(i) Copy

- the ability to copy access rights from one domain (row) to another for the same object is denoted by an asterisk (*) appended the access right

- Eg: (a) can be modified to (b)

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

(a)

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

(b)

- tranfer: if a right is copied from Access (i, j) to Access (k, j), it is removed from Access (i, j)

- limited copy: new copy does not get an asterisk (*) and is not copyable

(ii) Owner

- addition & removal of some rights

- if Access (i,j) contains owner as a right, then a process executing in D_i can add/remove any right in any entry of column j

- Eg: (a) can be modified to (b) — domain D_1 is the owner of F_1 and domain D_2 is the owner of F_2 & F_3 and they can add/remove rights in their respective columns (F_1 for D_1 and F_2 & F_3 for D_2)

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

(a)

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

(b)

(iii) Control

- applicable only to domain objects

- if control ∈ Access (D_i, D_j), then a process executing in D_i can remove any entry from row D_j

- Eg: (a) can be modified to (b) — process executing in D_2 can modify D_4

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	control, switch
D_3		read	execute					
D_4	read write		read write		switch			

(a)

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 control
D_3		read	execute					
D_4	write		write		switch			

(b)

- Access matrix does not solve confinement problem (preventing process from taking disallowed access)

IMPLEMENTATION of ACCESS RIGHTS

1. Global Table

- Set of ordered triples <domain, object, rights-set>
- Simplest implementation

- Whenever operation M executed on object O_j within domain D_i, the global table is searched for <D_i, O_j, R_k> where $M \in R_k$

- If triple found, operation allowed to continue. Otherwise exception raised

- Drawbacks
  - table too large to be kept in main memory
  - difficult to group object / domains (eg: access to everyone requires new entry in all domains)

2. Access Lists for Objects

- Each column of the matrix can be implemented as an access list for an object

- Each object is a list of pairs <domain, rights-set>

- Can be extended to have an entry for default set of access rights

- Whenever operation M executed on object O_j within domain D_i, access list of O_j searched for <D_i, R_k> such that $M \in R_k$

- If entry found, operation performed. Otherwise, default set is checked. If $M \in R_{default}$, performed. Else, exception raised.

3. Capability Lists for Domains

- Each row of matrix represented as list of pairs <object, rights-set> (each pair called access capability)

- Object represented by its physical name/address
- Whenever operation M executed on object O_j within domain D_i, the capability for the object is specified as a parameter
- Possession of the capability means access is allowed
- Capability list never directly accessible to processes running in that domain ; it is a protected object maintained by OS and indirectly accessed by user
- Protection of capability list is ensured in one of two ways

  (i) A tag bit is associated with each object to specify if it is capability or accessible data. The tag is not accessible by apps and is only accessed by OS (usually more than one bit for extra hardware information)

  (ii) Address space of program split into two parts — one containing normal program data & instructions (accessible to program) and one containing capability list (accessible to OS)

4. Lock-Key Mechanism

- Each object has a list of unique bit patterns called locks
- Each domain has list of unique bit patterns called keys
- Process executing in a domain can only access those objects for which the domain possesses a key (that matches the lock)
- List of keys managed by OS

comparision of implementations

- Global table — simple but large, cannot easily add special groups of objects or domains

- Access lists — correspond directly to user needs; at the time of object creation, user can specify which domains can access and what operations. However, determining set of access rights for a domain is difficult

- Capability lists — useful for localising info about a process, but do not correspond to user needs & revoking access is not simple

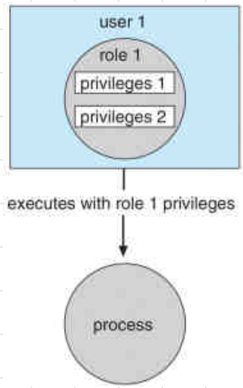
- Lock-key Mechanism — revoking is easy (changing locks), keys can be passed freely between domains

ACCESS CONTROL

- Solaris 10 — Role Based Access Control (RBAC)

- Privileges assigned to users and processes following the principle of least privilege

- Users assigned roles / can take on roles based on passwords to the roles (eg: sudo commands)



Revocation of Access Rights

- Questions about revocation

  (i) Immediate vs delayed — if delayed, when does revocation take place
  (ii) Selective vs general — does revocation affect all users or a group
  (iii) Partial vs total — all rights to an object or a subset
  (iv) Temporary vs permanent — is revocation permanent

- Revocation in lists simple; access list searched for domain and access removed from access-rights set (can be immediate, general or selective, partial or total, permanent or temporary)

- Revocation in capabilities — schemes must include the following

  1. Reacquisition
     - if process wants to use a capability that has been deleted during periodic deletion from domains, it can try to reacquire the capability
     - if access revoked, process cannot reacquire

  2. Back-pointers
     - each object contains list of pointers to capabilities
     - when revocation required, pointers can be followed and capabilities modified
     - costly implementation (MULTICS)

  3. Indirection
     - capabilities point to unique entry in global table which points to object (indirect)
     - revocation performed by deleting entry from global table
     - no selective revocation

4. Keys

- key is unique bit pattern associated with a capability
- Defined when capacity created
- Process that owns key cannot modify/ inspect it
- Technique #1 — master key
  - Each object has a master key
  - When capacity created, key is master key
  - When capability exercised, if key = master key, it is allowed.
  - Else, exception
  - Revocation: change master key with set-key
  - No selective revocation

- Technique # 2 — global table of keys
  - All keys on global table of keys
  - Capability valid if its key matches some key on the global table
  - Revocation: remove matching key from table
  - More flexible revocation

- Policy decision: who can modify object keys

SECURITY

- System is secure only if only its resources are used and accessed as intended

	SECURITY	PROTECTION
Basic	Provides the system access to legitimate users only	Controls the access to system resources
Policy	Describes which person is allowed to use the system	Specifies what files can be accessed by a particular user
Type of Threat Involved	External	Internal
Mechanism	Authentication and encryption are performed	Set or alter the authorization information

- Security violations maybe intentional (attacks from intruders or crackers) or accidental (easier to protect against)

- Threat: potential for violation; a vulnerability

Security Violations

1. Breach of Confidentiality
   - unauthorised reading / stealing of data/ information

2. Breach of Integrity
   - unauthorised modification of data

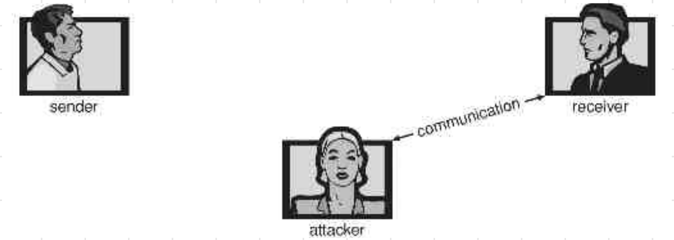
3. Breach of Availability
   - unauthorised destruction of data

4. Theft of Service
   - unauthorised use of service

5. Denial-of-Service
   - preventing legitimate usage of system (networking: SYN flooding)

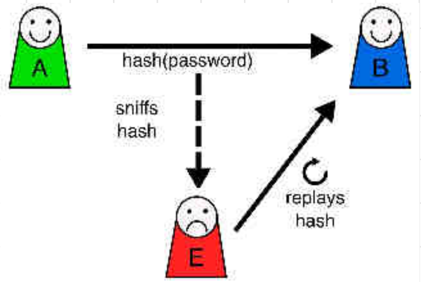
security violation methods

1. Masquerading
   - attacker prentends to be someone else
   - breach authentication and gain access



2. Replay attack

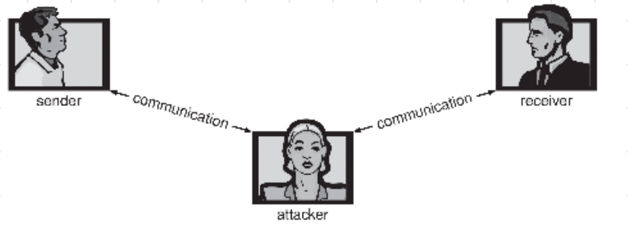
- malicious repeat of a valid data transmission
- illegally obtain information / resources
- can be done with message modification for more access



source: wikipedia

3. Man-in-the-middle attack

- attacker sits in data flow of communication
- masquerades as receiver to sender and sender to receiver
- in networking: preceded by session hijacking



SECURITY MEASURES

1. Physical Level

- data centres / computer sites must be physically secure against unauthorised entry

2. Human Level

- authorised users should have access to system
- phishing: users tricked into revealing confidential information
- dumpster diving: searching trash for sensitive data

3. OS Level

- OS must protect itself from breaches (DOS attack, stack overflow etc.)

4. Network Level

- protection from interception of data over communication links

PROGRAM THREATS

- Programs written to create security breach
- Common methods to cause security breach

1. Trojan Horse

- Code segment that misuses its environment
- Misuse ability of users to execute programs written by other users
- Search path: list of directories to search when program name given
- Variation: program emulating login prompt
- Variation: spyware — comes bundled with user-installed software
- Spyware creates ads, popups, captures user information and sends to central server
- Covert channel: attack that allows transfer of information between processes that are not allowed to communicate (eg: spam email)

2. Trap Door

- Intentional hole in the software written by the designer of a program that only they can access for their benefit
- Eg: banking program with rounding errors that credit the extra money to the attacker's account
- Compilers generating trap doors — hard to detect

3. Logic Bomb

- Malicious code intentionally inserted into a program
- Activated on host only when certain conditions met
- "Explodes" after condition, such as termination of employee, met

4. Stack and Buffer Overflow

- Exploits a bug in a program and sends excess data to the program
- Steps taken by attacker
  1. Overflow input field, command-line argument or input buffer until it writes onto stack
  2. Write exploit code with commands execute as part of the attack
  3. Overwrite current return address on stack with address of exploit code

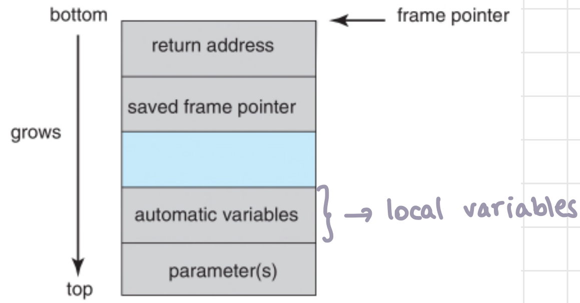
- Potential buffer overflow exploit: if argv[1] ≥ BUFFER-SIZE

```
#include <stdio.h>
#define BUFFER_SIZE 256

int main (int argc, char *argv[]) {
    char buffer[BUFFER_SIZE];

    if (argc < 2) {
        return -1;
    }
    else {
        strcpy(buffer, argv[1]);
        return 0;
    }
}
```

- Solution: use strncpy(dest, source, size)

```
strncpy(buffer, argv[1], sizeof(buffer)-1);
```

- Possible security vulnerabilities in stack overflow — stack structure known & return address can be changed

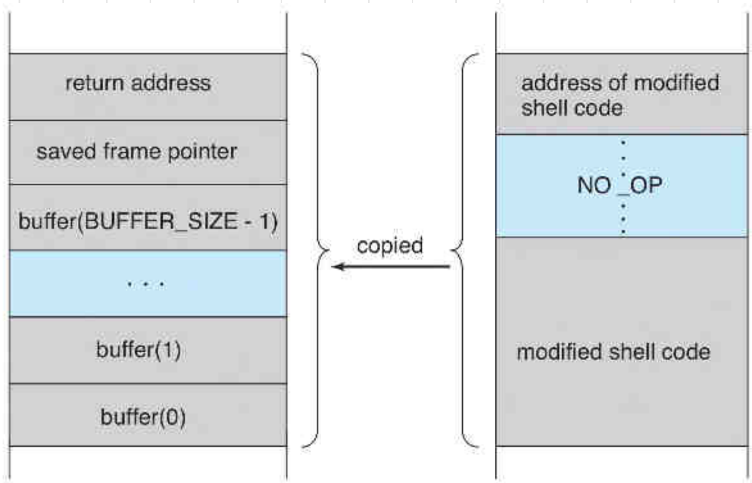


typical stack frame

- Program written by attacker that runs shell program

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    execvp(“\bin\sh”,”\bin \sh”, NULL);
    return 0;
}
```

- If user program runs with system-wide permissions, attacker code can be run maliciously



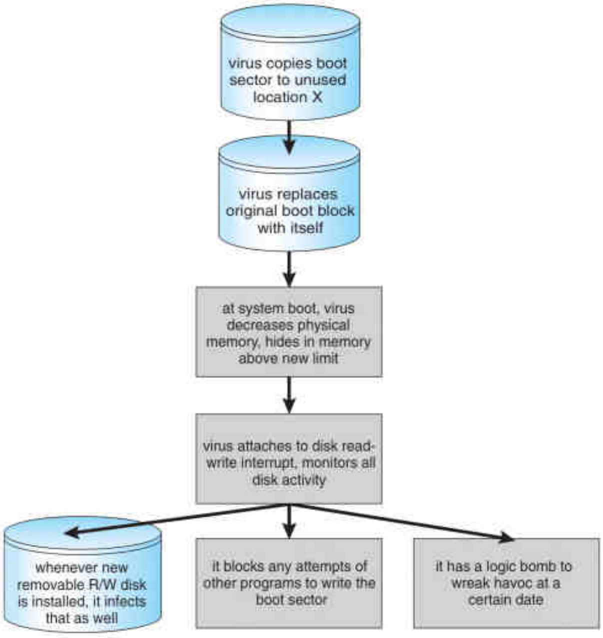
5. Viruses

- Fragment of code embedded in a legitimate program
- Self-replicating
- PCs more susceptible than UNIX-based systems
- Virus dropper (can be Trojan House) injects virus into system

- Categories of viruses

(a) File Virus
- virus appended to a file of executable code
- changes start of a program so that virus code executed
- after execution, returns control to program
- can go unnoticed

(b) Boot virus
- infects boot sector of system & executes every time system is booted, before OS loads
- do not appear in file system
- also called memory virus



(c) Macro virus
- written in high-level language
- triggered when a program (like Word, Excel) that executes macros automatically is run

(d) Source code virus
- looks for source code & includes virus in it

(e) Polymorphic virus
- changes virus signature each time it is installed
- avoid detection by antivirus

(f) Encrypted virus
- avoids detection
- virus decrypted by its decrypted code before execution

(g) Stealth Virus
- modifies parts of system that can be used to detect virus
- eg: modify read() system call to display non-infected code

(h) Tunnelling virus
- bypasses detection by installing itself in interrupt handler chain

(i) Multipartite virus
- able to infect multiple parts of system

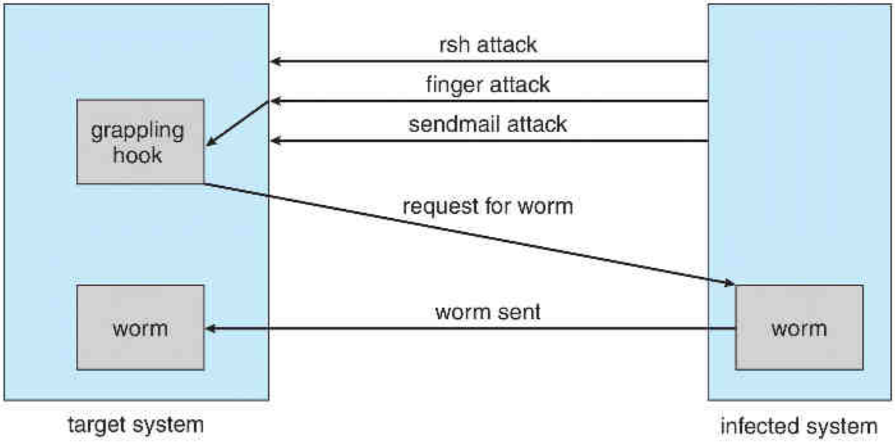
(j) Armoured virus
- coded to make it hard to understand by antivirus

System & Network THREATS

- To reduce threat, system's attack surface to be reduced
- OSes strive to be secure by default, where most services must be explicitly enabled by users

1. WORMS

- Program that duplicates itself
- 1988, Morris worm made up of grappling hook (also called vector or bootstrap) program and the main program.
- Grappling hook connected to origin machine and copied main program onto hooked system
- Exploited rsh (remote access), finger (telephone directory) and sendmail (sends, receives, routes email) programs



- Utilised password guessing to break into multiple user accounts

- 2003, Sobig worm — spread via email and attacked Windows systems

2. PORT SCANNING

- Not attack; means for cracker to detect a system's vulnerabilities to attack

- Automated attempt to connect to TCP/IP ports on a range of IP addresses

- nmap can determine OS, running programs etc

- Port scans are detectable; run from zombie systems — systems that have already been compromised by hackers & used as a remote host by them

3. DENIAL-OF-SERVICE

- Disrupting legitimate use of a service

- Eg: SYN-flooding for initiation of sockets for TCP connection with fake source IP addresses

- Distributed DOS attacks even harder to detect & use zombies as multiple sources

- Authentication blocking by multiple wrong passwords