File System interface

Juan Felipe Medina Lee, Ph.D.

Computer Science and Engineering
University of Puerto Rico
Mayagüez Campus

Outline

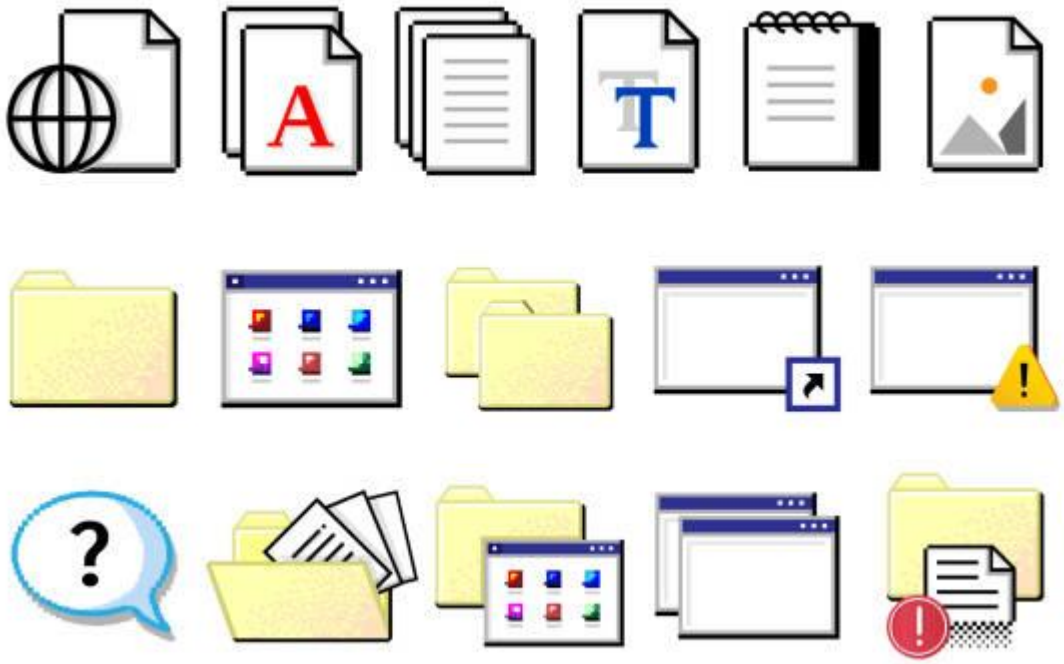
- File Concept
- Access Methods
- Disk and Directory Structure
- Protection
- Memory mapped files

File Concept

- It’s the standard way for the OS to write data on disk

- Types:
  - Data
    - Numeric
    - Character
    - Binary
  - Program

- Contents defined by file’s creator
  - Many types
    - text file,
    - music file,
    - executable file…



File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files is kept in the directory structure, which is maintained on the disk
- Information is kept in the directory structure

File info Window on Mac OS X

File Operations

- Create
- Write – at **write pointer** location
- Read – at **read pointer** location
- Reposition within file - **seek**
- Delete
- Truncate
- *Open* (F_i) – search the directory structure on disk for entry F_i, and move the content of entry to memory
- *Close* (F_i) – move the content of entry F_i in memory to directory structure on disk

Open Files

Several pieces of data are needed to manage open files:

- **Open-file table:** Tracks open files
- **File pointer:** pointer to last read/write location per process that has the file open
- **File-open count:** counter of the number of times a file is open – to allow removal of data from the open-file table when the last process closes it
- **Disk location of the file:** cache of data access information
- **Access rights:** per-process access mode information

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File Types – Name, Extension

Access Methods

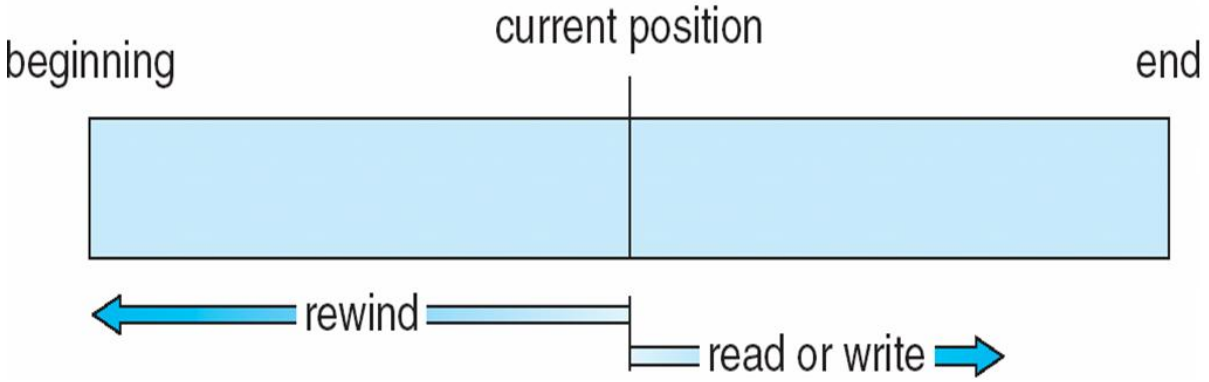
A file is a fixed length of **logical records**

- Sequential Access
- Direct Access
- Other Access Methods

Sequential Access

Operations

- **`read next`**
- **`write next`**
- **`Reset`**
- no read after last write (rewrite)

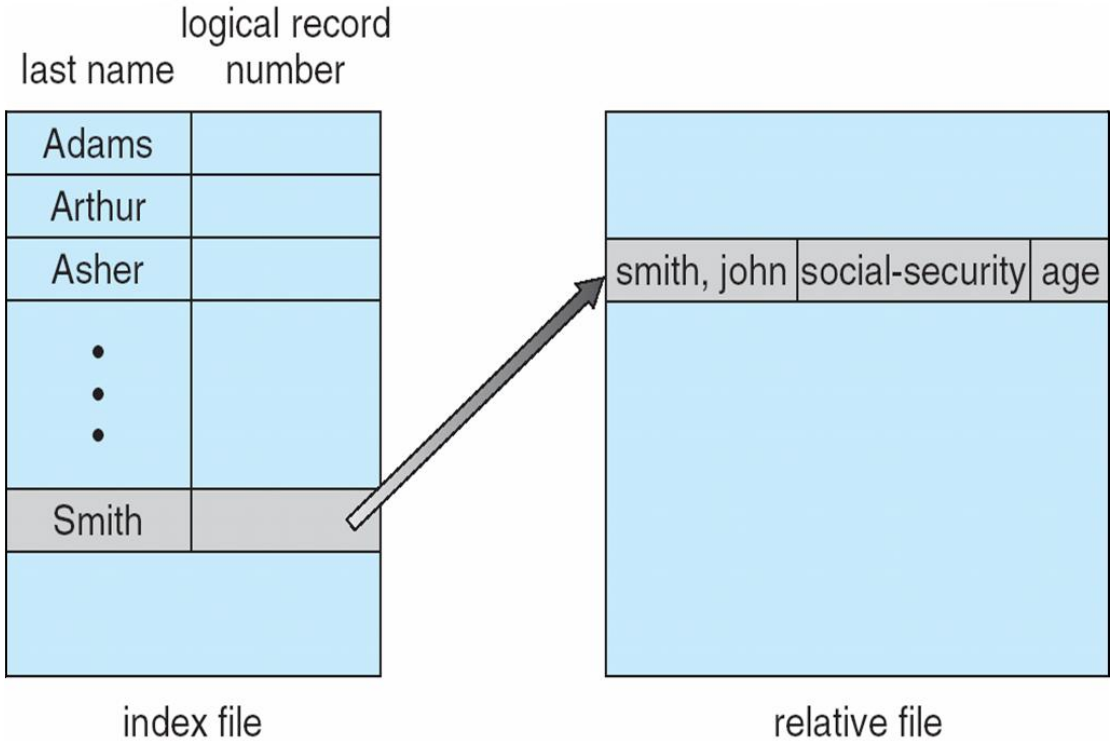


Direct Access

- **Approach 1:**
  - `Read(n)`
  - `Write(n)`
- **Approach 2:**
  - `position(n-1)`
  - `Read(next)`
  - `Write(next)`
- The file is accessed using consecutive relative block numbers, starting from 0.
- These are not necessarily consecutive in the disk.

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Simulation of Sequential Access on Direct-access File

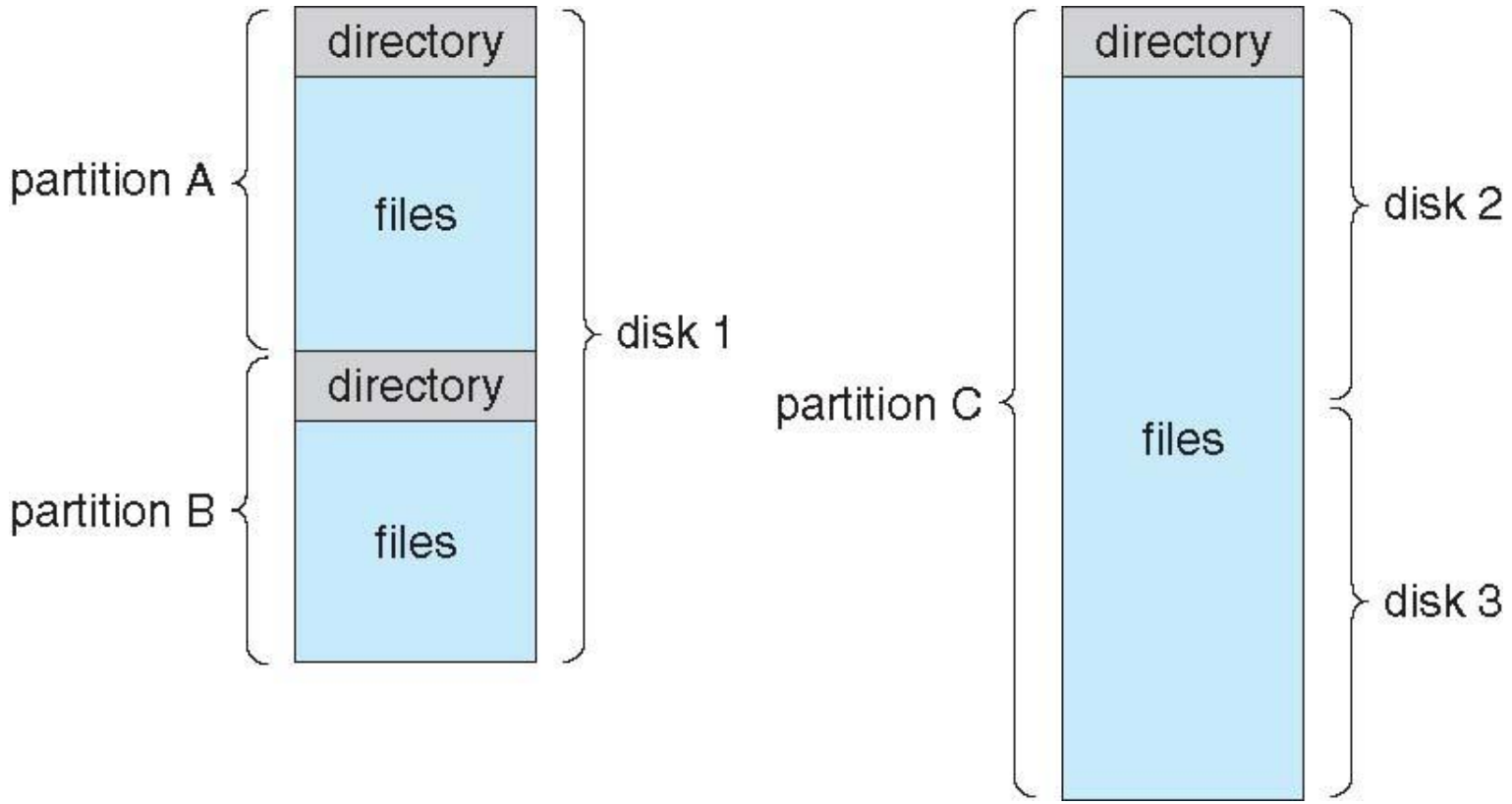


Example of Index and Relative Files

Disk Structure

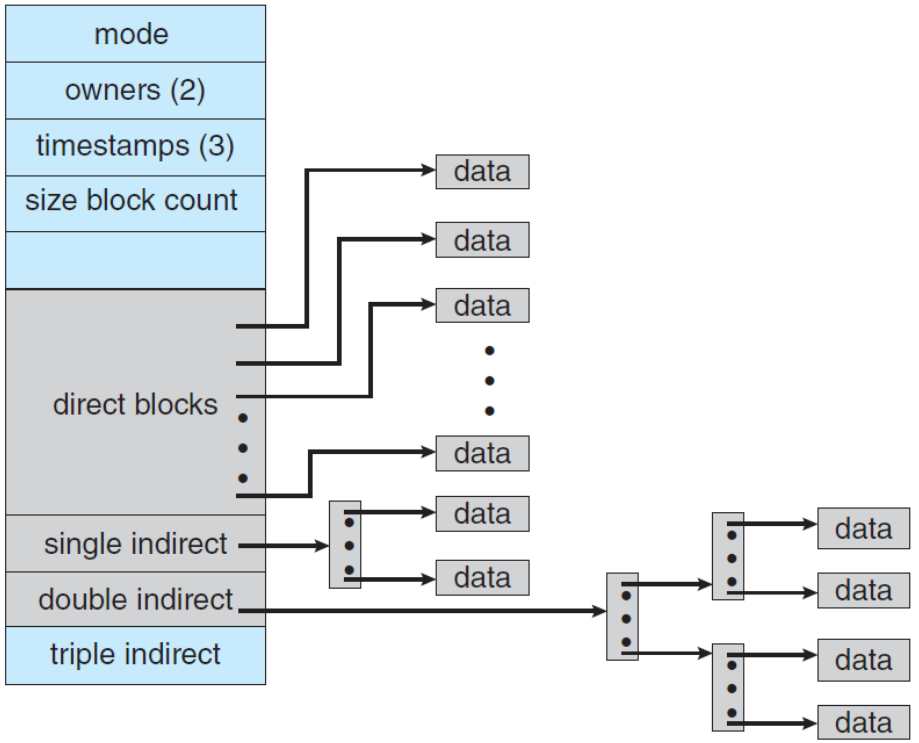
- The disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- A disk or partition can be used **raw** – without a file system, or **formatted** with a file system
  - Partitions are also known as minidisks or slices
- Entity-containing file system is known as a **volume**
- Each volume containing a file system also tracks that file system's info in the **device directory** or **volume table of contents**
- In addition to **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

A Typical File-system Organization



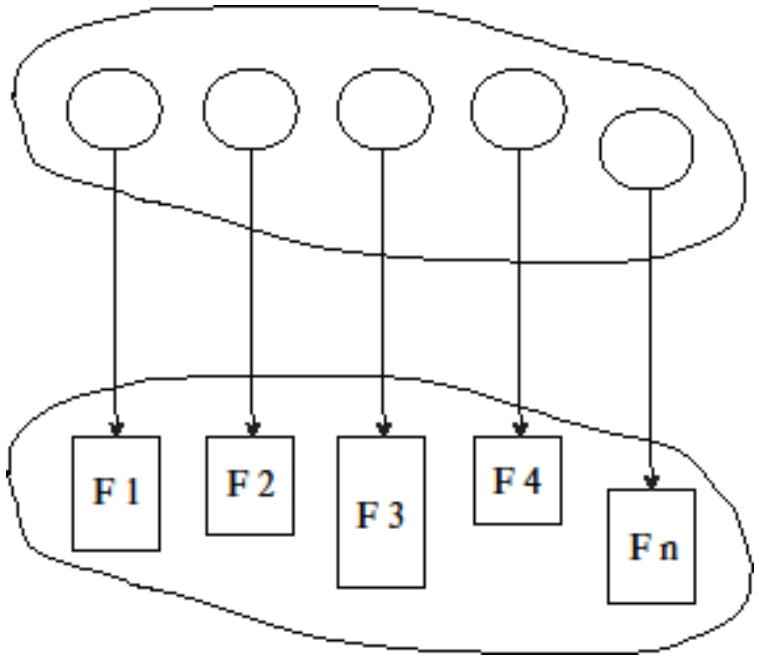
Unix Inode

The inode (index node) is a data structure in a Unix-style file system that describes a file-system object such as a file or a directory



Directory Structure

A collection of nodes containing information about all files



Both the directory structure and the files reside on disk

Operations Performed on Directory

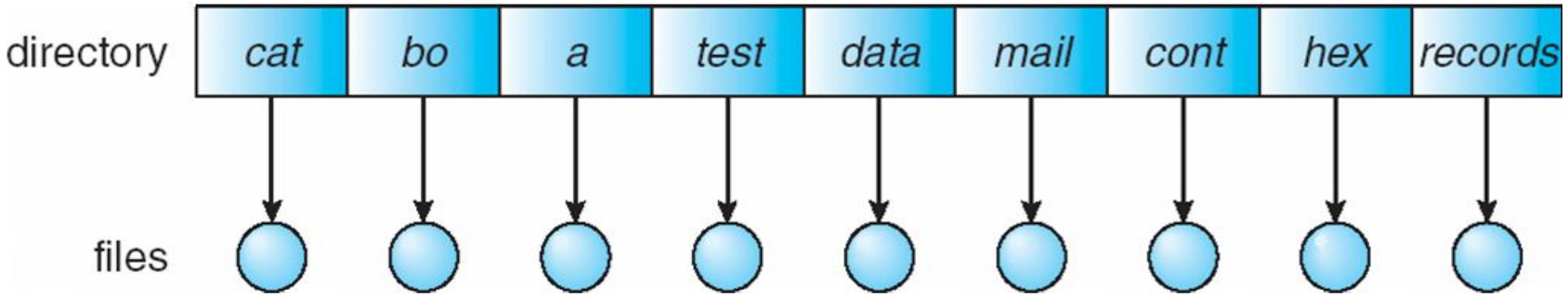
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Directory Organization

- **Efficiency** – locating a file quickly
- **Naming** – convenient to users
  - Two users can have the same name for different files
  - The same file can have several different names
- **Grouping** – logical grouping of files by properties, (e.g., all Java programs, all games,…)

Single-Level Directory

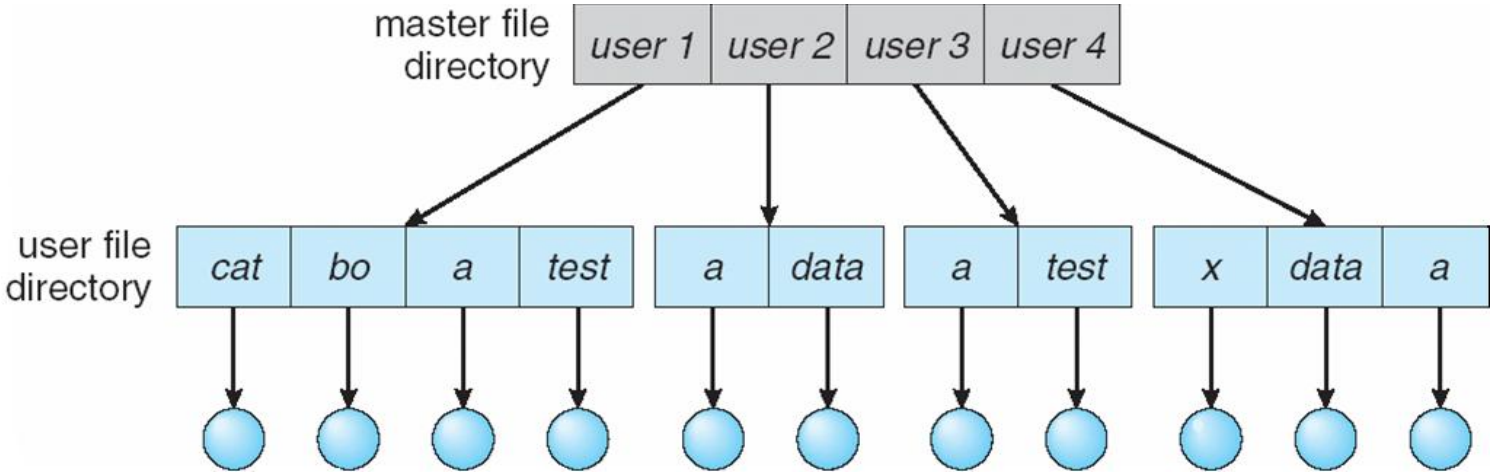
A single directory for all users



- Naming problem
- Grouping problem

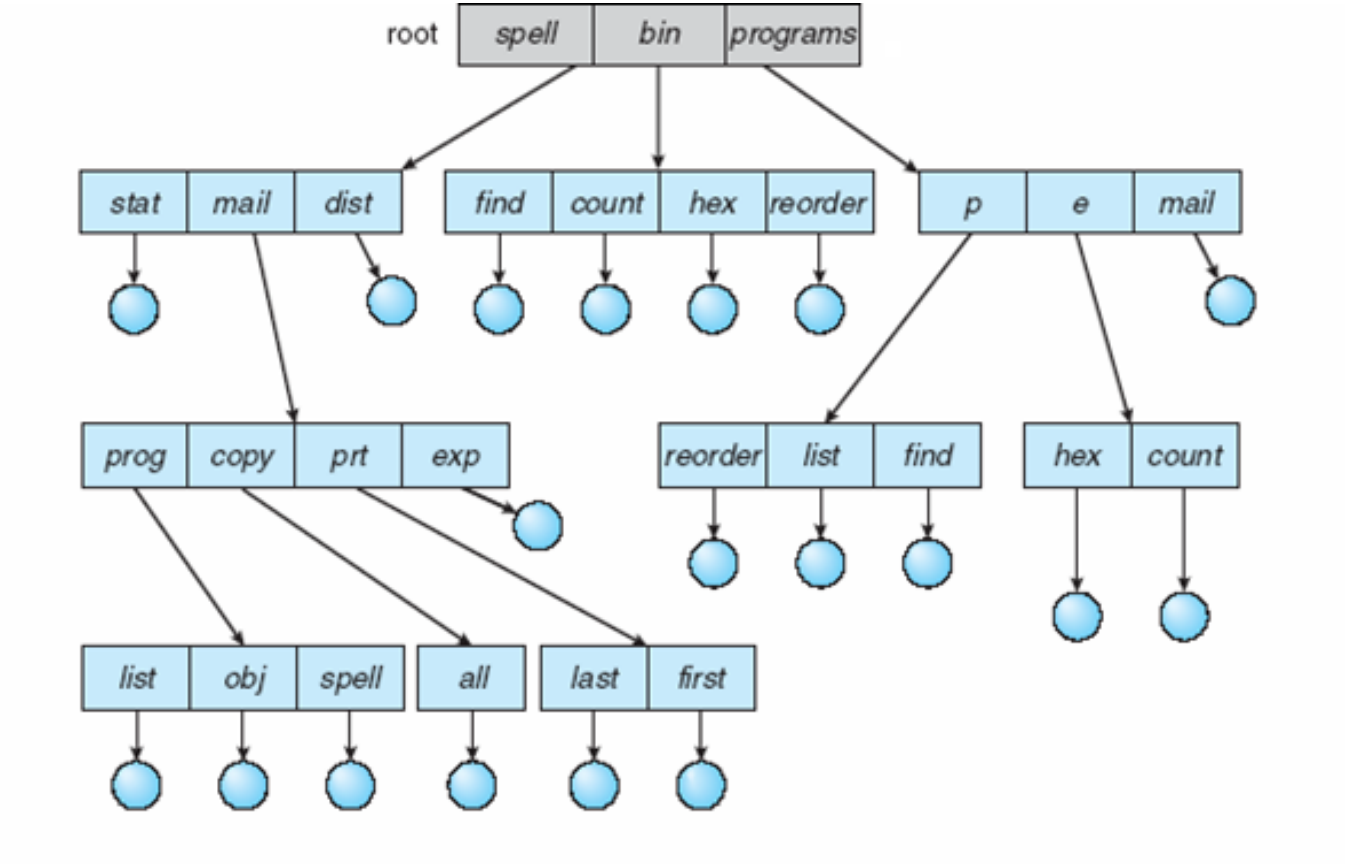
Two-Level Directory

Separate directory for each user



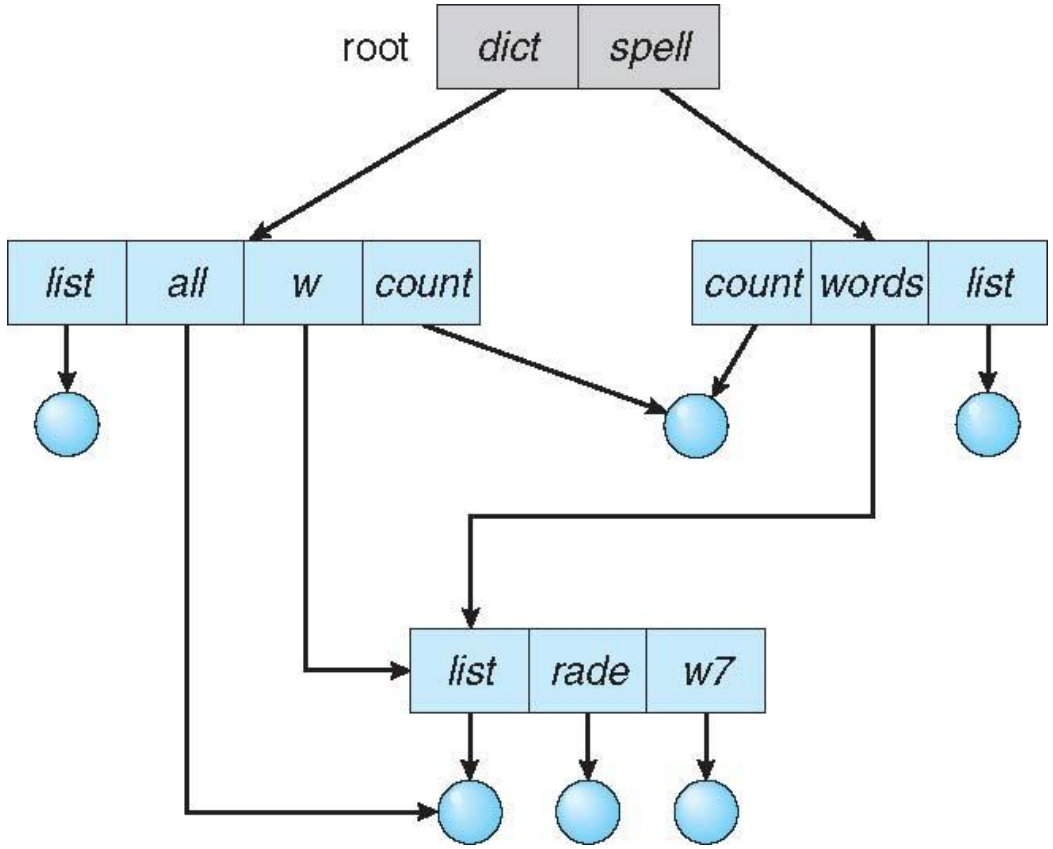
- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories

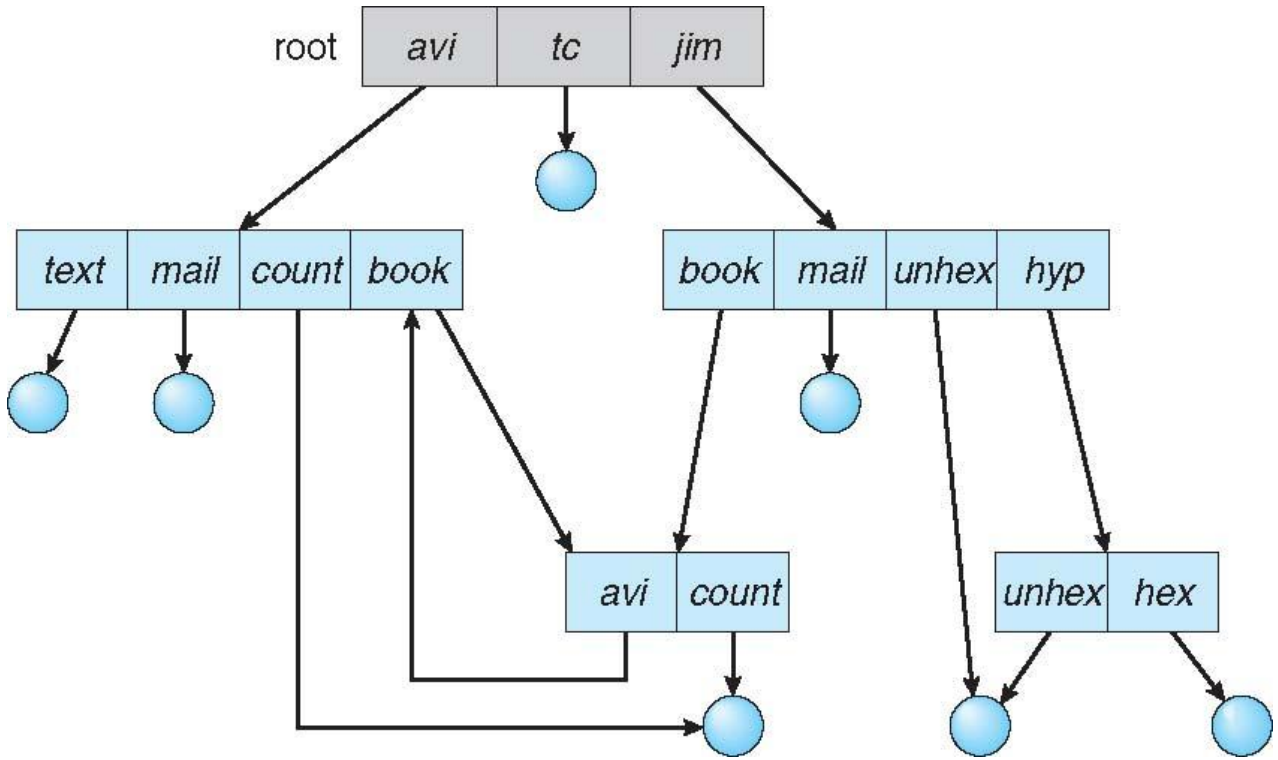


Acyclic-Graph Directories

Have shared subdirectories and files



General Graph Directory



General Graph Directory (Cont.)

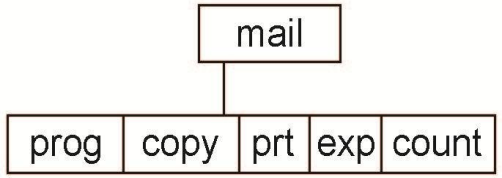
How do we guarantee no cycles?

- Allow only links to files not subdirectories
- **Implementing cycle-safe algorithms**
- Every time a new link is added use a cycle detection algorithm to determine whether it is OK

Current Directory

- Can designate one of the directories as the current (working) directory
  - **`cd /spell/mail/prog`**
- Creating and deleting a file is done in current directory
- Example of creating a new file
  - If the current directory is **`/mail`**
  - The command

    mkdir <dir-name>
  - Results in:



  - Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”

Protection

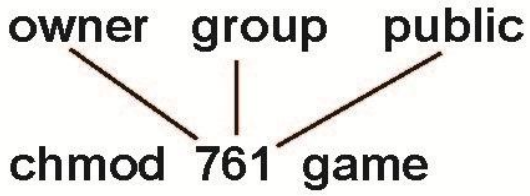
- File owner/creator should be able to control:
  - What can be done
  - By whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups in Unix

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a file (say *game*) or subdirectory, define an appropriate access.

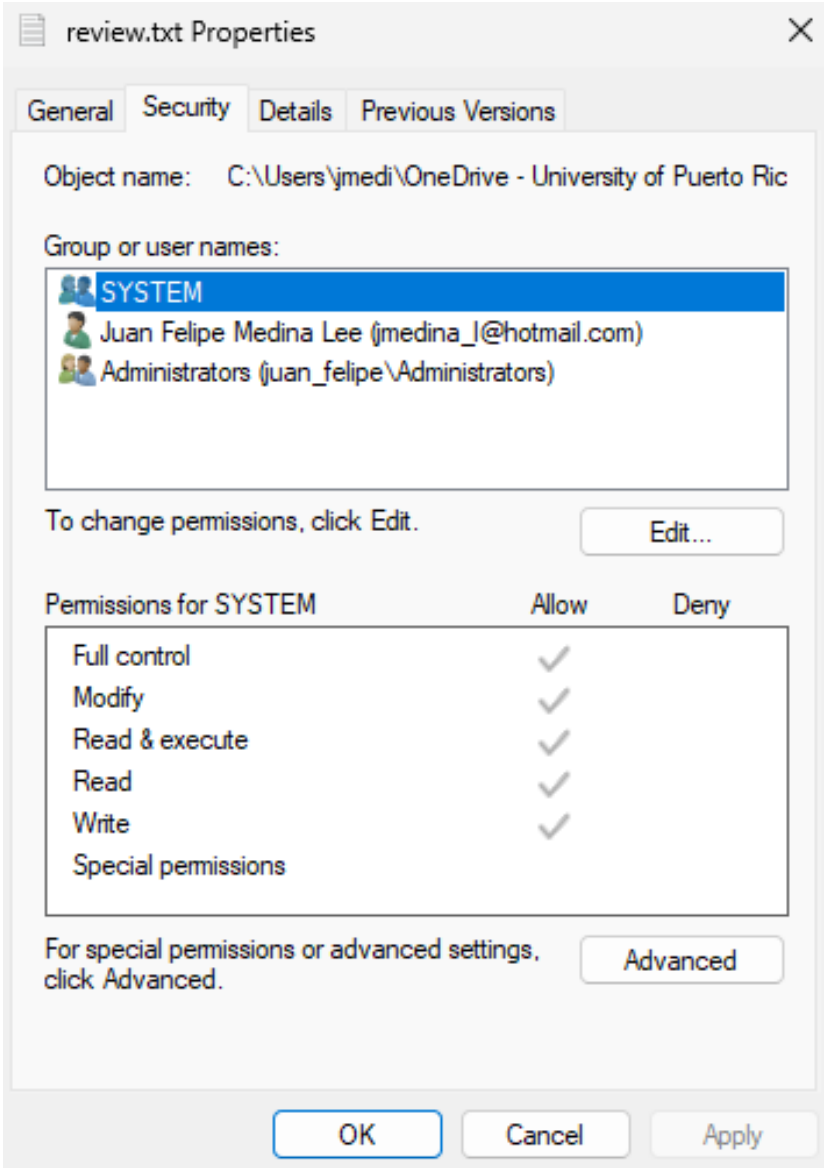


Attach a group to a file

```
chgrp      G      game
```

```
-rw-rw-r--    1 pbg    staff      31200  Sep 3 08:30     intro.ps
drwx------    5 pbg    staff        512  Jul 8 09.33     private/
drwxrwxr-x    2 pbg    staff        512  Jul 8 09:35     doc/
drwxrwx---    2 pbg    student      512  Aug 3 14:13     student-proj
-rw-r--r--    1 pbg    staff       9423  Feb 24 2003     program.c
-rwxr-xr-x    1 pbg    staff      20471  Feb 24 2003     program
drwx--x--x    4 pbg    faculty      512  Jul 31 10:31    lib/
drwx------    3 pbg    staff       1024  Aug 29 06:52    mail/
drwxrwxrwx    3 pbg    staff        512  Jul 8 09:35     test/
```

A Sample UNIX Directory Listing

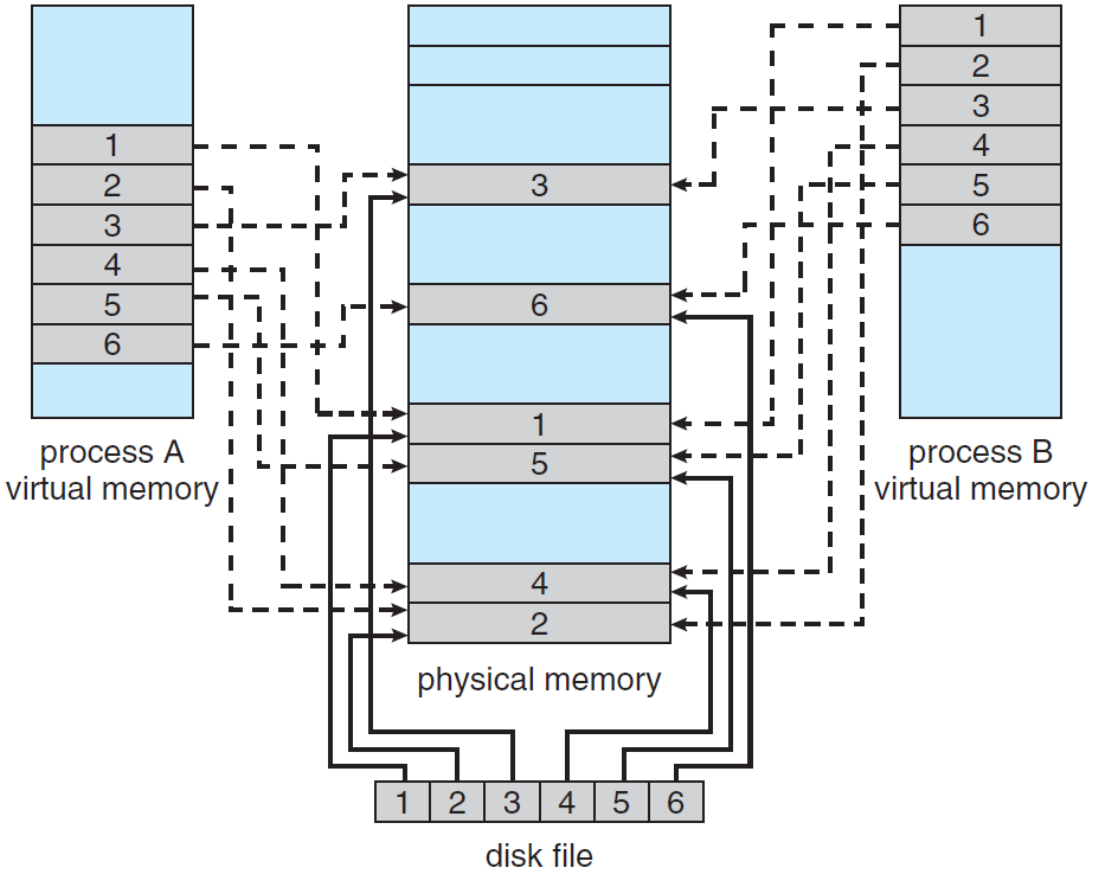


Windows 11 Access-Control List Management

Memory-Mapped files

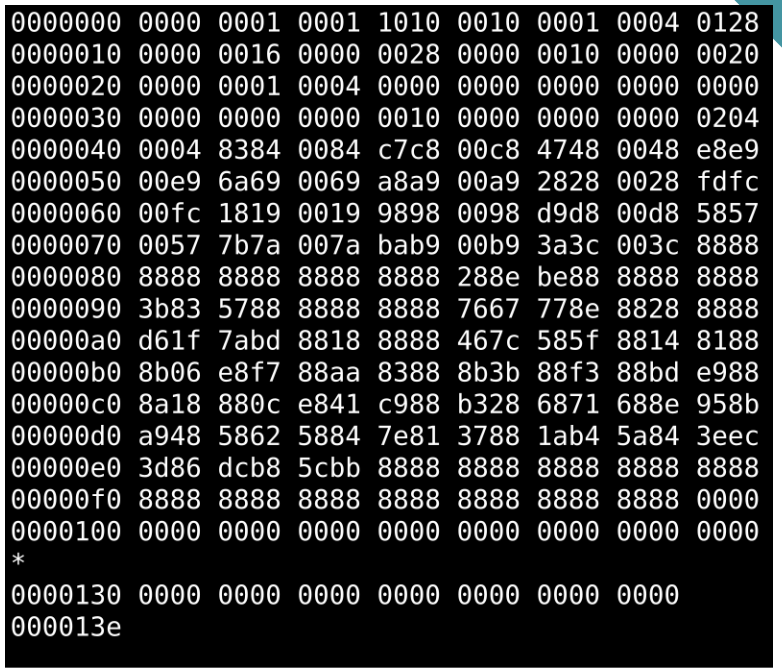
- Memory mapping a file is accomplished by mapping a disk block to a page (or pages) in memory
- Subsequent **reads** and **writes** to the file are handled as **routine memory accesses.**
- Manipulating files through memory rather than `read()` and `write()` **simplifies and speeds up file access and usage.**
- **Files can be shared by multiple processes using this technique.**
- In Linux, this operation can be done using the function `mmap()`

Memory-Mapped files



Binary files

- Store data in its raw, binary format (1s and 0s),
  - **Less space compared to text files**
- Written and read directly in their native format: no need for conversion between human-readable strings and machine-readable numbers.
  - **This makes binary file I/O faster compared to text files.**
- Binary files preserve the exact representation of data, such as floating-point numbers or complex data structures.
  - **Text files may introduce rounding errors or loss of precision during conversion.**



Thank you

Juan F. Medina

Juan.medina26@upr.edu

I/O Systems

Juan Felipe Medina Lee, Ph.D.

Computer Science and Engineering
University of Puerto Rico Mayagüez Campus

Overview

- I/O management is a major component of operating system design and operation
  - Important aspect of computer operation
  - I/O devices vary greatly
    - Various methods to control them
  - Performance management
  - New types of devices frequent
- Ports, buses, and device controllers connect to various devices
- **Device drivers** encapsulate device details
  - Present uniform device-access interface to the I/O subsystem

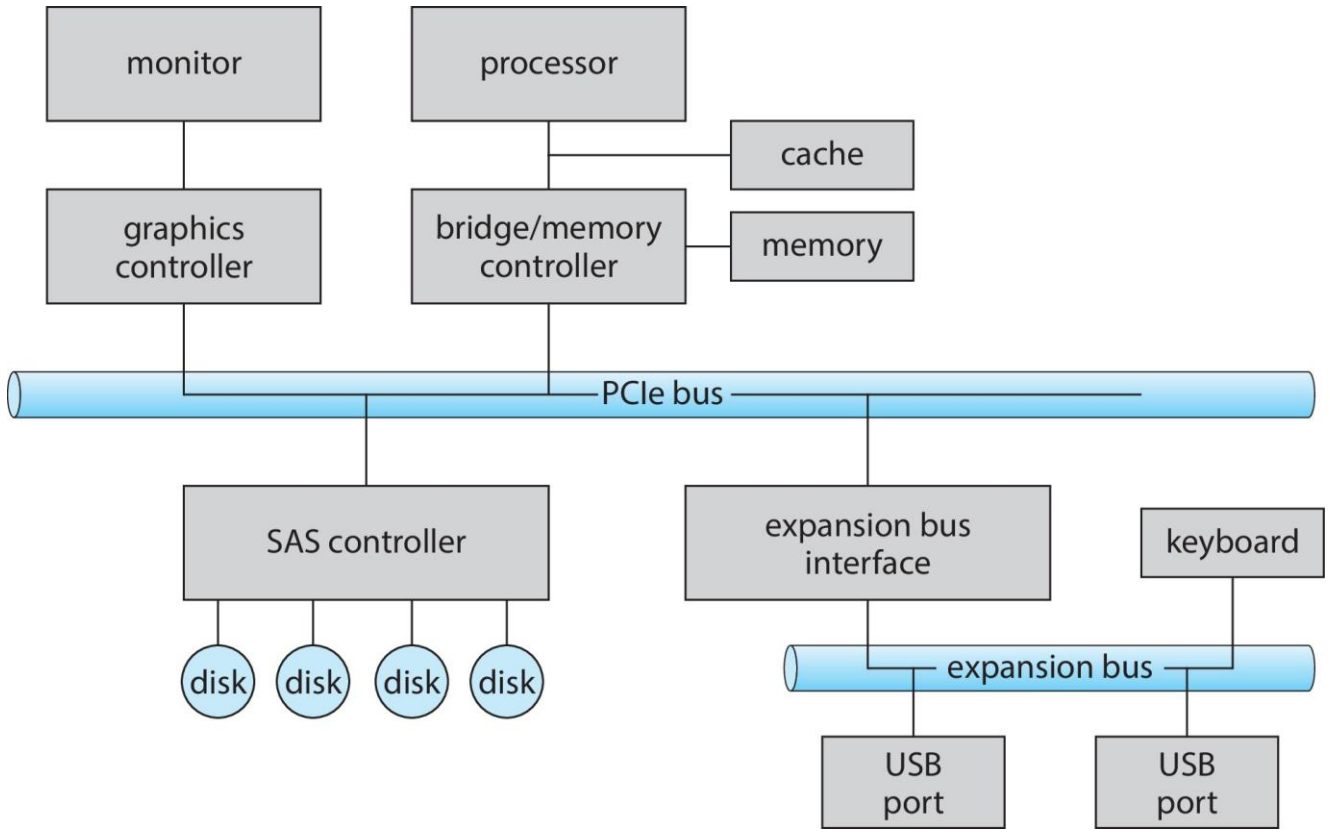
I/O Hardware

- Incredible variety of I/O devices
  - Storage
  - Transmission
  - Human-interface
- Common concepts – signals from I/O devices interface with computer
  - **Port** – connection point for device
  - **Bus** - **daisy chain** or shared direct access
    - **PCI** bus common in PCs and servers, PCI Express (**PCIe**)
    - **expansion bus** connects relatively slow devices
    - **Serial-attached SCSI** (**SAS**) common disk interface

I/O Hardware (Cont.)

- **Controller** (**host adapter**) – electronics that operate port, bus, device
  - Sometimes integrated
  - Sometimes separate circuit board (host adapter)
  - Contains processor, microcode, private memory, bus controller, etc.
    - Some talk to per-device controller with bus controller, microcode, memory, etc.

A Typical PC Bus Structure

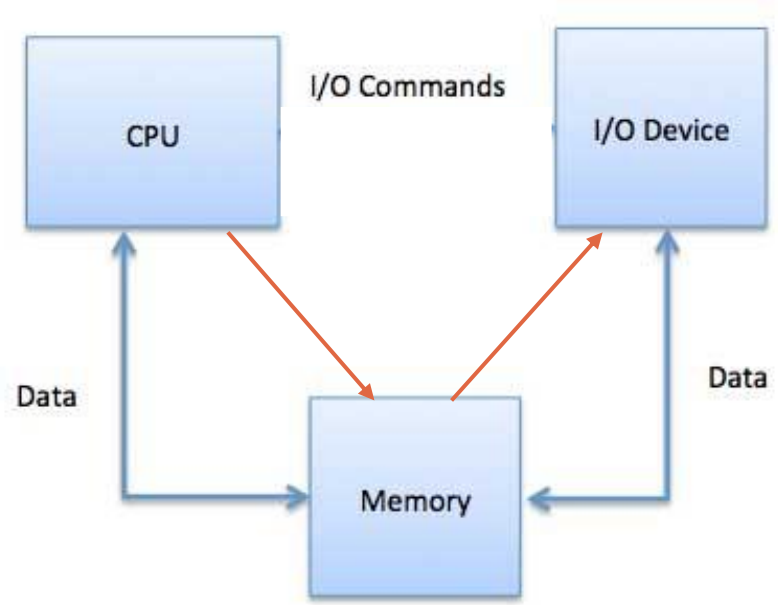


I/O Hardware (Cont.)

- **Fibre channel** (**FC**) is complex controller, usually separate circuit board (**host-bus adapter**, **HBA**) plugging into bus
- I/O instructions control devices
- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - **Data-in register, data-out register, status register, control register**
  - Typically 1-4 bytes, or FIFO buffer

I/O Hardware (Cont.)

- Devices have addresses, used by
  - Direct I/O instructions
  - **Memory-mapped I/O**
    - Device data and command registers mapped to processor address space
    - Especially for large address spaces (graphics)



Polling

For each byte of I/O

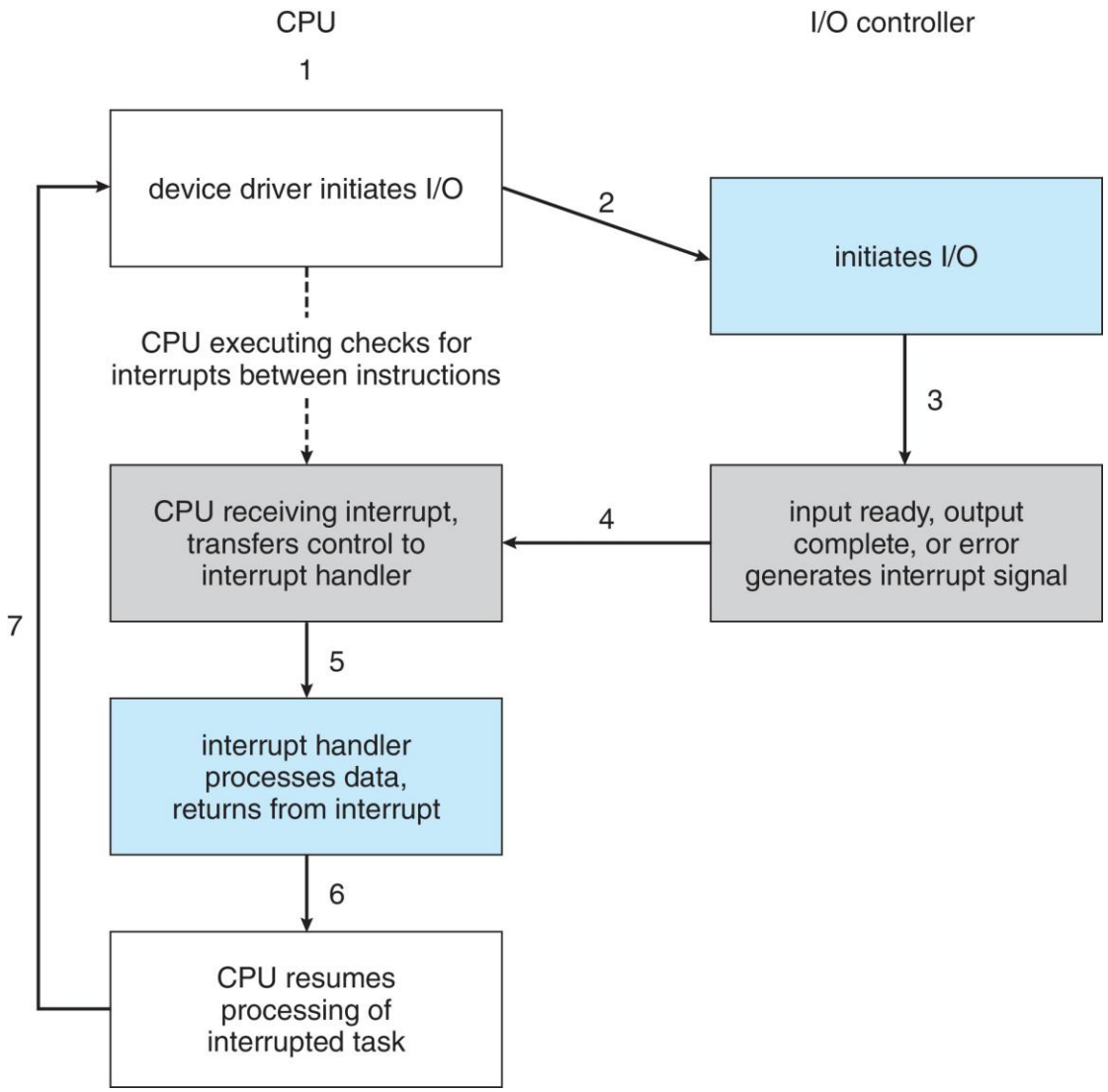
1. Read the busy bit from the status register until 0
2. Host sets read or write bit and if write copies data into data-out register
3. Host sets command-ready bit
4. Controller sets the busy bit, executes a transfer
5. Controller clears the busy bit, error bit, and command-ready bit when the transfer is done

Step 1 is **busy wait** cycle to wait for I/O from the device

- Reasonable if the device is fast
- But inefficient if the device slow
- **Does the CPU switch to other tasks?**
    - But if miss a cycle is missed, the data is overwritten/lost

Interrupts

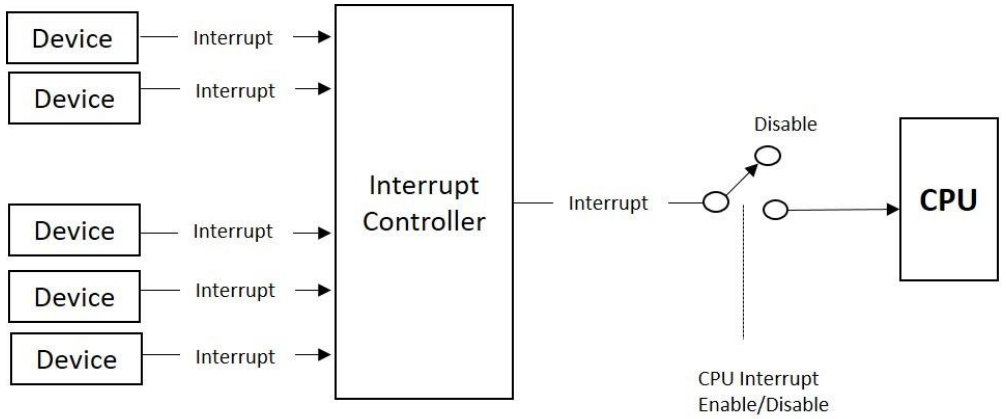
- Polling can happen in 3 instruction cycles
  - Read status, logical-and to extract status bit, branch if not zero
  - How to be more efficient if non-zero infrequently?
- CPU **Interrupt-request line** triggered by I/O device
  - Checked by processor after each instruction
- **Interrupt handler** receives interrupts
  - **Maskable** to ignore or delay some interrupts
- **Interrupt vector** to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some **nonmaskable**
  - Interrupt chaining if more than one device at same interrupt number



Interrupt-Driven I/O Cycle

Interrupts (Cont.)

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



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

Intel Pentium Processor Event-Vector Table

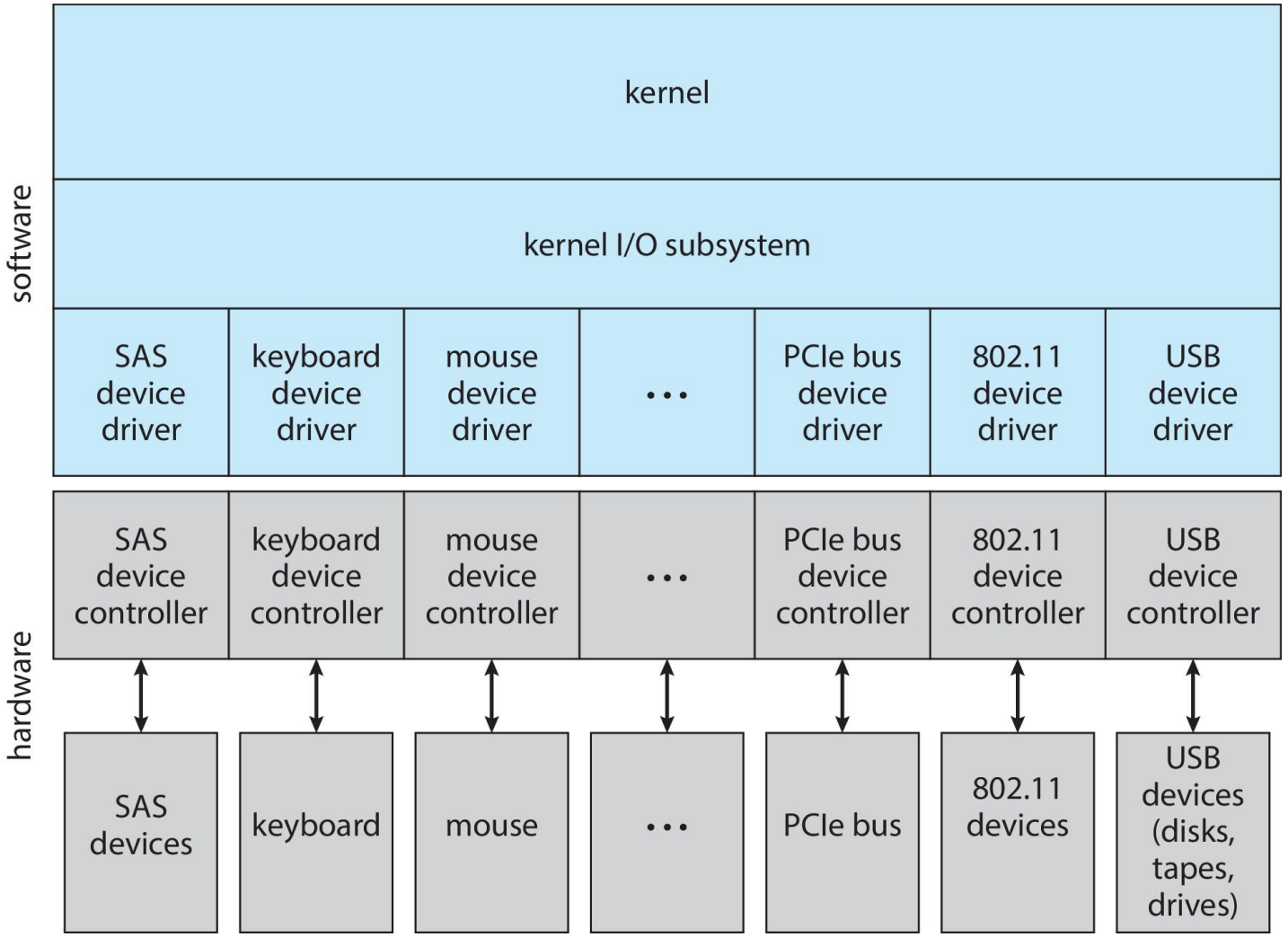
Latency

- Stressing interrupt management because even single-user systems manage hundreds of interrupts per second and servers hundreds of thousands
- For example, a quiet ubuntu desktop generated ~7,000 interrupts over 10 seconds

```
procs -----------memory---------- ---swap-- -----io---- -system-- -------cpu-------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st gu
 2  0      0 6865472 195348 2389164    0    0    51    68 1015    3  3  3 94  0  0  0
 0  0      0 6865472 195348 2389164    0    0     0     0  235  339  1  1 99  0  0  0
 0  0      0 6865472 195352 2389164    0    0     0    56  543  838  0  1 99  0  0  0
 0  0      0 6865472 195352 2389164    0    0     0     0  666 1119  1  1 99  0  0  0
 0  0      0 6865472 195352 2389164    0    0     0     0  550  934  1  1 99  0  0  0
 0  0      0 6865472 195352 2389164    0    0     0     0 1038 1795  1  1 98  0  0  0
 0  0      0 6865472 195352 2389164    0    0     0     0  630 1328  1  1 98  0  0  0
 0  0      0 6865472 195352 2389164    0    0     0    40  746 1426  1  1 98  0  0  0
 0  0      0 6865472 195352 2389164    0    0     0     0  369  608  0  1 99  0  0  0
 0  0      0 6865472 195352 2389164    0    0     0     0  623 1319  0  1 98  0  0  0
```

Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks
- Devices vary in many dimensions
  - **Character-stream** or **block**
  - **Sequential** or **random-access**
  - **Synchronous** or **asynchronous** (or both)
  - **Sharable** or **dedicated**
  - **Speed of operation**
  - **read-write**, **read only**, or **write only**



A Kernel I/O Structure

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Characteristics of I/O Devices

Characteristics of I/O Devices (Cont.)

Subtleties of devices handled by device drivers

Broadly I/O devices can be grouped by the OS into

- Block I/O
- Character I/O (Stream)
- Memory-mapped file access
- Network sockets

For direct manipulation of I/O device specific characteristics, usually an escape / back door

- Unix `ioctl()` call to send arbitrary bits to a device control register and data to device data register

Block and Character Devices

Block devices include disk drives

- Commands include read, write, seek
- **Raw I/O**, **direct I/O**, or file-system access
- Memory-mapped file access possible
  - File mapped to virtual memory and clusters brought via demand paging
- DMA

Character devices include keyboards, mice, serial ports

- Commands include `get(), put()`
- Libraries layered on top allow line editing

Network Devices

- Varying enough from block and character to have their own interface
- Linux, Unix, Windows, and many others include **socket** interface
  - Separates network protocol from network operation
  - Includes `select()` functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

Clocks and Timers

- Provide current time, elapsed time, timer
- Normal resolution about 1/60 second
- Some systems provide higher-resolution timers
- **Programmable interval timer** used for timings, periodic interrupts

Nonblocking and Asynchronous I/O

Blocking - process suspended until I/O completed

- Easy to use and understand
- Insufficient for some needs

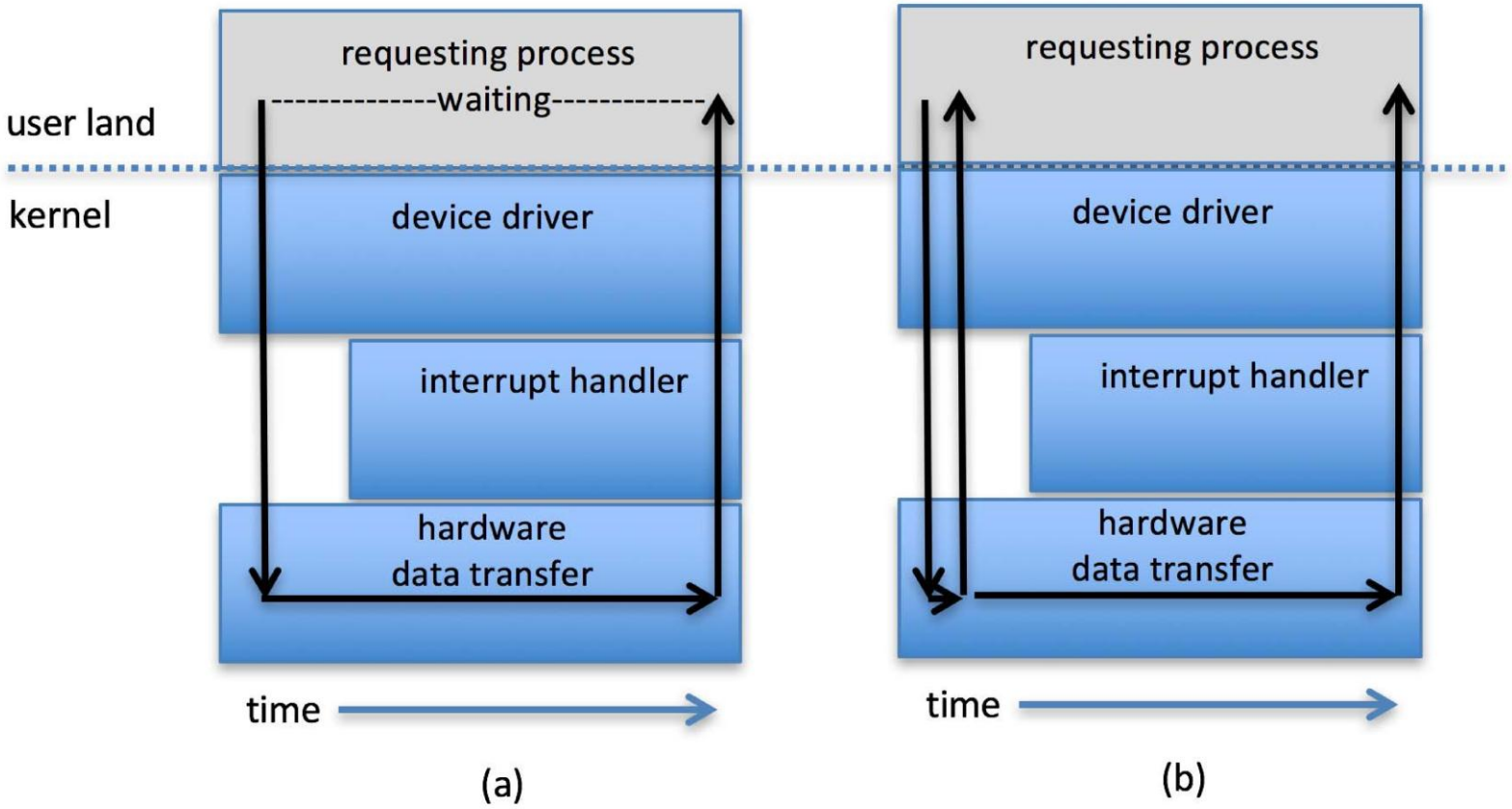
Nonblocking - I/O call returns as much as available

- User interface, data copy (buffered I/O)
- Implemented via multi-threading
- Returns quickly with count of bytes read or written
- **`select()`** to find if data ready then **`read()`** or **`write()`** to transfer

Asynchronous - process runs while I/O executes

- Difficult to use
- I/O subsystem signals process when I/O completed

Two I/O Methods



Kernel I/O Subsystem

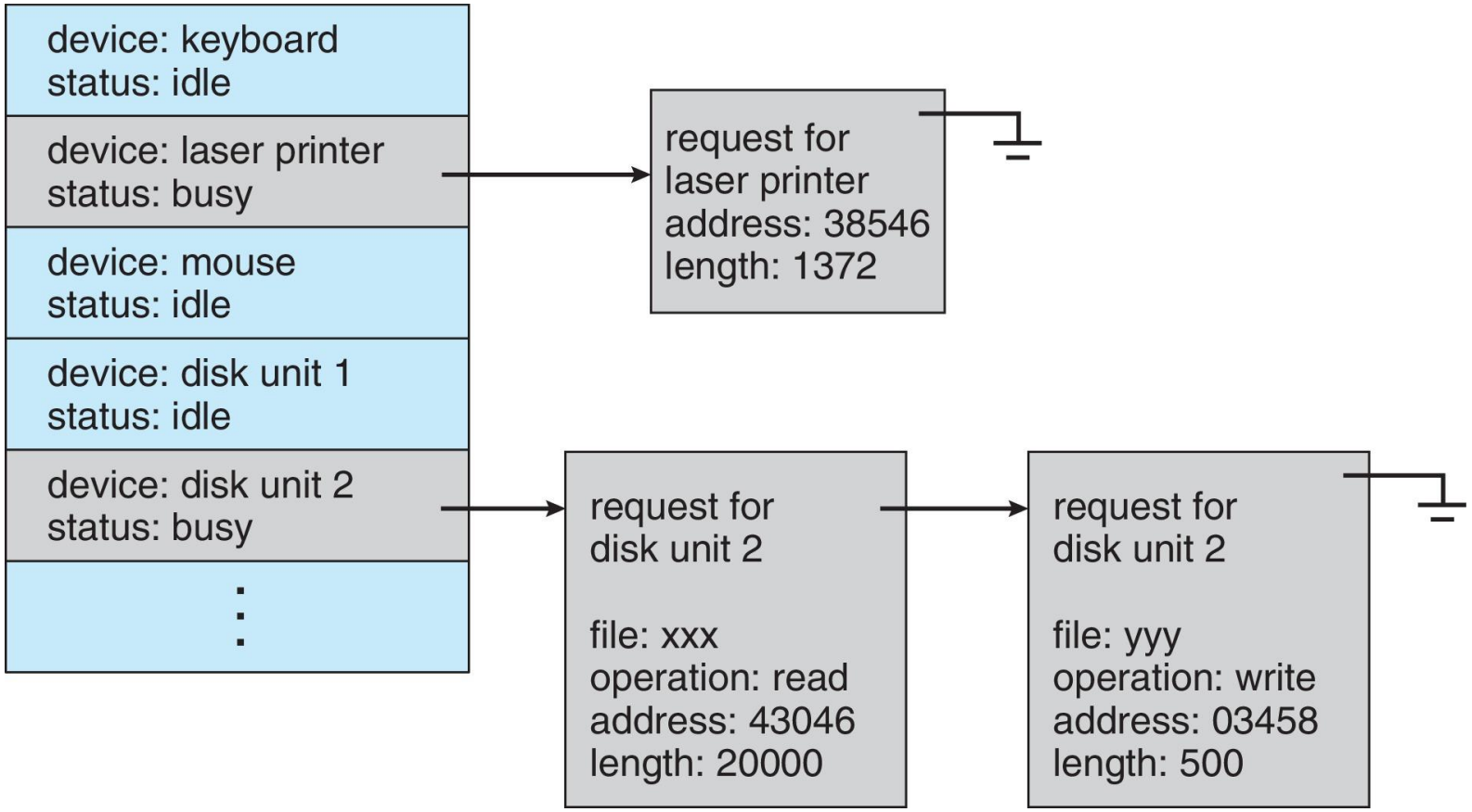
Scheduling

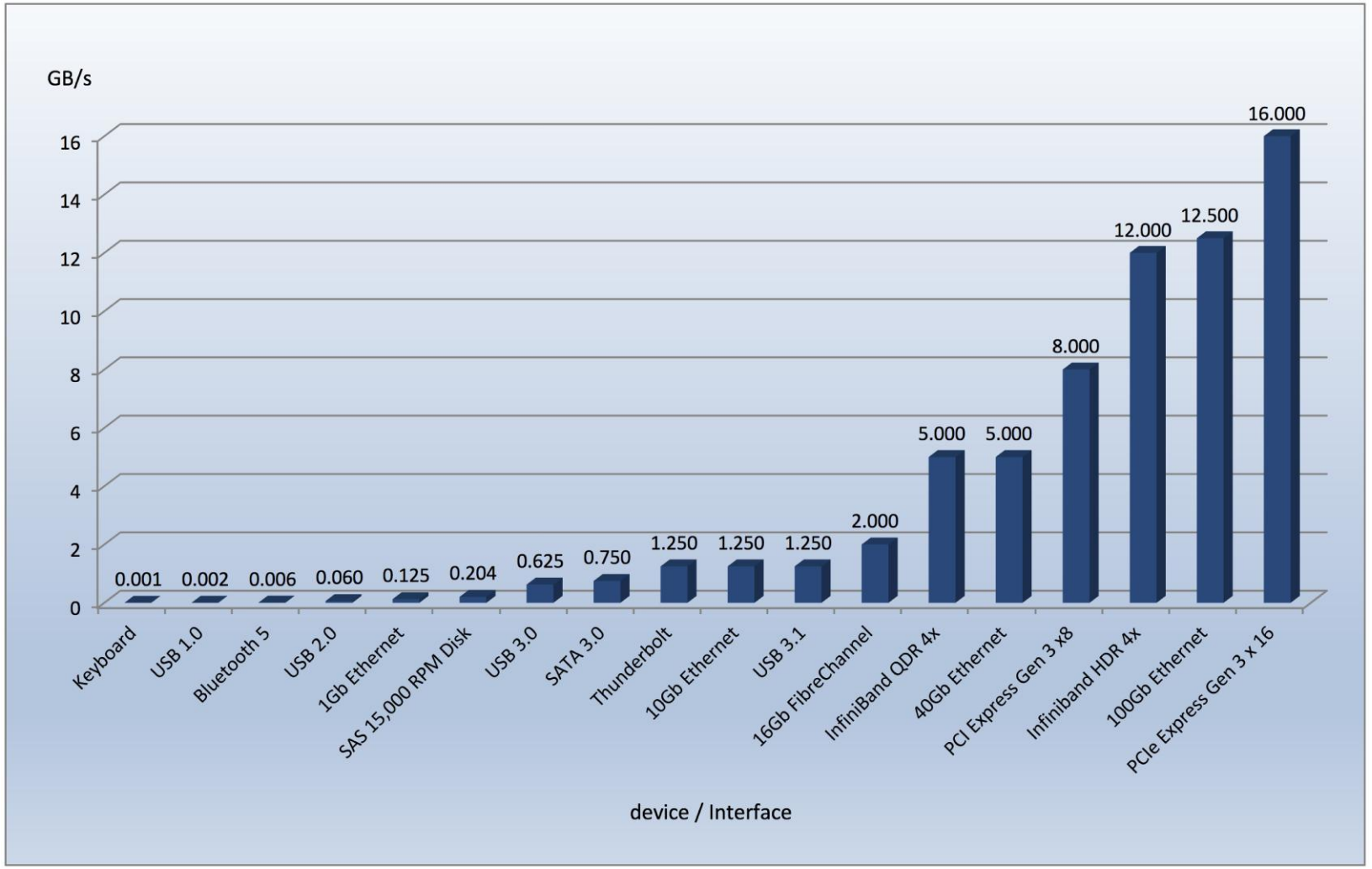
- Some I/O request ordering via per-device queue
- Some OSs try fairness
- Some implement Quality Of Service (i.e. IPQOS)

Buffering - store data in memory while transferring between devices

- To cope with device speed mismatch
- To cope with device transfer size mismatch
- To maintain “copy semantics”
- **Double buffering** – two copies of the data
  - Kernel and user
  - Full / being processed and not-full / being used
  - Copy-on-write can be used for efficiency in some cases

Device-status Table





Common PC and Data-center I/O devices and Interface Speeds

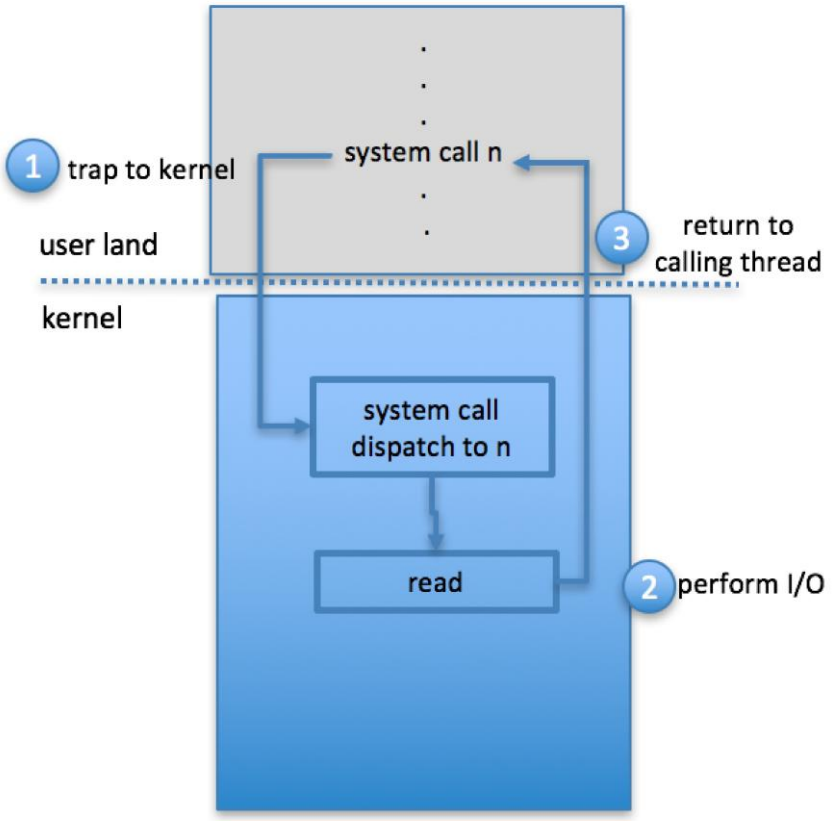
Kernel I/O Subsystem

- **Caching** - faster device holding copy of data
  - Always just a copy
  - Key to performance
  - Sometimes combined with buffering
- **Spooling** - hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing
- **Device reservation** - provides exclusive access to a device
  - System calls for allocation and de-allocation
  - Watch out for deadlock

I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - **I/O must be performed via system calls**
    - Memory-mapped and I/O port memory locations must be protected too

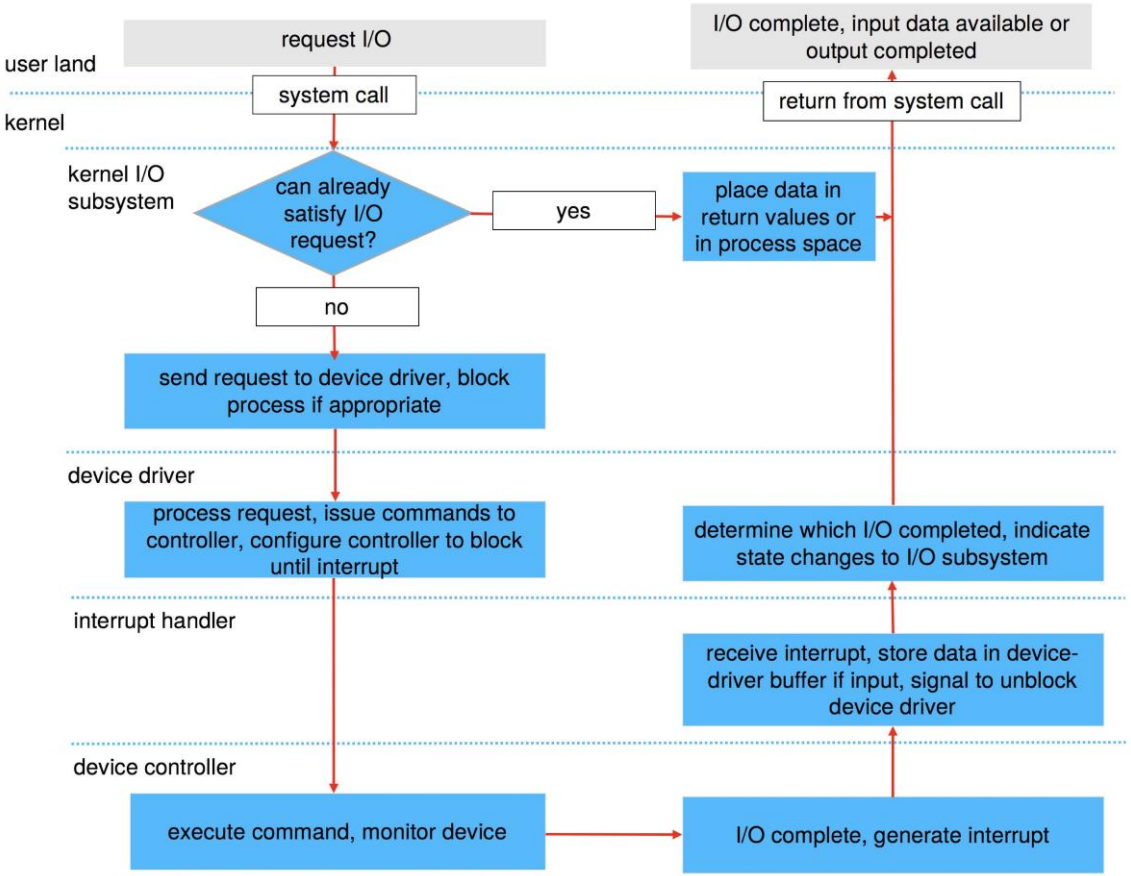
Use of a System Call to Perform I/O



Transforming I/O Requests to Hardware Operations

Consider reading a file from disk for a process:

- Determine the device holding file
- Translate the name to the device representation
- Physically read data from disk into the buffer
- Make data available to the requesting process
- Return control to the process



Life Cycle of An I/O Request

I/O using Input_event structure

- The `input_event` structure in Linux is the fundamental way the kernel reports input events from various devices (keyboards, mice, joysticks, touchscreens, etc.) to user-space applications.
- It provides a standardized format for describing what kind of event occurred, when it happened, and its value.

```
struct input_event {
        struct timeval time;
        __u16 type;
        __u16 code;
        __s32 value;
};
```

I/O using Input_event structure

- The `input_event` structure in Linux is the fundamental way the kernel reports input events from various devices (keyboards, mice, joysticks, touchscreens, etc.) to user-space applications.
- It provides a standardized format for describing what kind of event occurred, when it happened, and its value.

```
struct input_event {
        struct timeval time;
        __u16 type;
        __u16 code;
        __s32 value;
};
```

Members of `input_event`

- `struct timeval time;`
  - This member is a structure itself, defined in `<sys/time.h>`, and it records the timestamp of when the input event occurred. It has two fields:
    - **`long tv_sec`**: Represents the number of seconds that have elapsed since the Unix epoch (January 1, 1970, 00:00:00 UTC).
    - **`long tv_usec`**: Represents the number of microseconds (millionths of a second) that have elapsed within the current second.

Members of `input_event` (2)

- `__u16 type;`
  - This is an unsigned 16-bit integer (unsigned short) that indicates the type of event that occurred.
    - **`EV_KEY`:** Key press and release events (including buttons on mice and joysticks).
    - **`EV_REL`**: Relative axis movement events (e.g., mouse motion, joystick movements). The value field indicates the change in position.
    - **`EV_ABS`**: Absolute axis movement events (e.g., touchscreen coordinates, joystick axes with absolute positioning). The value field indicates the absolute position.
    - **`EV_MSC`**: Miscellaneous events (e.g., scancodes from keyboards).
    - **`EV_SW`**: Switch state changes (e.g., headphone jack insertion).
    - **`EV_LED`**: LED state changes (e.g., keyboard LEDs).

Members of `input_event` (3)

- `__u16 code;`
  - This is an unsigned 16-bit integer (unsigned short) that specifies the specific event code within the given type. The meaning of the code depends on the type.
    - If type is `EV_KEY`, code might be BTN_LEFT (left mouse button), `BTN_RIGHT` (right mouse button), `KEY_A` (the 'A' key on the keyboard), `KEY_SPACE`, etc.
    - If type is `EV_REL`, code might be `REL_X` (relative movement along the X-axis), `REL_Y` (relative movement along the Y-axis), `REL_WHEEL` (vertical scroll wheel), `REL_HWHEEL` (horizontal scroll wheel), etc.
    - If type is `EV_ABS`, code might be `ABS_X` (absolute X coordinate), `ABS_Y` (absolute Y coordinate), `ABS_PRESSURE` (pressure on a touchscreen), etc.

Members of `input_event` (4)

- `__u32 value;`
  - This is a signed 32-bit integer (int) that represents the value associated with the event. The meaning of value depends on the type and code:
    - For `EV_KEY` events:
      - 0: Key release.
      - 1: Key press.
      - 2: Key repeat (if the key is held down).
    - For `EV_REL` events: This is the signed change in the relative position (e.g., -2 for moving the mouse 2 pixels to the left).
    - For `EV_ABS` events: This is the current absolute position of the axis. The valid range for these values can be queried for each device.

Members of `input_event` (4)

User-space applications typically interact with input devices by:

1. **Opening an Event Device File:** Applications open a file in the `/dev/input/event*` directory corresponding to the input device they want to monitor (e.g., `/dev/input/event0` for a keyboard, `/dev/input/event1` for a mouse).

2. **Reading `input_event` Structures:** They then use the `read()` system call to read data from this file. Each successful read() usually returns one or more `input_event` structures that the kernel placed in the file's buffer.

3. **Interpreting the Events:** The application then examines the type, code, and value members of the received `input_event` structures to determine what action occurred on the input device and takes appropriate action.

Thank you

Juan F. Medina

Juan.medina26@upr.edu

Introduction to Security

Juan Felipe Medina Lee, Ph.D.

Computer Science and Engineering
University of Puerto Rico Mayagüez Campus

Outline

- The Security Problem
- Program Threats
- System and Network Threats
- Cryptography as a Security Tool
- User Authentication

Objectives

- Discuss security threats and attacks
- Explain the fundamentals of encryption, authentication, and hashing
- Examine the uses of cryptography in computing
- Describe the various countermeasures to security attacks

The Security Problem

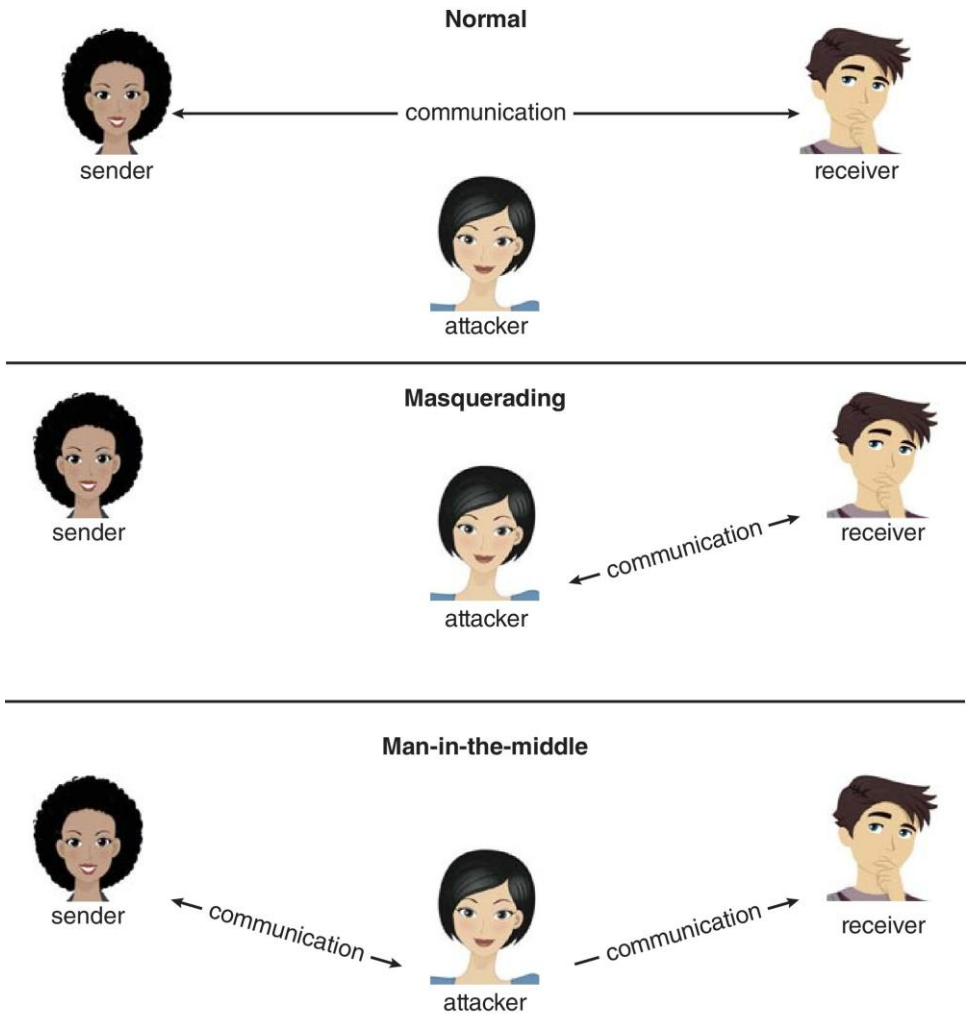
- A system **secure** if its resources are used and accessed as intended under all circumstances
    - Unachievable
- **Intruders** (**crackers**) attempt to breach security
- **Threat** is a potential security violation
- **Attack** is an attempt to breach security
    - Attack can be accidental or malicious
    - Easier to protect against accidental than malicious misuse

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

Security Violation Methods

- **Masquerading** (breach **authentication**)
  - Pretending to be an authorized user to escalate privileges
- **Replay attack**
  - As is or with **message modification**
- **Man-in-the-middle attack**
  - Intruder sits in data flow, masquerading as sender to receiver and vice versa
- **Session hijacking**
  - Intercept an already-established session to bypass authentication
- **Privilege escalation**
  - Common attack type with access beyond what a user or resource is supposed to have

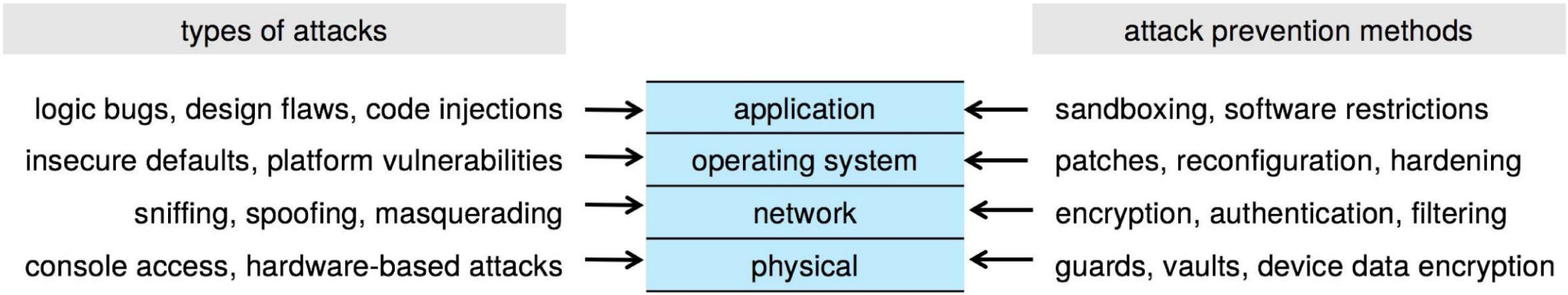


Standard Security Attacks

Security Measure Levels

- Impossible to have absolute security, but make the cost to perpetrators sufficiently high to deter most intruders
- Security must occur at four levels to be effective:
  - **Physical**
    - Data centers, servers, connected terminals
  - **Network**
    - Intercepted communications, interruption, DOS
  - **Operating System**
    - Protection mechanisms, debugging
  - **Application**
    - Benign or malicious apps can cause security problems
- Security is as weak as the weakest link in the chain
- Humans a risk too via **phishing** and **social-engineering** attacks

Four-layered Model of Security



Program Threats

- **Malware** - Software designed to exploit, disable, or damage computer
- **Trojan Horse** – Program that acts in a clandestine manner
  - **Spyware** – Program frequently installed with legitimate software to display adds, capture user data
  - **Ransomware** – locks up data via encryption, demanding payment to unlock it
- Others include trap doors, logic bombs
- **All try to violate the Principle of Least Privilege:**

THE PRINCIPLE OF LEAST PRIVILEGE

"The principle of least privilege. Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job. The purpose of this principle is to reduce the number of potential interactions among privileged programs to the minimum necessary to operate correctly, so that one may develop confidence that unintentional, unwanted, or improper uses of privilege do not occur."—Jerome H. Saltzer, describing a design principle of the Multics operating system in 1974: https://pdfs.semanticscholar.org/1c8d/06510ad449ad24fbdd164f8008cc730cab47.pdf.

- The goal frequently is to leave behind a Remote Access Tool (RAT) for repeated access

Program Threats

Many variations, many names

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**
- Up to 80% of spam delivered by spyware-infected systems

Trap Door

- Specific user identifier or password that circumvents normal security procedures
- Could be included in a compiler
- How to detect them?

```c
#include <stdio.h>
#include <string.h>
// Predefined password
#define PASSWORD "securepassword"

int main() {
    char input[256];

    printf("Enter the password: ");
    // Read user input
    gets(input);

    // Validate the password
    int v = validate(input);
    if (v || strcmp(input, PASSWORD) == 0) {
        printf("Access granted!\n");
    } else {
        printf("Access denied!\n");
    }

    return 0;
}
```

Trapdoor example

C Program with Buffer-overflow Condition

```c
#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;
    }
}
```

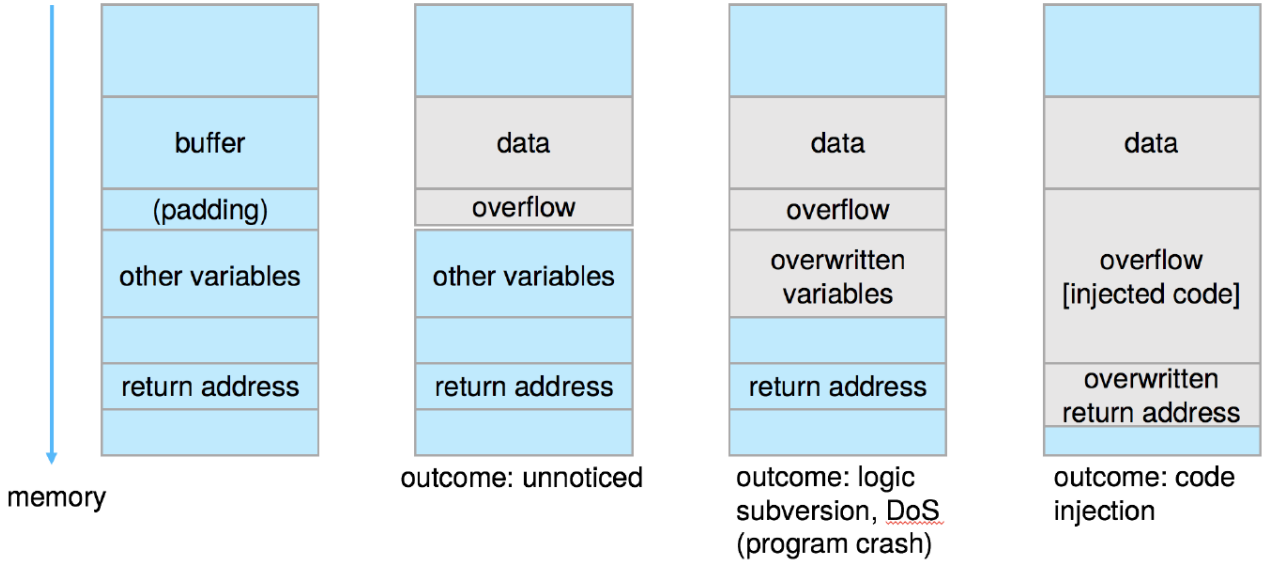
- **Code review** can help – programmers review each other's code, looking for logic flows, programming flaws

Code Injection

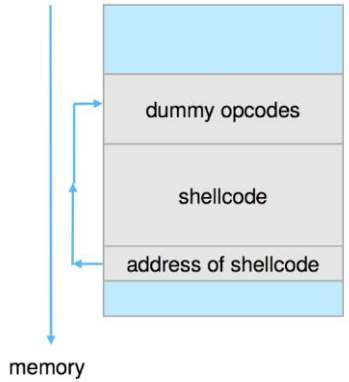
- **Code-injection attack** occurs when system code is not malicious but has bugs allowing executable code to be added or modified
  - Results from poor or insecure programming paradigms, commonly in low-level languages like C or C++ which allow for direct memory access through pointers
  - Goal is a buffer overflow in which code is placed in a buffer and execution caused by the attack

Code Injection (Cont.)

- Outcomes from code injection include:



- Frequently use trampoline to code execution to exploit buffer overflow:



Great Programming Required?

- For the first step of determining the bug, and the second step of writing exploit code, yes
- **Script kiddies** can run pre-written exploit code to attack a given system
- Attack code can get a shell with the processes' owner's permissions
  - Or open a network port, delete files, download a program, etc.
- Depending on the bug, the attack can be executed across a network using allowed connections, bypassing firewalls
  - **Morris worm**
- Buffer overflow can be disabled by **disabling stack execution** or adding a bit to the page table to indicate a "non-executable" state
  - Available in SPARC and x86
  - But still have security exploits

Program Threats (Cont.)

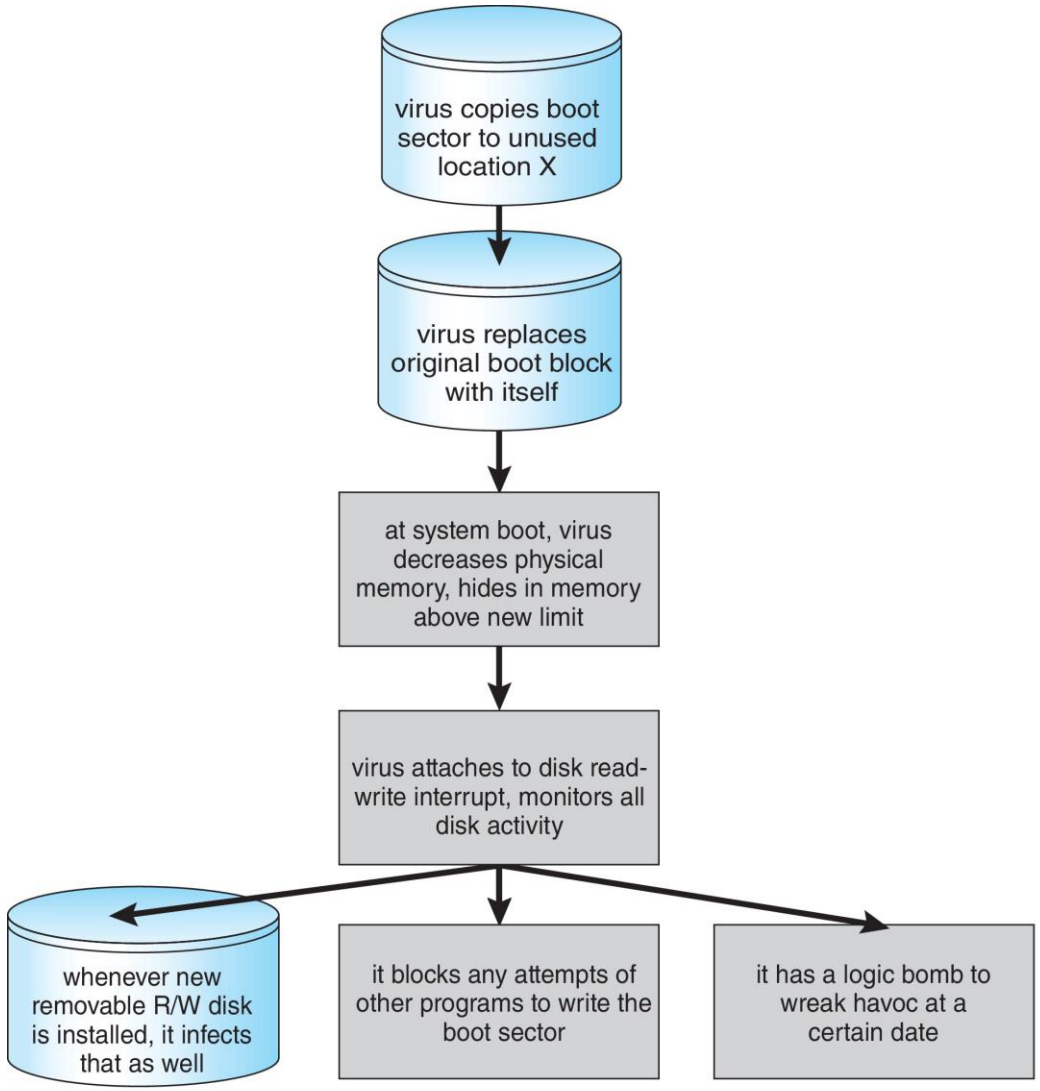
Viruses

- Code fragment embedded in a legitimate program
- Self-replicating, designed to infect other computers
- Very specific to CPU architecture, operating system, applications
- Usually borne via email or as a macro
- Visual Basic Macro to reformat hard drive

```
Sub AutoOpen()
Dim oFS
   Set oFS = CreateObject(''Scripting.FileSystemObject'')
   vs = Shell(''c:command.com /k format c:'',vbHide)
End Sub
```

Program Threats (Cont.)

- **Virus dropper** inserts a virus into the system
- Many categories of viruses
  - File/parasitic
  - Boot/memory
  - Macro
  - Source code
  - Polymorphic to avoid having a **virus signature**
  - Encrypted
  - Stealth
  - Tunneling
  - Multipartite
  - Armored



A Boot-sector Computer Virus

The Threat Continues

- Attacks are still common, still occurring
- Attacks moved over time from science experiments to tools of organized crime
  - Targeting specific companies
  - Creating botnets to use as tools for spam and DDOS delivery
  - **Keystroke logger** to grab passwords, credit card numbers
- Why is Windows the target for most attacks?
  - Most common
  - **Monoculture** is considered harmful

System and Network Threats

- Some systems are “open” rather than **secure by default**
  - Reduce **attack surface**
  - But more complicated to use, and more knowledge is needed to administer

- Network threats are more complex to detect and prevent
  - Protection systems are weaker
  - More difficult to have a shared secret on which to base access
  - No physical limits once the system is attached to the internet
    - Or on a network with a system attached to the Internet
  - Even determining the location of the connecting system is difficult
    - IP address is the only knowledge

System and Network Threats (Cont.)

- Denial of Service
  - Overload the targeted computer, preventing it from doing any useful work
  - **Distributed Denial-of-Service** (**DDoS**) comes from multiple sites at once
  - Consider the start of the IP-connection handshake (SYN)
- How many started connections can the OS handle?
  - Consider traffic to a website
- How can you tell the difference between being a target and being really popular?
  - Accidental – CS students writing bad `fork()` code
  - Purposeful – extortion, punishment

System and Network Threats (Cont.)

Port scanning

- Automated attempt to connect to a range of ports on one or a range of IP addresses
- Detection of answering service protocol
- Detection of OS and version running on system
- `nmap` scans all ports in a given IP range for a response
- `nessus` has a database of protocols and bugs (and exploits) to apply against a system
- Frequently launched from **zombie systems**
    - To decrease trace-ability

Cryptography as a Security Tool

- Broadest security tool available
  - Internally to a given computer, the source and destination of messages can be known and protected
    - OS creates, manages, and protects process IDs, communication ports
- The source and destination of messages on a network cannot be trusted without cryptography
    - Local network – IP address?
      - Consider unauthorized host added
    - WAN / Internet – how to establish authenticity
      - Not via IP address

Cryptography

- Means to constrain potential senders (*sources*) and/or receivers (*destinations*) of *messages*
  - Based on secrets (**keys**)
  - Enables
    - Confirmation of source
    - Receive only at a specific destination
    - A trust relationship between the sender and the receiver

Encryption

Constrains the set of possible receivers of a message

Encryption algorithm consists of

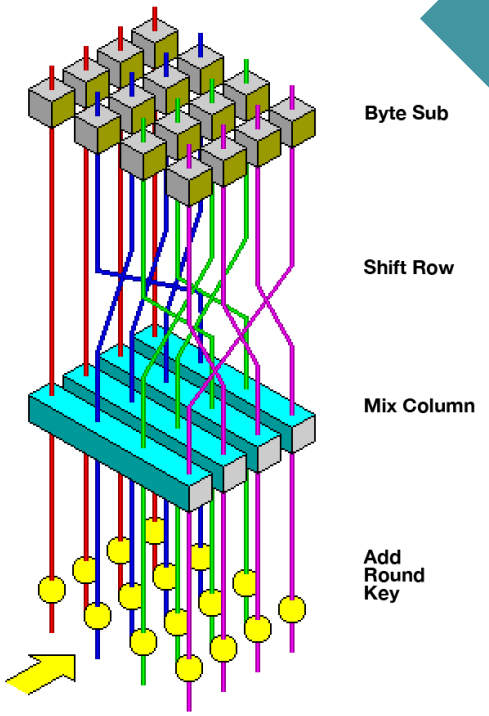
- Set K of keys
- Set M of Messages
- Set C of ciphertexts (encrypted messages)
- A function $E : K \rightarrow (M \rightarrow C)$. That is, for each $k \in K$, E_k is a function for generating ciphertexts from messages
    - Both E and E_k for any k should be efficiently computable functions
- A function $D : K \rightarrow (C \rightarrow M)$. That is, for each $k \in K$, D_k is a function for generating messages from ciphertexts
    - Both D and D_k for any k should be efficiently computable functions

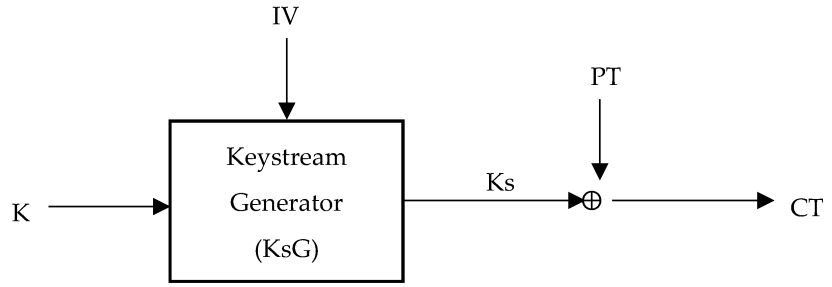
Encryption (Cont.)

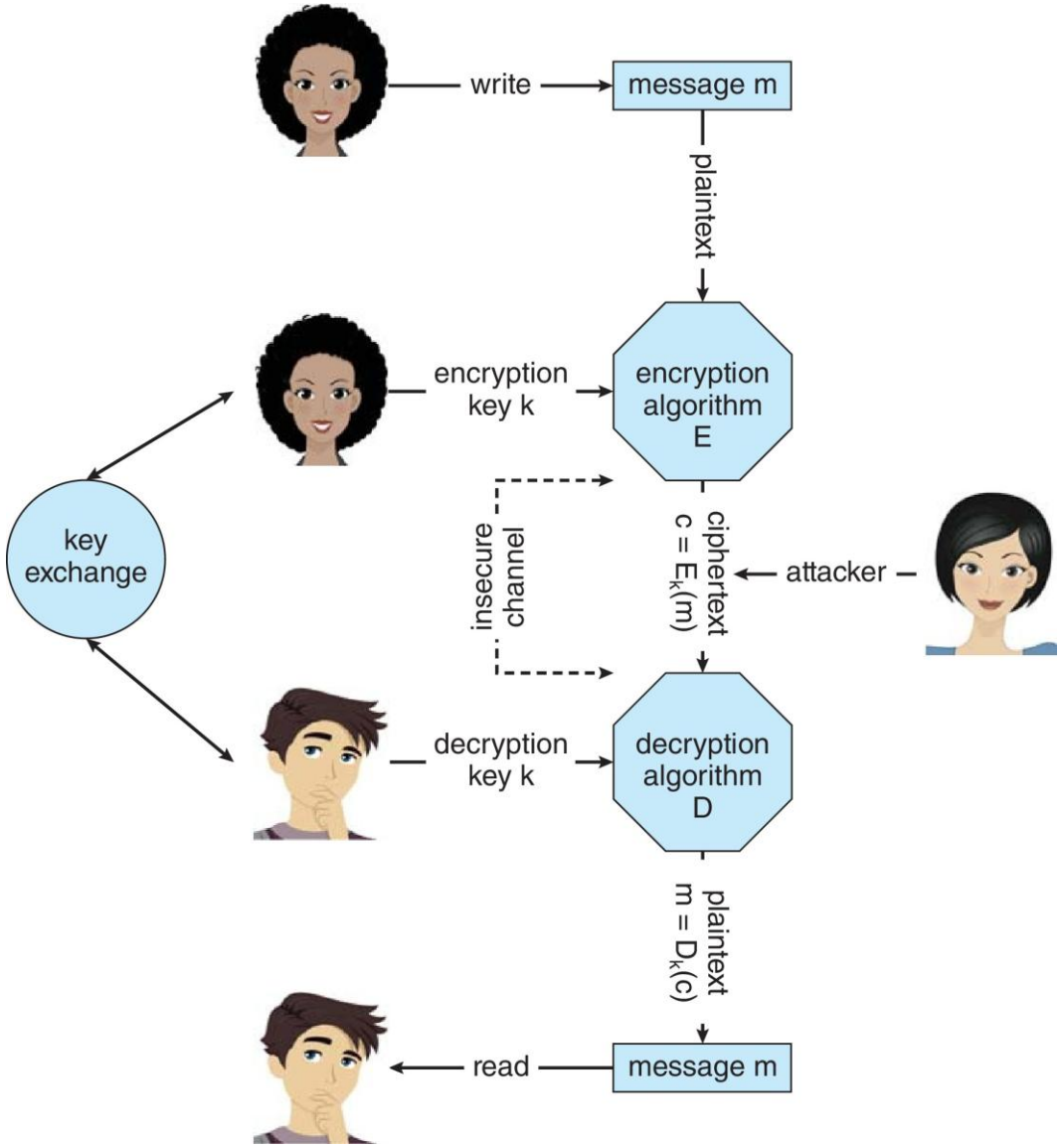
- An encryption algorithm must provide this essential property: **Given a ciphertext c ∈ C, a computer can compute m such that $E_k(m) = c$ only if it possesses *k***
  - Thus, a computer holding *k* can decrypt ciphertexts to the plaintexts used to produce them, but a computer not holding *k* cannot decrypt ciphertexts
  - Since ciphertexts are generally exposed (for example, sent on the network), it is important that it be infeasible to derive *k* from the ciphertexts

Symmetric Encryption

- The same key is used to encrypt and decrypt
  - Therefore, *k* must be kept secret
- **DES** was the most commonly used **symmetric block-encryption** algorithm (created by US Govt)
  - Encrypts a block of data at a time
  - Keys are too short, so now considered insecure
- Triple-DES considered more secure
  - Algorithm used 3 times using 2 or 3 keys
  - For example $c = E_{k3}(D_{k2}(E_{k1}(m)))$
- 2001 NIST adopted new block cipher - Advanced Encryption Standard (**AES**)
  - Keys of 128, 192, or 256 bits, works on 128 bit blocks
- RC4 is the most common symmetric stream cipher, but is known to have vulnerabilities
  - Encrypts/decrypts a stream of bytes (i.e., wireless transmission)
  - Key is an input to a pseudo-random-bit generator
    - Generates an infinite **keystream**







Any problems?

Secure Communication over Insecure Medium

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
- It 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

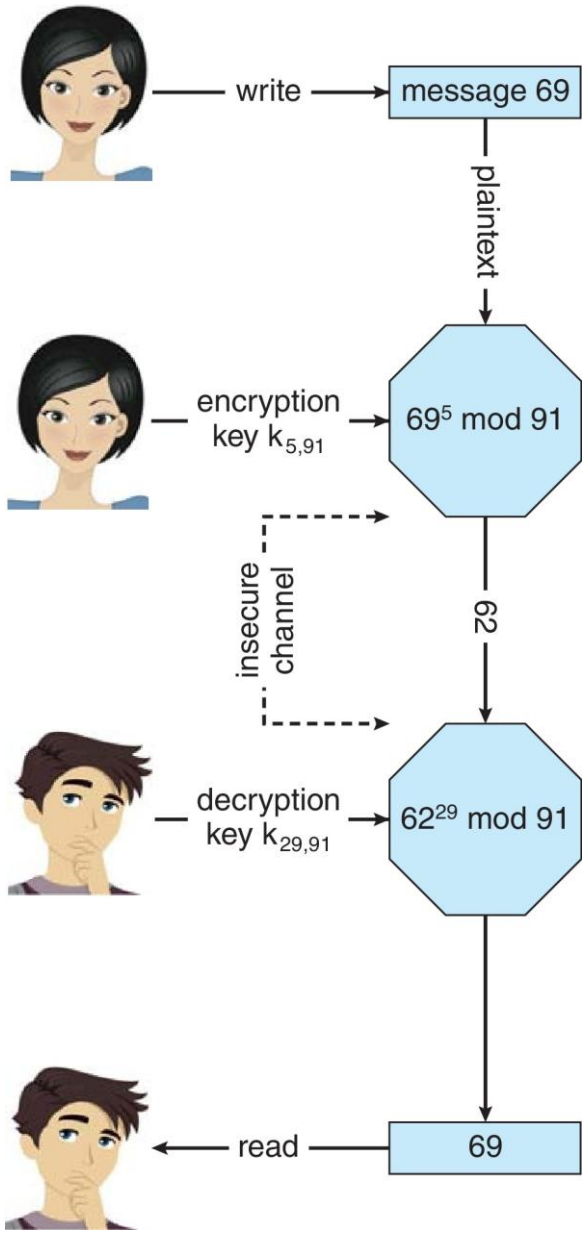
Asymmetric Encryption (Cont.)

Formally, it is computationally infeasible to derive $k_{d,N}$ from $k_{e,N}$, and so k_e need not be kept secret and can be widely disseminated

- k_e is the **public key**
- k_d is the **private key**
- N is the product of two large, randomly chosen prime numbers p and q (for example, p and q are 2048 bits each)
- Encryption algorithm is $E_{ke,N}(m) = m^{k_e} \bmod N$, where k_e satisfies $k_e k_d \bmod (p-1)(q-1) = 1$
- The decryption algorithm is then $D_{kd,N}(c) = c^{k_d} \bmod N$

Asymmetric Encryption Example

- For example. make $p = 7$and $q = 13$
- We then calculate $N = 7*13 = 91$ and $(p-1)(q-1) = 72$
- We next select k_e relatively prime to 72 and< 72, yielding 5
- Finally, we calculate k_d such that $k_e k_d \bmod 72 = 1$, yielding 29
- We how have our keys
  - Public key, $k_{e,N} = 5, 91$
  - Private key, $k_{d,N} = 29, 91$
- Encrypting the message 69 with the public key results in the ciphertext 62
- Cyphertext can be decoded with the private key
  - Public key can be distributed in cleartext to anyone who wants to communicate with holder of public key



Encryption using RSA Asymmetric Cryptography

Cryptography (Cont.)

- Note that symmetric cryptography is based on transformations, asymmetric cryptography on mathematical functions
  - Asymmetric, much more computationally intensive
  - Typically not used for bulk data encryption

- Asymmetric encryption algorithms are used for:
  - encryption of small amounts of data
  - Authentication
  - Confidentiality
  - **key distribution**

Authentication

- It constrains the set of **potential senders** of a message
  - Complementary to encryption
  - Also, it can prove that the message is unmodified
- Algorithm components
  - A set K of keys
  - A set M of messages
  - A set A of authenticators
  - A function $S : K \rightarrow (M \rightarrow A)$
    - That is, for each $k \in K$, S_k is a function for generating authenticators from messages
    - Both S and S_k for any k should be efficiently computable functions
  - A function $V : K \rightarrow (M \times A \rightarrow \{\text{true, false}\})$. That is, for each $k \in K$, V_k is a function for verifying authenticators on messages
    - Both V and V_k for any k should be efficiently computable functions

Authentication (Cont.)

- For a message *m*, a computer can generate an authenticator $a \in A$ such that $V_k(m, a)$ = `true` only if it possesses *k*
- Thus, the computer holding *k* can generate authenticators on messages so that any other computer possessing *k* can verify them
- Computers not holding *k* cannot generate authenticators on messages that can be verified using V_k
- Since authenticators are generally exposed (for example, they are sent on the network with the messages themselves), it must not be feasible to derive *k* from the authenticators
- Practically, if $V_k(m,a)$ = **`true`** then we know *m* has not been modified and that the sender of the message has *k*
  - If we share *k* with only one entity, we know where the message originated

Authentication – Hash Functions

- Basis of authentication
- Creates small, fixed-size block of data **message digest** (**hash value**) from *m*
- Hash Function *H* must be collision resistant on *m*
  - Must be infeasible to find an $m' \neq m$ such that $H(m) = H(m')$
- If $H(m) = H(m')$, then $m = m'$
  - The message has not been modified
- Common message-digest functions include **MD5**, which produces a 128-bit hash, and **SHA-1**, which outputs a 160-bit hash
- Not useful as authenticators
  - For example *H(m)* can be sent with a message
    - But if *H* is known someone could modify *m* to *m'* and recompute *H(m')* and modification not detected
    - **So must authenticate *H(m)***

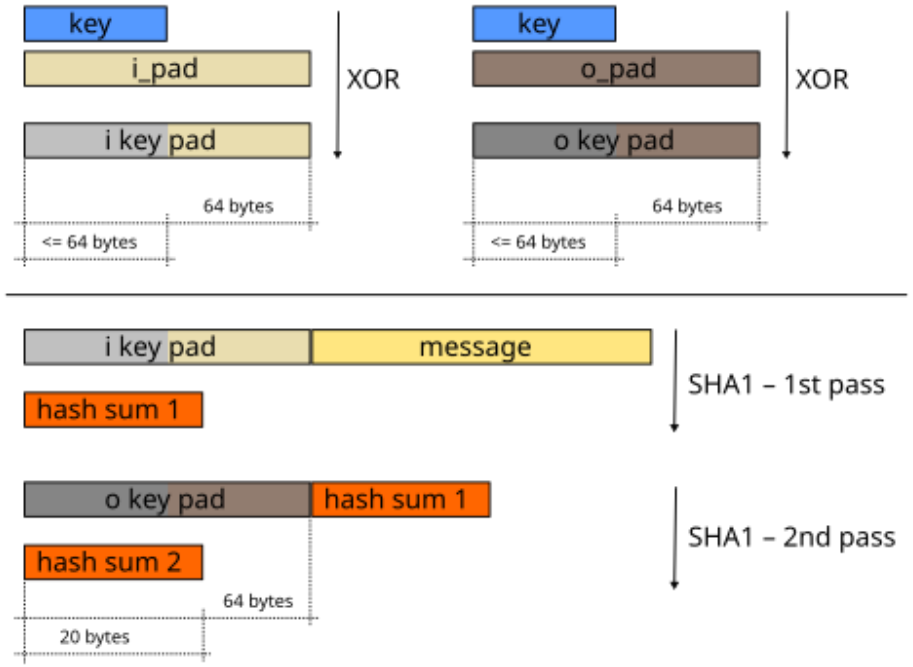
Authentication - MAC

- Symmetric encryption is used in the **message-authentication code** (**MAC**) authentication algorithm
- Cryptographic checksum generated from the message using the secret key
  - Can securely authenticate short values
- If used to authenticate *H(m)* for an *H* that is collision resistant, then obtain a way to securely authenticate long messages by hashing them first
- Note that *k* is needed to compute both S_k and V_k, so anyone able to compute one can compute the other

MAC example

- **Sender:**
  - Message: `Hello`
  - Secret key: `key123`
  - Algorithm: `HMAC-SHA256`
  - Output MAC: `e2f5... (hash value)`
  - Sent: `[Hello][e2f5...]`

- **Recipient:**
  - Receives: `[Hello][e2f5...]`
  - Recomputes HMAC using `Hello` and `key123`.
  - Compares with received MAC:
    - If match → Message is valid.
    - If mismatch → Message is tampered or unauthenticated.



Authentication – Digital Signature

- Based on asymmetric keys and digital signature algorithm
- Authenticators produced are **digital signatures**
- Very useful – ***anyone*** can verify authenticity of a message
- In a digital-signature algorithm, computationally infeasible to derive k_s from k_v
  - V is a one-way function
  - Thus, k_v is the public key and k_s is the private key
- Consider the RSA digital signature algorithm
  - Similar to the RSA encryption algorithm, but the key use is reversed
  - Digital signature of message $S_{ks}(m) = H(m)^{k_s} \bmod N$
  - The key k_s is again a pair (d, N), where N is the product of two large, randomly chosen prime numbers p and q
  - Verification algorithm is $V_{kv}(m, a) \stackrel{?}{=} (a^{k_v} \bmod N = H(m))$
    - Where k_v satisfies $k_v k_s \bmod (p-1)(q-1) = 1$

Authentication (Cont.)

Why is authentication a subset of encryption?

- Fewer computations (except for RSA digital signatures)
- Authenticator is usually shorter than a message
- Sometimes want authentication but not confidentiality
    - Signed patches et al
- Can be basis for **non-repudiation**

Key Distribution

- Delivery of a symmetric key is a huge challenge
  - Sometimes done **out-of-band** (Exchanged in paper)
- Asymmetric keys can proliferate – stored on **key ring**
  - Even asymmetric key distribution needs care – man-in-the-middle attack

User Authentication

Crucial to identify user correctly, as protection systems depend on user ID

- User identity most often established through **passwords**, can be considered a special case of either keys or capabilities
- Passwords must be kept secret
  - Frequent change of passwords
  - History to avoid repeats
  - Use of “non-guessable” passwords
  - Log all invalid access attempts (but not the passwords themselves)
  - Unauthorized transfer
- Passwords may also either be encrypted or allowed to be used only once
  - Does encrypting passwords solve the exposure problem?
    - Might solve **sniffing**
    - Consider Trojan horse keystroke logger
    - How are passwords stored at the authenticating site?

Passwords

- Encrypt to avoid having to keep a secret
  - But keep secret anyway (i.e. Unix uses superuser-only readable file `/etc/shadow`)
  - Use an algorithm that is easy to compute but difficult to invert (hash)
  - Only encrypted password stored, never decrypted
- One-time passwords
  - Use a function based on a seed to compute a password, both for the user and the computer
  - Hardware device/calculator / key fob to generate the password
    - Changes very frequently
    - **Double Factor Authentication**
- Biometrics
  - Some physical attributes (fingerprint, hand scan, face recognition)
- Multi-factor authentication
  - Need two or more factors for authentication
    - i.e., USB “dongle”, biometric measure, and password

Passwords (Cont.)

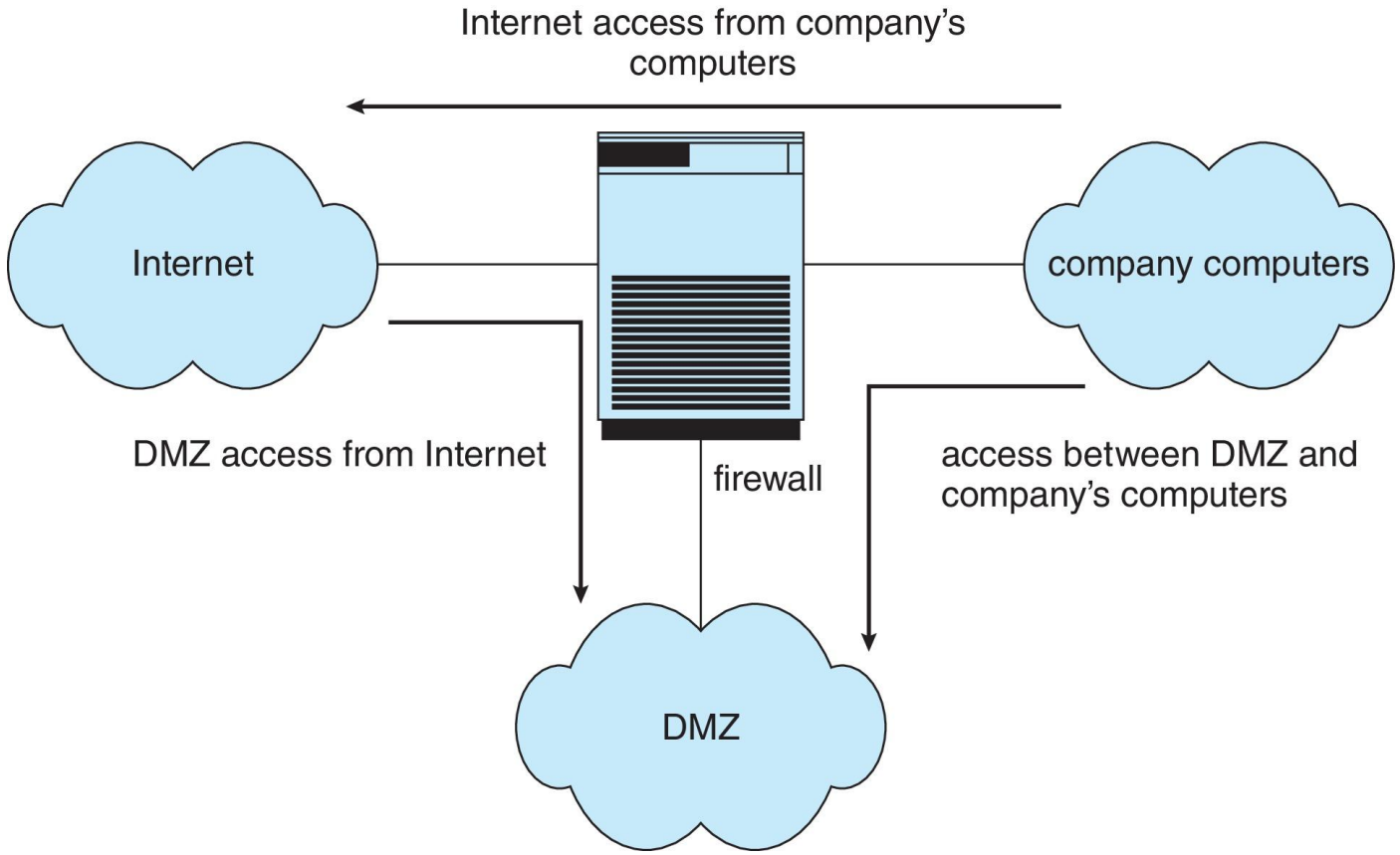
STRONG AND EASY TO REMEMBER PASSWORDS

It is extremely important to use strong (hard to guess and hard to shoulder surf) passwords on critical systems like bank accounts. It is also important to not use the same password on lots of systems, as one less important, easily hacked system could reveal the password you use on more important systems. A good technique is to generate your password by using the first letter of each word of an easily remembered phrase using both upper and lower characters with a number or punctuation mark thrown in for good measure. For example, the phrase "My girlfriend's name is Katherine" might yield the password "Mgn.isK!". The password is hard to crack but easy for the user to remember. A more secure system would allow more characters in its passwords. Indeed, a system might also allow passwords to include the space character, so that a user could create a **passphrase** which is easy to remember but difficult to break.

Firewalling to Protect Systems and Networks

- A network **firewall** is placed between trusted and untrusted hosts
  - The firewall limits network access between these two **security domains**
- Can be tunneled or spoofed
  - Tunneling allows disallowed protocol to travel within allowed protocol (i.e., telnet inside of HTTP)
  - Firewall rules are typically based on hostname or IP address which can be spoofed
- **Personal firewall** is a software layer on a given host
  - Can monitor/limit traffic to and from the host
- **Application proxy firewall** understands application protocols and can control them (i.e., SMTP)
- **System-call firewall** monitors all important system calls and applies rules to them (i.e., this program can execute that system call)

Network Security Through Domain Separation Via Firewall



Ethical considerations in OS development

A general review

Real-World Ethical Challenges in OS Development

1. **Vulnerability Disclosure:** To what extent do we reveal the knowledge of a potential security flaw that an entity can be exploited?. Is it better to publicly release to raise awareness or is it better to contact the affected party and give them a chance to fix the problem?

2. **Data Collection vs. Privacy:** How do we balance “gathering usage data for improvement” with “respecting user privacy and minimizing surveillance”?

3. **Security vs. Accessibility:** How do we ensure security measures do not unduly burden or exclude users with disabilities? What efforts are being made to meet this goal?

4. **Backdoors and Government Access:** What are the ethical implications of building backdoors or providing privileged access to law enforcement or intelligence agencies?

5. **Addressing Legacy Systems:** What responsibility do vendors have to support and secure older operating systems that are still in use, even if they are no longer profitable to maintain?

OS Security: A Foundation Built on Ethics

•**The OS as a Critical Trust Anchor:** Operating systems are at the heart of our digital world. We entrust them with our data, privacy, and system security.

•**Developers Hold Significant Power:** OS creators make choices that directly impact millions of users, both positively and negatively.

•**Beyond Compliance: Ethics is Key:** Legal compliance is essential but not enough. Ethical considerations demand proactive commitment to user well-being.

•**Key Ethical Principles in OS Development:**

- **Honesty & Transparency:** Being forthright about potential vulnerabilities and design choices.
- **Responsibility:** Taking ownership of the consequences of your work.
- **Competence:** Maintaining the skills to build secure systems.
- **Respect for Privacy:** Minimizing data collection and maximizing user control.
- **Fairness:** Ensuring security measures are accessible and equitable for all users.

Upholding Ethical Practices: Responsibility and Impact

•Consequences of Ignoring Ethical Principles:

- **Data Breaches:** Exposure of sensitive personal information, leading to financial and reputational damage.
- **Privacy Violations:** Erosion of trust in technology and potential for misuse of personal data.
- **System Instability:** Vulnerabilities that can be exploited by malicious actors.
- **Erosion of Trust:** Users losing confidence in the operating system and its developers.

•Building an Ethical Culture:

- **Promote Education and Training:** Equip developers with the knowledge and skills to address security issues in an ethical manner.
- **Establish Clear Codes of Conduct:** Develop guidelines for responsible security practices and informed decision-making.
- **Encourage Open Communication and Whistleblowing:** Create a safe environment for reporting ethical concerns.
- **Foster Accountability:** Hold individuals and organizations responsible for ethical breaches.

Thank you

Juan F. Medina

Juan.medina26@upr.edu

Protection

Juan Felipe Medina Lee, Ph.D.

Computer Science and Engineering
University of Puerto Rico Mayagüez Campus

Objectives

- Discuss the goals and principles of protection in a modern computer system
- Explain how protection domains combined with an access matrix are used to specify the resources a process may access
- Examine capability and language-based protection systems
- Describe how protection mechanisms can mitigate system attacks

Goals of Protection

- In a protection model, a 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

Principles of Protection

Guiding principle – the **principle of least privilege**

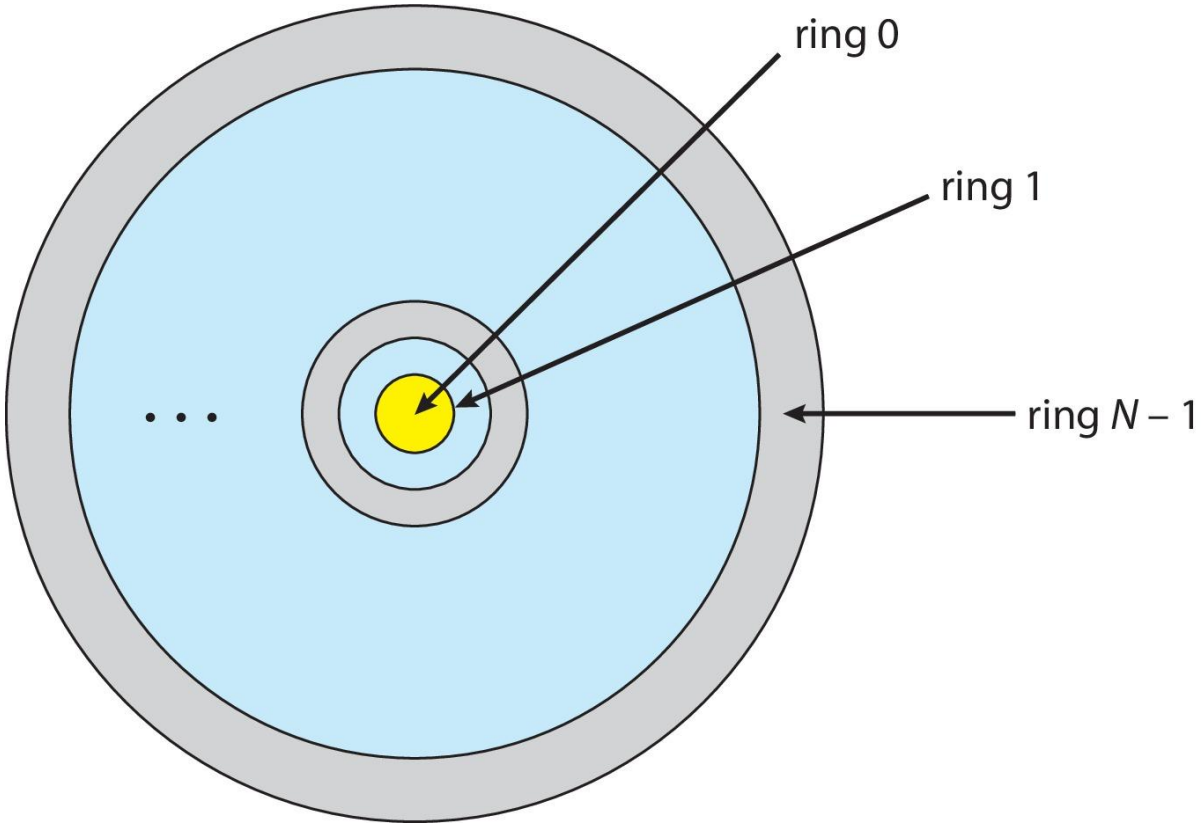
- Programs, users, and systems should be given just enough **privileges** to perform their tasks
- Properly set **permissions** can limit the damage if an entity has a bug, gets abused
- Can be static (during the life of the system, during the life of the process)
- Or dynamic (changed by a process as needed) – **domain switching, privilege escalation**
- **Compartmentalization** is a derivative concept regarding access to data
    - Process of protecting each system component through the use of specific permissions and access restrictions

Protection Rings

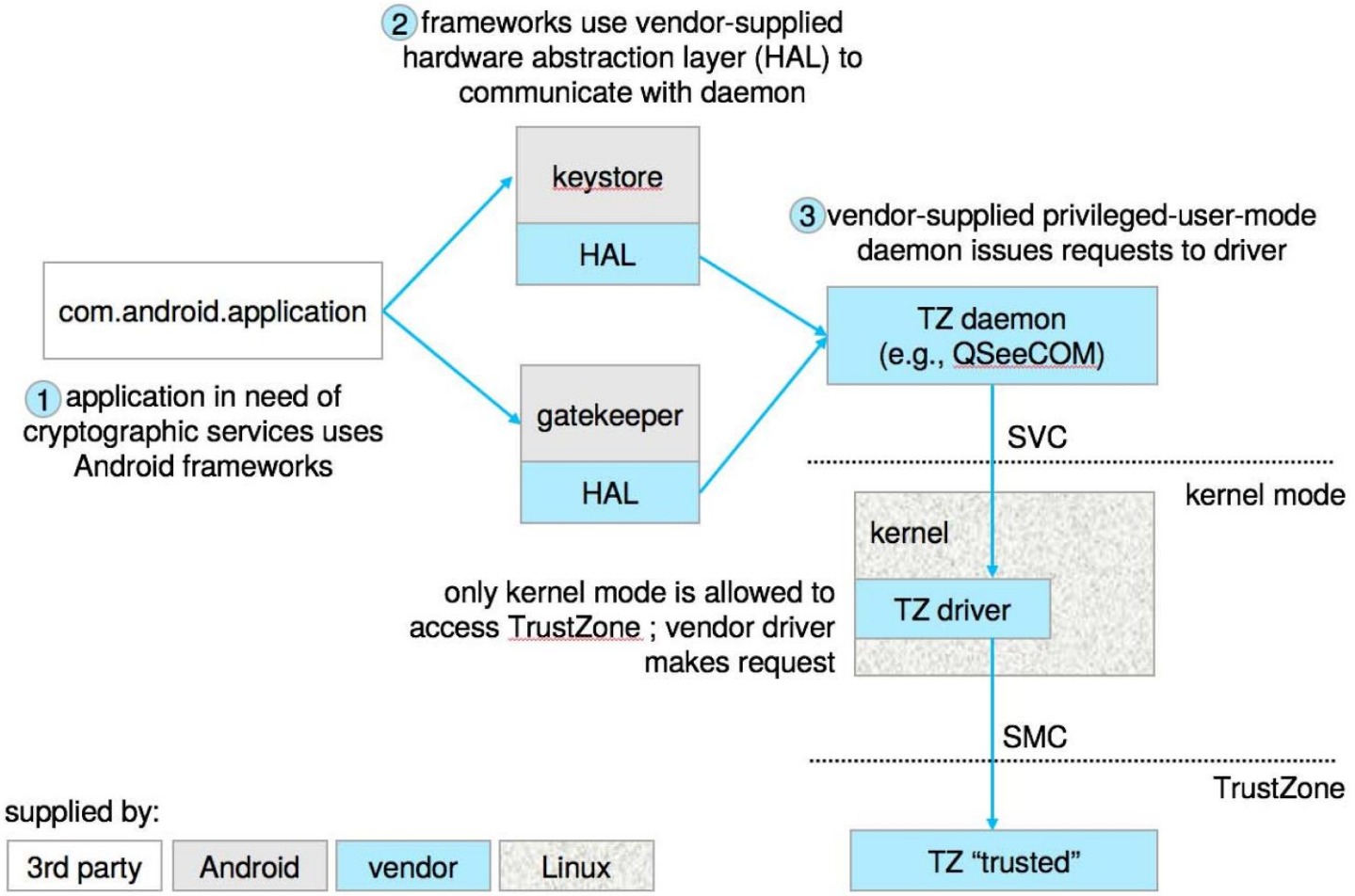
Components ordered by amount of privilege and protected from each other

- For example, **the kernel is in one ring and user applications in another**
- This privilege separation requires hardware support
- Gates used to transfer between levels, for example, the syscall instruction
- Also traps and interrupts
- **Hypervisors** introduced the need for yet another ring
- ARMv7 processors added **TrustZone**(**TZ**) ring to protect crypto functions with access via new **Secure Monitor Call** (**SMC**) instruction
    - Protecting NFC secure element and crypto keys from even the kernel

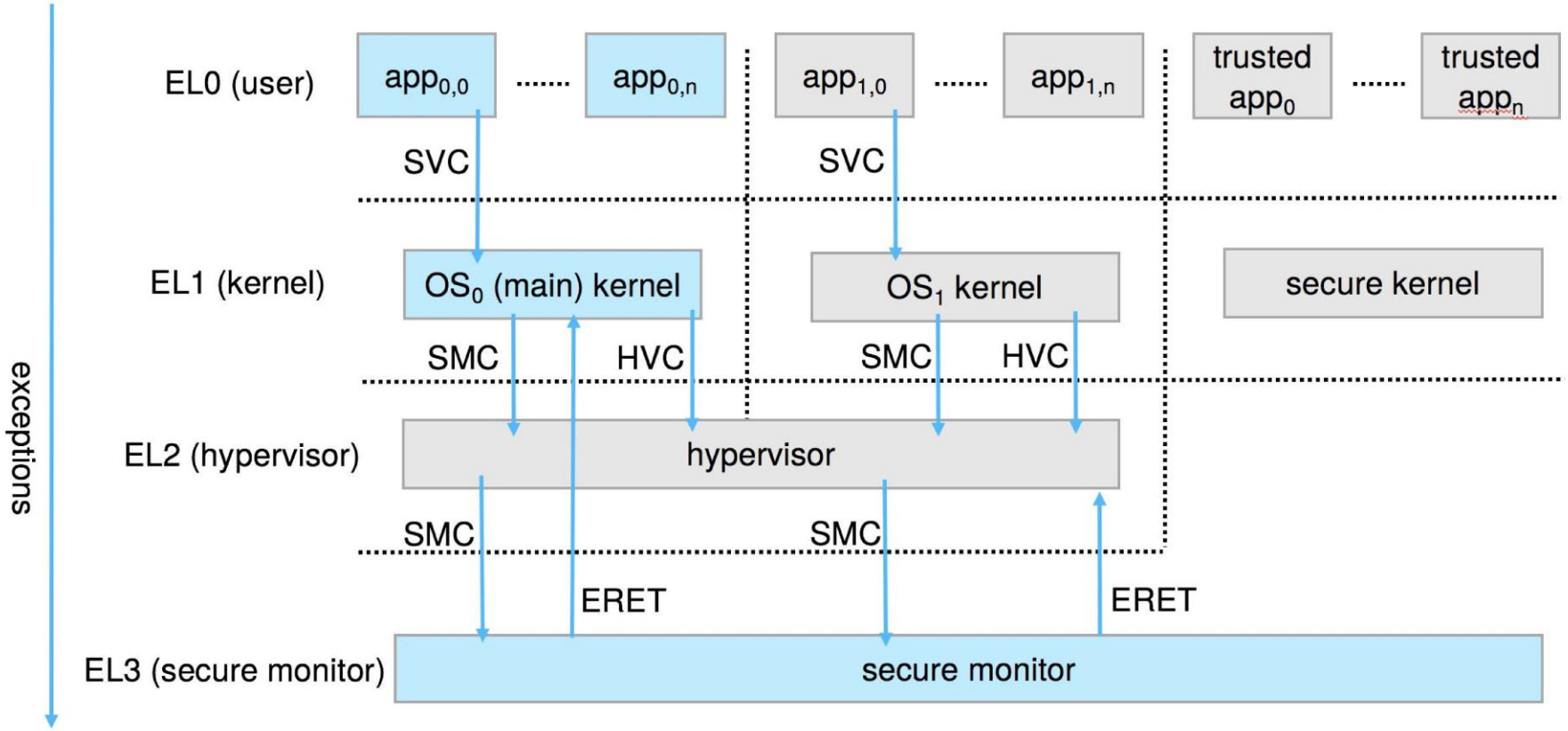
1. Let D_i and D_j be any two domain rings
2. If $j < I \Rightarrow D_i \subseteq D_j$



Protection Rings (MULTICS)



Android use of TrustZone



ARM CPU Architecture

Domain of Protection

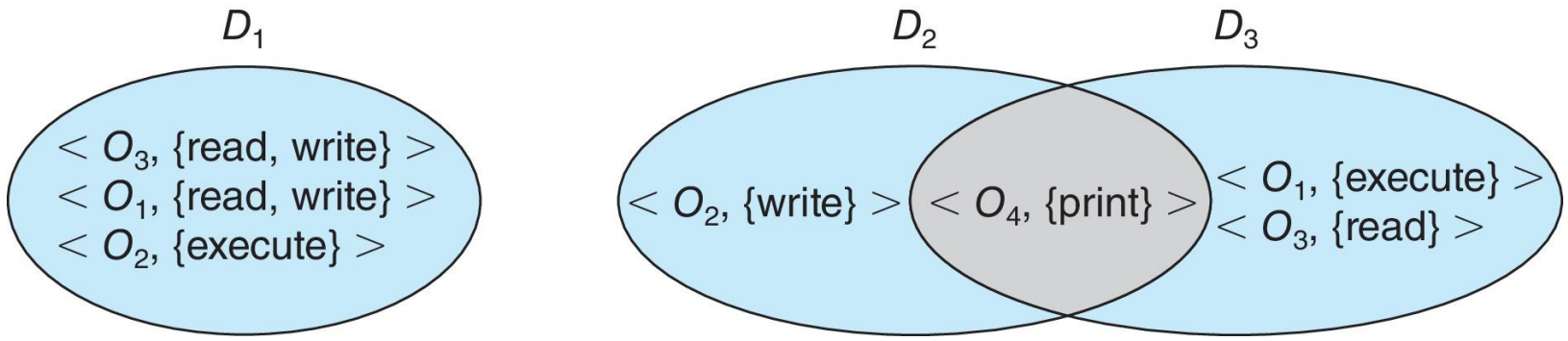
- Rings of protection separate functions into domains and order them hierarchically
- Computer can be treated as processes and objects
  - **Hardware objects** (such as devices) and **software objects** (such as files, programs, semaphores
- Process for example should only have access to objects it currently requires to complete its task – the **need-to-know** principle

Domain of Protection (Cont.)

- Implementation can be via a process operating in a **protection domain**
  - Specifies the resources that the process may access
  - Each domain specifies a set of objects and types of operations on them
  - The ability to execute an operation on an object is an **access right**
    - <object-name, rights-set>
  - Domains may share access rights
  - Associations can be **static** or **dynamic**
  - If dynamic, processes can **domain switch**

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



Domain Implementation (UNIX)

- Domain = **`user-id`**
- Domain switch accomplished via the file system
  - Each file is associated with it a domain bit (*setuid bit*)
  - When a file is executed and setuid = on, then **user-id is set to the owner of the file** being executed
  - When execution completes user-id is reset
- Domain switch accomplished via passwords
  - `su` command temporarily switches to another user's domain when the other domain's password provided
- Domain switching via commands
  - `Sudo` command prefix executes a specified command in another domain (if the original domain has privilege or password given)

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

- If a process in Domain D_i tries to do “op” on object O_j, then “op” must be in the access matrix
- **The user who creates an object can define an access column for that object**
- Can be expanded to dynamic protection
  - Operations to add or delete access rights
  - Special access rights:
    - *owner of* O_i
    - *copy op from* O_i *to* O_j *(denoted by “*”)*
    - *control – can* D_i *modify* D_j *access rights*
    - *transfer – switch from domain* D_i *to* D_j

Use of Access Matrix (Cont.)

- **Access matrix** design separates the mechanism from the policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - If it 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
- But it doesn't solve the general confinement problem

Access Matrix: Domain switch

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			

Access Matrix: Domain control

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			

Access Matrix with *Copy* Rights

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)

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)

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 ∈ 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 -> If M ∈ default set, also allow access

- Each column = Access-control list for one object
  Defines who can perform what operation

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

Implementation of Access Matrix (Cont.)

Option 3 – Capability list for domains

- Instead of object-based, the list is domain-based
- **Capability list** for the domain is a list of objects, together with operations allowed on them
- Object represented by its name or address, called a **capability**
- Execute operation M on object O_j, process requests operation, and specify capability as a parameter
  - Possession of capability means access is allowed
- **Capability list associated with domain but never directly accessible by domain**
  - Rather, protected object, maintained by OS and accessed indirectly
  - Like a “secure pointer”
  - Idea can be extended up to applications

Implementation of Access Matrix (Cont.)

- **Option 4 – Lock-key**
  - Compromise between access lists and capability lists
  - Each object has list of unique bit patterns, called **locks**
  - Each domain as list of unique bit patterns called **keys**
  - Process in a domain can only access object if domain has key that matches one of the locks

Comparison of Implementations

Many trade-offs to consider

- **Global table** is simple but can be large
- **Access lists** correspond to the needs of users
  - Determining a set of access rights for domains is non-localized, so it is difficult
  - Every access to an object must be checked
    - Many objects and access rights -> slow
- **Capability lists** are useful for localizing information for a given process
  - But revocation capabilities can be inefficient
- **Lock-key** is practical and flexible, keys can be passed freely from domain to domain, and easy revocation

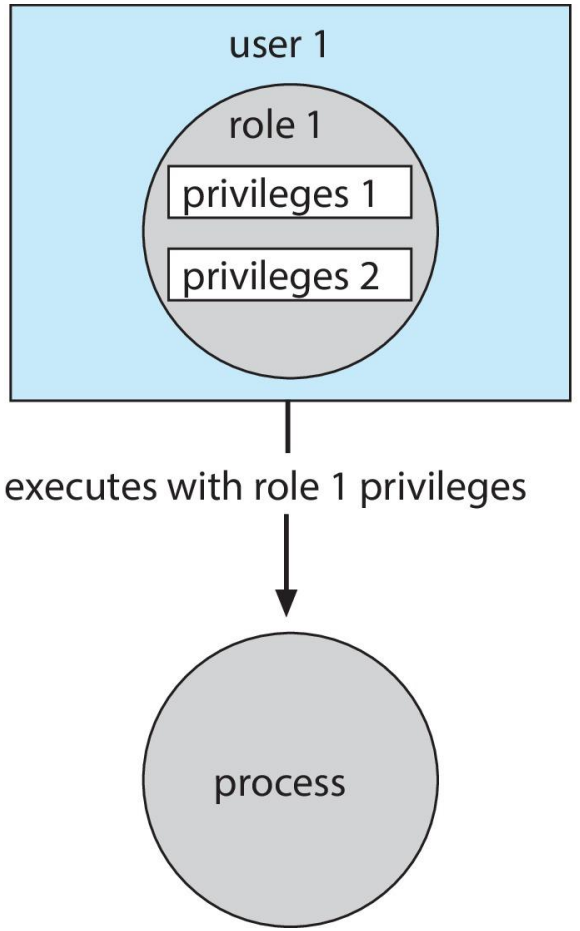
Comparison of Implementations (Cont.)

Most systems use a combination of access lists and capabilities

- First access to an object -> access list searched
  - If allowed, a capability is created and attached to the process
    - Additional accesses need not be checked
  - After the last access, the capability is destroyed
  - Consider a file system with ACLs per file

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



Thank you

Juan F. Medina

Juan.medina26@upr.edu