

# OPERATING SYSTEM STRUCTURE

## CHAPTER # 2

## Operating Systems

# Chapter 2 - Outline

- ❑ Process Management
- ❑ Memory Management
- ❑ File Management
- ❑ I/O System Management
- ❑ Secondary Storage Management
- ❑ Networking
- ❑ System Protection
- ❑ Operating System Services
- ❑ OS Layered Approach
  - ❑ OS/2 Layer Structure
- ❑ Virtual Machines
- ❑ System Design Goals
- ❑ Mechanisms and Policies
- ❑ Operating System Implementation
- ❑ System Generation (SYSGEN)

# Process Management

- A process is a program in execution
- A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task
- The operating system is responsible for the following activities in connection with process manager
  - ▣ Process creation and termination
  - ▣ Process suspension and resumption
  - ▣ Provision of mechanisms for:
    - process synchronization
    - process communication

# Memory Management

- Memory is a large array of words or bytes, each with its own address
- It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a volatile storage device
  - ▣ It loses its contents in the case of system failure
- The operating system is responsible for the following activities in connections with memory management:
  - ▣ Keep track of which parts of memory are currently being used and by whom
  - ▣ Decide which processes to load when memory space becomes available
  - ▣ Allocate and deallocate memory space as needed

# File Management

- A file is a collection of related information defined by its creator
- Commonly, files represent programs (both source and object forms) and data
- The operating system is responsible for the following activities in connections with file management:
  - ▣ File creation and deletion
  - ▣ Directory creation and deletion
  - ▣ Support of primitives for manipulating files and directories
  - ▣ Mapping files onto secondary storage
  - ▣ File backup on stable (nonvolatile) storage media

# I/O System Management

- Operating system is responsible for I/O system management
- The I/O system consists of:
  - ▣ A buffer-caching system
  - ▣ A general device-driver interface
  - ▣ Drivers for specific hardware devices

# Secondary Storage Management

- Since main memory (primary storage) is volatile and too small to accommodate all data and programs permanently, the computer system must provide secondary storage to back up main memory
- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data
- The operating system is responsible for the following activities in connection with disk management:
  - ▣ Free space management
  - ▣ Storage allocation
  - ▣ Disk scheduling

# Networked System

- A networking system is a collection of processors
- The processors in the system are connected through a communication network
- Communication takes place using a protocol
- This system provides user access to various system resources
- Access to a shared resource allows:
  - ▣ Computation speed-up
  - ▣ Increased data availability
  - ▣ Enhanced reliability



# System Protection

- Protection refers to a mechanism for controlling access by programs, processes, or users to both system and user resources
- The protection mechanism must:
  - ▣ distinguish between authorized and unauthorized access
  - ▣ specify the controls to be imposed
  - ▣ provide a means of enforcement

# Operating System Services

- Operating systems provide an environment for execution of programs and services to programs and users
- A set of operating-system services provides functions that are helpful to the user
- **User interface**
  - ▣ Almost all operating systems have a user interface (**UI**)
    - Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **Batch**
- **Program execution**
  - ▣ The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)

# Operating System Services

SHEHERYAR MALIK

## □ I/O operations

- since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O

## □ File-system manipulation

- Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management
- Operating system helps the programs to achieve that

# Operating System Services

## □ Communications

- ▣ Processes may exchange information, on the same computer or between computers over a network
- ▣ Communications may be via shared memory or through message passing (packets moved by the OS)

## □ Error detection

- ▣ OS needs to be constantly aware of possible errors
  - May occur in the CPU and memory hardware, in I/O devices, in user program
  - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
  - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

# Operating System Services

## □ Resource allocation

- ▣ When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
  - Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code

## □ Accounting

- ▣ To keep track of which users use how much and what kinds of computer resources

# Operating System Services

## □ Protection and security

- ▣ The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other

### ▣ Protection

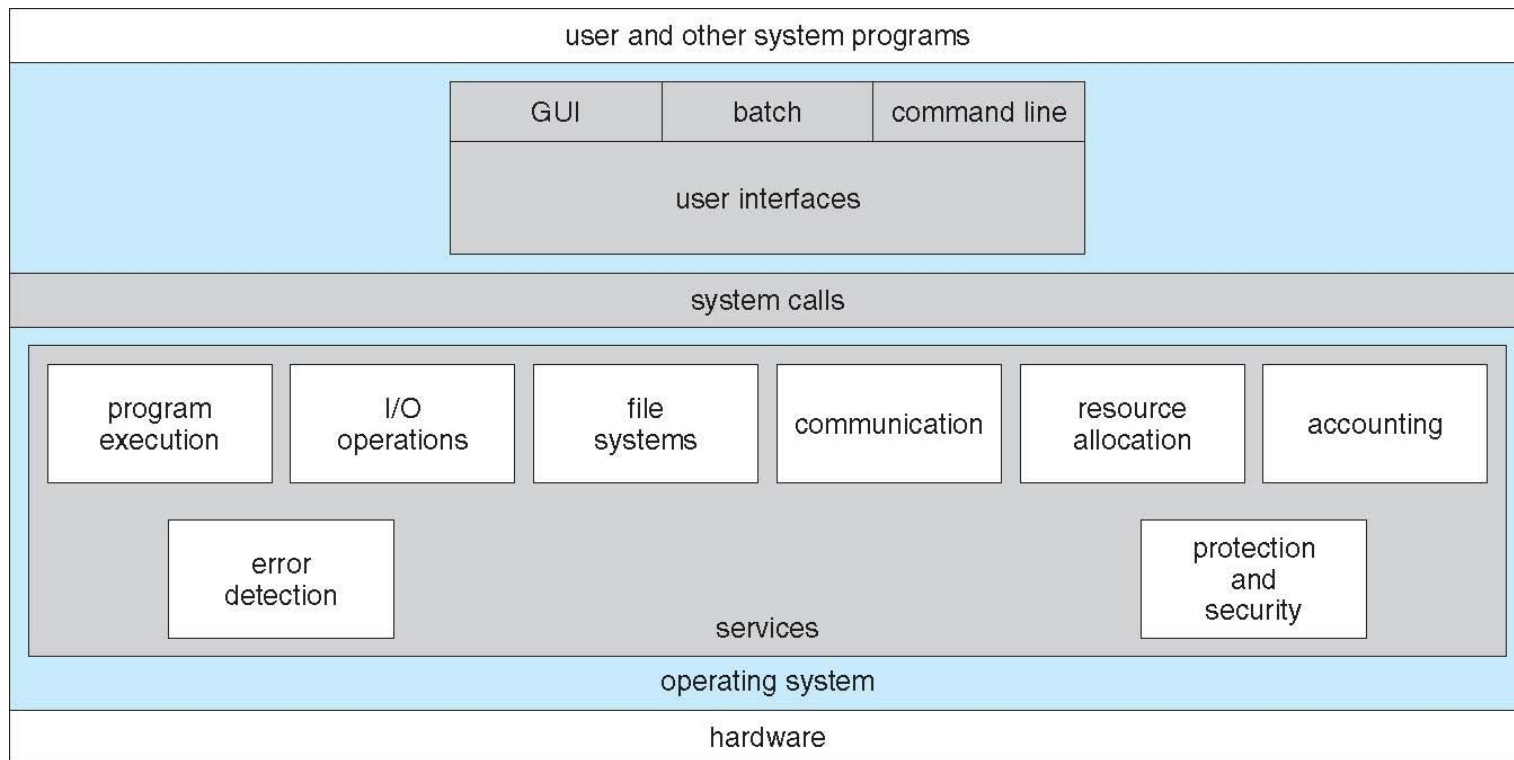
- involves ensuring that all access to system resources is controlled

### ▣ Security

- of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
  - If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link

# A View of Operating System Services

SHEHERYAR MALIK



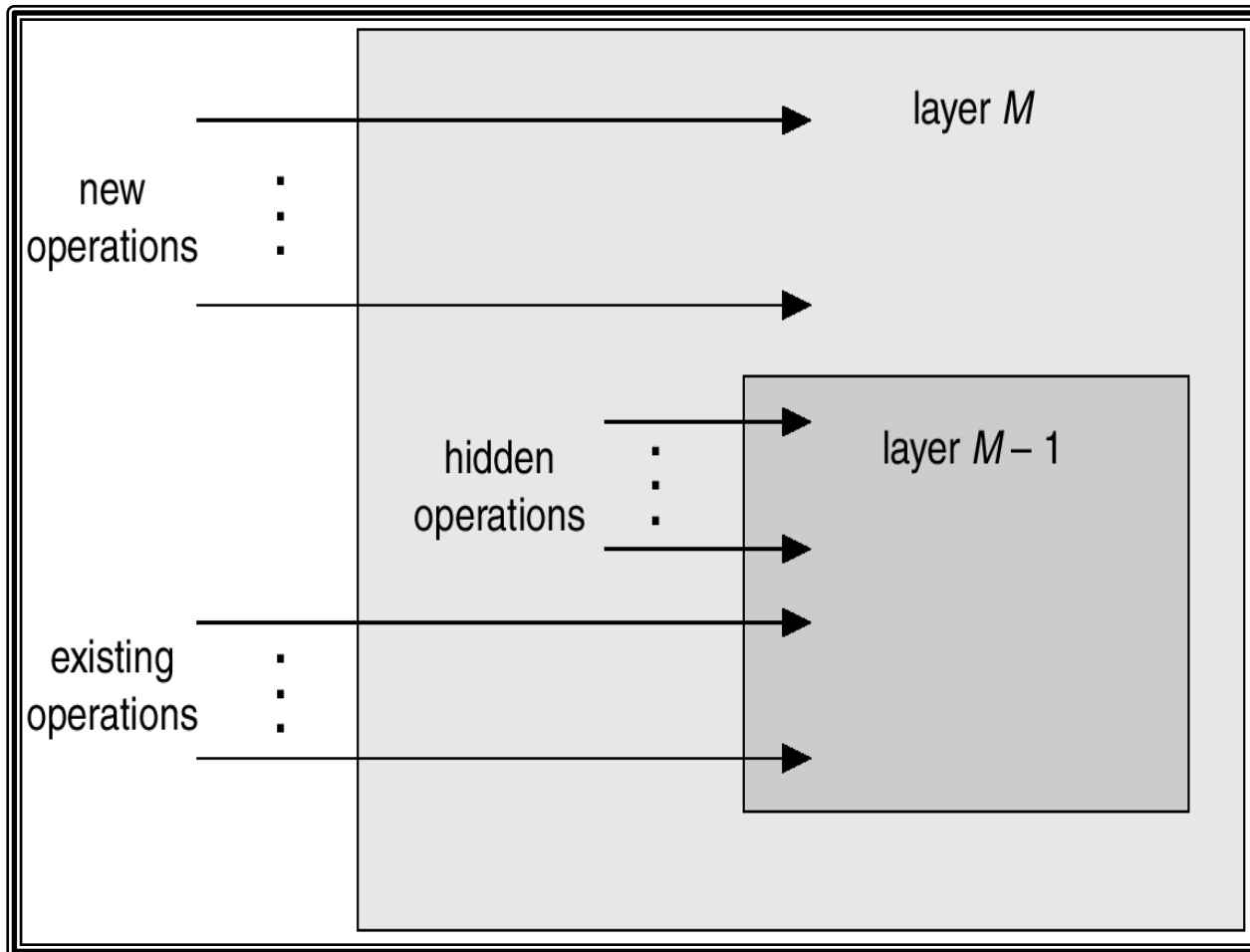
# OS Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers
  - ▣ The bottom layer (layer 0), is the hardware
  - ▣ The highest (layer N) is the user interface
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

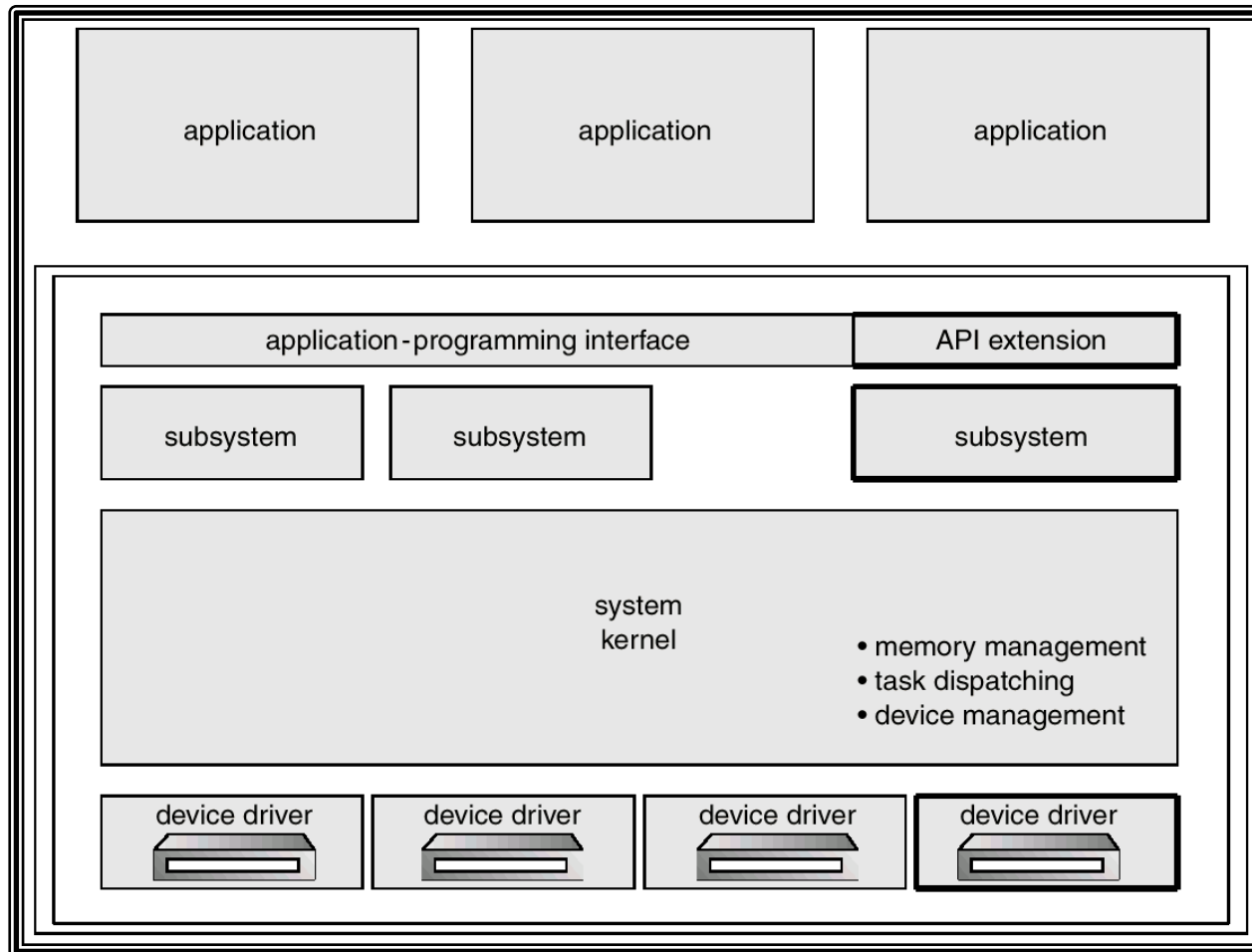


# An Operating System Layer

SHEHRYAR MALIK



# OS/2 Layer Structure

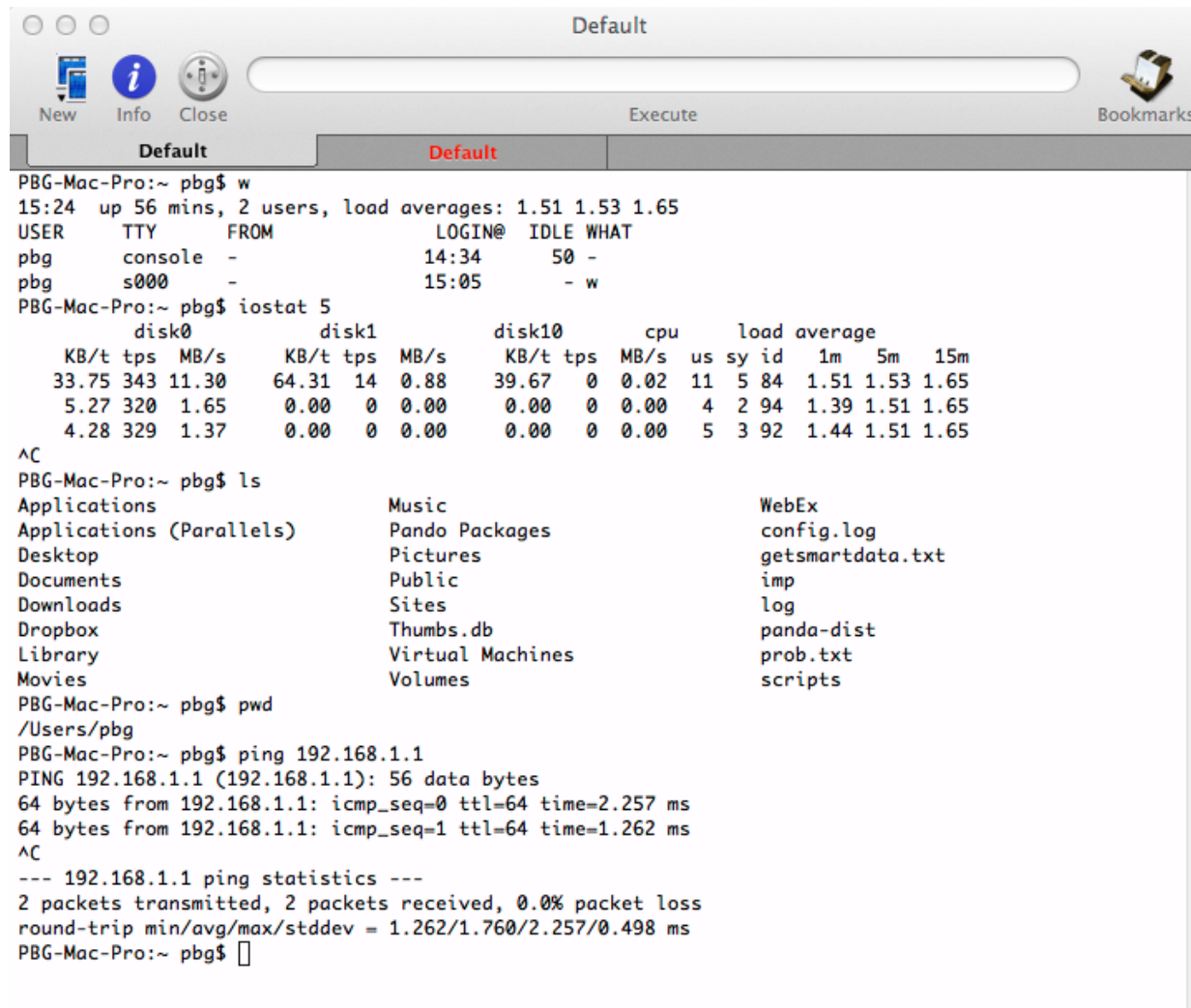


# User Operating System Interface - CLI

SHEHERYAR MALIK

- CLI or **command interpreter** allows direct command entry
  - Sometimes implemented in kernel, sometimes by systems program
  - Sometimes multiple flavors implemented – **shells**
  - Primarily fetches a command from user and executes it
    - Sometimes commands built-in, sometimes just names of programs
      - If the latter, adding new features doesn't require shell modification

# Bourne Shell Command Interpreter



```
Default
New Info Close Execute Bookmarks
Default Default
PBG-Mac-Pro:~ pbg$ w
15:24 up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER      TTY      FROM          LOGIN@  IDLE WHAT
pbg       console -            14:34    50 -
pbg       s000    -            15:05    - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0          disk1          disk10          cpu          load average
      KB/t tps MB/s      KB/t tps MB/s      KB/t tps MB/s  us sy id  1m  5m  15m
      33.75 343 11.30      64.31 14  0.88      39.67 0  0.02  11  5 84  1.51 1.53 1.65
      5.27 320  1.65        0.00 0  0.00        0.00 0  0.00   4  2 94  1.39 1.51 1.65
      4.28 329  1.37        0.00 0  0.00        0.00 0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications          Music                  WebEx
Applications (Parallels)  Pando Packages        config.log
Desktop               Pictures               getsmartdata.txt
Documents              Public                 imp
Downloads              Sites                  log
Dropbox                Thumbs.db              panda-dist
Library                Virtual Machines       prob.txt
Movies                 Volumes                scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$
```

# User Operating System Interface - GUI

SHEHERYAR MALIK

- User-friendly **desktop** metaphor interface
  - ▣ Usually mouse, keyboard, and monitor
  - ▣ **Icons** represent files, programs, actions, etc
  - ▣ Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**))
  - ▣ Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - ▣ Microsoft Windows is GUI with CLI “command” shell
  - ▣ Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
  - ▣ Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

# Touch-screen Interfaces

SHEHERYAR MALIK

- Touch-screen devices require new interfaces
  - ▣ Mouse not possible or not desired
  - ▣ Actions and selection based on gestures
  - ▣ Virtual keyboard for text entry



# System Programs

- System programs provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- They can be divided into:
  - ▣ File manipulation
  - ▣ Status information sometimes stored in a File modification
  - ▣ Programming language support
  - ▣ Program loading and execution
  - ▣ Communications
  - ▣ Background services
- Most users' view of the operation system is defined by system programs, not the actual system calls

# System Programs

## □ File management

- ▣ Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

## □ Status information

- ▣ Some ask the system for info - date, time, amount of available memory, disk space, number of users
- ▣ Others provide detailed performance, logging, and debugging information
- ▣ Typically, these programs format and print the output to the terminal or other output devices
- ▣ Some systems implement a **registry** - used to store and retrieve configuration information



# System Programs

## □ File modification

- ▣ Text editors to create and modify files
- ▣ Special commands to search contents of files or perform transformations of the text

## □ Programming-language support

- ▣ Compilers, assemblers, debuggers and interpreters sometimes provided

## □ Program loading and execution

- ▣ Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language

## □ Communications

- ▣ Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

## □ Background Services

- ▣ Launch at boot time
  - Some for system startup, then terminate
  - Some from system boot to shutdown
- ▣ Provide facilities like disk checking, process scheduling, error logging, printing
- ▣ Run in user context not kernel context
- ▣ Known as **services, subsystems, daemons**

# Operating System Design and Implementation

SHEHERYAR MALIK

- Design and Implementation of OS not “solvable”, but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system
- **User** goals and **System** goals

# OS Design Goals

- User goals
  - ▣ Operating system should be
    - convenient to use
    - easy to learn
    - reliable
    - safe
    - fast
- System goals
  - ▣ Operating system should be
    - easy to design
    - implement
    - maintain
    - flexible
    - reliable
    - error-free
    - efficient

# Operating System Design and Implementation

SHEHERYAR MALIK

- Important principle to separate
  - **Policy:** *What* will be done?
  - **Mechanism:** *How* to do it?
- Mechanisms determine how to do something, policies decide what will be done
  - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later
- Specifying and designing OS is highly creative task of **software engineering**

# Operating System Implementation

SHEHERYAR MALIK

- Much variation
  - ▣ Early OSES in assembly language
  - ▣ Then system programming languages like Algol, PL/1
  - ▣ Now C, C++
- Actually usually a mix of languages
  - ▣ Lowest levels in assembly
  - ▣ Main body in C
  - ▣ Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- Code written in a high-level language:
  - ▣ can be written faster
  - ▣ is more compact
  - ▣ is easier to understand and debug
- An operating system is far easier to port (move to some other hardware) if it is written in a high-level language

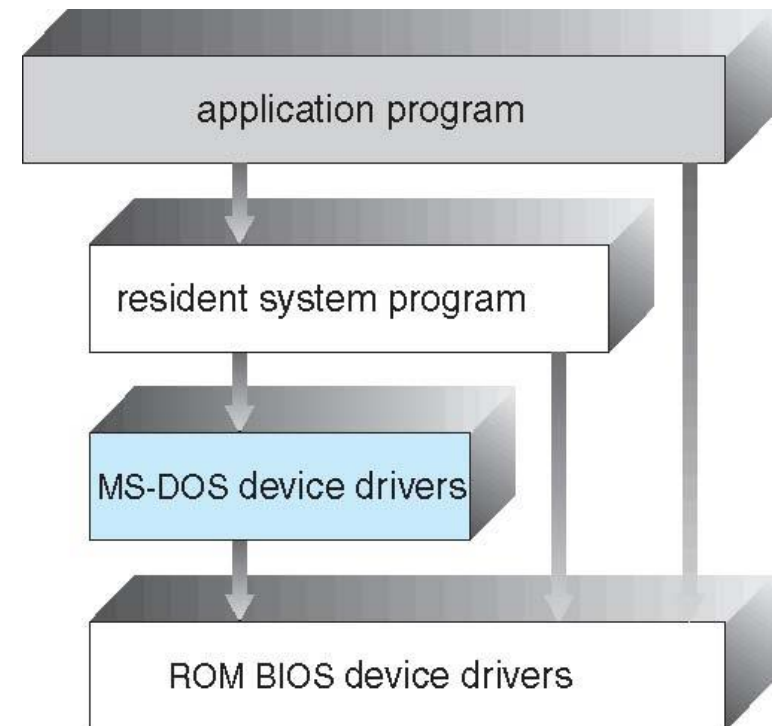
# Operating System Structure

SHEHERYAR MALIK

- General-purpose OS is very large program
- Various ways to structure one as follows

# Simple Structure

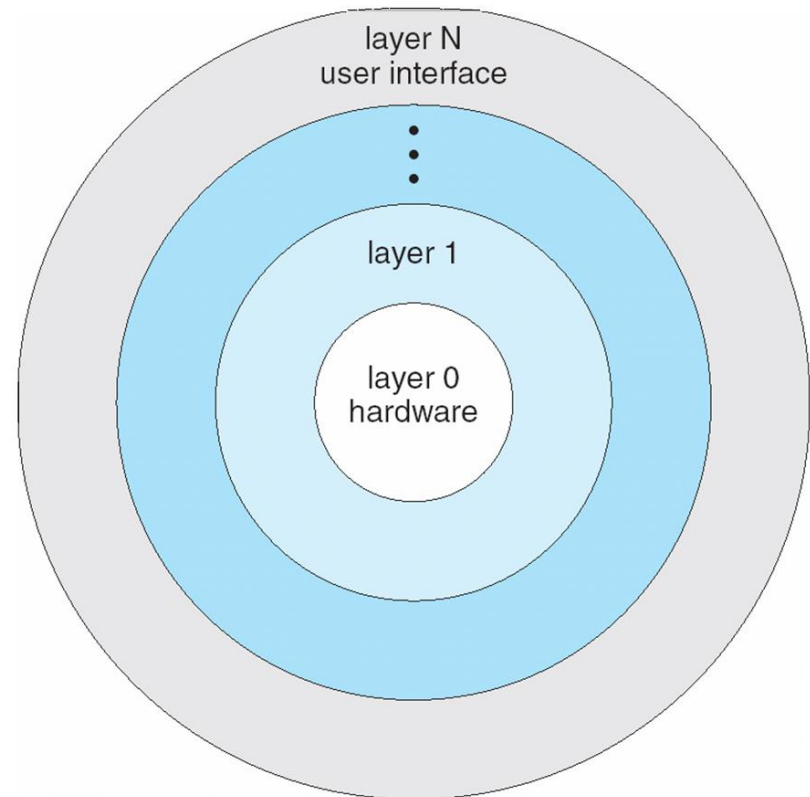
- I.e. MS-DOS – written to provide the most functionality in the least space
  - ▣ Not divided into modules
  - ▣ Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated





# Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

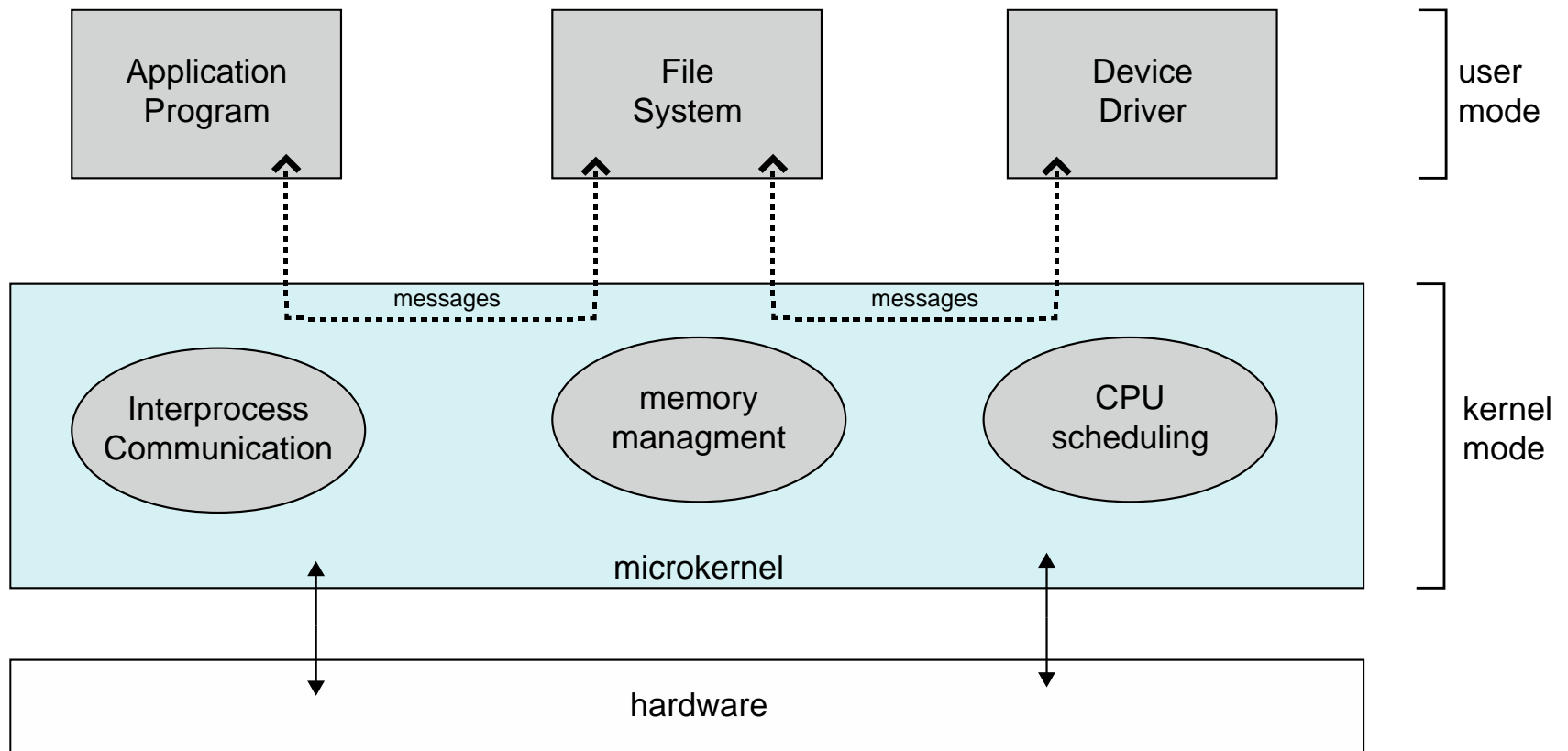


# Microkernel System Structure

- Monolithic (Old approach)
  - Basic system services like process and memory management, interrupt handling etc. were packed into a single module in kernel space.
  - Some serious problems like,
    - Size was huge,
    - Poor maintainability (bug fixing and addition of new features resulted in recompilation of the whole kernel code which could consume hours.
  - E.g. MS-DOS
- Moves as much from the kernel into user space
- **Mach** example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

# Microkernel System Structure

SHEHERYAR MALIK

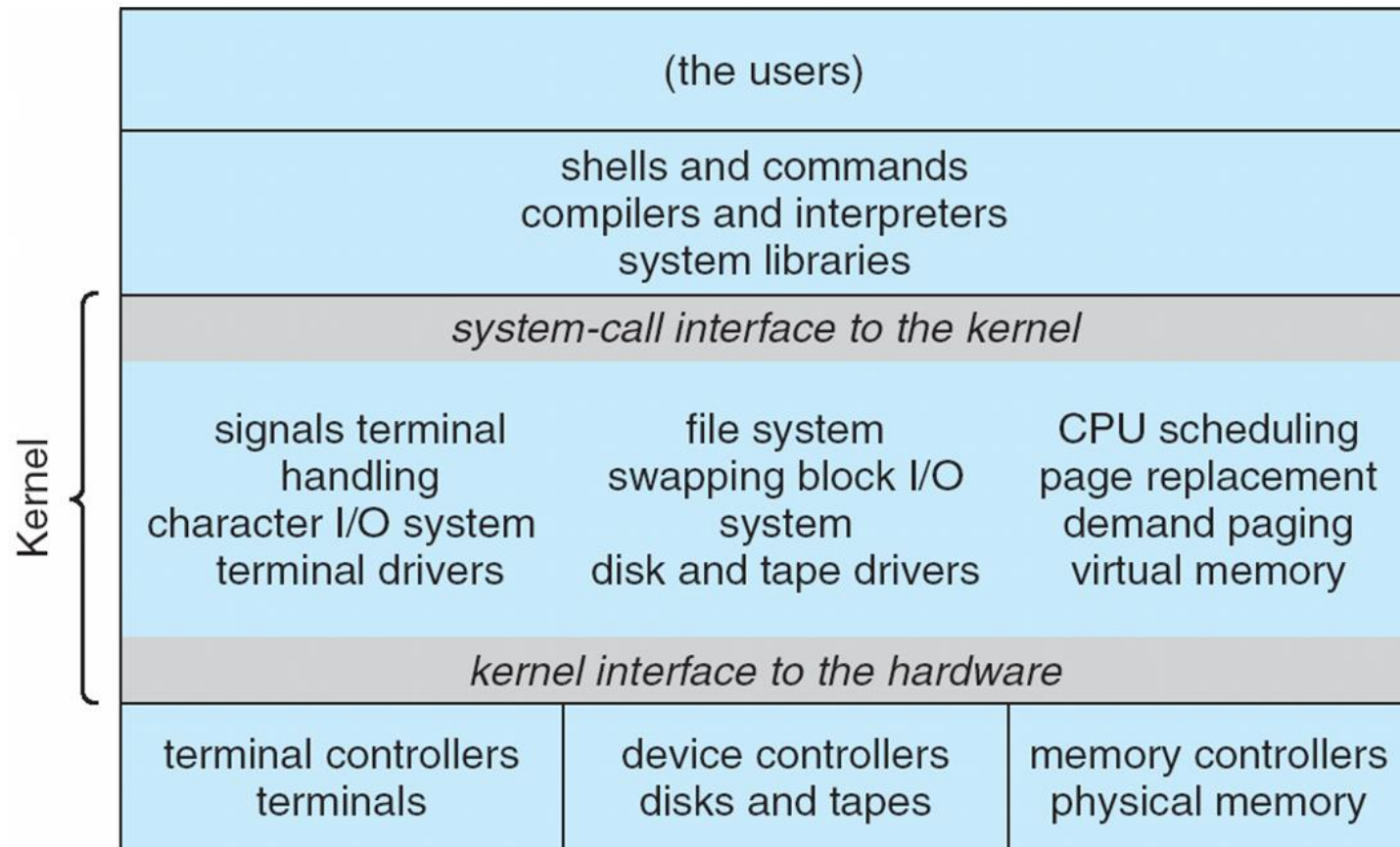


- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring
- The UNIX OS consists of two separable parts
  - ▣ Systems programs
  - ▣ The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

# Traditional UNIX System Structure

SHEHERYAR MALIK

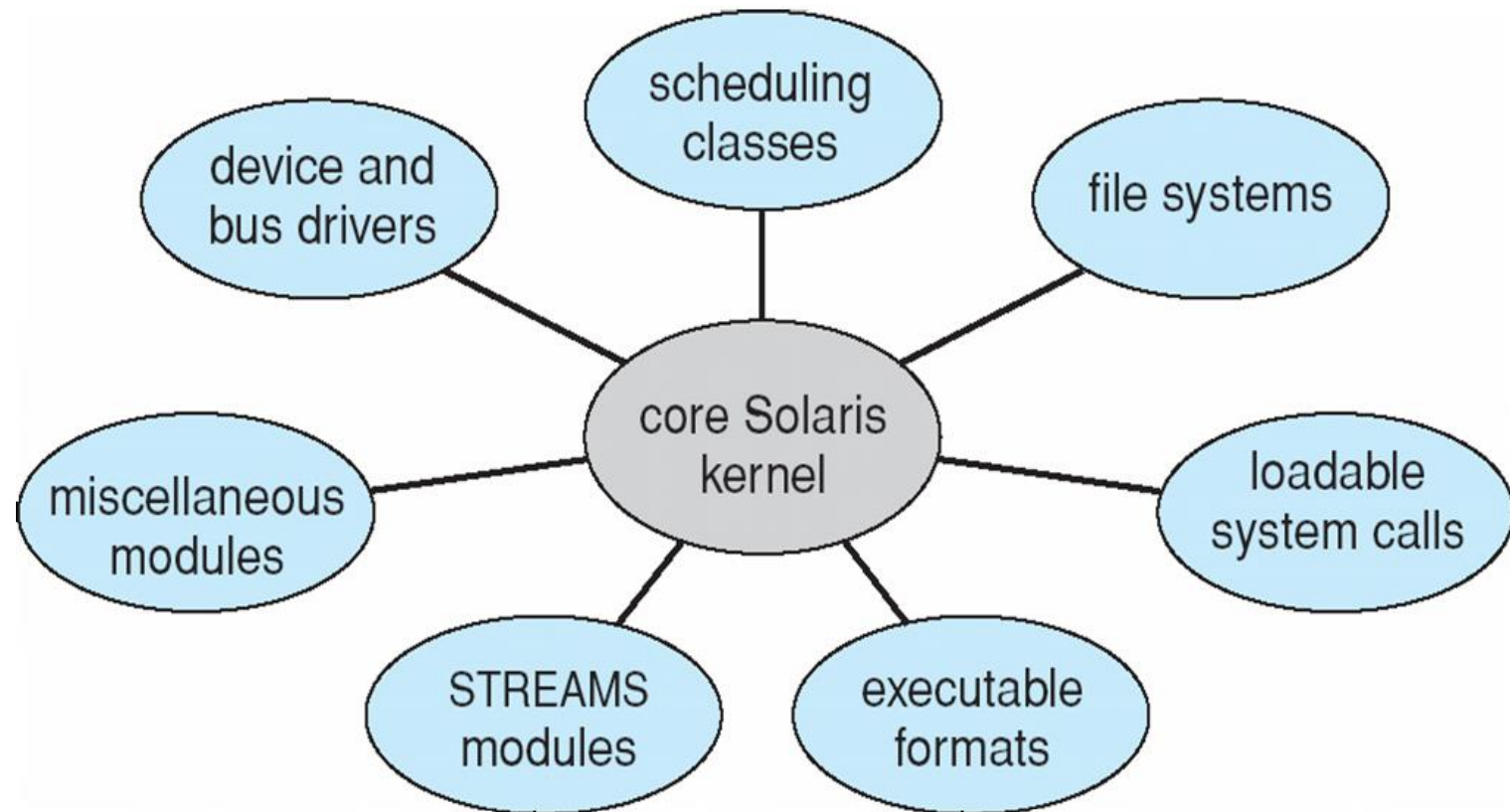
Beyond simple but not fully layered



# Modules

- Most modern operating systems implement **loadable kernel modules**
  - ▣ Uses object-oriented approach
  - ▣ Each core component is separate
  - ▣ Each talks to the others over known interfaces
  - ▣ Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
  - ▣ Linux, Solaris, etc

# Solaris Modular Approach



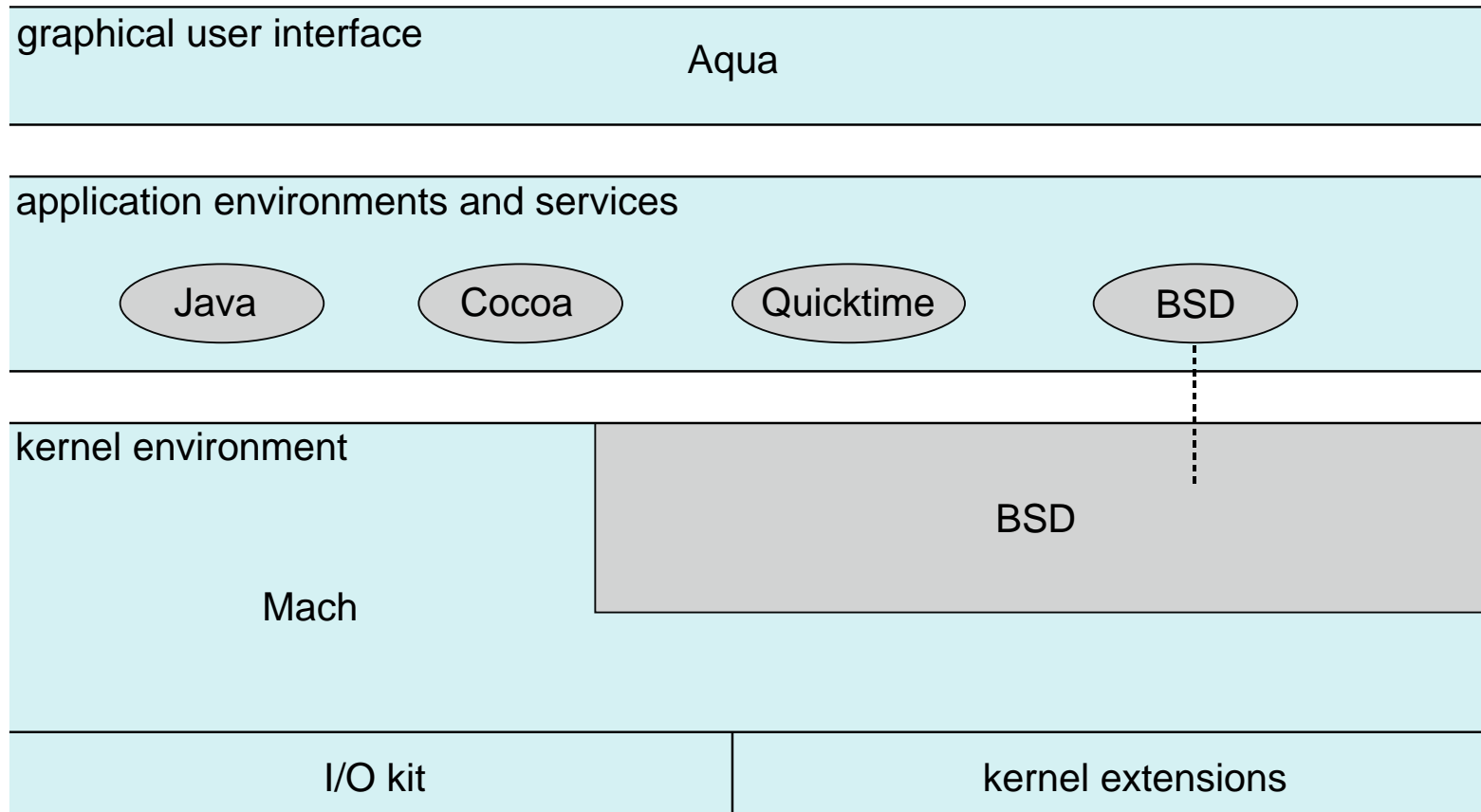
# Hybrid Systems

- Most modern operating systems actually not one pure model
  - ▣ Hybrid combines multiple approaches to address performance, security, usability needs
  - ▣ Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - ▣ Windows mostly monolithic, plus microkernel for different subsystem *personalities*
- Apple Mac OS X hybrid, layered, Aqua UI plus Cocoa programming environment
  - ▣ Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)



# Mac OS X Structure

SHEHERYAR MALIK



- Apple mobile OS for *iPhone, iPad*
  - ▣ Structured on Mac OS X, added functionality
  - ▣ Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - ▣ **Cocoa Touch** Objective-C API for developing apps
  - ▣ **Media services** layer for graphics, audio, video
  - ▣ **Core services** provides cloud computing, databases
  - ▣ Core operating system, based on Mac OS X kernel

Cocoa Touch

Media Services

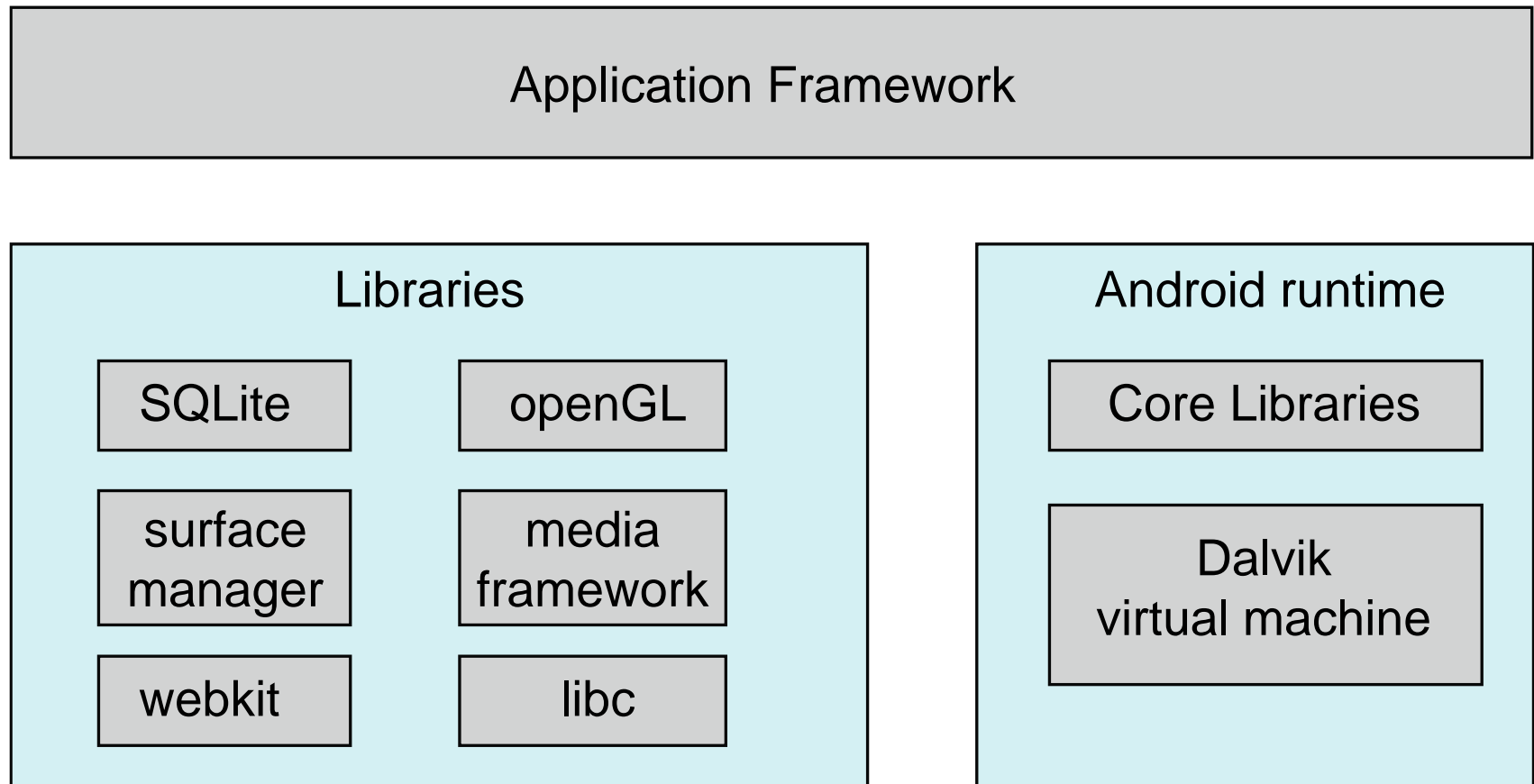
Core Services

Core OS

# Android

- Developed by Open Handset Alliance (mostly Google)
  - ▣ Open Source
- Similar stack to iOS
- Based on Linux kernel but modified
  - ▣ Provides process, memory, device-driver management
  - ▣ Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - ▣ Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

# Android Architecture



# Operating-System Debugging

SHEHERYAR MALIK

- **Debugging** is finding and fixing errors, or **bugs**
- OS generates **log files** containing error information
- Core Dump
  - ▣ Failure of an application can generate **core dump** file capturing memory of the process
- Crash Dump
  - ▣ Operating system failure can generate **crash dump** file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
  - ▣ Sometimes using ***trace listings*** of activities, recorded for analysis
  - ▣ **Profiling** is periodic sampling of instruction pointer to look for statistical trends

# Operating System Generation (SYSGEN)

SHEHERYAR MALIK

- Operating systems are designed to run on any of a class of machines
  - ▣ the system must be configured for each specific computer site
- **SYSGEN** program obtains information concerning the specific configuration of the hardware system
  - ▣ Used to build system-specific compiled kernel or system-tuned
  - ▣ Can generate more efficient code than one general kernel

# Operating System Generation (SYSGEN)

SHEHERYAR MALIK

- When power initialized on system, execution starts at a fixed memory location
  - ▣ Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - ▣ Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - ▣ Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**

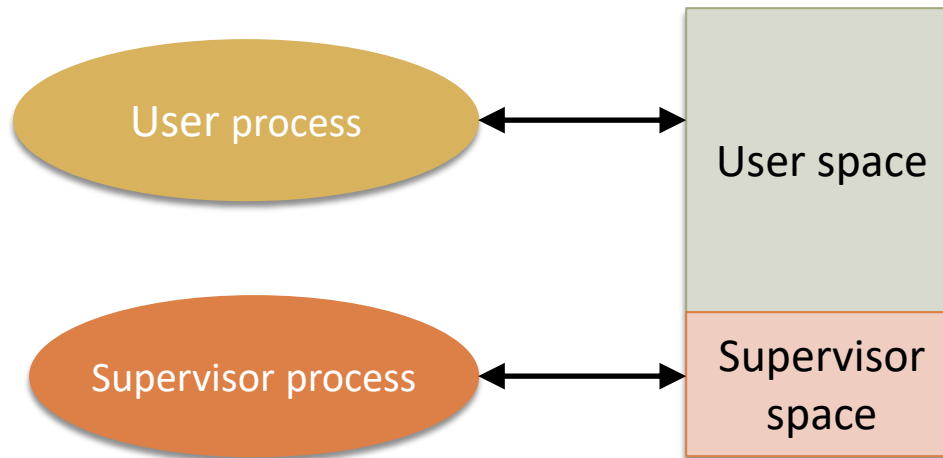
# Kernel/Kernel Mode

- Kernel is part of operating system that includes most heavily used functions
- Generally kernel is permanently in main memory
- A privileged, nucleus or supervisory mode into which the machine switches when the OS is running
- Enables the execution of such privileged instructions
  - ▣ Such as in the area of I/O operations, which are not available to user mode
- The techniques by which a user program requests the kernel's services are;
  - ▣ System calls
  - ▣ Message passing
- The process can only be switched into kernel mode by the above actions



# Kernel/Kernel Mode

- No other way that a user program can execute kernel instructions
- This facility is not given to user program in order to enforce system security



**Supervisor & user memory**

# System Calls

- User program communicate with OS and request its services by making system calls
- Corresponding to each system call is a library procedure that user program can call
- System calls provide the interface between a running program and the OS
- These calls are generally available as assembly language instruction
- All instruction with the hardware is implemented by system calls
- OS provides an additional layer of subroutines called an API (application programming interface), between programmer and system call interface

# System Calls

- ❑ To compile a program, a process calls the command interpreter or shell, reads command from the terminal
- ❑ Shell will create a new process to compile the program
- ❑ When the process has completed compilation, it executes a system call to terminate itself
- ❑ Other process system calls are also available to;
  - ❑ Request for more memory
  - ❑ Released unused/extra memory
  - ❑ Process and memory management

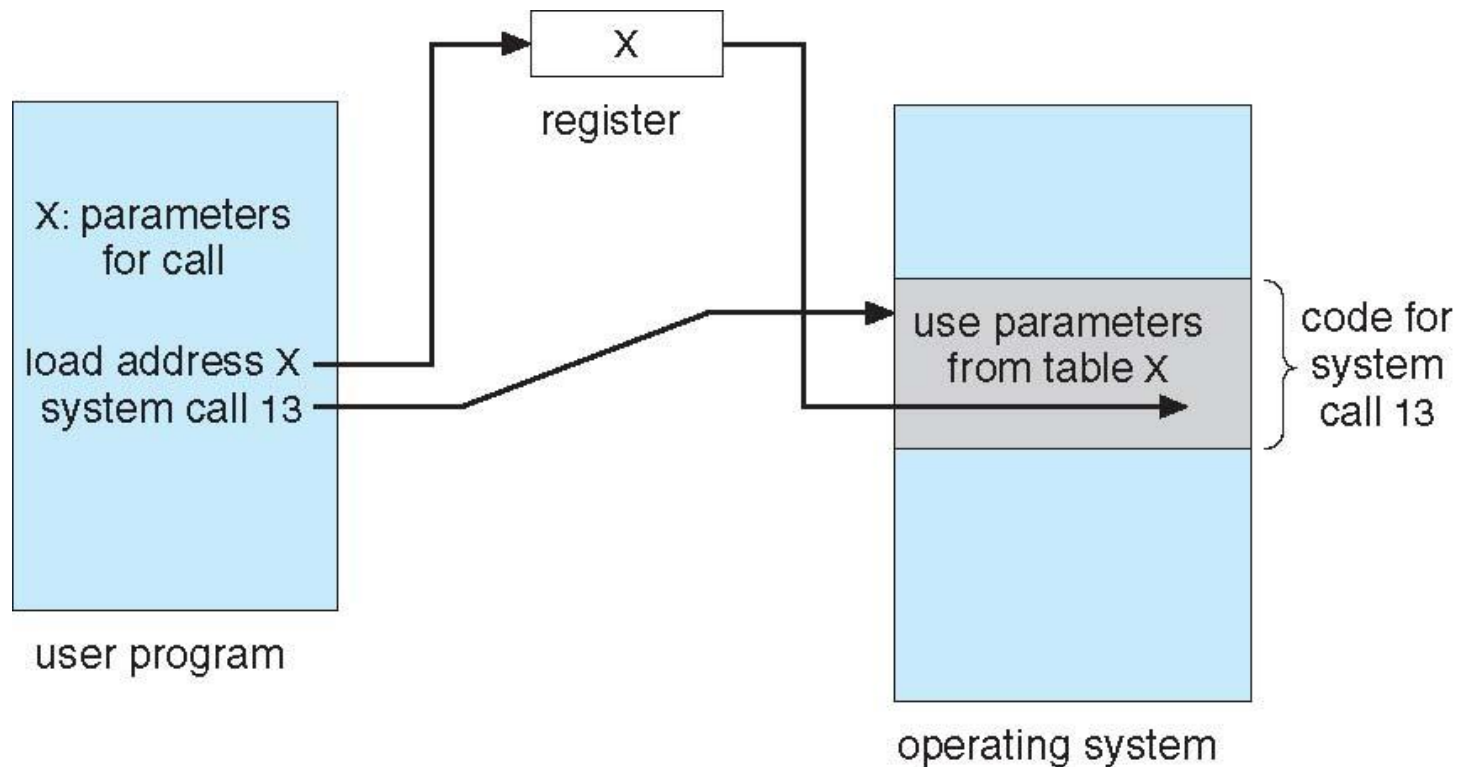
# System Calls

- System calls related to file systems;
  - ▣ System calls are needed to create, read, move and write a file
  - ▣ A file should be opened before read it
  - ▣ Similarly, after reading file must be closed which is done by system calls
  - ▣ System calls are required to
    - create and remove a directory
    - manipulate file with in different directories
- System V Unix provides about 64 system calls e.g.
  - ▣ `create(name, amode)`
  - ▣ `open(name, oflag, amode)`
  - ▣ `close(fd)`
- MS-DOS uses software interrupts similar to system calls

# System Call Processing

- Three general methods are used to pass parameters between a running program and the operating system
  - ▣ Pass parameters in registers
  - ▣ Store the parameters in a table in memory, and the table address is passed as a parameter in a register
  - ▣ Push (store) the parameters onto the stack by the program, and pop off the stack by operating system

# System Call Processing



# Types of System Calls

- ❑ Process control
- ❑ File management
- ❑ Device management
- ❑ Information maintenance
- ❑ Communications

# Types of System Calls

- Process control
  - ▣ end, abort
  - ▣ load, execute
  - ▣ create process, terminate process
  - ▣ get process attributes, set process attributes
  - ▣ wait for time
  - ▣ wait event, signal event
  - ▣ allocate and free memory
  
- ▣ Dump memory if error
- ▣ **Debugger** for determining **bugs, single step** execution
- ▣ **Locks** for managing access to shared data between processes



# Types of System Calls

- File management
  - ▣ create file, delete file
  - ▣ open, close file
  - ▣ read, write, reposition
  - ▣ get and set file attributes
  
- Device management
  - ▣ request device, release device
  - ▣ read, write, reposition
  - ▣ get device attributes, set device attributes
  - ▣ logically attach or detach devices

# Types of System Calls (Cont.)

- Information maintenance
  - ▣ get time or date, set time or date
  - ▣ get system data, set system data
  - ▣ get and set process, file, or device attributes
  
- Communications
  - ▣ create, delete communication connection
  - ▣ send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  - ▣ **Shared-memory model** create and gain access to memory regions
  - ▣ transfer status information
  - ▣ attach and detach remote devices

# Types of System Calls (Cont.)

SHEHRYAR MALIK

- Protection
  - ▣ Control access to resources
  - ▣ Get and set permissions
  - ▣ Allow and deny user access

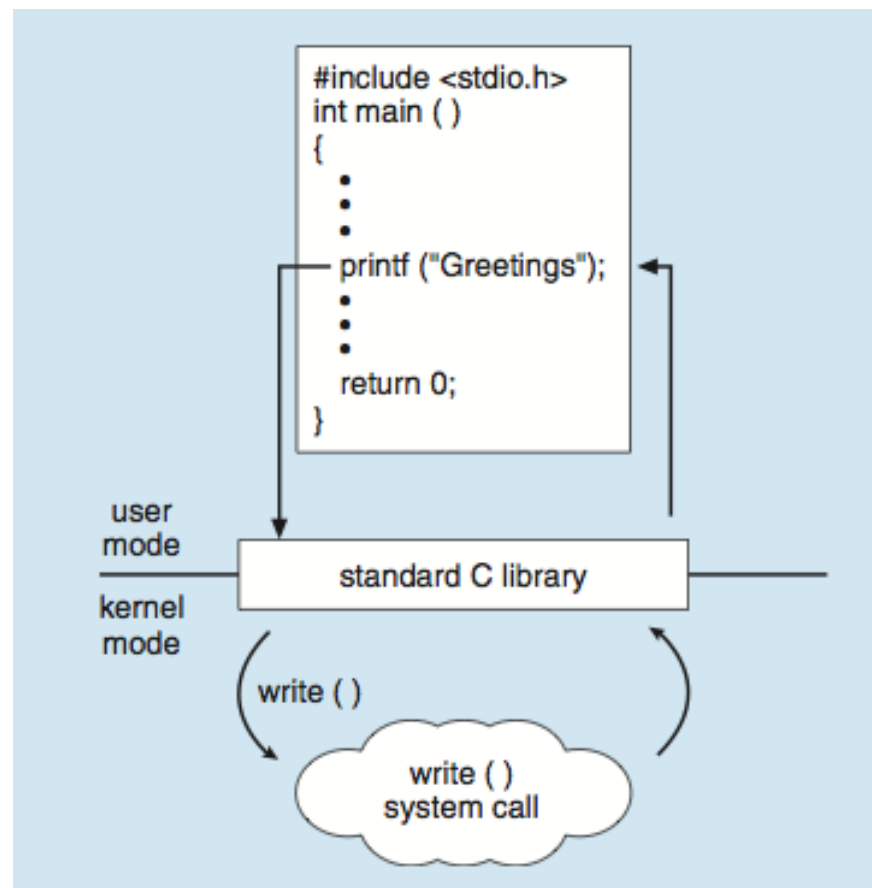
# Examples of Windows and Unix System Calls

SHEHERYAR MALIK

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

# Standard C Library Example

- C program invoking printf() library call, which calls write() system call



# Interrupts

- Mechanism by which hardware like I/O devices, memory modules may interrupt the normal processing of the processor
- Interrupts are generated by various agents to notify the OS of the occurrence of some event
  - ▣ such as completion of an I/O activity
- When an interrupt occurs, the execution flow of processor is diverted into specific part of OS code which deals with the event – *interrupt handler*
- Interrupts are used to permit several programs and I/O activities to process independently and asynchronously
  - ▣ Hence improve the processing speed

# Interrupts

- Advantages
  - ▣ It provides a low overheads means of gaining the attention of the CPU
  - ▣ *Interrupt Vector Table* is a location near the bottom of memory contains the address of interrupt services procedures of I/O devices
- As each instruction terminates, the processor may checks for the occurrence of interrupts
  - ▣ If an interrupt received, the actual program is temporarily suspended, and the processor is diverted to interrupt handling routine
  - ▣ When the interrupt is served, the execution will return to the interrupted program

# Interrupts

## □ Example:

- ▣ External devices like printer are much slower than the processor
- ▣ When processor is transferring data to the printer, the processor pause and remain idle until the printer is reading data
- ▣ I/O modules will send an interrupt request signal to the processor to suspend its current processing and send more data



# Interrupt Processing

- Following events occurs in processor hardware and software due to an I/O interrupt
  - ▣ The device issue an interrupt signal to processor
  - ▣ The processor finishes execution of current instruction, saves states of interrupted process before responding to the interrupt
  - ▣ The processor sends an acknowledgment signal to the device that issued the interrupt and device remove its interrupt signal/flag
  - ▣ Processor transfer control to the appropriate interrupt handler routine and it starts processing interrupt
  - ▣ When the interrupt id served, the state of interrupted process is restored
  - ▣ Next process starts execution

# Types of Interrupts

- Interrupts may be generated by a number of sources, which are following
- **I/O:** Generated by I/O device controller, to signal normal completion of event or the occurrence of an error or failure condition
- **Time:** Generated by an internal clock within the processor due to expiration of a process's time quantum or receipt of a signal from another processor on a multiprocessor system
- **Hardware error:** Generated by hardware faults such as power failure
- **Program:** Generated in a user program as a result of instruction execution, such as;
  - ▣ Arithmetic overflow
  - ▣ Divide by zero
  - ▣ Attempt to execute an illegal machine instruction
  - ▣ Reference out side a user allowed memory space