Operating Systems

CS235AI

Semester: III						
OPERATING SYSTEMS **(Theory and Lab)** **(Common to CS, IS, CD & CY)**						
Course Code	:	22CS35		CIE	:	100 Marks
Credits: L:T:P	:	3:0:1		SEE	:	100 Marks
Total Hours	:	45L + 30P		SEE Duration	:	3 Hours

Unit-I	09 Hrs.
Introduction- Perspectives Business domain: Virtualization and Cloud Computing Application: Traditional computing, Mobile computing, Distribut systems Introduction What Operating System do, Operating System structure, Operating system Operations. System Structures Operating system services, System Calls, Types of System calls Process Management Process concept, Process scheduling, Operations on processes	
Unit – II	09 Hrs.
Multithreaded programming Overview, Multicore programming, Multithreading models, Thread libraries - pthreads CPU scheduling and Process Synchronization Basic concepts, scheduling criteria, scheduling algorithms-FCFS, SJF, RR, priority, Real-time CPU scheduling	
Unit –III	09 Hrs.
Process Synchronization Background, The Critical section problem, Peterson's Solution Process Synchronization Synchronization hardware, Mutex locks, Semaphores, Classic problems of synchronization Case study: Implementation of classic synchronization problem using semaphores	
Unit –IV	09 Hrs.
Main Memory Management Background, Swapping, Contiguous memory allocation, Segmentation, Paging, Structure of page table. Virtual memory Background, Demand Paging, Copy-on-write, Page replacement, Allocation of frames, Thrashing	
Unit –V	09 Hrs.
File Systems File Naming, File Structure, File Types, File Access, File Attributes, File Operations, An example program using File-System calls, File-System Layo Implementing Files. The Virtual File System: The role of the Virtual File System (VFS), VFS data structure, Filesystem Types, Filesystem handling, Pathname looku Implementation of VFS System calls, File Locking.	

Course Outcomes

Course Outcomes: After completing the course, the students will be able to:-	
CO 1	Demonstrate the fundamental concepts of operating system like process management, file management, memory management and issues of synchronization.
CO 2	Analyze and interpret operating system concepts to acquire a detailed understanding of the course.
CO 3	Apply the operating systems concepts to address related new problems in computer science Domain.
CO 4	Design or develop solutions to solve applicable problems in operating systems domain.
CO5	Extend the theoretical knowledge acquired through the course to understand the real-world implementation of operating system.

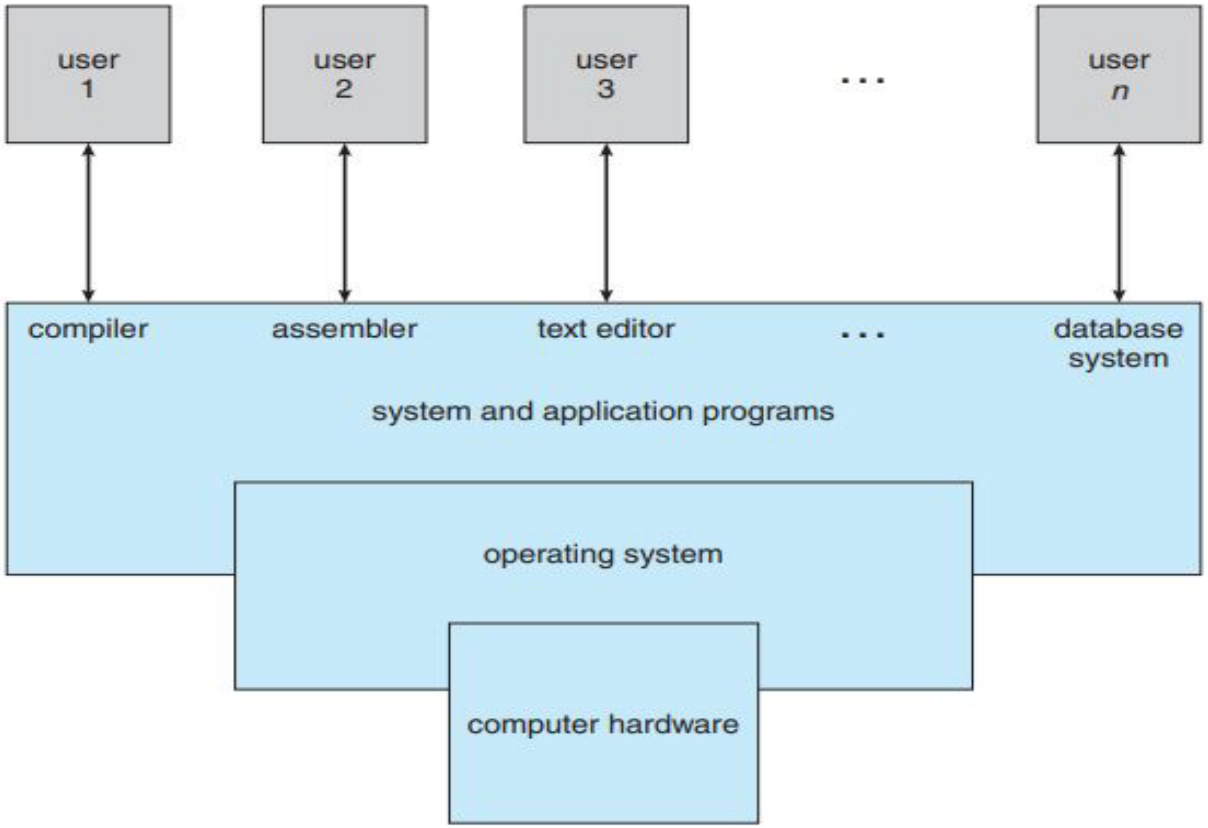
Reference Books	
1.	Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin , Greg Gagne, 9th Edition, Incorporated, 2018, John Wiley & Sons, ISBN 978-1-265-5427-0
1.	Modern operating systems, Tanenbaum, Andrew, 4 th Edition, Pearson Education, Inc 2009. ISBN 013359162X, 978-0133591620
1.	UNIX System Programming Using C++, Terrence Chan, 2011, Prentice Hall India, ISBN: 9788120314689 978-8120314689.
1.	Operating systems - A concept based Approach, D.M Dhamdhere, 3rd Edition, 2017, Tata McGraw-Hill, ISBN: 1259005585, 978-1259005589
1.	"xv6: a simple, Unix-like teaching operating system", https://pdos.csail.mit.edu/6.828/2014/xv6/book-rev8.pdf
1.	Understanding the LINUX Kernal, Daniel P Bovet and Marco Cesati, 3rd Edition, 17 November 2005, O'Reilly Publication, 9780596554910, 0596554915. (For Virtual File System of fifth unit)

CONTINUOUS INTERNAL EVALUATION		
ASSESSMENT AND EVALUATION PATTERN Theory & quizzes questions are to be framed using Bloom's Taxonomy Levels - Remembering, Understanding, Applying, Analyzing, Evaluating, and Creating		
WEIGHTAGE	CIE (50%)	SEE (50%)
A. QUIZZES: Each quiz is evaluated for 10 marks		
Quiz-I for 10 Marks	20	*****
Quiz-II for 10 Marks		
A. TESTS: Each test will be conducted for 50 Marks adding upto 100 marks. Final test marks will be reduced to 40		
Test – I for 50 Marks	40	*****
Test – II for 50 Marks		
A. EXPERIENTIAL LEARNING: Experiential Learning comprises of the following components viz Online MooC Course on Operating System tools and System Calls : 10 Marks Case Study Implementation- 20 Marks; Video based Presentation(4-5 minutes per student) with Report– 10 Marks	40	*****
LAB: Conduction of laboratory exercises (20Marks), Labtest (10Marks) and Innovative project implementation (20Marks) adding upto 50Marks. THE FINAL MARKS WILL BE REDUCED TO 30 MARKS	50	50
MAXIMUM MARKS FOR THE THRORY (A+B+C)	100	100
TOTAL MARKS FOR THE COURSE	100	150

Contents

- What operating system do
- Operating system structure
- Operating system operations
- **System structures**
- Operating system services
- System calls
- Types of system calls
- **Process management**
- Process concept
- Process scheduling
- Operations on processes

Abstract view



What operating system do

- The operating system controls the hardware and coordinates its use among the various application programs for the various users.
- operating system provides the means for proper use of the resources in the operation of the computer system.
- It simply provides an environment within which other programs can do useful work.
- Two views of OS:
- **User view**
- **System view**

What operating system do(Contd..)

- Case 1:
- user's view of the computer varies according to the interface being used.
- Emphasis is for ease of use than resource utilization
- Case 2:
- user sits at a terminal connected to a mainframe or a minicomputer.
- users share resources and may exchange information
- Goal is resource utilization— to assure that all available CPU time, memory, and I/O are used efficiently

What operating system do(Contd..)

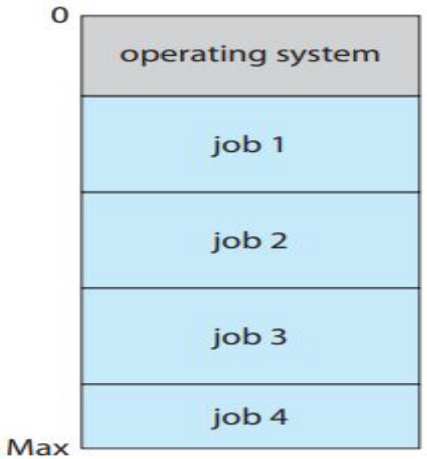
- Case 3:
- users sit at workstations connected to networks of other workstations and servers.
- Users share resources such as networking and servers
- Case 4:
- embedded computers in home devices
- No user intervention

System view

- operating system is the program most intimately involved with the hardware.
- operating system can be viewed as a resource allocator.
- The operating system acts as the manager of the resources.
- resource allocation is especially important where many users access the same mainframe or minicomputer.
- An operating system is a control program. A control program manages the execution of user programs to prevent errors and improper use of the computer

Operating system structure

- An operating system provides the environment within which programs are executed.



- The operating system keeps several jobs in memory , as main memory is small jobs are kept on the disk in the job pool.

Operating system structure

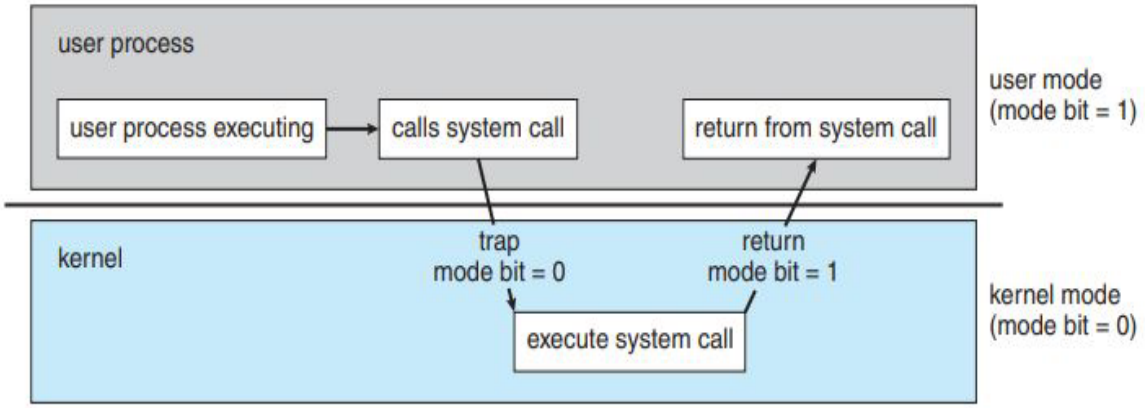
- Multiprogrammed systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively.
- Time sharing (or multitasking) is a logical extension of multiprogramming.
- In a time-sharing system, the operating system must ensure reasonable response time. This goal is sometimes accomplished through swapping
- A more common method for ensuring reasonable response time is virtual memory

Operating-System Operations

- operating systems are interrupt driven
- Events are almost always signaled by the occurrence of an interrupt or a trap.
- For each type of interrupt, separate segments of code in the operating system determine what action should be taken.
- Mode of operation
- **Dual-mode**
- **Multi-mode**

Operating-System Operations

- Fig below shows two modes of operation:



- Kernel mode(supervisor mode, system mode, or privileged mode)
- User mode

Operating-System Operations

- mode bit, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1)

Operating System Services

- Operating systems provide an environment for execution of programs and services to programs and users
- One 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)
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device

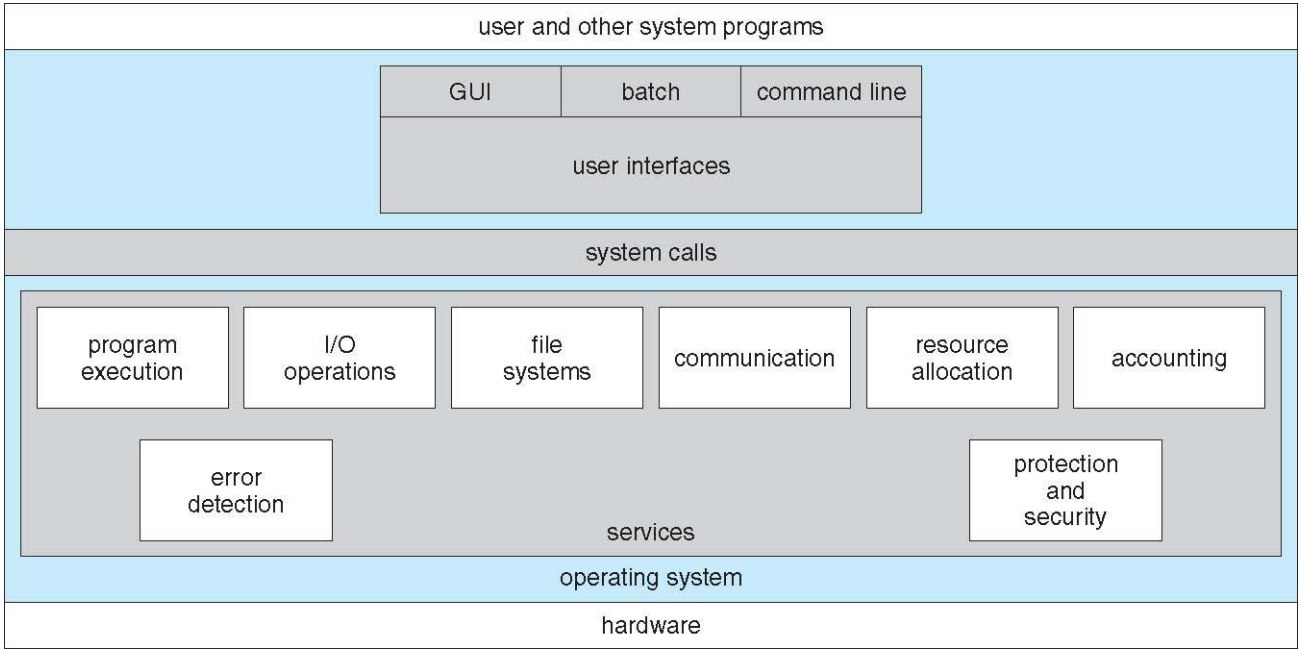
Operating System Services

- **File-system manipulation** - Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
- **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 - CPU cycles, main memory, file storage, I/O devices.
- **Accounting -** To keep track of which users use how much and what kinds of computer resources
- **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

A View of Operating System Services

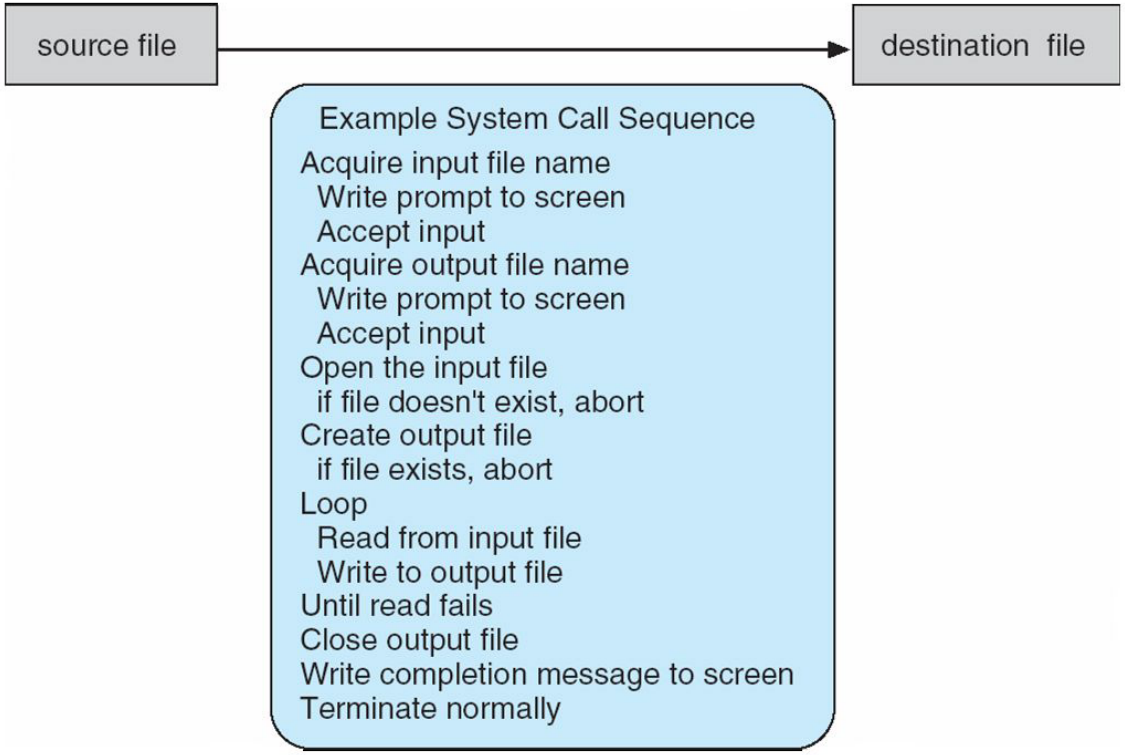


System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use

Example of System Calls

- System call sequence to copy the contents of one file to another file



System Call Implementation

- Typically, a number associated with each system call
  - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
    - Managed by run-time support library

Types of system calls

- Process control,
- File manipulation,
- Device manipulation,
- Information maintenance,
- Communications, and
- Protection.

Types of System Calls

- Process control
  - create process, terminate process
  - end, abort
  - load, execute
  - 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.)

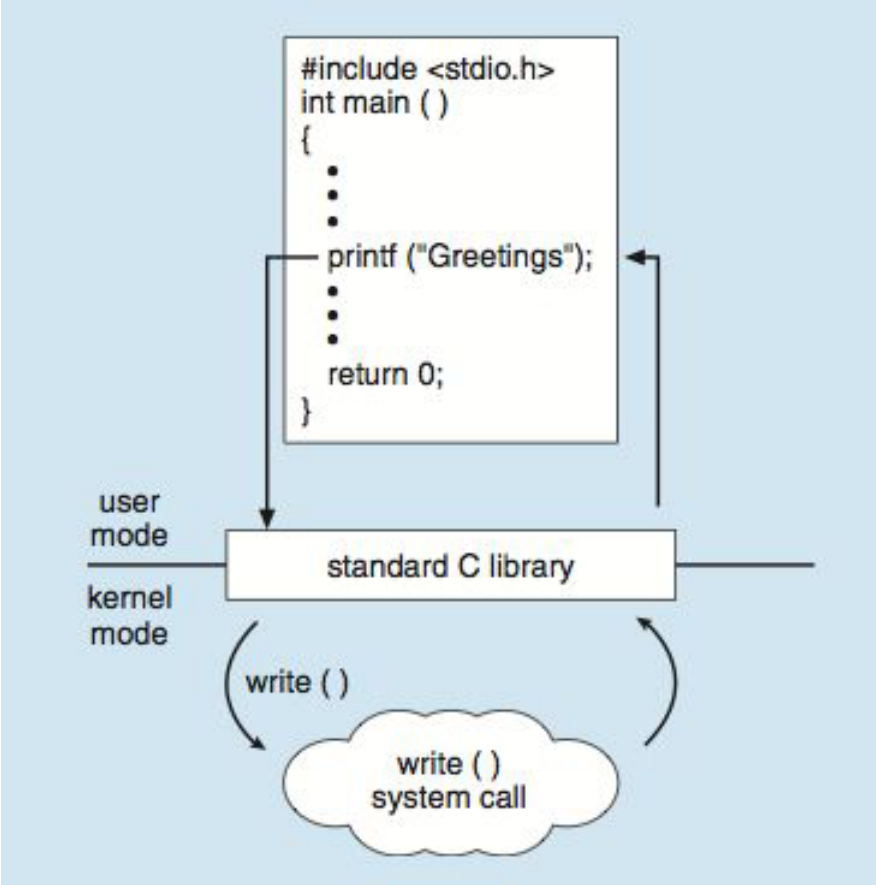
- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

Examples of Windows and Unix System Calls

	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() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Standard C Library Example

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

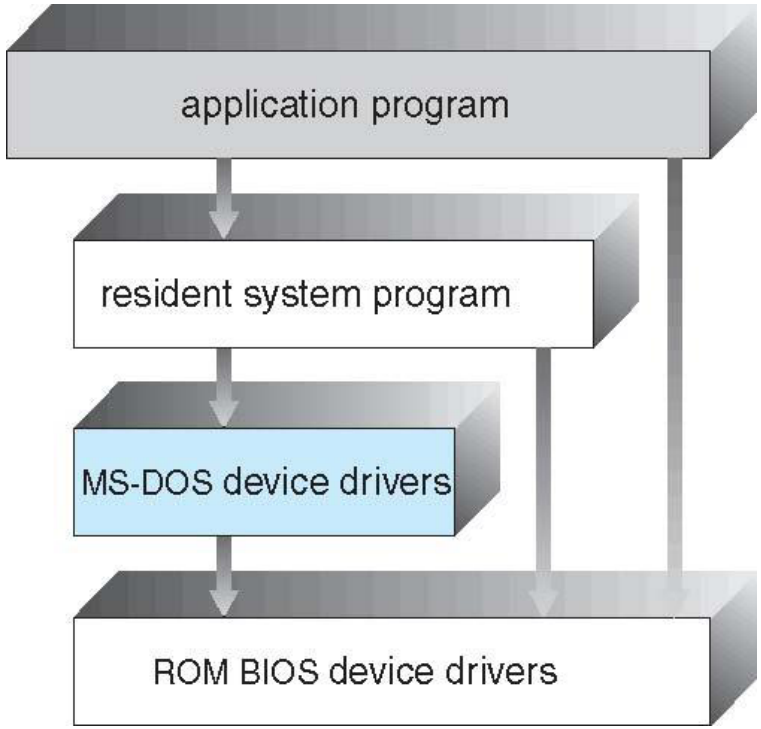


Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex -- UNIX
  - Layered – an abstrcation
  - Microkernel -Mach

Simple Structure -- MS-DOS

- 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
  - Limited by hardware of its time



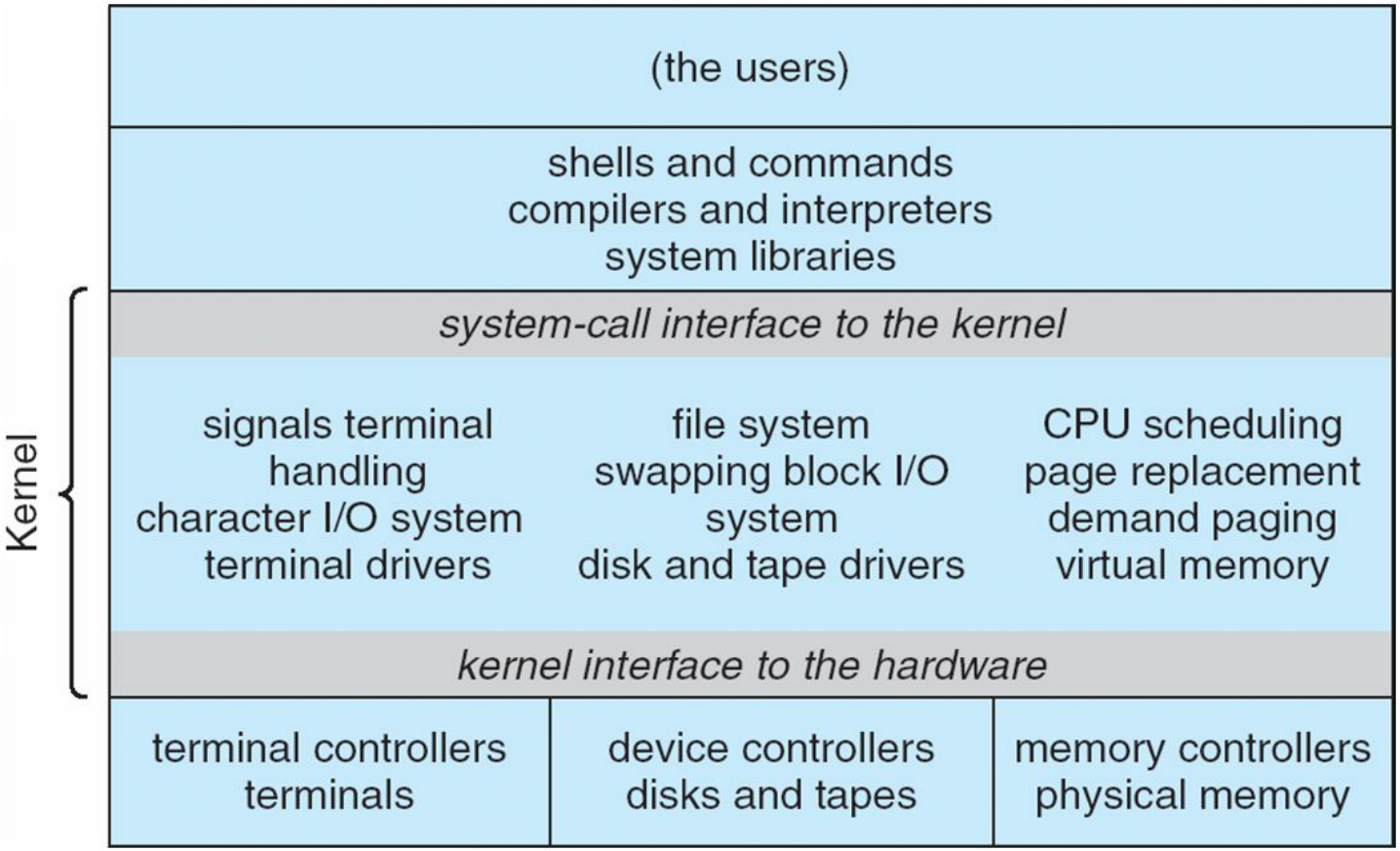
Non Simple Structure -- UNIX

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

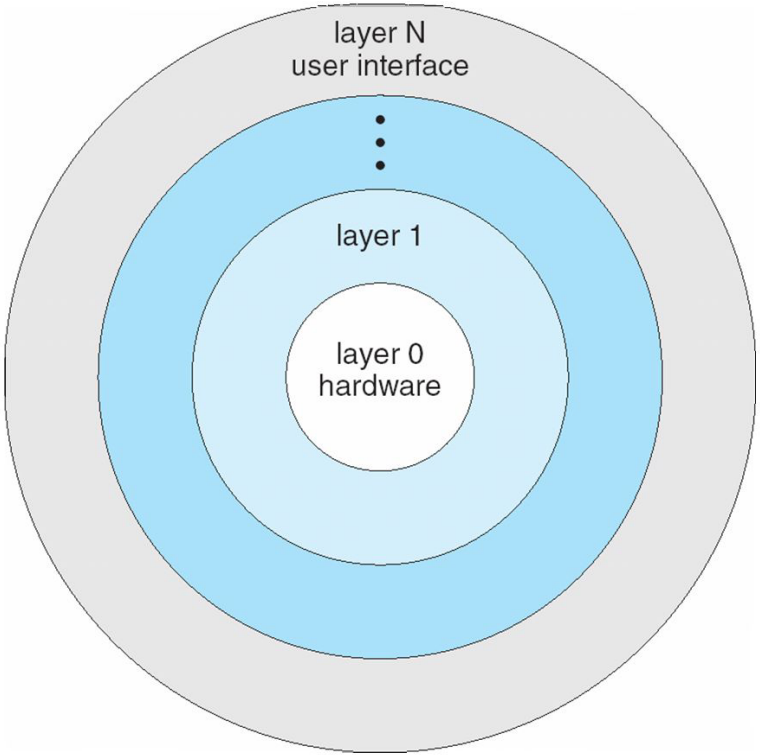
Traditional UNIX System Structure

Beyond simple but not fully layered-monolithic architecture



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.



Microkernel System Structure

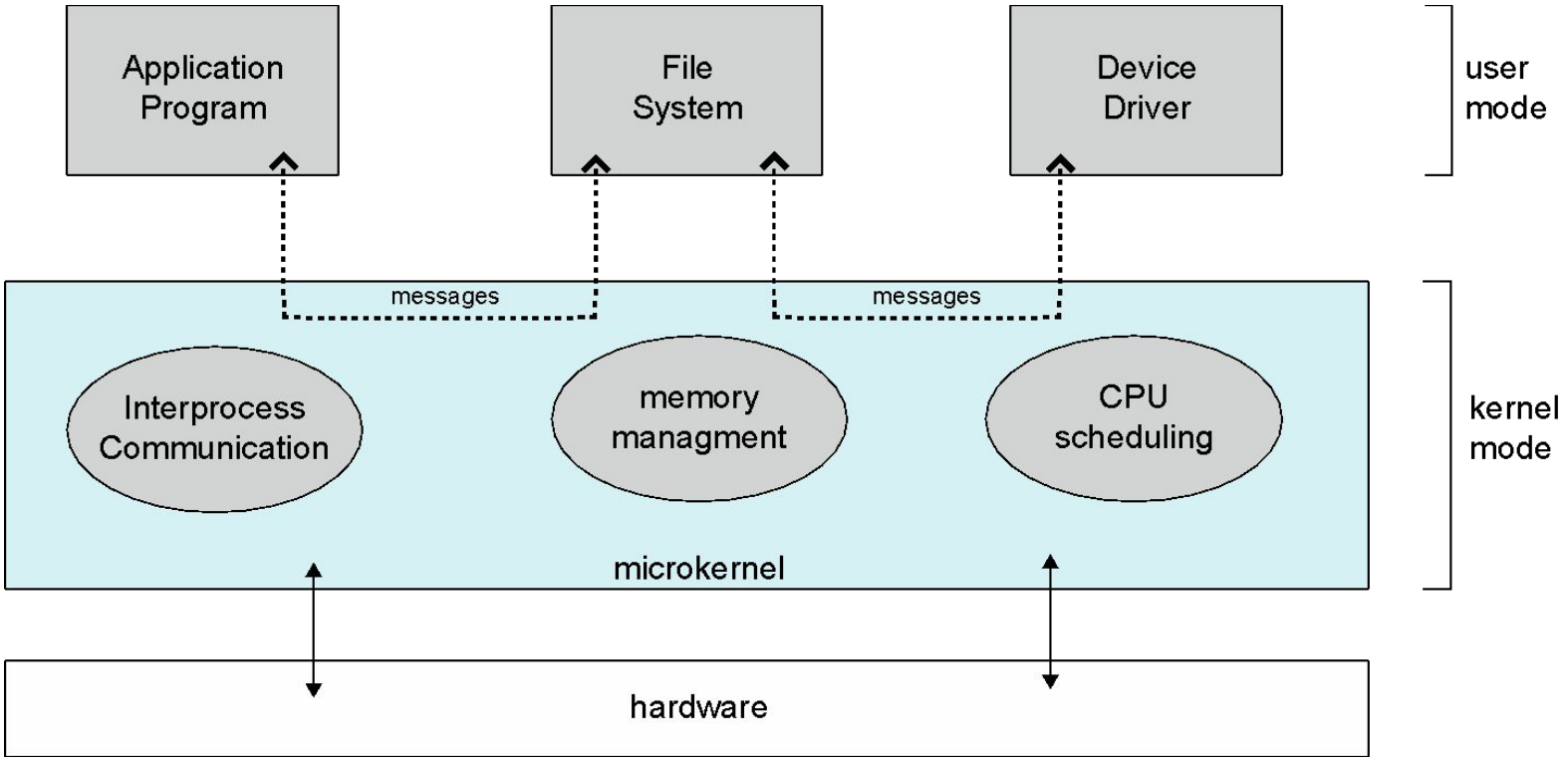
- Moves as much from the kernel into user space
- **Mach** example of **microkernel**
- **Developed at Carnegie Mellon University**
  - Mac OS X kernel (**Darwin**) partly based on Mach
  - structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs.
- Communication takes place between user modules using **message passing**

Microkernel System Structure

- 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

  - Windows NT, QNX, Windows 95

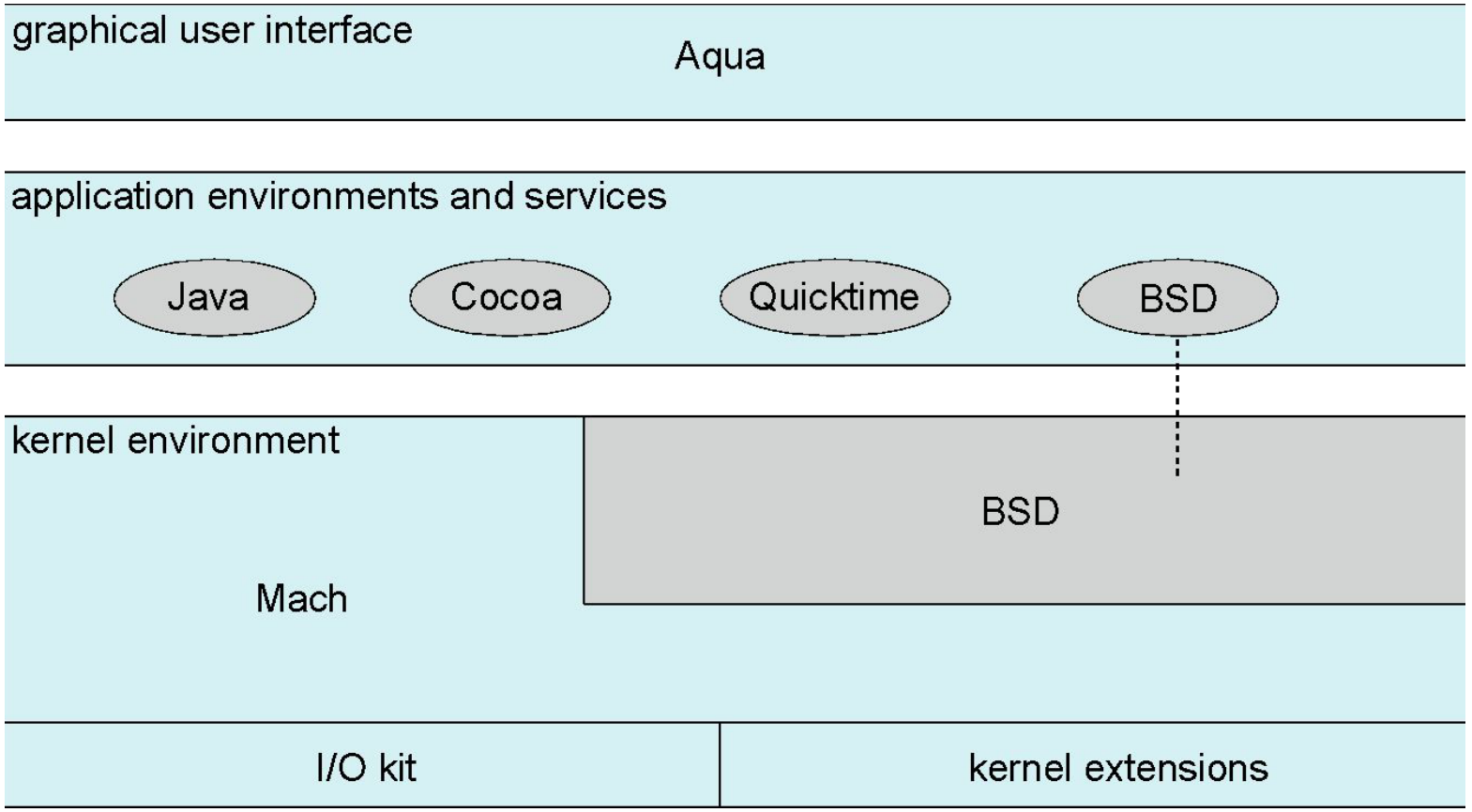
Microkernel System Structure



Hybrid Systems

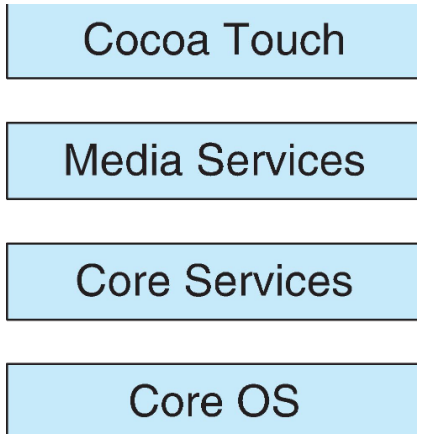
- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem ***personalities***
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment

Mac OS X Structure



iOS

- 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



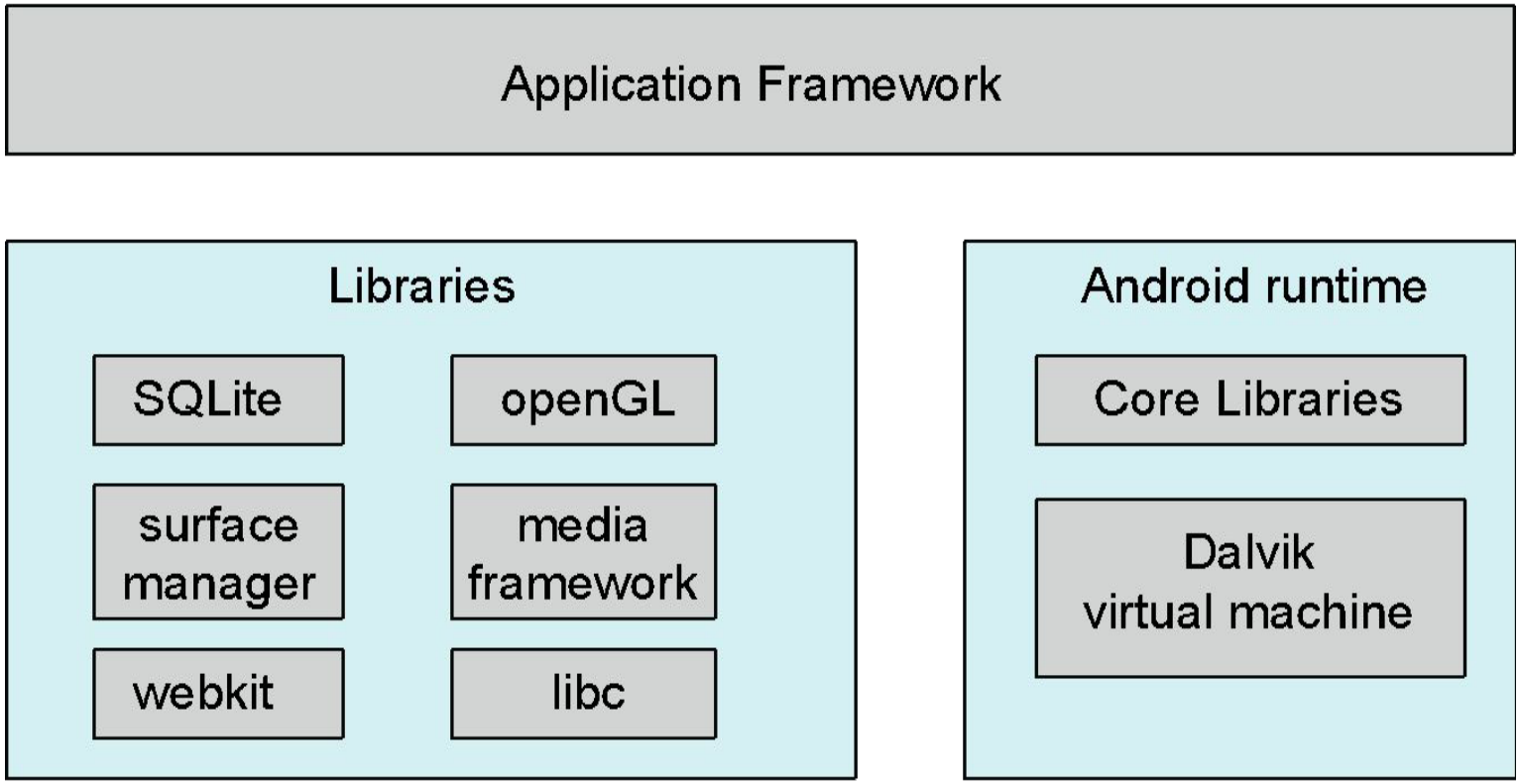
Android

- 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

Android

- 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



Process Management

- Process concept,
- Process scheduling,
- Operations on processes

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 management.
  - Process creation and deletion.
  - process suspension and resumption.
  - Provision of mechanisms for:
    - process synchronization
    - process communication

Process Management

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Process – a program in execution; process execution must progress in sequential fashion.
- A process includes:
  - program counter
  - stack
  - data section

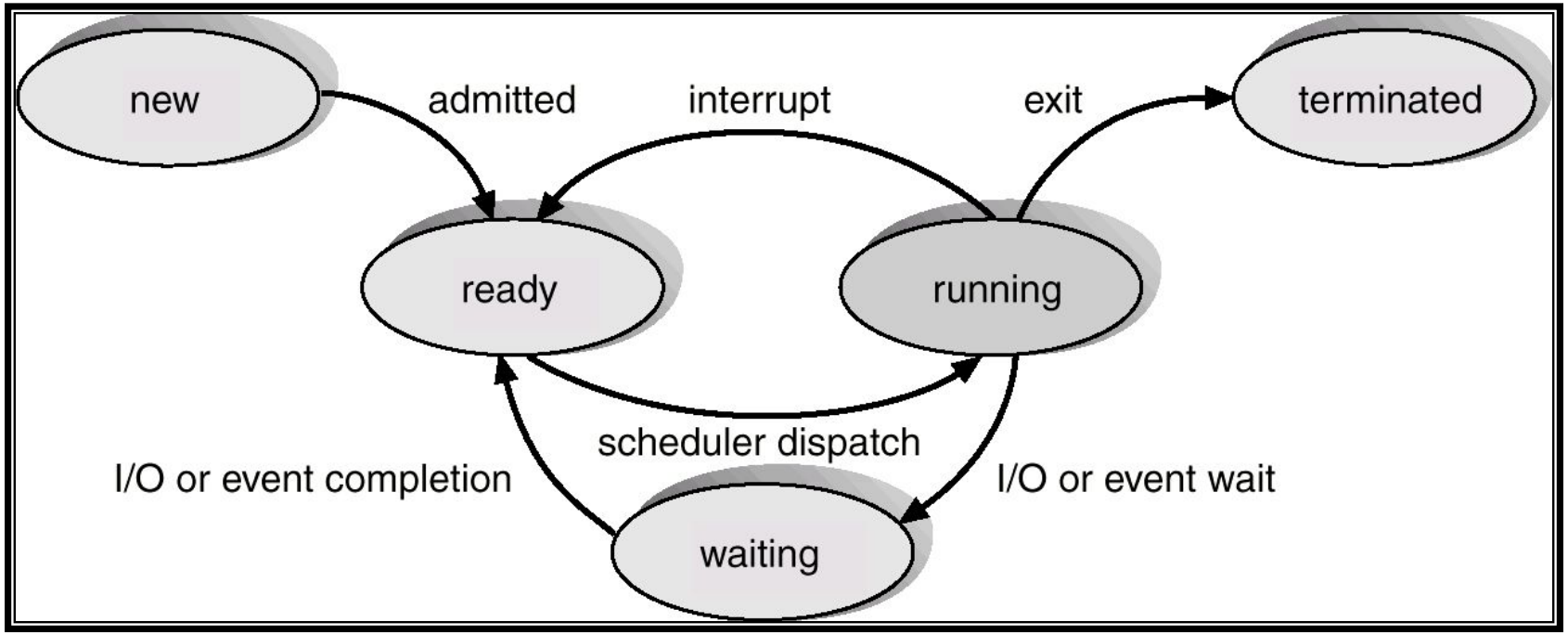
Terms

- Job
- Task
- Process

State of process

- As a process executes, it changes *state*
  - **new**: The process is being created.
  - **running**: Instructions are being executed.
  - **waiting**: The process is waiting for some event to occur.
  - **ready**: The process is waiting to be assigned to a processor.
  - **terminated**: The process has finished execution.

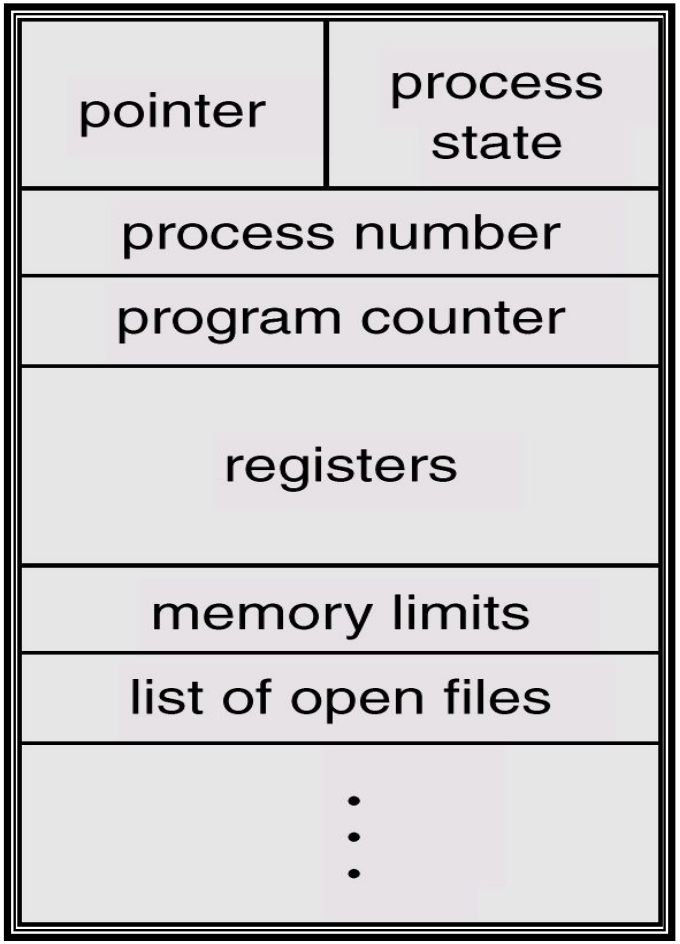
State of process



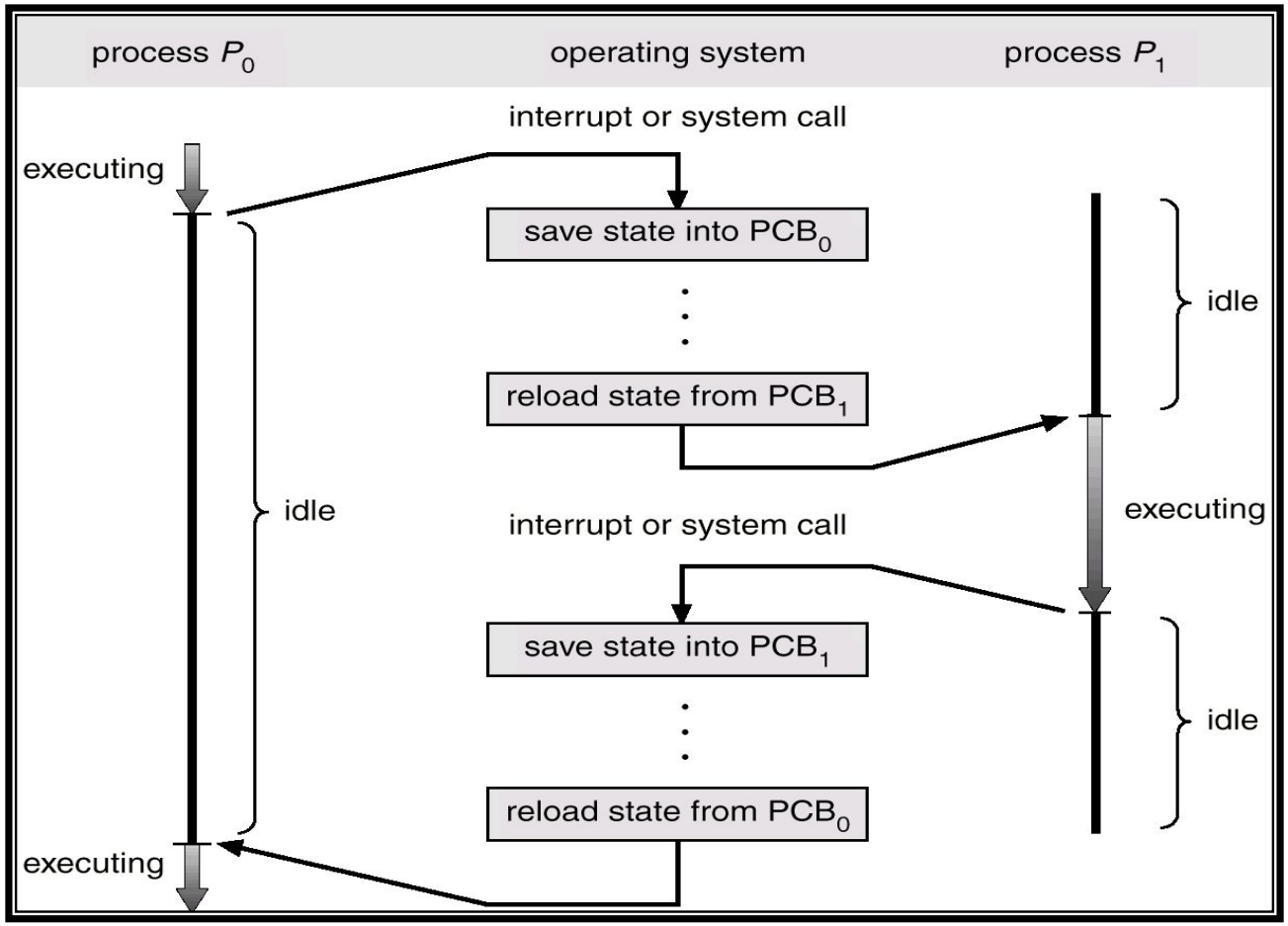
Process Control Block (PCB)

Information associated with each process.

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information


pointer
process state
process number
program counter
registers
memory limits
list of open files

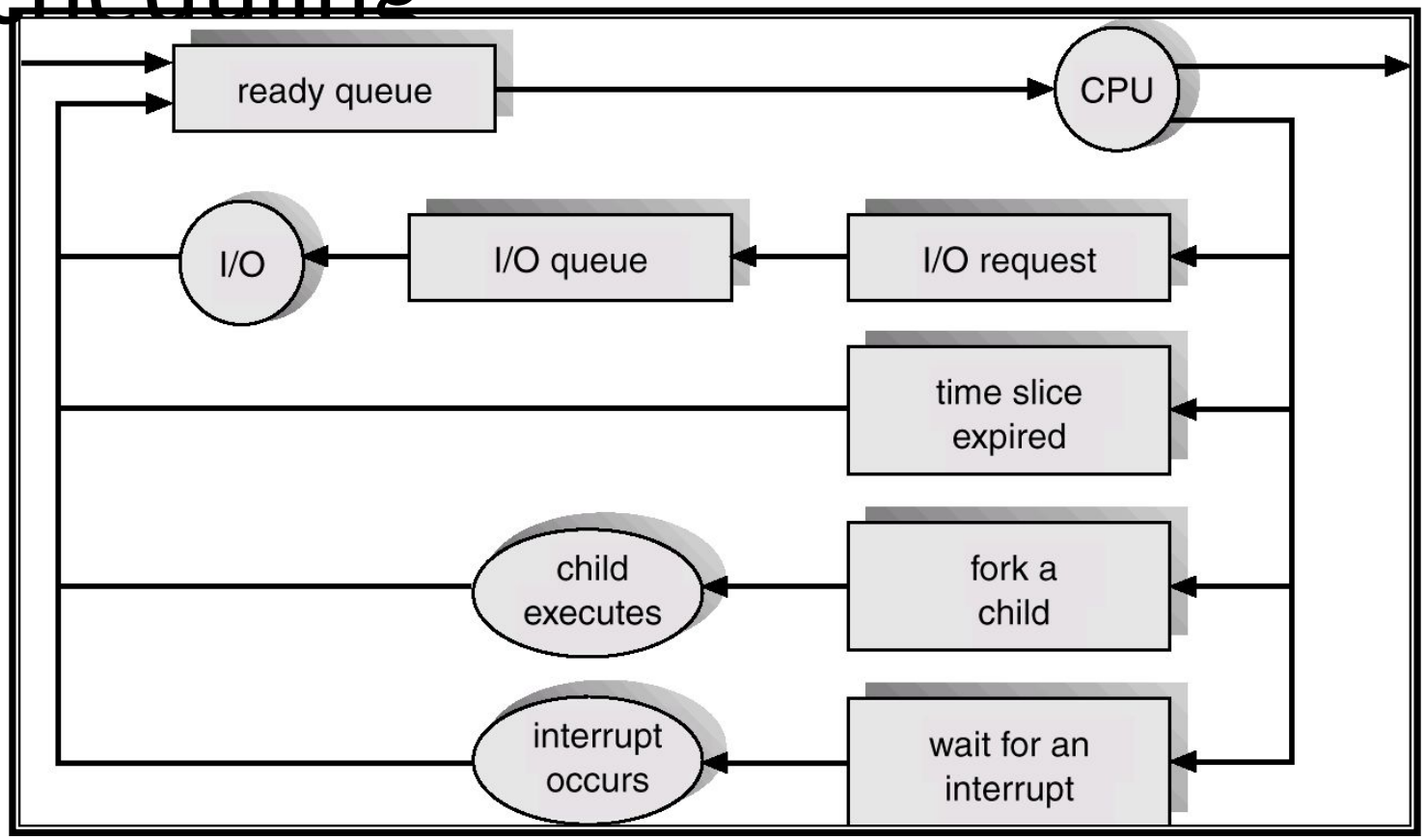
CPU Switch From Process to Process



Process Scheduling Queues

- Job queue – set of all processes in the system.
- Ready queue – set of all processes residing in main memory, ready and waiting to execute.
- Device queues – set of processes waiting for an I/O device.
- Process migration between the various queues.

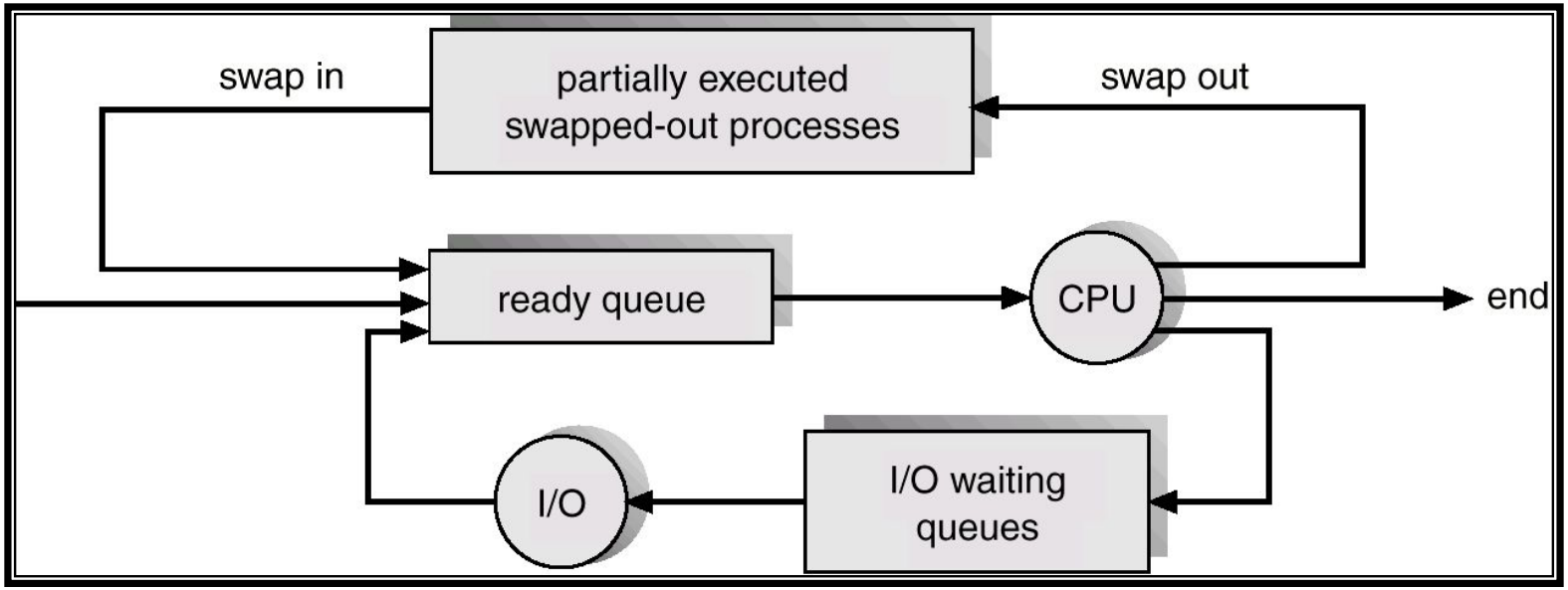
Representation of Process Scheduling



Schedulers

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.
- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU.

Addition of Medium Term Scheduling



Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast).
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow).
- The long-term scheduler controls the *degree of multiprogramming.*
- Processes can be described as either:
  - I/O-*bound process* – spends more time doing I/O than computations, many short CPU bursts.
  - *CPU-bound process* – spends more time doing computations; few very long CPU bursts.

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support.

Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent's resources.
  - Parent and child share no resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

Process Creation (Cont.)

- Address space
  - Child duplicate of parent.
  - Child has a program loaded into it.
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process' memory space with a new program.

Process Termination

- Process executes last statement and asks the operating system to decide it (**exit**).
  - Output data from child to parent (via **wait**).
  - Process' resources are deallocated by operating system.
- Parent may terminate execution of children processes (**abort**).
  - Child has exceeded allocated resources.
  - Task assigned to child is no longer required.
  - Parent is exiting.
    - Operating system does not allow child to continue if its parent terminates.
    - Cascading termination.