Chapter 1.1 Operating System Overview

Outline

- What is an operating system
- Views of operating system
- The history of Operating System
- Evolution of Operating System

Introduction

- A modern computer consists of :
  - One or more processors ,Main memory ,Disks , Printers, Various input/output devices
- Managing all these components requires a layer of software – the **operating system**
- It is impossible for every application programmer to understand every detail.
- A layer of computer software is introduced to provide a better, simpler, cleaner model of the resources and manage them.

What is an Operating System?

- An OS is a system program that controls the execution of application programs and acts as an interface between applications and the computer hardware.
- The operating system acts as a **resources manager** and allocates them to specific programs and users, whenever necessary to perform a particular task. Therefore operating system is the resource manager.
- A computer system has many resources (hardware and software), which may be require to complete a task. The commonly required resources are input/output devices, memory, file storage space, CPU etc.

- It can be thought of as having three objectives:
  - **Convenience :** An OS makes a computer more convenient to use.
  - **Efficiency:** An OS allows the computer system resources to be used in an efficient manner.
  - **Ability to evolve:** An OS should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions without interfering with service.
- Most computers have two modes of operation: **kernel mode** and **user mode.** The operating system, the most fundamental piece of software, runs in **kernel mode** (also called **supervisor mode**).

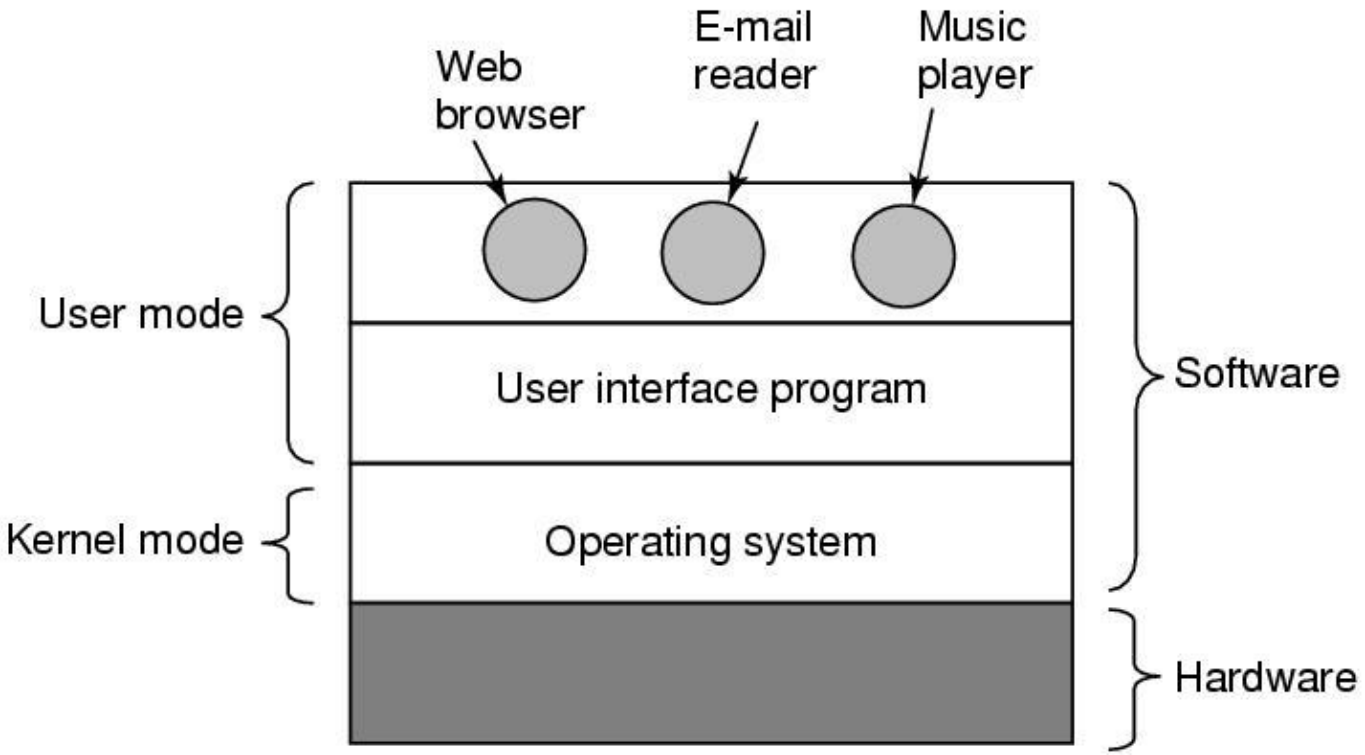


Figure : Where the operating system fits in.

- The hardware consists of chips, boards, disks, a keyboard, a monitor, and similar physical objects.
- On top of the hardware is the software.
- OS runs in **kernel mode**, which has complete access to all the hardware and can execute any instruction the machine is capable of executing.
- The rest of the software runs in **user mode**, in which only a subset of the machine instructions is available.
- The user interface program, shell or GUI, is the lowest level of user-mode software, and allows the user to start other programs, such as a Web browser, email reader, or music player .

Computer System Components

- Computer system can be divided into four components

Hardware:

  - Provides basic computing resources (CPU, memory, I/O devices).

- **Operating System :**
  - Controls and coordinates the use of hardware among application programs.
- **Application Programs :**
  - Solve computing problems of users (compilers, database systems, video games, business programs such as banking software).
- **Users :** People, machines, other computers

Two functions:

- From top to down: provide application programmers a clean abstract set of resources instead of hardware ones.
- From down to top: Manage these hardware resources.

The Operating System as a Resource Manager

- Allow multiple programs to run at the same time.
- Manage and protect memory, I/O devices, and other resources.
- Resource management includes **multiplexing** (sharing) resources in two different ways:
  - In time
  - In space
- Modern OS runs multiple programs of multiple users at the same time
  - Imagine what would happen if several programs want to print at the same time?
  - How to account the resource usage of each process?
  - Resources can be multiplexed:
    - How to ensure fairness and efficiency?

The Operating System as an Extended Machine

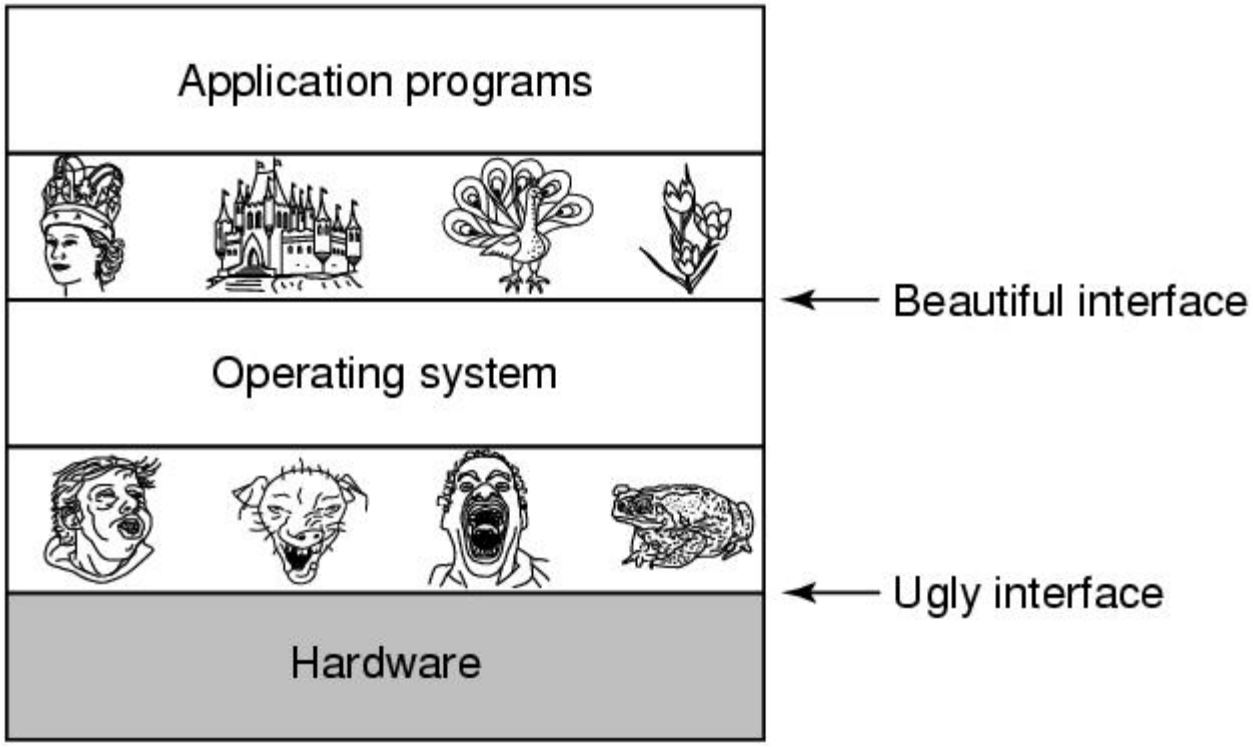


Figure : Operating systems turn ugly hardware into beautiful abstractions.

- Abstraction:
  - CPU—process
  - Storage –- files
  - Memory– address space
- 4 types of people:
  - Industrial engineer: design hardware
  - Kernel designer
  - Application programmer: OS’s user
  - End users

User View

Two Views of Operating System

- The goal of the Operating System is to maximize the work and minimize the effort of the user.
- Most of the systems are designed to be operated by single user, however in some systems multiple users can share resources, memory. In these cases Operating System is designed to handle available resources among multiple users and CPU efficiently.
- Operating System must be designed by taking both usability and efficient resource utilization into view.
- In embedded systems(Automated systems) user view is not present.
- There are some devices that contain very less or no user view because there is no interaction with the users. Examples are embedded computers in home devices, automobiles etc.

System View

- The operating system can be viewed as a resource allocator also. A computer system consists of many resources like - hardware and software - that must be managed efficiently.
- The operating system acts as the manager of the resources, decides between conflicting requests, controls the execution of programs, etc.
- It controls application programs that are not part of Kernel.

Operating System functions

- **Process management**:- Process management helps OS to create and delete processes. It also provides mechanisms for synchronization and communication among processes.
- **Memory management** Memory management module performs the task of allocation and de-allocation of memory space to programs in need of this resources.
- **Device management** Device management keeps tracks of all devices. This module also responsible for this task is known as the I/O controller
- **I/O System Management:** One of the main objects of any OS is to hide the peculiarities of that hardware devices from the user.
- **Graphics User Interface (GUI)** management: Provides and manages the user interface that interacts with graphics and visual content on a computing device.

- **Secondary-Storage management** which includes primary storage, secondary storage, and cache storage.
- **Application** which allows standard communication between software and your computer.
- **User interface** which allows you to communicate with your computer.
- **Job accounting**: Keeping track of time & resource used by various job and users.
- **Networking:** The processors communicate with one another through the network.
- **Security** module protects the data and information of a computer system against malware threat and authorized access.

History of Operating Systems

Generations:

- (1945–55) Vacuum Tubes
- (1955–65) Transistors and Batch Systems
- (1965–1980) ICs and Multiprogramming
- (1980–Present) Personal Computers
- (1990-present) Mobile Computers

1st: vacuum tubes

- Large and slow
- Engineers design, build, operate and maintain the computer
- All programming is done with machine language, or by wiring circuits using cables
- Insert plugboards into the computer and operate
- The work is mainly numerical calculations

2nd: transistors and batch systems

- Also called mainframes
- Computers are managed by professional operators
- To run a **job** (i.e., a program or set of programs), a programmer would first write the program on paper (in FORTRAN or assembler),then punch it on cards.
- Collect a batch of jobs in the input room, then read them into a magnetic tape; the same for output
- Assembly language and high-level programming languages like FORTRAN, COBOL were used.

3rd: IC and Multiprogramming

- OS/360:
  - Aims to adapts 1401/7904, covers all trades of life
  - However, OS/360 introduces several key techniques
    - Multi-programming: solve the problem of CPU idling
    - Spooling: simultaneous peripheral operation on line
      - Whenever a job finishes, OS load a new job from disk to the empty-partition
  - High-level languages (FORTRAN-II TO IV, COBOL, PASCAL PL/1, BASIC, ALGOL-68 etc.)

3rd: ICs and Multiprogramming

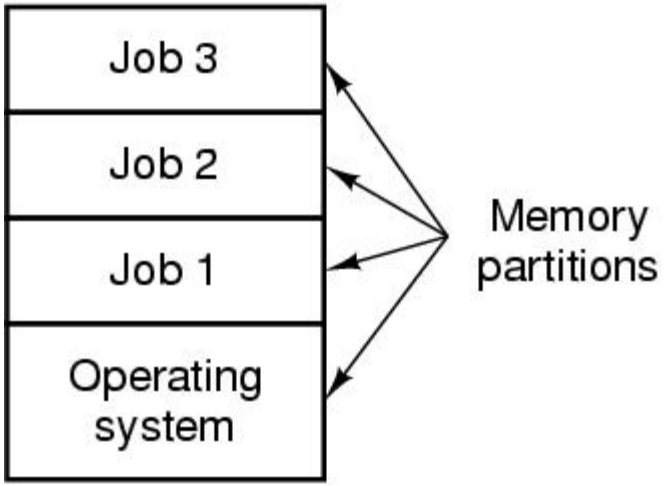


Figure . A multiprogramming system with three jobs in memory.

3rd: Ics and Multiprogramming

- Problems:
  - 3rd generation OS was well suited for big scientific calculations and massive data processing
  - But many programmers complain a lot… for not be able to debug quickly…. Why?
  - And the solution to this problem would be….?
  - Timesharing

4th: personal computers

- Computers of fourth generation used Very Large Scale Integrated (VLSI) circuits
- CP/M(**Control Program for Microcomputers**)
  - First disk-based OS
- 1980, IBM PC, Basic Interpreter, DOS, MS-DOS
- GUI--Lisa—Apple: user friendly (UNIX)
- MS-DOS with GUI– Win95/98/me(**Millennium Edition**)—winNT
- Time sharing, real time networks, distributed operating system were used.

5th: Mobile Computers

- VLSI technology became ULSI (Ultra Large Scale Integration) technology, resulting in the production of microprocessor chips having ten million electronic components.
- The first real smartphone did not appear until the mid-1990s when Nokia released the N9000: a phone and a **PDA** (Personal Digital Assistant).
- **Symbian** OS popular brands like Samsung, Sony Ericsson, Motorola, and especially Nokia.
- Other operating systems like **RIM's** Blackberry OS (introduced for smartphones in 2002) and Apple's iOS (released for the first **iPhone** in 2007)
- Linux-based operating system released by Google in 2008.

Types of Operating Systems

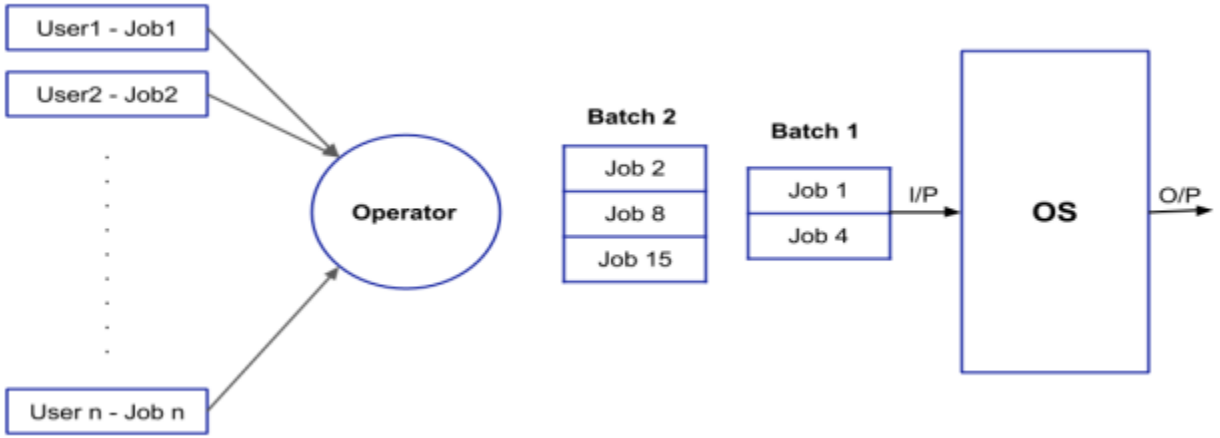
1. Mainframe Operating Systems

- The operating systems for mainframes are heavily oriented toward processing many jobs at once, most of which need prodigious amounts of I/O. They typically offer three kinds of services: batch, transaction processing, and timesharing.
- A **batch system** is one that processes routine jobs without any interactive user present. Claims processing in an insurance company or sales reporting for a chain of stores is typically done in batch mode.
- **Transaction-processing systems** handle large numbers of small requests, for example, check processing at a bank or airline reservations. Each unit of work is small, but the system must handle hundreds or thousands per second.
- **Timesharing systems** allow multiple remote users to run jobs on the computer at once, such as querying a big database.

2. Simple Batch Systems

- To improve utilization, the concept of a batch OS was developed. It appears that the first batch OS was developed in the mid-1950s.
- In a Batch Operating System, the similar jobs are grouped together into batches with the help of some operator and these batches are executed one by one.
- The central idea behind the simple batch-processing scheme is the use of a piece of software known as the **monitor** .
- The monitor is always in the main memory and available for execution.
- In this type of system, there is **no direct interaction between user and the computer**.
- Instead, the user submits the job on cards or tape to a computer operator, who batches the jobs together sequentially and places the entire batch on an input device.

- For example, let us assume that we have 10 programs that need to be executed. Some programs are written in C++, some in C and rest in Java. Now, every time when we run these programmes individually then we will have to load the compiler of that particular language and then execute the code. But what if we make a batch of these 10 programmes.
- The benefit with this approach is that, for the C++ batch, you need to load the compiler only once. Similarly, for Java and C, the compiler needs to be loaded only once and the whole batch gets executed.



- Resident Monitor is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

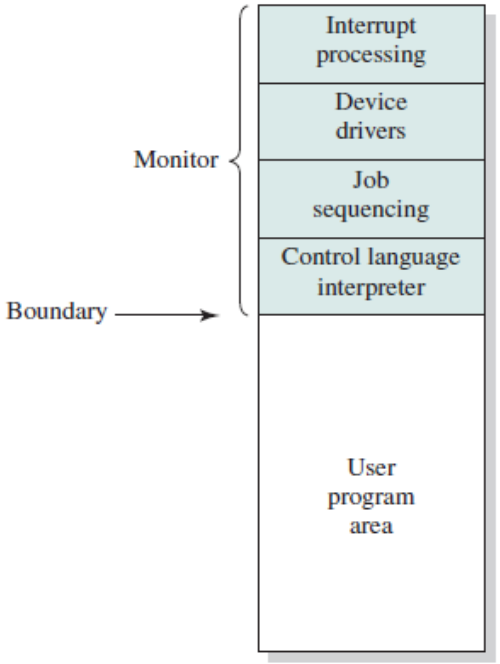


Figure : Memory Layout for a Resident Monitor

Advantages:

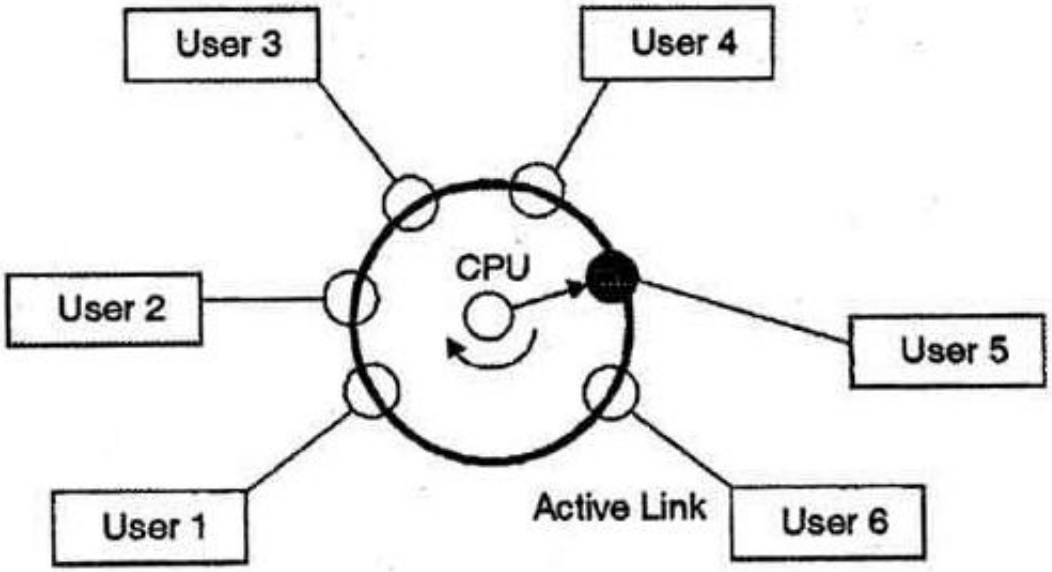
- The overall time taken by the system to execute all the programmes will be reduced.
- The Batch Operating System can be shared between multiple users.

Disadvantages:

- Manual interventions are required between two batches.
- The CPU utilization is low because the time taken in loading and unloading of batches is very high as compared to execution time.

3. Time-Sharing Systems

- In a Multi-tasking Operating System, more than one processes are being executed at a particular time with the help of the time-sharing concept. So, in the time sharing environment, we decide a time that is called **time quantum/slicing** and when the process starts its execution then the execution continues for only that amount of time and after that, other processes will be given chance for that amount of time only.
- In the next cycle, the first process will again come for its execution and it will be executed for that time quantum only and again next process will come. This process will continue.
- Time Sharing' is no longer commonly used, it has been replaced by 'Multitasking System’ also known as multitasking.



- In above figure the user 5 is active but user 1, user 2, user 3, and user 4 are in waiting state whereas user 6 is in ready status.
- As soon as the time slice of user 5 is completed, the control moves on to the next ready user i.e. user 6. In this state user 2, user 3, user 4, and user 5 are in waiting state and user 1 is in ready state. The process continues in the same way and so on.

- The time sharing system provides the direct access to a large number of users where CPU time is divided among all the users on scheduled basis.

Advantages:

- Since equal time quantum is given to each process, so each process gets equal opportunity to execute.
- The CPU will be busy in most of the cases and this is good to have case.

Disadvantages:

- Process having higher priority will not get the chance to be executed first because the equal opportunity is given to each process.

4. Server operating System

- They run on servers, which are either very large personal computers, workstations, or even mainframes.
- They serve multiple users at once over a network and allow the users to share hardware and software resources. Servers can provide print service, file service, or Web service.
- Internet providers run many server machines to support their customers and Websites use servers to store the Web pages and handle the incoming requests.
- Typical server operating systems are Solaris, FreeBSD, Linux and Windows Server 201x.

5. Multiprogramming Operating system

- The I/O devices are much slower than the processor, leaving the processor idle most of the time waiting for the I/O devices to finish their operations.
- It is used to have several jobs to execute which should be held in main memory. If several jobs are ready to run at the same time, then the system chooses which one to run through the process of **CPU Scheduling**.
- In Multiprogramming system, CPU will never be idle and keeps on processing.
- In **Uniprogramming** only one program sits in the main memory so it has a small size. But in the case of **multiprogramming** main memory needs more space.
- Uniprogramming system runs smoothly as only one task is run at a time. The slow processor can also work well in Uniprogramming but in multiprogramming processor needs to be fast. In multiprogramming large space of RAM is needed.
- Fixed-size partition is used in Uniprogramming. Both fixed and variable size partitions can be used in multiprogramming systems.

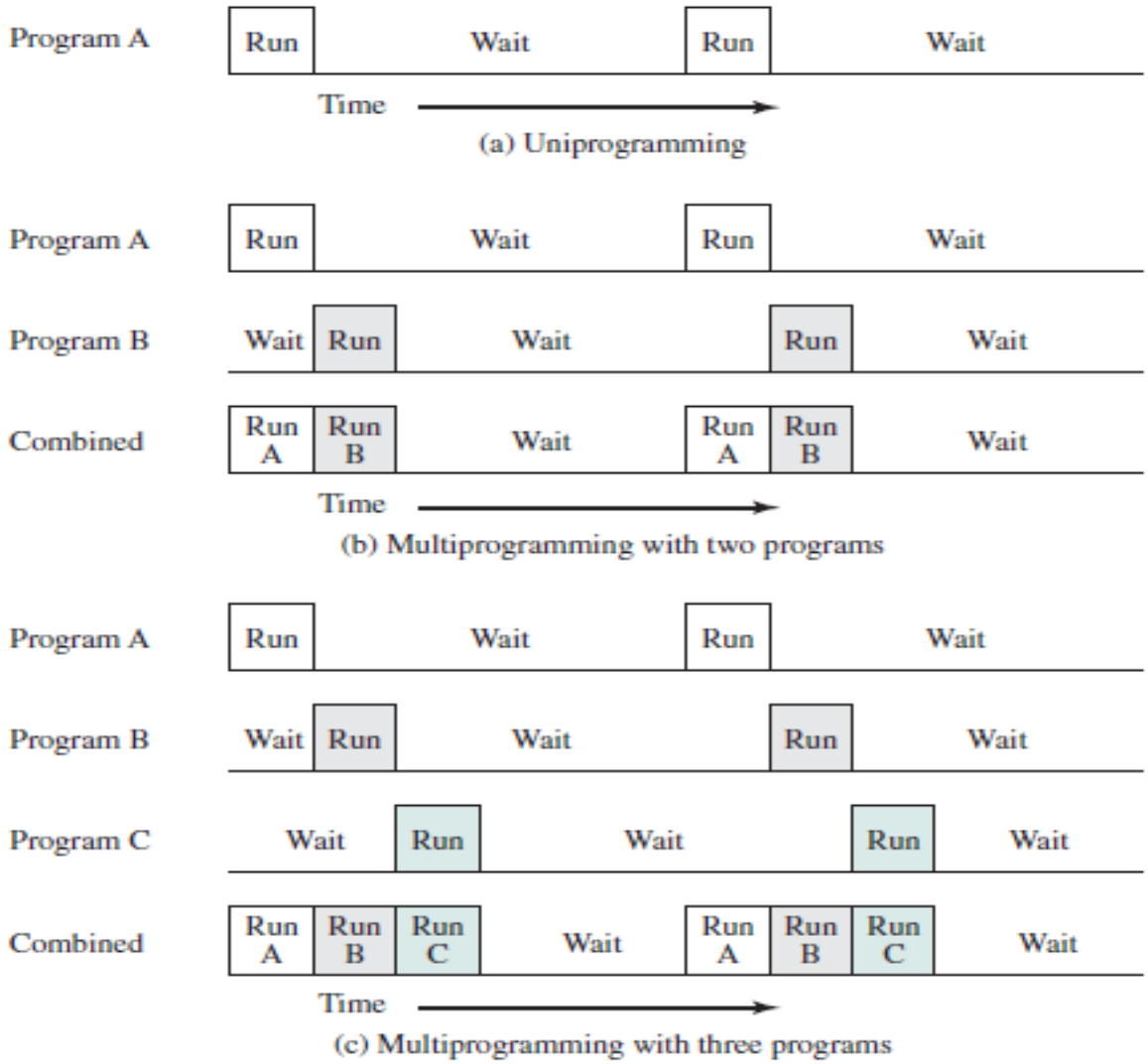


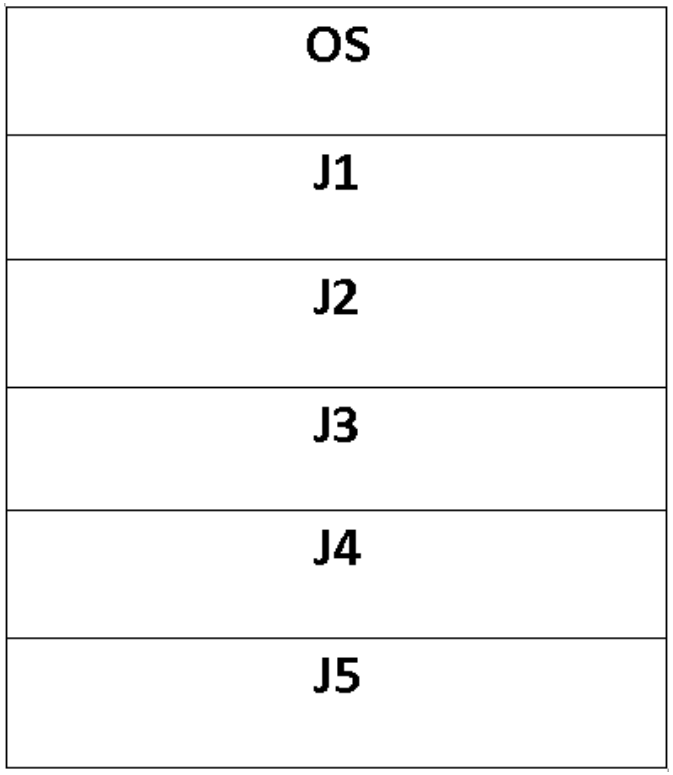
Figure : Multiprogramming Example

Example of Uniprogramming

- Batch processing in old computers and
- mobiles
- The old operating system of computers
- Old mobile operating system

Example of Multiprogramming

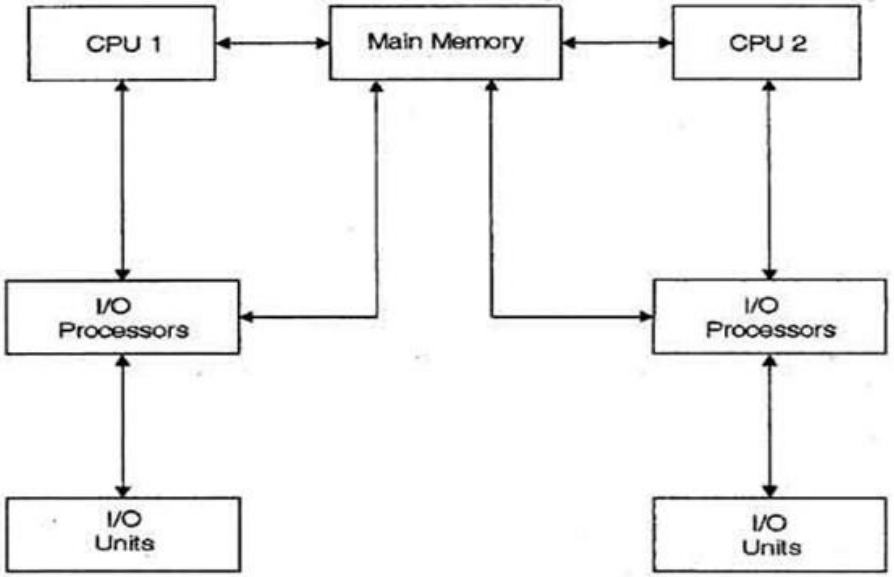
- Modern operating systems like Windows
- XP and Windows 7,8,10



- The main memory consisting 5 jobs at a time ,the CPU execute one by one.

6. Multiprocessor Operating system

- Multiprocessor system means, there are more than one processor which work parallel to perform the required operations.
- It allows the multiple processors, and they are connected with physical memory, computer buses, clocks, and peripheral devices.
- The main objective of using a multiprocessor operating system is to increase the execution speed of the system and consume high computing power.



Advantages:

- More processes can be executed parallel at the same time, increase throughput of the system will increase.
- Multiprocessor systems are more reliable. in case of failure of any one processor will not make the system come to a halt.

7. Personal Computer Operating Systems

- Modern ones all support multiprogramming, often with dozens of programs started up at boot time.
- Their job is to provide good support to a single user. They are widely used for word processing, spreadsheets, games, and Internet access.
- Personal computer operating system are made only for personal. You can say that your laptops, computer systems, tablets etc.
- Common examples are Linux, FreeBSD, Windows 7, Windows 8, and Apple’s OS X.

8. Handheld Systems

- A handheld computer, originally known as a **PDA** (**Personal Digital Assistant**), is a small computer that can be held in your hand during operation.
- Smartphones and tablets are the best-known examples.
- The conventional Handheld Computer Operating Systems are Android, iOS, and Windows.
- Most of these devices boast multicore CPUs, GPS, cameras and other sensors, copious amounts of memory, and sophisticated operating systems.

9. Embedded Operating Systems

- Embedded systems run on the computers that control devices that are not generally thought of as computers and which do not accept user-installed software.
- Typical examples are microwave ovens, TV sets, cars, DVD recorders, traditional phones, and MP3 players.
- The main property which distinguishes embedded systems from handhelds is the certainty that no untrusted software will ever run on it.

10. Real-time operating system (RTOS)

- It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.
- It is developed for real-time applications where data should be processed in a fixed, small duration of time.
- It is used in an environment where multiple processes are supposed to be accepted and processed in a short time.
- RTOS requires quick input and immediate response, e.g., in a petroleum refinery, if the temperate gets too high and crosses the threshold value, there should be an immediate response to this situation to avoid the explosion. Similarly, this system is used to control scientific instruments, missile launch systems, traffic lights control systems, air traffic control systems, etc.

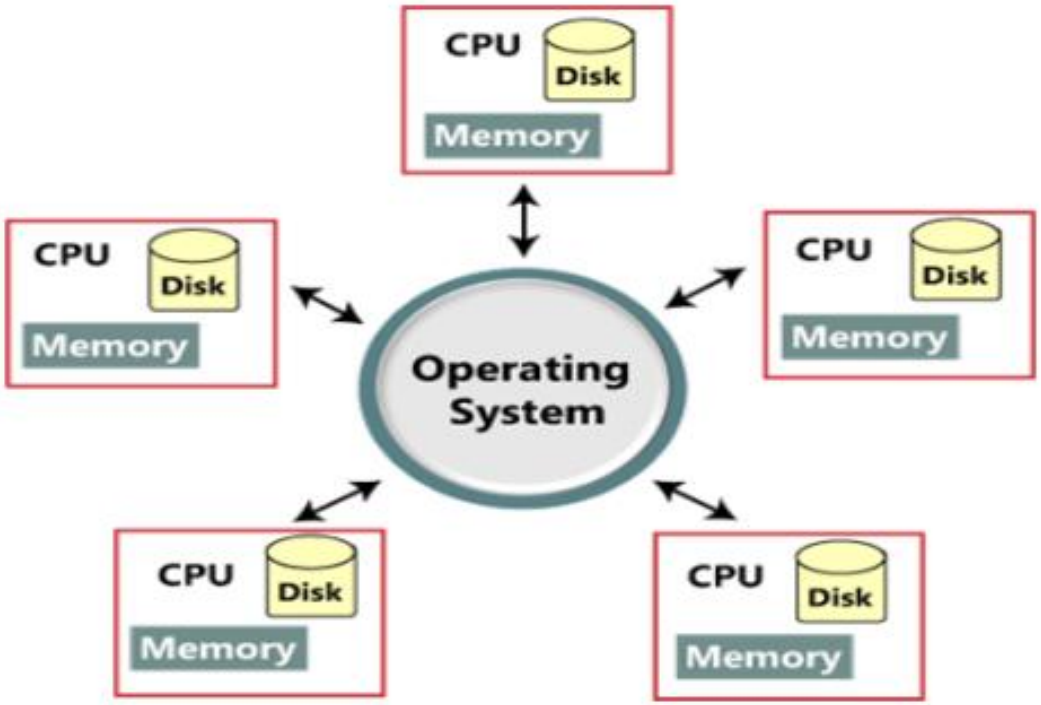
- This system is further divided into two types based on the time constraints:
- **Hard Real-Time Systems:** These are used for the applications where timing is critical or response time is a major factor; even a delay of a fraction of the second can result in a disaster.
- Many of these are found in industrial process control, avionics, military, airbags and automatic parachutes and similar application areas.
- A soft real time system is a system in which one or more failures to meet the deadline is not considered as complete system failure but that performance is considered to be degraded. These systems are referred to as **Soft Real-Time Operating Systems**.
- For example: Multimedia streaming, advanced scientific projects, and virtual reality, smart phones etc.

11. Smart Card Operating Systems

- The smallest operating systems run on smart cards, which are credit-card-sized devices containing a CPU chip.
- They have very severe processing power and memory constraints. Some are powered by contacts in the reader into which they are inserted, but contactless smart cards are inductively powered, which greatly limits what they can do.
- Some of them can handle only a single function, such as electronic payments, but others can handle multiple functions. Often these are proprietary systems.
- Some smart cards are Java oriented. This means that the ROM on the smart card holds an interpreter for the Java Virtual Machine (JVM). Java applets (small programs) are downloaded to the card and are interpreted by the JVM interpreter.
- Resource management and protection also become an issue when two or more applets are present at the same time. These issues must be handled by the (usually extremely primitive) operating system present on the card.

12. Distributed Operating System

- These types of the operating system is a recent advancement in the world of computer technology and are being widely accepted all over the world and, that too, with a great pace.
- Various autonomous interconnected computers communicate with each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred to as **loosely coupled systems** or distributed systems.
- The major benefit of working with these types of the operating system is that it is always possible that one user can access the files or software which are not actually present on his system but some other system connected within this network i.e., remote access is enabled within the devices connected in that network.


CPU
Disk
Memory
CPU
Disk
Memory
CPU
Disk
Memory
Operating System
CPU
Disk
Memory
CPU
Disk
Memory

Advantages:

- Failure of one will not affect the other network communication, as all systems are independent from each other
- Electronic mail increases the data exchange speed
- Since resources are being shared, computation is highly fast and durable
- Load on host computer reduces
- These systems are easily scalable as many systems can be easily added to the network
- Delay in data processing reduces

Disadvantages:

- Failure of the main network will stop the entire communication
- To establish distributed systems the language which is used are not well defined yet
- These types of systems are not readily available as they are very expensive.

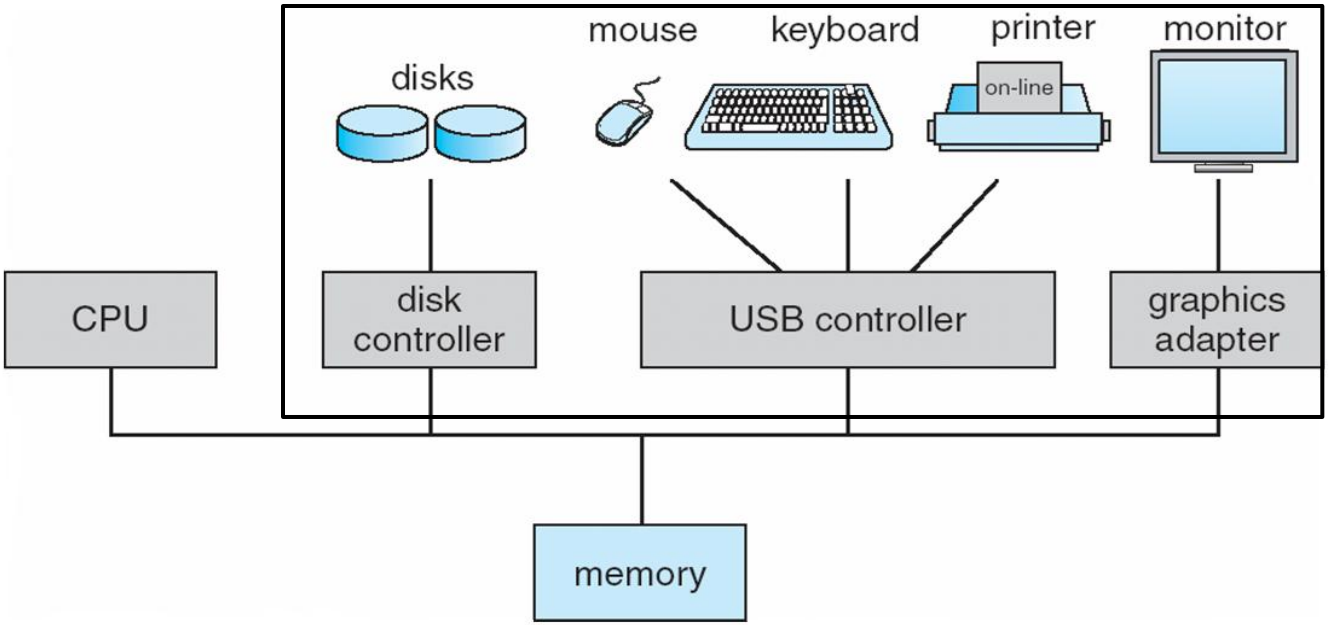
Chapter 1.2: System call and OS Structures

Outline

- Computer System Organization
- Operating System Services
- System calls
- Operating System Structure
- Client-Server Model
- Virtual Machines
- The Shell

Computer System Organization

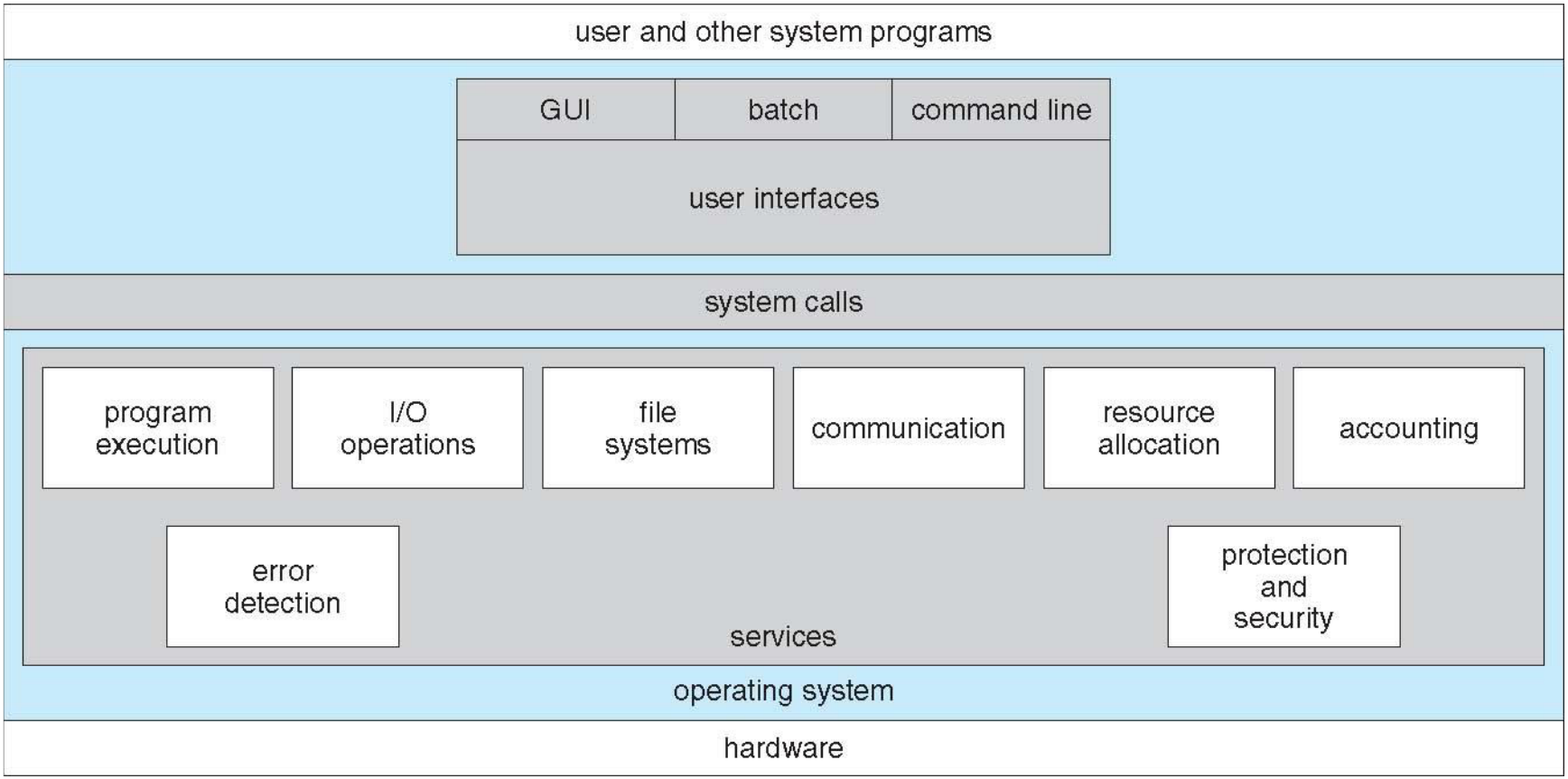
I/O devices



Operating System Services

- Operating systems provide
  - an environment for execution of programs and
  - services to programs and users
- Services may differ from one OS to another
- What are the common classes?
  - Convenience of the user
  - Efficiency of the system

Overview of Operating System Services



OS Services for Helping Users

User interface

-Almost all operating systems have a user interface (**UI**).

–Three forms

• **Command-Line (CLI)**

–Shell command

• **Batch**

–Shell script

• **Graphics User Interface (GUI)**

–Windows system

Program execution

–Load a program into memory

–Run the 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

–Common I/Os: read, write, etc.

–Special functions: recording CD/DVD

Notes: Users usually cannot control I/O devices directly, so OS provides a mean to do I/O

File-system manipulation -The file system is of particular interest

–OS provides a variety of file systems

• Major services

–read and write files and directories

–create and delete files and directories

–search for a given file

–list file Information

–permission management: allow/deny access

Communications: information exchange between processes

–Processes on the same computer

–Processes between computers over a network

Implementations

–Shared memory

• Two or more processes read/write to a shared section of mem.

–Message passing

• Packets of information are moved between processes by OS

Error detection –OS needs to be constantly aware of possible errors

• Error types

–CPU

–memory hardware: memory error, power failure, etc.

–I/O devices: parity error, connection failure, etc.

–user program: arithmetic overflow, access illegal mem.

• Error handling

–Ensure correct and consistent computing

–Halt the system, terminate an error-causing process etc.

Resource allocation

–Resources must be allocated to each user/job

–Resource types -CPU cycles, main memory, file storage, I/O devices

–Special allocation code may be required, e.g., CPU scheduling routines depend on

• Speed of the CPU, jobs, number of registers, etc.

Accounting -To keep track of

–which users use how much and what kinds of resources

• Usage

–Accounting for billing users

–Accumulating usage statistics, can be used for

• Reconfiguration of the system

• Improvement of the efficiency

Protection and security

–Concurrent processes should not interfere w/ each other

–Control the use of computer

•**Protection**

–Ensure that all access to system resources is controlled

•**Security**

–User authentication by password to gain access

–Extends to defending external I/O devices from invalid access attempts

User Operating System Interface –CLI(Command line interface or command interpreter)

–Allows direct command entry

–Included in the kernel or treated as a special program

• Sometimes multiple flavors implemented –**shells**

–Linux: multiple shells (C shell, KornShell etc.)

–Third-party shell or free user-written shell

–Most shells provide similar functionality (personal preference) Main function of CLI

–Get and execute the next user-specified command

–Many commands manipulate files

GUI

- User-friendly graphical user interface

–Mouse-based window-and-menu system (desktopmetaphor)

–**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 in early 1970s

• 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

–Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

Touchscreen devices require new interfaces

–Mouse not possible or not desired

–Actions and selection based on gestures

–Virtual keyboard for text entry

–Voice commands

Choices of Interfaces

- Personal preference

CLI: more efficient, easier for repetitive tasks

–System administrator

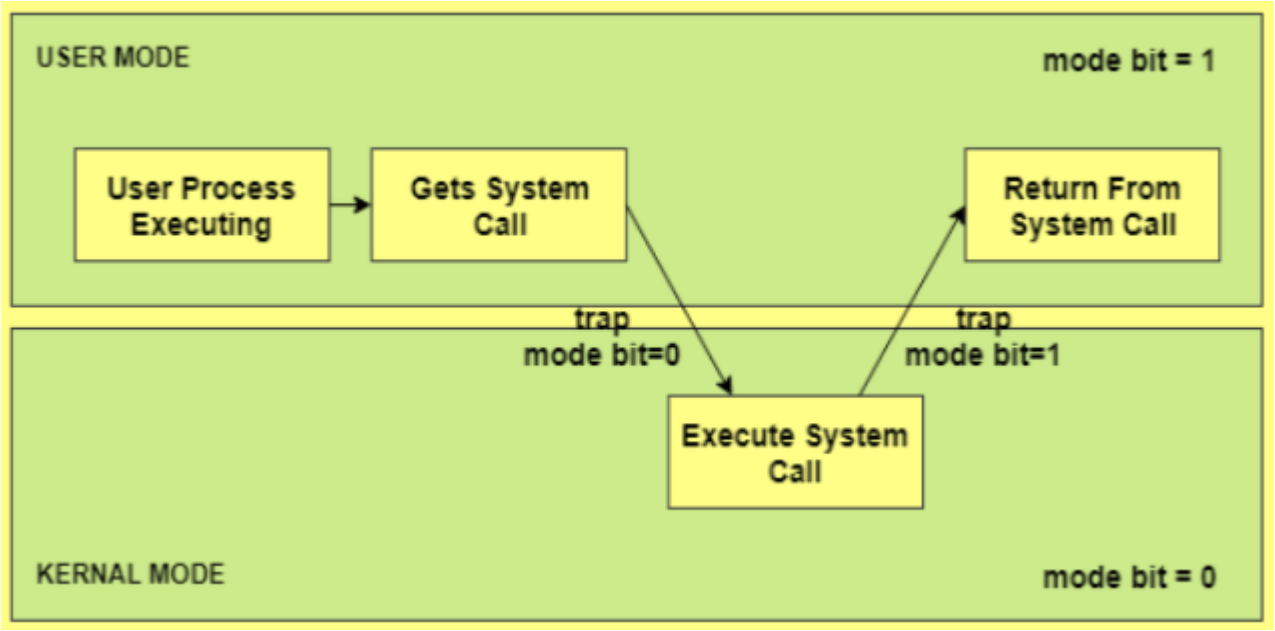
–Power users who have deep knowledge of a system

–Shell scripts

GUI: user-friendly

- The design and implementation of user interface is not a direct function of the OS

User mode and kernel mode



System Calls

- Programming interface to the services provided by the OS.
- A **system call** is a mechanism that provides the interface between a process and the operating system.
- Implementation language

–Typically written in a high-level language (C or C++)

–Certain low-level tasks (direct hardware access) are written using assembly language

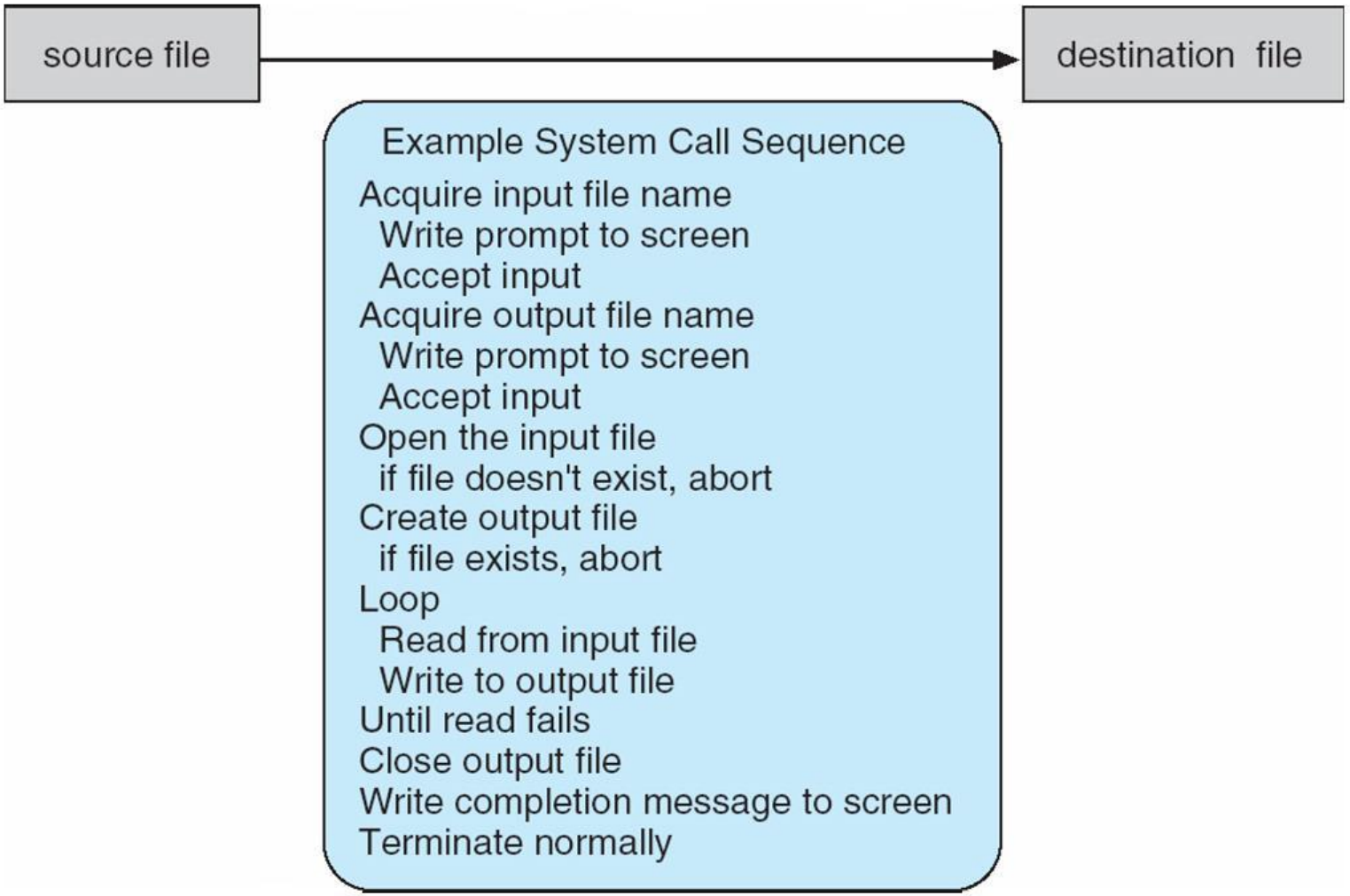
- Example of using system call

–Read data from a file and copy to another file

–open()+ read() + write()?

Example of System Calls

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



• Simple programs may make heavy use of the OS

–A system executes thousands of system calls per second

–Not user-friendly

• Each OS has its own name for each system call

–This course/textbook uses generic examples

How to use?

–Mostly accessed by programs via a high-level **API** rather than direct system call use

• **Why prefer API rather than invoking system call?**

–Easy of use: Simple programs may make heavy use of the OS

–Program portability: Compile and run on any system that supports the same API

Application Programming Interface (API)

–A set of functions that are available to application programmers

- Three most common APIs

–Win32 API for Windows

–POSIX API for POSIX-based systems

- including virtually all versions of UNIX, Linux, and Mac OS X

–Java API for the Java virtual machine (JVM)

- How to use API?

–Via a library of code provided by OS

–Libc: UNIX/LINUX with C language

Who invokes system call: System call interface

–Provided by the run-time support system, which is a set of functions built into libraries within a compiler

- How?

–intercepts function calls in the API

–invokes necessary system calls

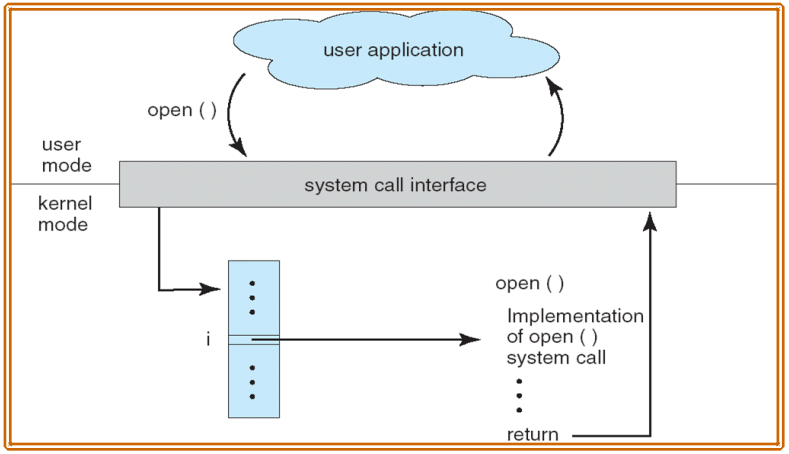
- Implementation

–Typically, a number associated with each system call

–System-call interface maintains a table indexed according to the numbers

System Calls

- Interface between applications and the OS.
  - Application uses an assembly instruction to trap into the kernel
  - Some higher level languages provide wrappers for system calls (e.g., C)or tables or stack.
- Linux has about 300 system calls
  - read(), write(), open(), close(), fork(), exec(), ioctl(),…..



Implementation Benefits

- The caller needs to know nothing about

–how the system call is implemented

–what it does during execution

–Just needs to obey API and understand what OS will do as a result call

- Most details of OS interface are hidden from programmer by API

–Managed by run-time support library

System Call Parameter Passing

Three general methods are used to pass parameters between a running program and the operating system.

–Simplest: pass the parameters in registers

- In some cases, may be more parameters than registers

–Table-based

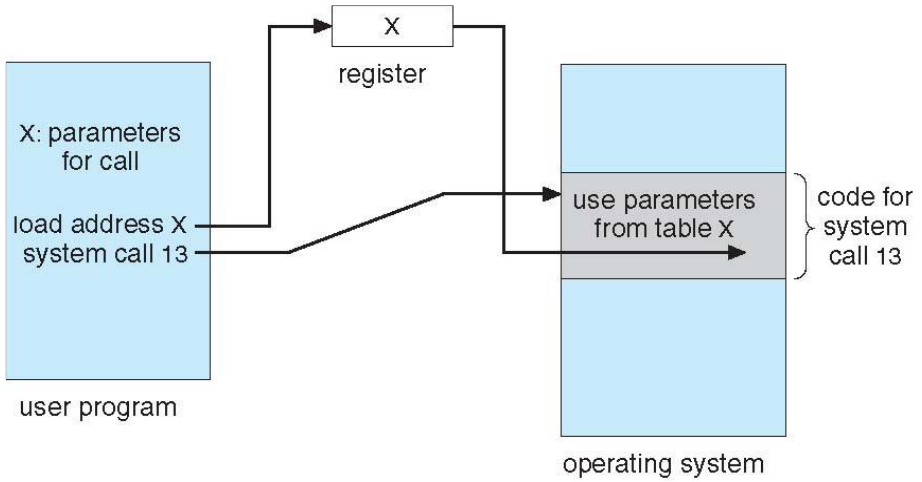
- Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
- This approach taken by Linux and Solaris

–Stack-based

- Parameters are placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system

Parameter Passing via Table

- Three general methods exist for passing parameters to the OS:
  - Parameters can be passed in registers.
  - When there are more parameters than registers, parameters can be stored in a block and the block address can be passed as a parameter to a register.
  - Parameters can also be pushed on or popped off the stack by the operating system.



How to Make a System Call in Operating Systems

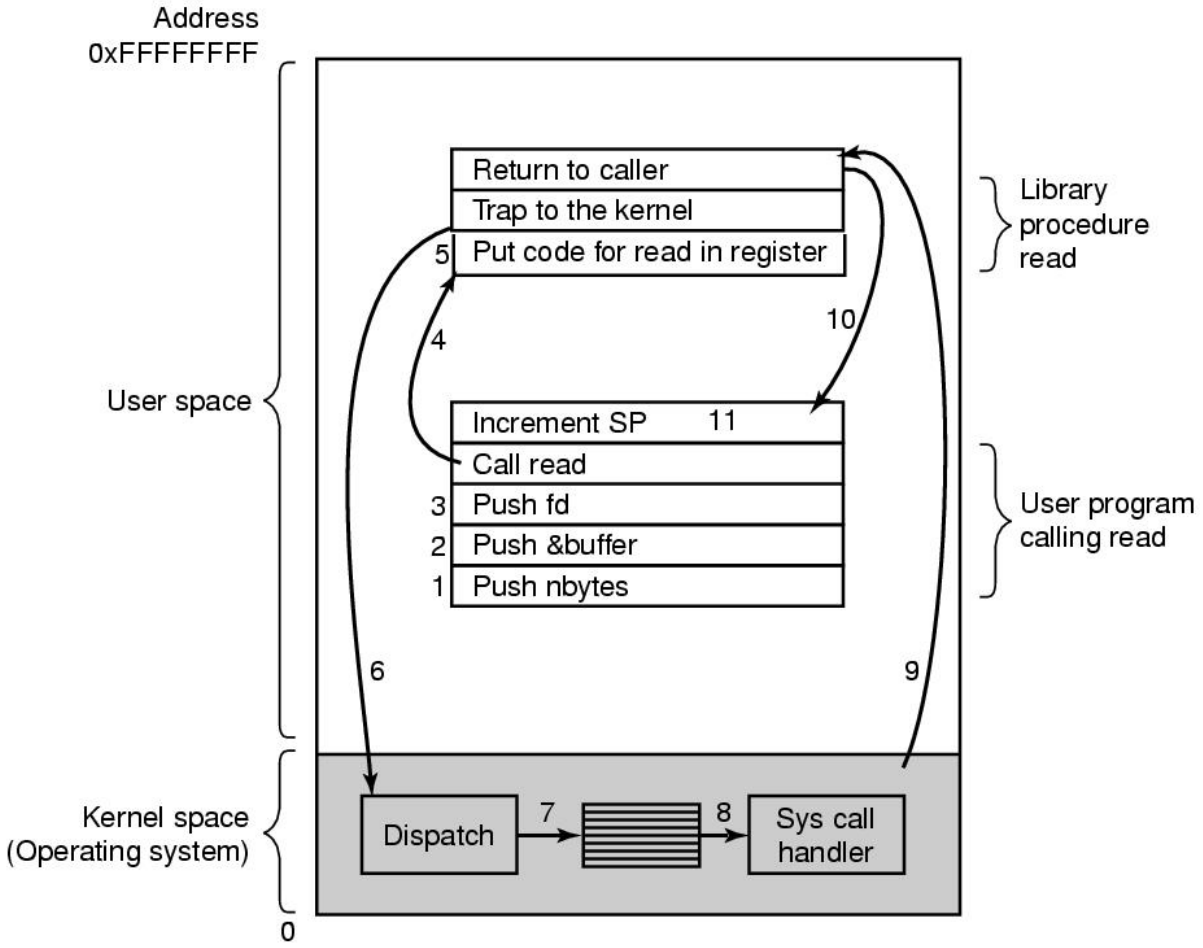


Figure: The 11 steps in making the system call read(fd, buffer, nbytes).

- The following actions occur when the user executes the (Unix) system call

  count = read(fd, buffer, nbytes)
- 1-3-the calling program pushes the parameters onto the stack. The first and third parameters are called by value, but the second one is called by its address as denoted by the & symbol.
- 4-Call the library routine, which involves pushing the return address on to the stack and jumping to the routine.
- 5-the library procedure places the system call number in a place where the operating system expects it, such as a register.
- 6-the library procedure executes a TRAP instruction to switch from user mode to kernel mode and start execution at a fixed address within the kernel.

- 7-the kernel examines the system call number and then dispatches it to the correct system call handler. This correct number is given in the table of system call handlers by pointers referenced at the system call number.
- 8-the system call handler runs.
- 9-the operation is completed, and the user is given back control once the TRAP instruction is set.
- 10-this procedure returns to the user program, like how all normal library procedures do.
- 11-the operating system has to clear the stack, so it increments it enough so that it is empty (The stack is popped).

Types of System calls

Here are the five types of system calls used in OS:

- Process Control
- File Management
- Device Management
- Information Maintenance
- Communications

Process control

This system calls perform the task of process creation, process These system calls are responsible for the **creation**, **termination**, and **management** of processes in an operating system.

Functions and Their Descriptions

- **end, abort**: Terminates a process normally (end) or abnormally (abort).
- **load, execute:** Loads a program into memory and starts its execution as a new process.
- **get process attributes, set process attributes:** Retrieves or modifies properties of a process (e.g., priority, PID, scheduling info).
- **wait for time**: Causes a process to pause or sleep for a specific period.
- **wait event, signal event:** Used for **inter-process communication** or **synchronization**. A process can wait for a certain event, and another can signal the occurrence of that event.

- **allocate and free memory**: Manages dynamic memory allocation and deallocation for processes.
- **dump memory if error:** Creates a memory dump when an error or crash occurs to help in debugging.
- **debugger for determining bugs, single step execution**: Allows debugging tools to analyze process behavior step by step.
- **locks for managing access to shared data between processes**: Prevents race conditions and ensures **mutual exclusion** when multiple processes access shared resources.

File management

These are used to **create, read, write, and manipulate files**. Examples:

Functions and Their Descriptions

- **open():** open a file
- **read() :** read from a file
- **write():** write to a file
- **close():** close a file
- **delete():** delete a file

Device management

Device management in an Operating System handles all operations related to **input/output (I/O) devices** like keyboards, mice, printers, hard drives, USBs, etc. These devices need to be **requested**, **used**, and **released** properly by the OS.

Functions and Their Descriptions

- **request device :** The process asks the OS to use a particular device (e.g., printer, scanner).
- **release device:** Frees the device when done, so others can use it.
- **read / write:** read: Takes input from the device buffer, write: Sends output to device buffer
- **Reposition:** Changes the location of the read/write head (e.g., move file pointer on a disk).
- **get device attributes:** Reads properties of the device (like status, type, speed, etc.).
- **set device attributes:** Changes those properties (e.g., switch a printer to color mode).
- **logically attach or detach devices:** Connects/disconnects a device in the OS (without physically removing it).

Information maintenance

- These system calls deal with **metadata**—which is "data about data." They help in transferring **system-related information** between the **Operating System** and **user programs**.

Functions and Their Descriptions

- **get time or date:** Retrieve the current system time/date.
- **set time or date**: Modify the system clock (requires admin permissions).
- **get system data:** Access system-level info (like OS version, memory stats, etc.).
- **set system data:** Update configuration settings (usually by the admin/system programs).
- **get/set attributes:** Retrieve or update properties of files, processes, or devices (e.g., permissions, priority).

Communications

These system calls are used to **enable and manage communication between processes**, either:

- **On the same system (inter-process communication – IPC)** or
- **Across different systems (network communication)**.

Message Passing Model: This model does not use shared memory. Communication is done by sending and receiving messages.

Functions and Their Descriptions

- **create communication connection:** Establish a link/channel between communicating processes (e.g., socket setup).
- **delete communication connection:** Terminate the communication channel.
- **send message:** Send a message to another process or system.
- **receive message:** Receive a message from another process or system.
- **to host name or process name:** Identify the destination of the message.

Shared-Memory Model: In this model, two or more processes share a region of memory, and communication happens through that shared space.

Functions and Their Descriptions

- **create/gain access to memory region:** Allocate and access a shared memory segment.
- **transfer status information:** Exchange control/status flags, progress info, etc., through shared memory.
- **attach/detach remote devices:** Connect or disconnect remote devices for data transfer or communication.

System Calls for Process Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory- and file-system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

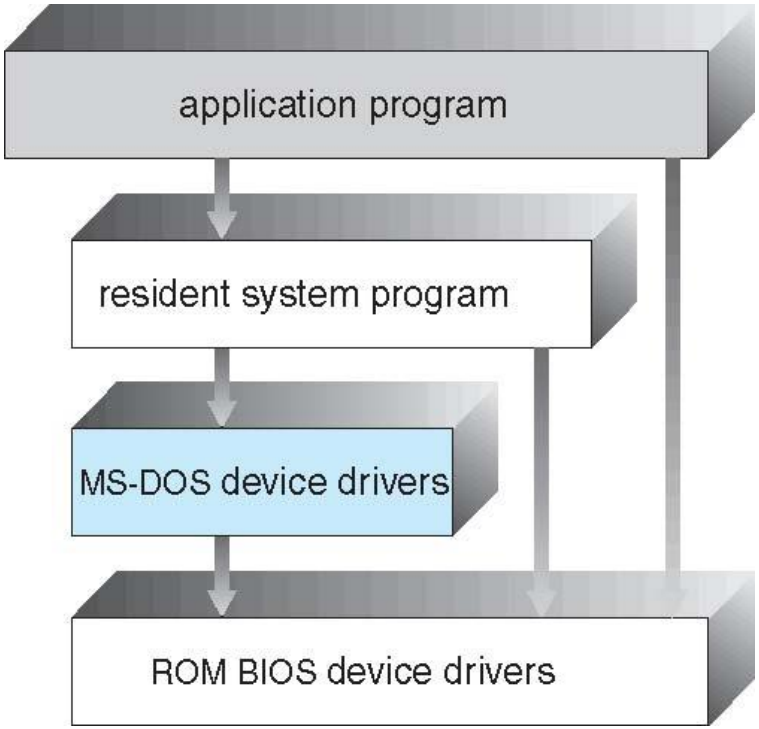
Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Operating System Structure

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 i.e.no CPU execution mode(user and kernel)



a. Application Program

- User-facing software (e.g., word processors, games, spreadsheets).
- Performs tasks by making requests to lower layers (e.g., reading files, printing documents).
- Microsoft Word, Lotus 1-2-3.

b. Resident System Program

- Core OS utilities and services that stay in memory.
- Provides file management (e.g., COMMAND.COM), memory management, and interrupt handling.
- MS-DOS kernel, shell commands (DIR, COPY).

c. MS-DOS Device Drivers

- Software drivers for hardware (loaded dynamically or via CONFIG.SYS).
- Translates OS requests into hardware-specific commands.
- Drivers for keyboards, disks (ANSI.SYS, HIMEM.SYS).

d. ROM BIOS Device Drivers

- Firmware stored in the motherboard's ROM.
- Low-level hardware control (bootup, disk access, screen output).
- BIOS interrupts (INT 13h for disk access, INT 10h for video).

Layered Approach

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Figure. Structure of the operating system.

- The layering was done by convention, i.e. there was no enforcement by hardware and the entire OS is linked together as one program. This is true of many modern OS systems as well (e.g., linux).

- **Layer 0** - This layer dealt with allocation of the processor, switching between processes when interrupts occurred or timers expired.
- **Layer 1** - This layer did the memory management.
- **Layer 2** - This layer handled the communication between each process and the operator console.
- **Layer 3** - This layer took care of managing the I/O devices and buffering the information streams to and from them.
- **Layer 4** - On this layer, user programs were found.
- **Layer 5** - On this layer, the system operator process was located.

Monolithic Approach

- In **monolithic kernel** the author tells us that it runs every basic system service in kernel space.

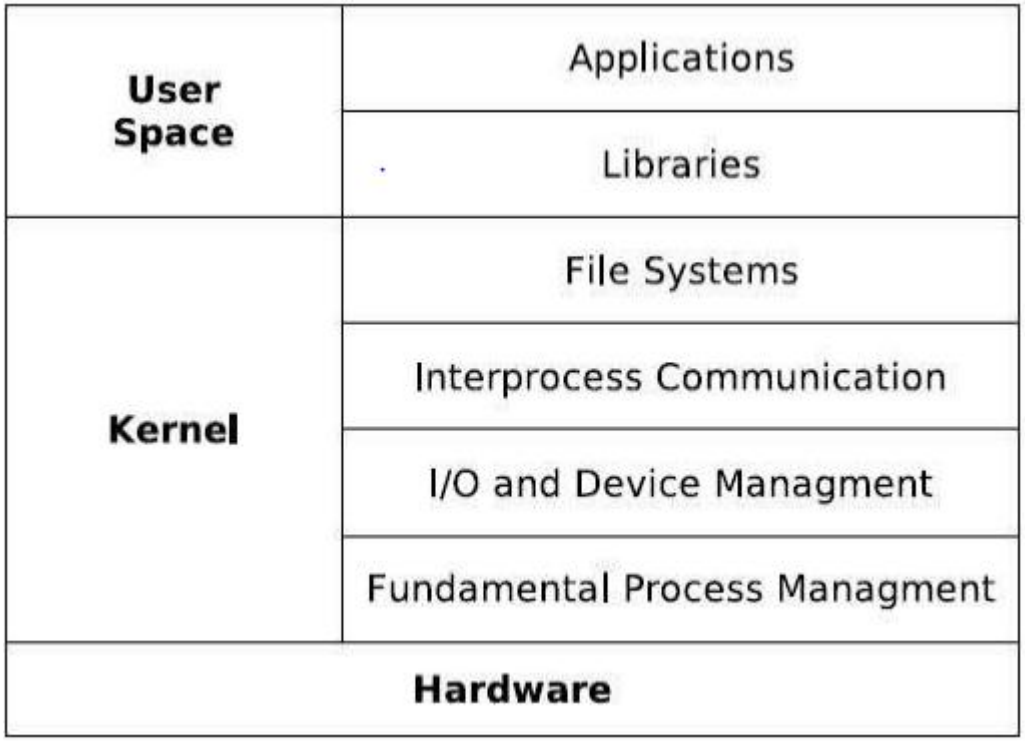


Figure: Monolithic kernel based operating system

Monolithic kernel

- The entire O.S. is placed inside the kernel
- It runs as a single large process
- As all the services are placed inside the kernel, they have a single address space
- It is bigger in size
- It is easy to implement/code
- Performance is high (As kernel can invoke any function directly as everything is placed in the kernel)
- Less Secure (If one service fails, entire system crashes)

Microkernel Approach

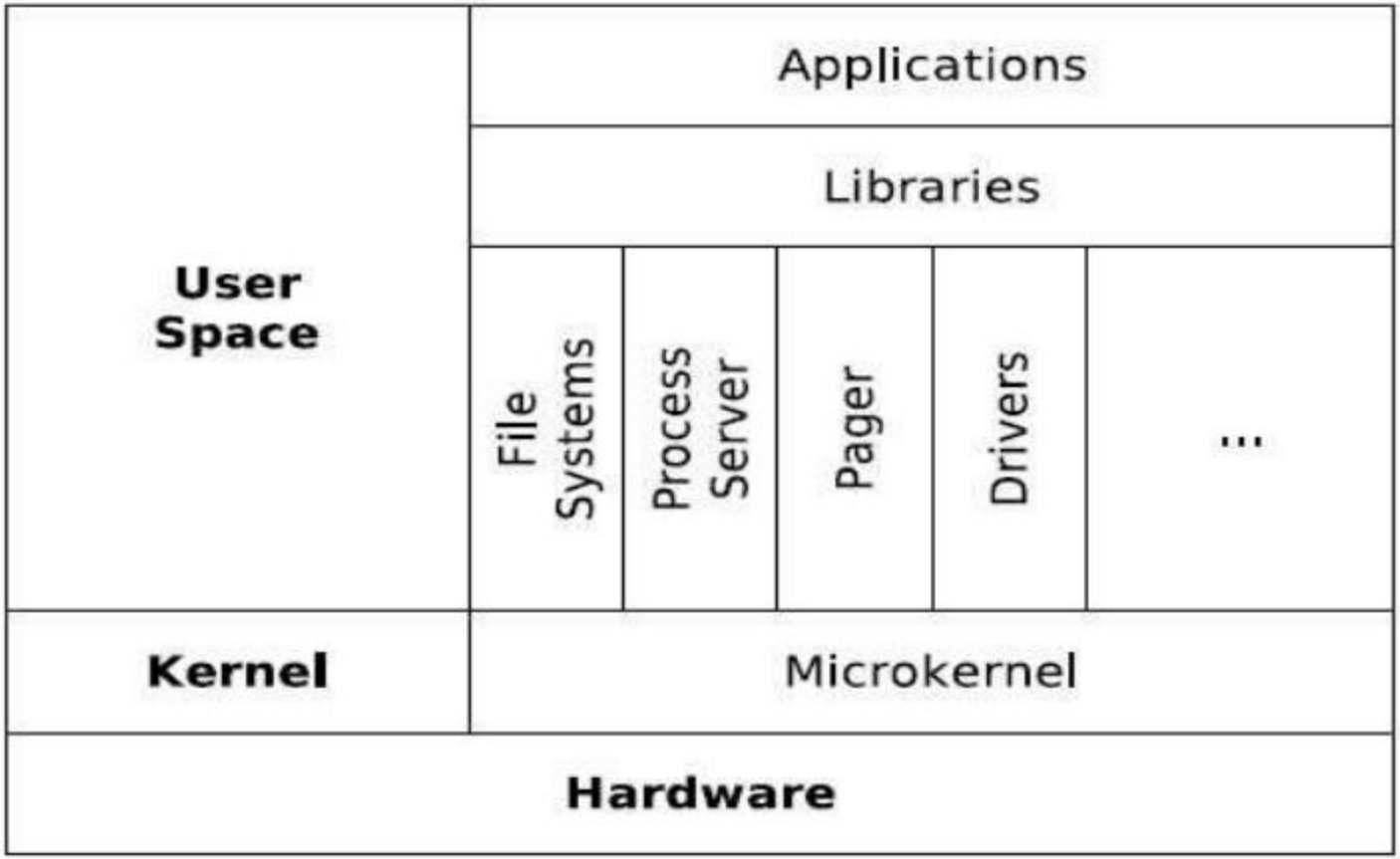
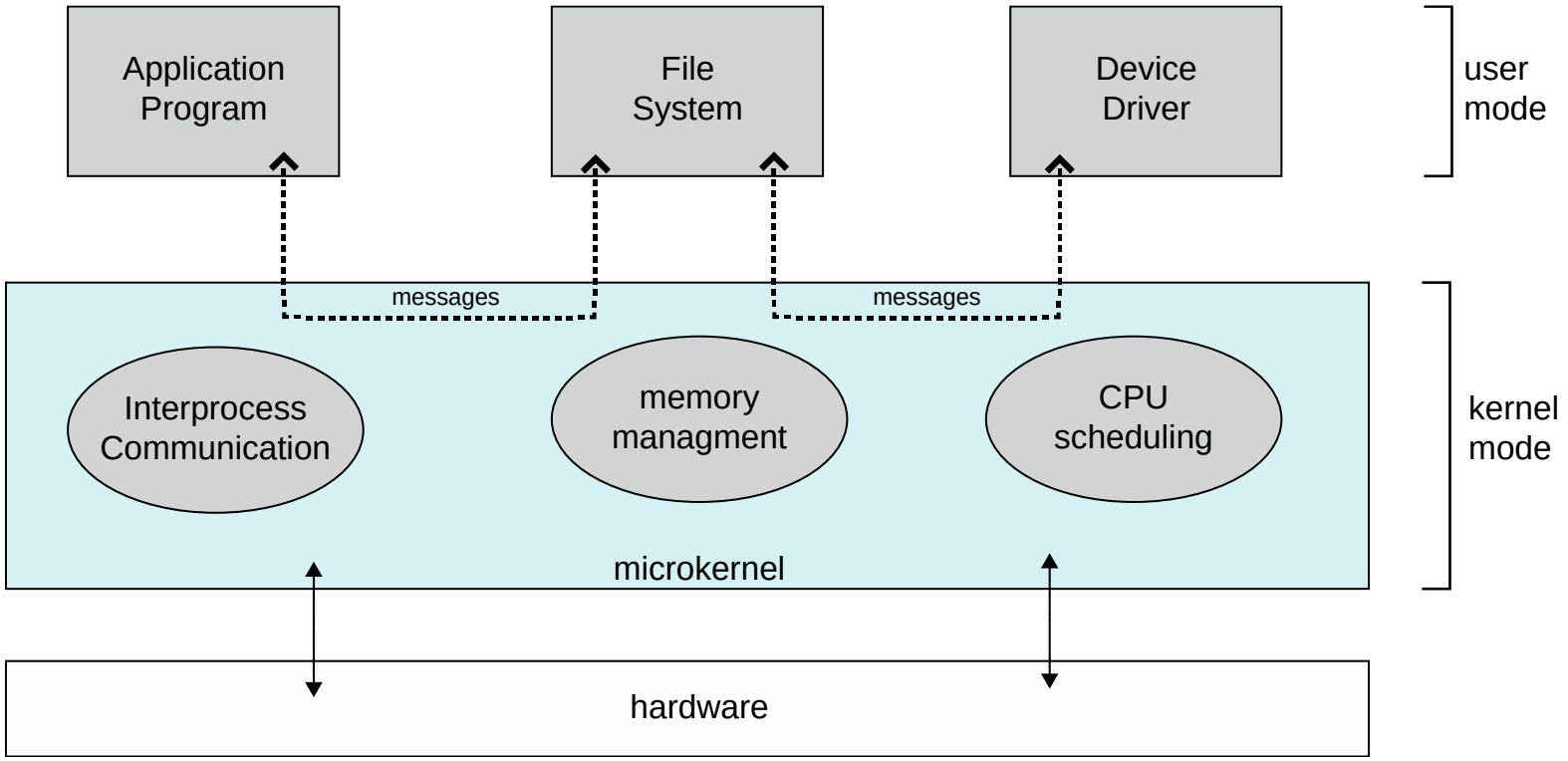


Figure: Micro kernel based operating system

Microkernel System Structure

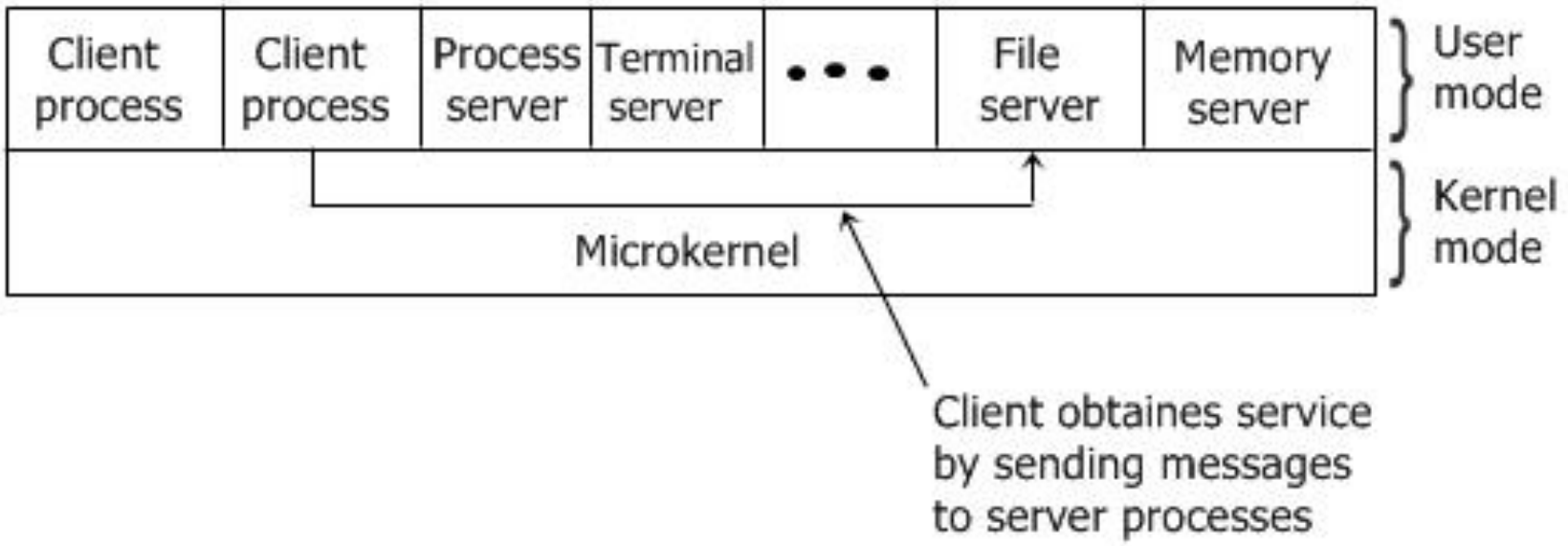


Microkernel

- Only bare minimum code is placed inside the kernel (only basic memory management and Inter Process Communication code) Here the kernel is broken down into processes called as servers
- As services(Servers provide services) are separated they have different address spaces -It is smaller in size
- It is tough to implement/code
- Performance is low (As servers are separated, so to invoke services from other servers IPC(Inter Process Communication) is needed which requires kernel's permission and thus increases access time and lowers the performance)
- More Secure (Even if one service crashes, others can function properly because of separation)

Client-Server Model

- A slight variation of the microkernel idea is to distinguish two classes of processes, the **servers**, each of which provides some service, and the **clients**, which use these services. This model is known as the **client-server** model.
- In the client-server model, as shown in the figure given below, all the kernel does is handle the communication between the clients and the servers.



- By splitting the operating system (OS) up into parts, each of which only handles one fact of the system, such as file service, terminal service, process service, or memory service, each part becomes small and manageable.
- The adaptability of the client-server model, to use in distributed system is the advantage of this model.

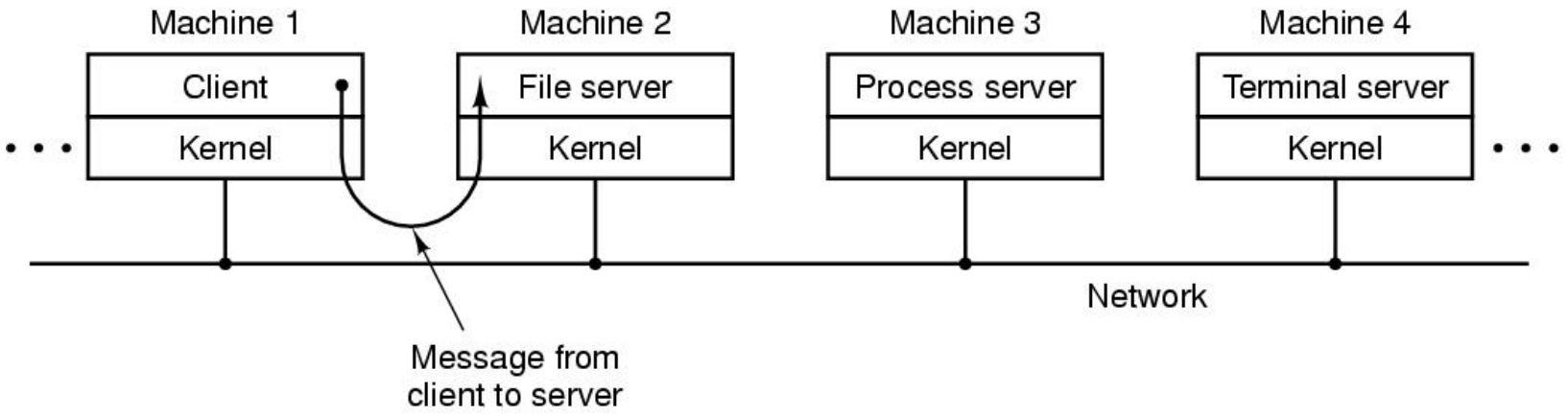
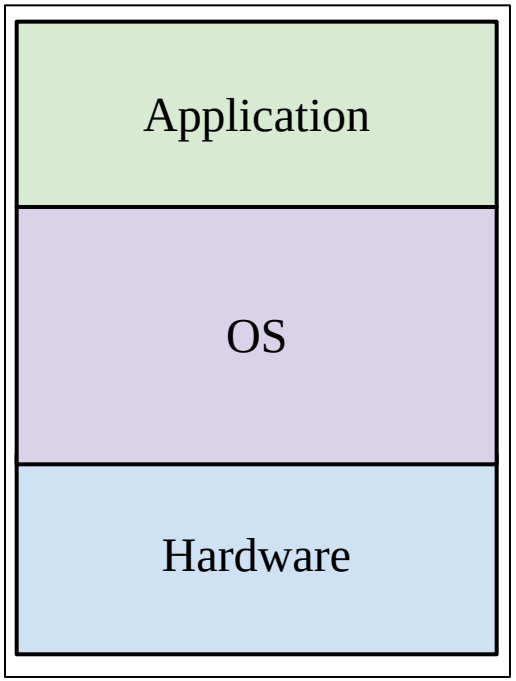


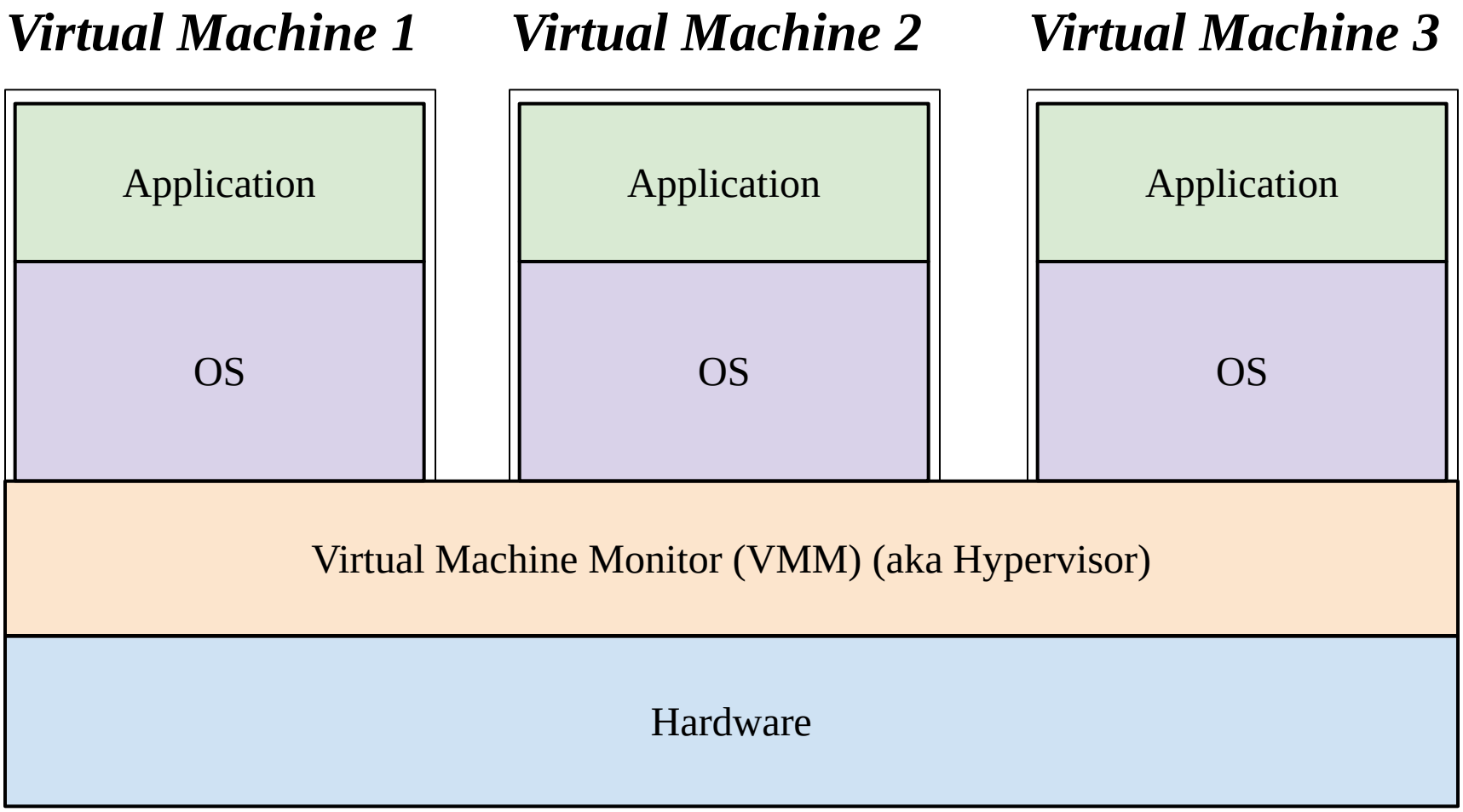
Figure : The client-server model over a network.

Virtual Machines

Physical Machine



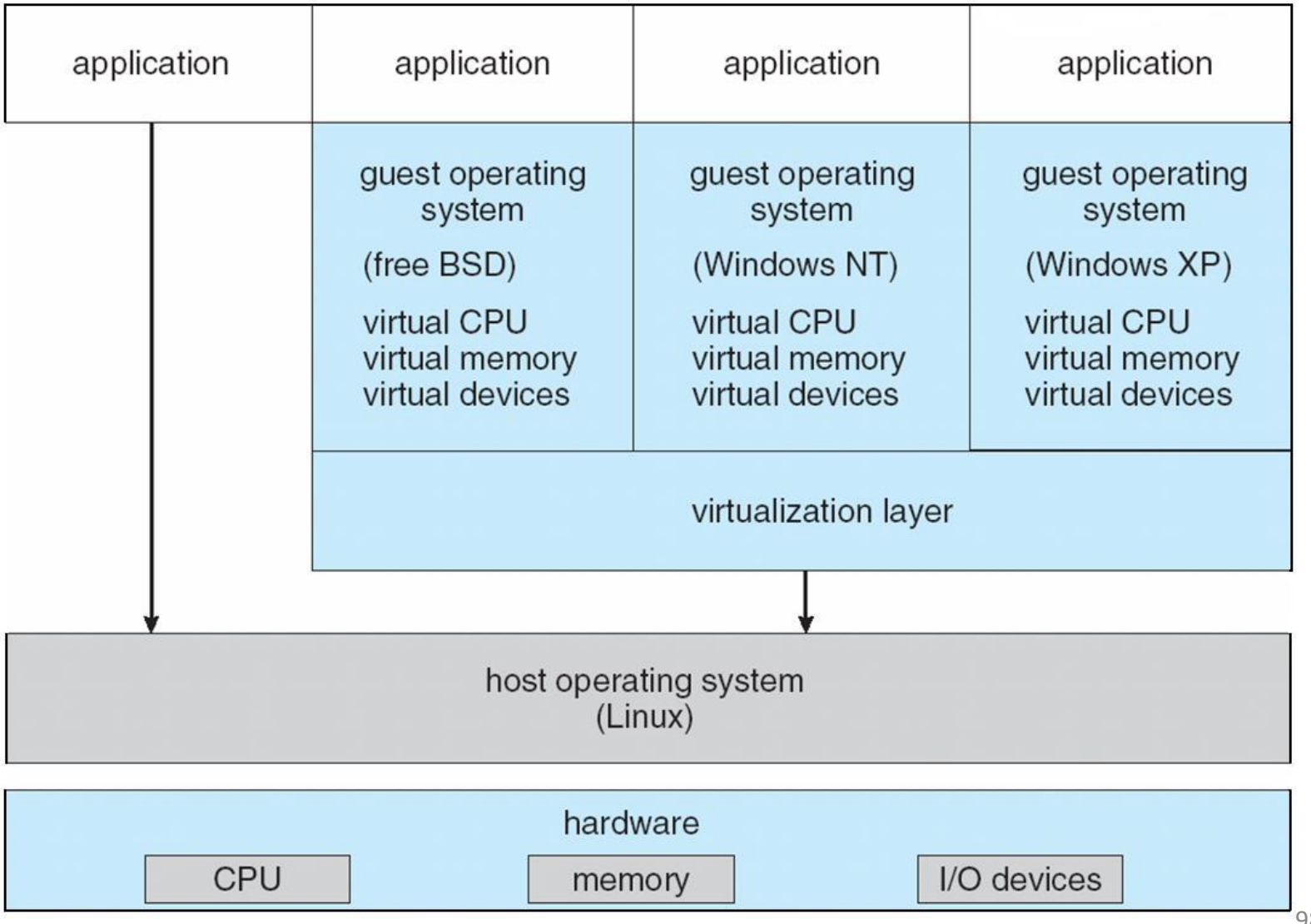
Virtual Machines



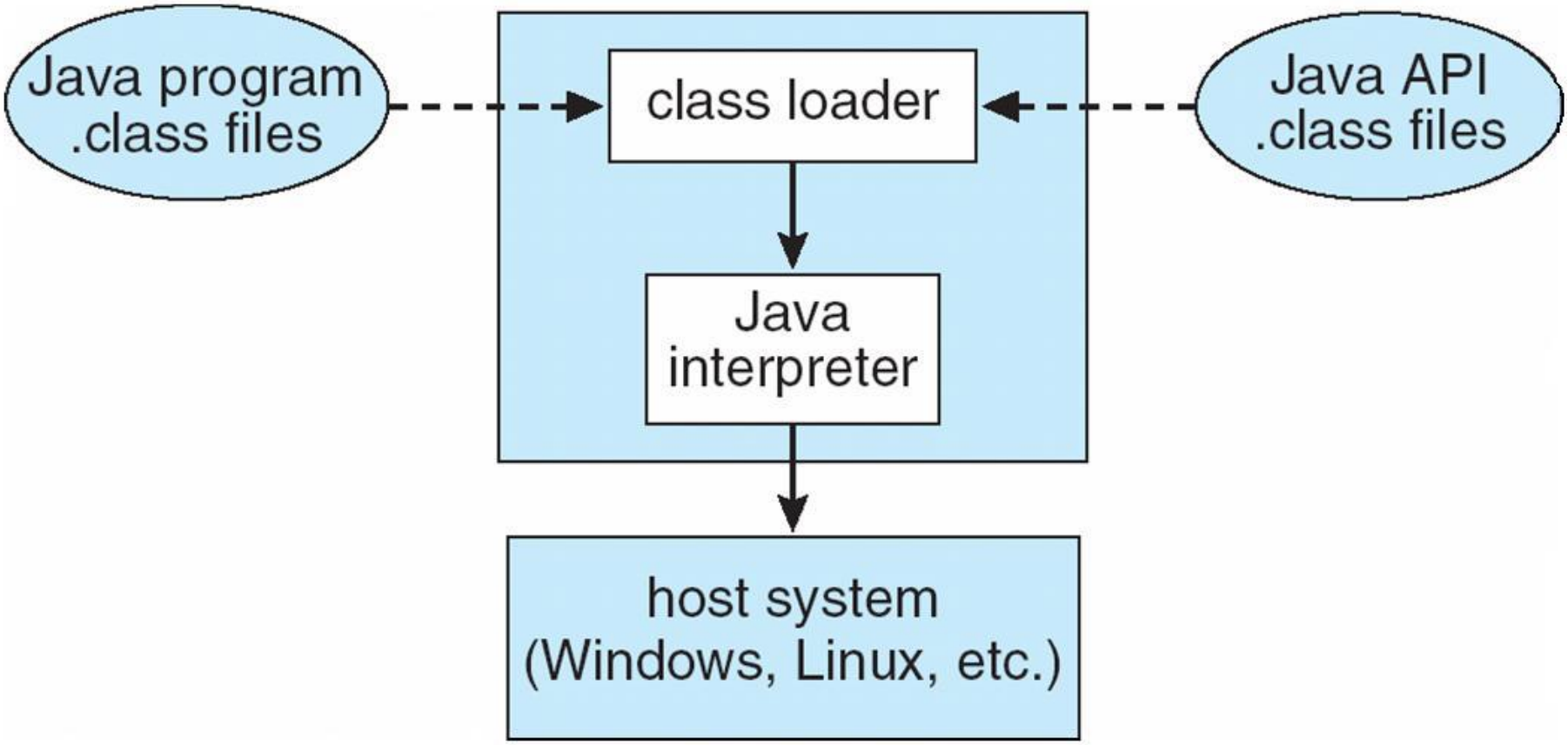
Virtual Machines

- Virtual machine is an illusion of a real machine. It is created by a real machine operating system, which make a single real machine appears to be several real machine.
- The best example of virtual machine architecture is IBM 370 computer.
- Vmware, Java virtual machine, .Net framework etc.
- Use cases
  - Resource configuration
  - Running multiple OSes, either the same or different OSes
  - Run existing OS binaries on different architecture

VMware Architecture



The Java Virtual Machine



The Shell

- The operating system is the code that carries out the system calls.
- Editors, compilers, assemblers, linkers, and command interpreters definitely are not part of the operating system, even though they are important and useful.
- A **Shell** provides you with an interface to the Unix system.
- Although it is not part of the operating system, it makes heavy use of many operating system features and thus serves as a good example of how the system calls can be used.
- It is also the primary interface between a user sitting at his terminal and the operating system

- A shell is software that provides an interface between users and operating system of a computer to access the services of a kernel.

There are two types of shell:

Command-Line Shell (CLI Shell): This type of shell allows users to interact with the system by typing commands into a text interface.

Examples:

- **Bash (Bourne Again Shell):**Common in Linux
- **sh (Bourne Shell):** Original UNIX shell
- **csh (C Shell):**C-like syntax
- **ksh (Korn Shell):**Combines features of sh and csh
- **zsh (Z Shell):** Extended features for scripting
- **Command Prompt (cmd.exe):** Windows command-line interpreter
- **PowerShell:** Advanced shell from Microsoft for automation and configuration

Graphical User Interface (GUI) Shell: This type of shell provides a **visual interface** that allows users to interact with the operating system using windows, icons, menus, and pointers.

Examples:

- **Windows Explorer / Windows Shell** – GUI shell for Microsoft Windows
- **GNOME Shell** – Common in Linux distributions using GNOME
- **KDE Plasma Shell** – GUI shell for KDE desktop environment

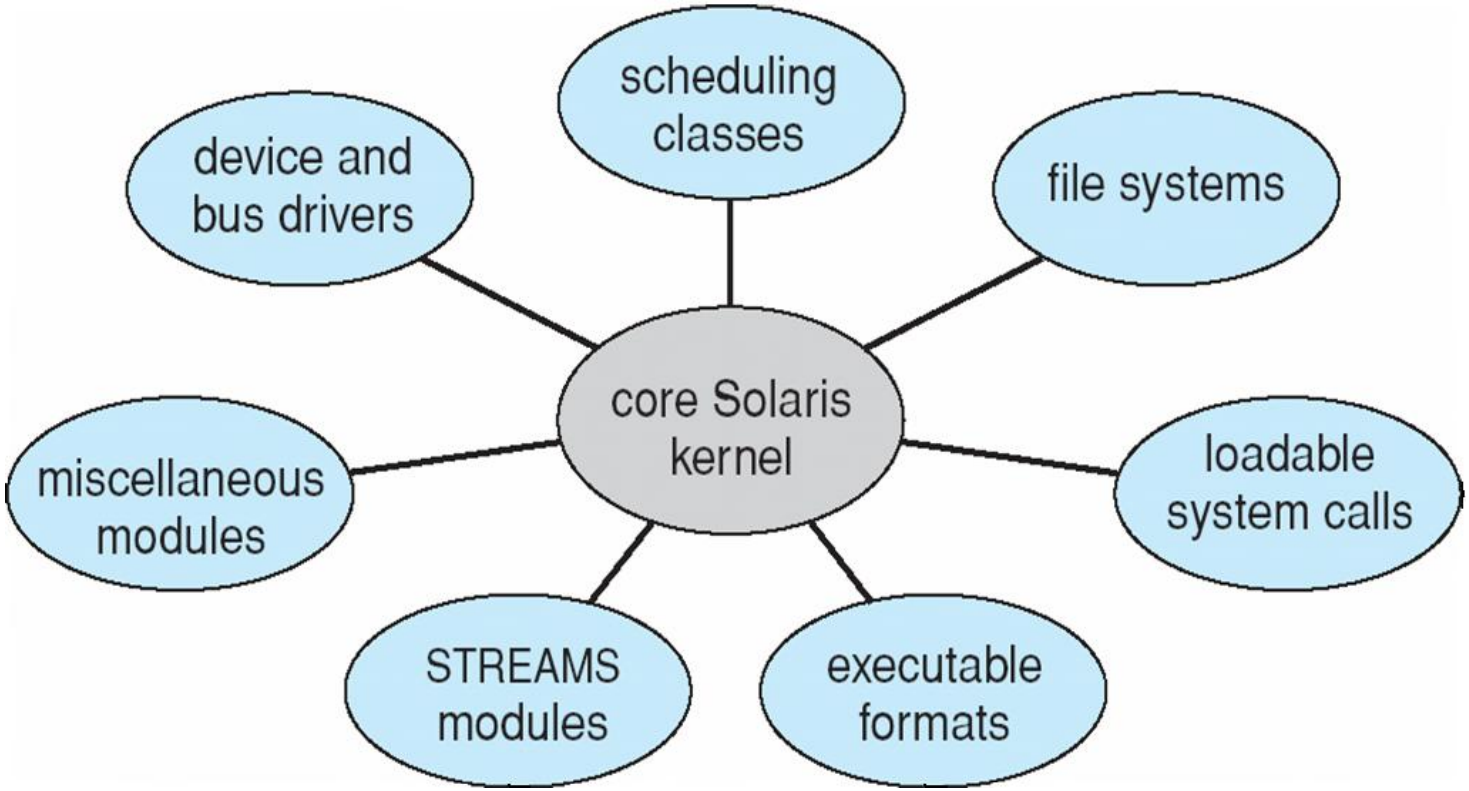
A Simple Shell

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
        type_prompt( );                         /* display prompt on the screen */
        read_command(command, parameters);      /* read input from terminal */

        if (fork( ) != 0) {                     /* fork off child process */
             /* Parent code. */
             waitpid(–1, &status, 0);           /* wait for child to exit */
        } else {
             /* Child code. */
             execve(command, parameters, 0);    /* execute command */
        }
}
```

Modules

- Many 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

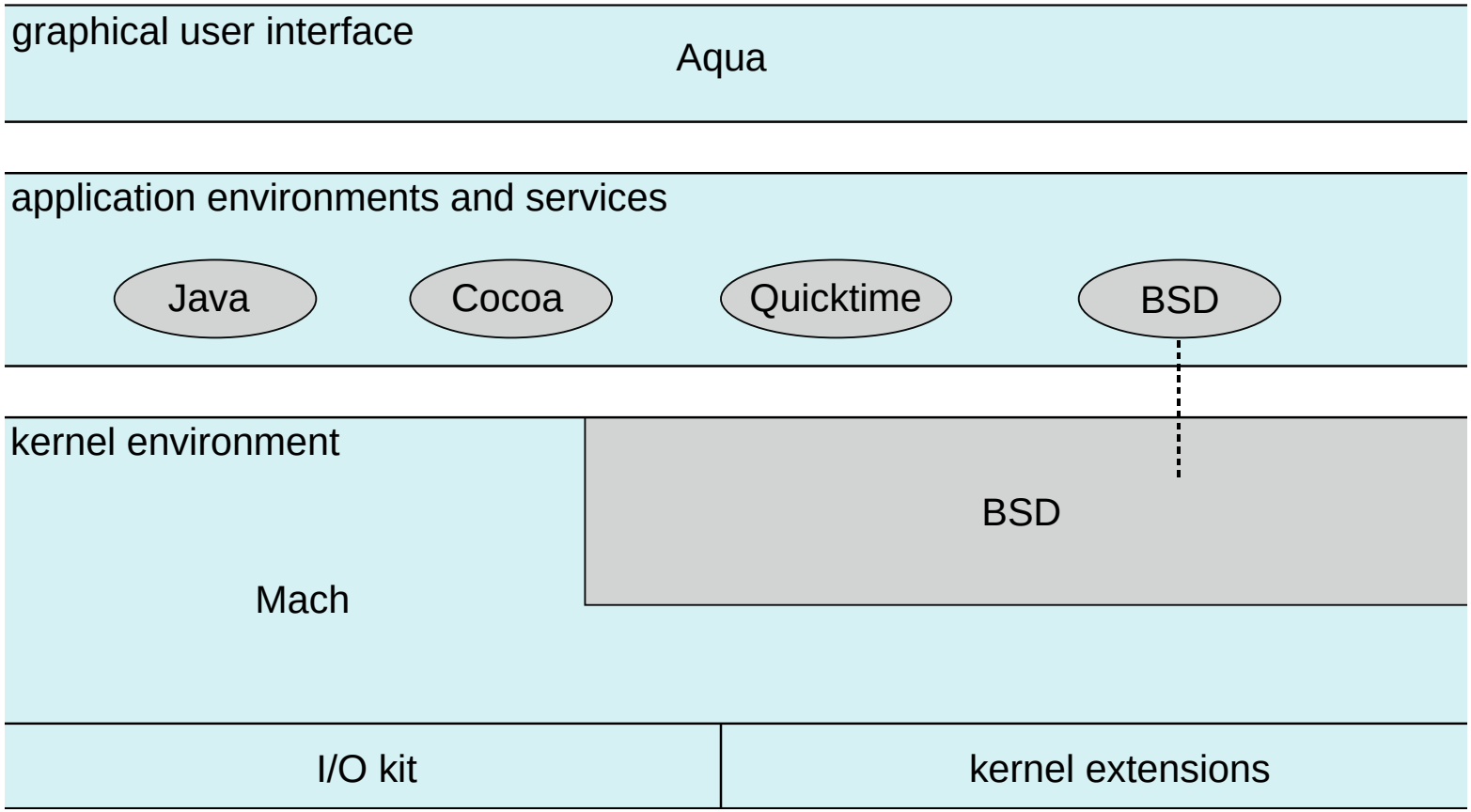
Solaris Modular Approach



Hybrid Systems

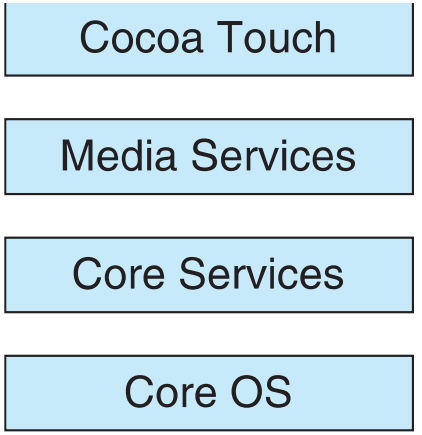
- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic,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



iOS

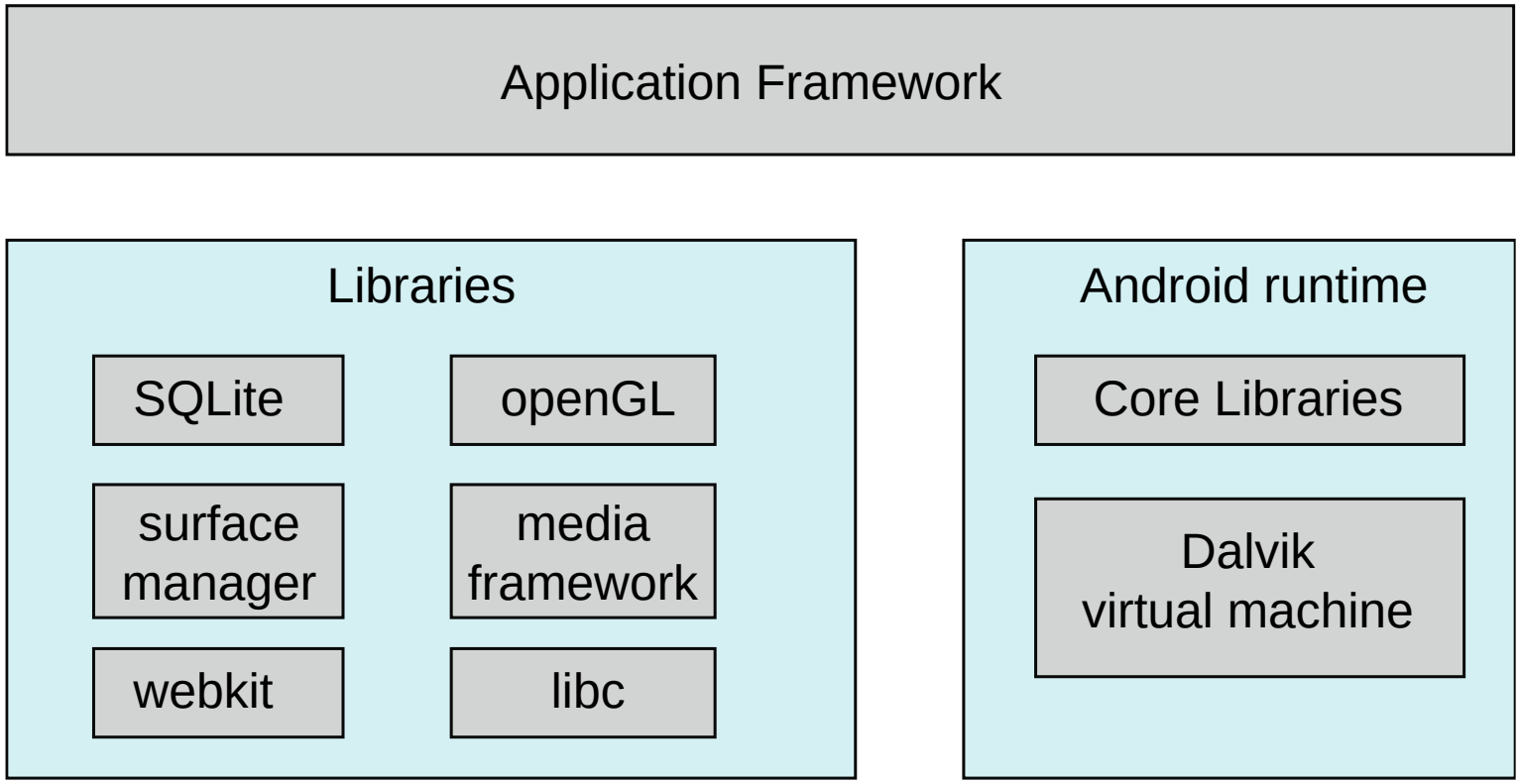
- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, 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
- 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



Thank You