

Unit - 1 Contents:-

Operating System Introduction, Structures - Simple Batch, Multi-programmed, Time-shared, Personal Computer, Parallel, Distributed Systems, Real-Time Systems, System components, Operating System services, System Calls.

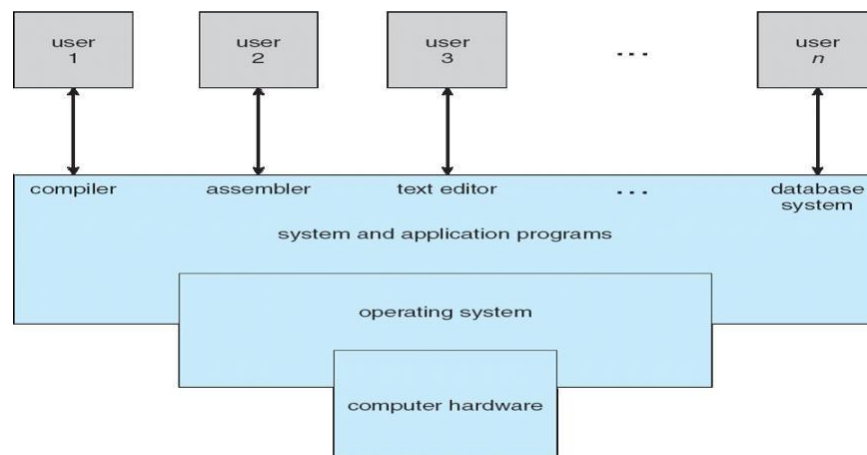
Operating System:-

A program that acts as an intermediary between a user of a computer and the computer hardware.

Operating system goals:

- Execute user programs and make solving user problems easier
- Make the computer system convenient to use
- Use the computer hardware in an efficient manner.

Computer System Structure or Components of a Computer System:-



Computer system can be divided into four components

- Hardware – provides basic computing resources
 - ☐ CPU, memory, I/O devices
- Operating system
 - ☐ Controls and coordinates use of hardware among various applications and users
- Application programs – define the ways in which the system resources are used to solve the computing problems of the users
 - ☐ Word processors, compilers, web browsers, database systems, video games
- Users
 - ☐ People, machines, other computers

Components of a Computer System

- To understand more fully the operating system's role, we next explore operating systems from two viewpoints: that of the **user** and that of the **system**.
- **User View**: The user's view of the computer varies according to the interface being used.
- **System View**: In this context, we can view an operating system as a **resource allocator**. **A computer system has many** resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices, and so on.
- The operating system acts as the manager of these resources.

Components of a Computer System

- A computer system can be divided roughly into four components: the **hardware**, the **operating system**, the **application programs**, and the **users**.
- The hardware—the central processing unit (CPU), the memory, and the input/output (I/O) devices—provides the basic computing resources for the system.
- The **application programs**—such as word processors, spreadsheets, compilers, and Web browsers—define the ways in which these resources are used to solve users' computing problems

G RAVI KUMAR CSE@CMRCET
OS

63

Operating System Definition

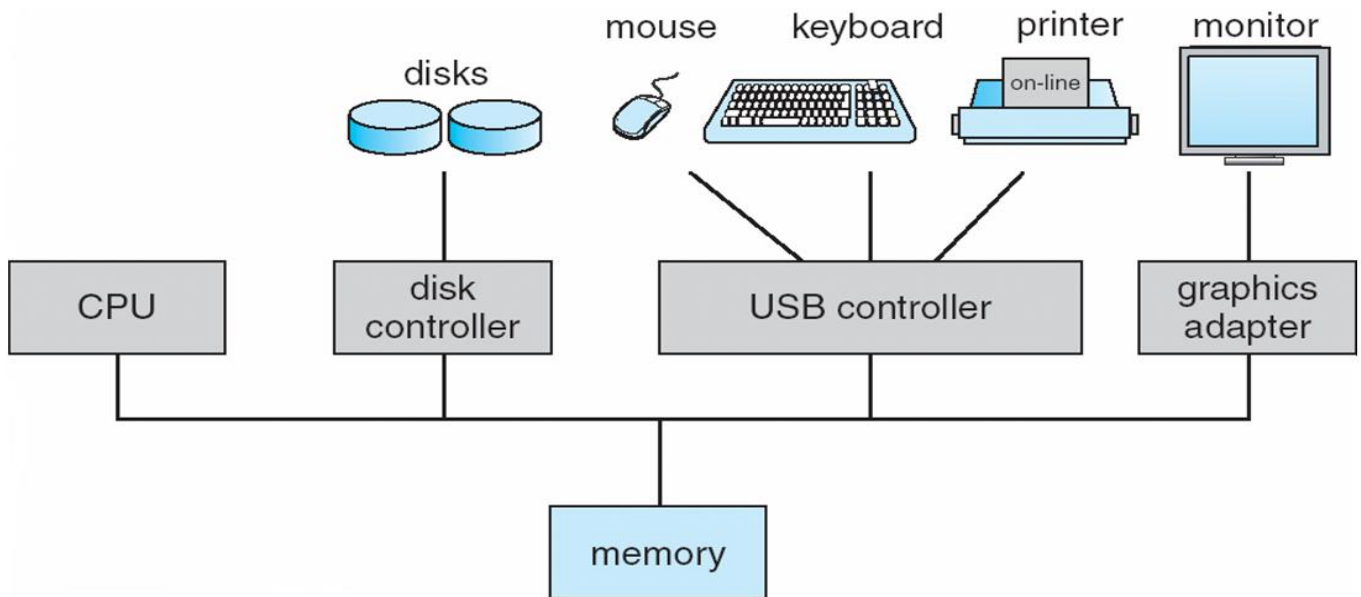
- OS is a resource allocator.
- Manages all resources
- Decides between conflicting requests for efficient and fair resource use.
- OS is a control program
- Controls execution of programs to prevent errors and improper use of the computer.
- No universally accepted definition.
- Everything a vendor ships when you order an operating system is good approximation but varies wildly
 - "The one program running at all times on the computer" is the kernel. Everything else is either a system program (ships with the operating system) or an application program

Computer Startup

- **Bootstrap program** is loaded at power-up or reboot
- Typically stored in ROM or EPROM, generally known as **firmware**
- Initializes all aspects of system
- Loads operating system kernel and starts execution.

Computer System Organization

- Computer-system operation
- One or more CPUs, device controllers connect through common bus providing access to shared memory
- Concurrent execution of CPUs and devices competing for memory cycles



Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt.

Ex: On UNIX, the first system process is — “init” and it starts many other daemons.

Operating-System Structure

• The Evolution Operating System

- Simple Batch Systems.
- Multi programmed Systems
- Time Shared Systems
- Personal Computers
- Parallel systems
- Distributed Systems
- Real time systems

1) Simple Batch Systems.

- Early computers are very expensive and therefore it was important to maximize processor utilization.
- To improve utilization the concept of Batch OS was developed.
- The first Batch OS was developed in the mid 1950's by General Motors for use on an IBM 701.
- It is refined & implemented on the IBM704. IBM 700 Series computers.



- Early 1960's, a no. of vendors had developed batch OS for their computer systems.
- Such as IBMSYS, IBM OS for 7090/7094 Computers.
- The idea behind this simple batch processing scheme is the use of piece of a s/w known as Monitor.
- With this type of OS, the user no longer has direct access to the processor.
- Instead, the user submits the job on cards or tape to a computer operator.
- Who batches the jobs together sequentially & places the entire batch on an input device, for use by the monitor, To understand how this scheme works?



A **punched card** or **punch card** is a piece of stiff paper that can be used to contain **digital data** represented by the presence or absence of holes in predefined positions. Digital data can be used for **data processing** applications or used to directly control **automated machinery**.

- Let us look at it from two points of view.
 - The monitor
 - The processor

Batch Operating System diagram is in class notes.

1) The monitor point of view:

- The monitor controls the sequence of events.
- For this, monitor must be in main memory & available for the execution.
- The portion is referred as Resident Monitor.
- The rest of the monitor consists of utilities and common functions that are loaded as subroutines to the user program at the beginning of any job that requires them.

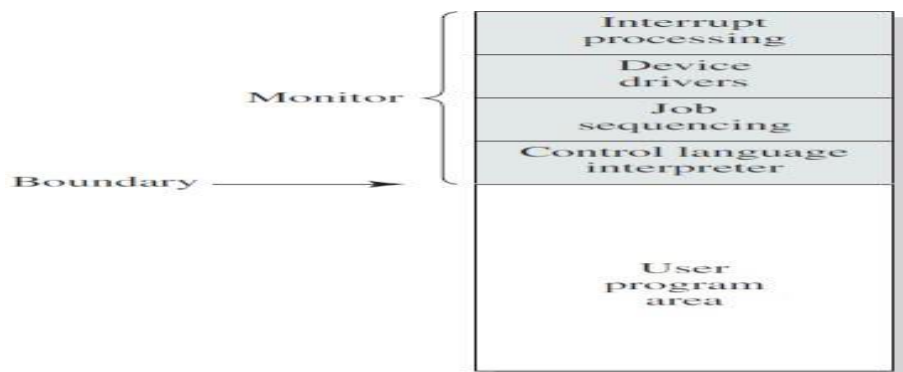


Figure 2.3 Memory Layout for a Resident Monitor

- The monitor reads in jobs one at a time from input device (Tape or Card reader). As it is read in, the current job is placed in the user program area and control is passed to this job. When the job is completed it returns controls to the monitor. Which immediately reads in the next job?
- The result of each job is sent to an o/p device such as a printer for delivery to the

user.

2) Processor point of view:-

- The processor is executing instructions from the portion of main memory containing the monitor. The processor will then execute the instructions in the user program until it encounters an ending or error condition.
- The monitor performs a scheduling function.
- A batch of jobs is queued up and jobs are executed. With each job, instructions are in a primitive form of JCL (Job control Language).
- This is a special type of programming language used to provide instructions to the monitor.

For ex. FORTRAN PLUS

- All FORTRAN instructions and data are on a separate punched cards or tape.
- In addition the job includes Job Control Instructions, which are denoted by the beginning \$.

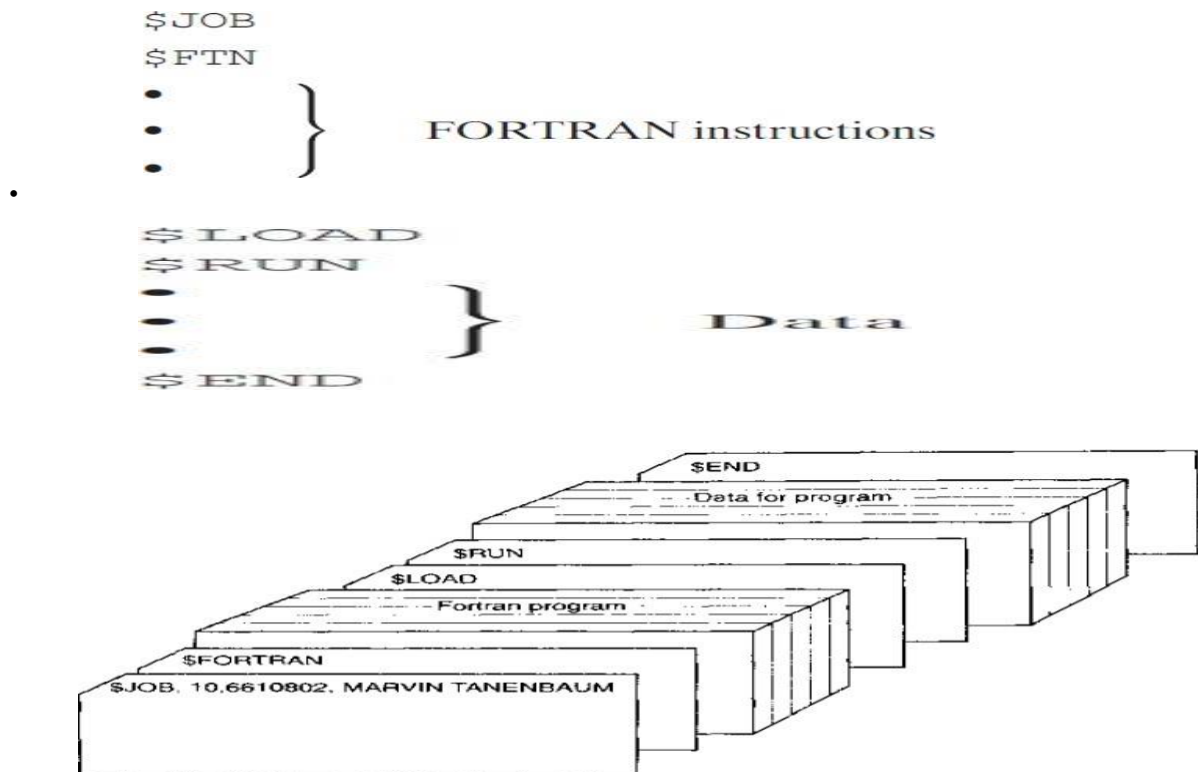


Figure 1-3. Structure of a typical FMS job.

- To execute this job, the monitor reads the \$FTN line and loads the appropriate language compiler from its mass storage(tape).
- The compiler translates the user's program into object code, which is stored in memory or storage.
- If it is stored in memory, the operation is referred to as "compile, load and go".
- If it is stored on tape, then the \$LOAD instructions is required.

OPERATING SYSTEMS UNIT-1

- The monitor invokes the loader, which loads the object program into memory and transfers control to it.
- During the execution of the user program, any input instruction causes one line of data to be read.
- The input instruction in the user program causes an input routine that is part of the OS to be invoked.
- The monitor or batch OS is simply a computer program.
- It relies on the ability of the processor to fetch instructions from various portions of main memory.
- Certain other hardware features are also desirable.
 - Memory protection
 - Timer
 - Privileged instructions
 - Interrupts

- **Memory protection:** While the user program is executing, it must not alter the memory area containing the monitor. If such an attempt is made, the processor hardware should detect an error and transfer control to the monitor. The monitor would then abort the job, print out an error message, and load in the next job.
- **Timer:** A timer is used to prevent a single job from monopolizing the system. The timer is set at the beginning of each job. If the timer expires, the user program is stopped, and control returns to the monitor.

Privileged instructions: Certain machine level instructions are designated privileged and can be executed only by the monitor. If the processor encounters such an instruction while executing a user program, an error occurs causing control to be transferred to the monitor. Among the privileged instructions are I/O instructions, so that the monitor retains control of all I/O devices. This prevents, for example, a user program from accidentally reading job control instructions from the next job. If a user program wishes to perform I/O, it must request that the monitor perform the operation for it.

- **Interrupts:** Early computer models did not have this capability. This feature gives the OS more flexibility in relinquishing control to and regaining control from user programs.

Considerations of memory protection and privileged instructions lead to the concept of modes of operation. A user program executes in a **user mode**, in which certain areas of memory are protected from the user's use and in which certain instructions may not be executed. The monitor executes in a system mode, or what has come to be called **kernel mode**, in which privileged instructions may be executed and in which protected areas of memory may be accessed.

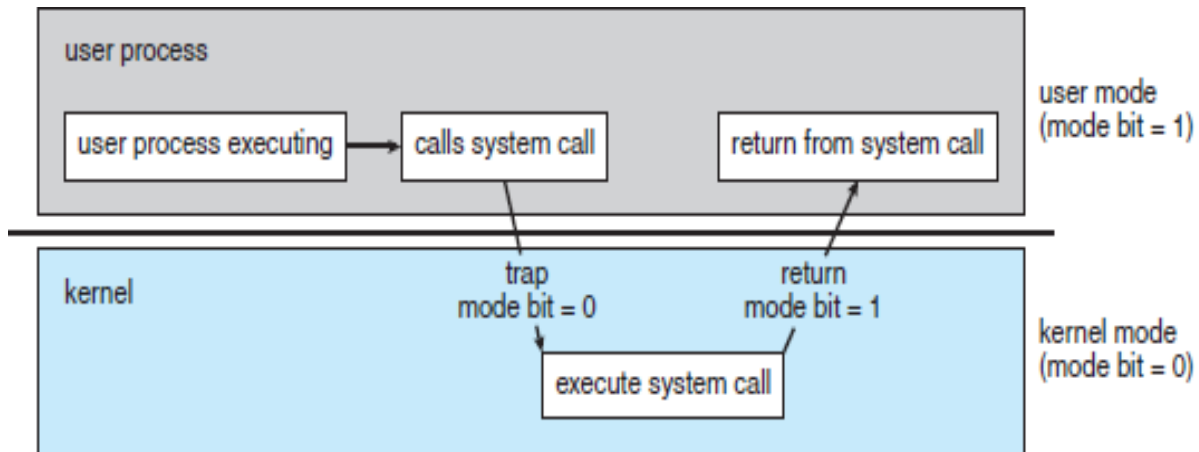


Figure 1.10 Transition from user to kernel mode.

OS Operations

1. Dual mode operation
2. Timer

1. Dual mode operation

In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and userdefined code. The approach taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution.

For that we need two separate *modes* of operation: **user mode** and **kernel mode** (also called **supervisor mode**, **system mode**, or **privileged mode**).

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

With the mode bit, we can distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user. When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), the system must transition from user to kernel mode to fulfill the request.

Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.

The dual mode of operation provides us with the means for protecting the operating system from errant users

2 Timer

We must ensure that the operating system maintains control over the CPU. We cannot allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the operating system. To accomplish this goal, we can use a **timer**.

A timer can be set to interrupt the computer after a specified period. The period may be fixed (for example, 1/60 second) or variable (for example, from 1 millisecond to 1 second). A **variable timer** is generally implemented by a fixed-rate clock and a counter. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.

We can use the timer to prevent a user program from running too long. A simple technique is to initialize a counter with the amount of time that a program is allowed to run.

A program with a 7-minute time limit, for example, would have its counter initialized to 420. Every second, the timer interrupts, and the counter is decremented by 1. As long as the counter is positive, control is returned to the user program. When the counter becomes negative, the operating system terminates the program for exceeding the assigned time limit.

Multiprocessing

Multiprocessing is basically executing multiple processes at the same time on multiple processors, whereas multiprogramming is keeping several programs in main memory and executing them concurrently using a single CPU only.

A program loaded into memory and executing is called a **process**.

If several jobs are ready to be brought into memory and if there is not enough room for all of them, then the system must choose among them. Making this decision is called “**Job Scheduling**”. If several jobs are ready to run at the same time, the system must choose among them. Making this decision is called “**CPU Scheduling**”.

ii) Multiprogrammed Batch Systems

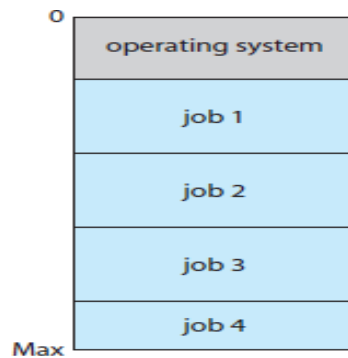


Figure 1.9 Memory layout for a multiprogramming system.

One of the most important aspects of operating systems is the ability to multiprogram. The idea is as follows: The operating system keeps several jobs in memory simultaneously (Figure 1.9).

In general, Main Memory is too small to accommodate all jobs, the jobs are kept initially on the disk is called **Job Pool**.

This pool consists of all processes residing on disk and waiting allocation of main memory.

The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete.

In a **Non-Multiprogrammed** system, the CPU would sit idle.

In a **Multiprogrammed** system, the operating system simply switches to and executes another job. When *that* job needs to wait, the CPU switches to *another* job, and so on.

Eventually, the first job finishes waiting and gets the CPU back. As long as at least one job needs to execute, the CPU is never idle.

Multiprogrammed system provides an environment in which various system resources(CPU, MEMORY, Peripheral Devices) are utilized effectively, but they don't provide, the user interaction with computer system.

Multiprogramming works on the concept of context switching.

Multitasking or Time Sharing: Multitasking is an logical extension of multiprogramming. In these systems, the CPU executes multiple jobs, by switching

OPERATING SYSTEMS UNIT-1

among them, but the switches occur so frequently that the users can interact with each program while it is running.

Multitasking is based on the time sharing alongside the concept of context switching.

In Time sharing system, each process is assigned some specific quantum of time for a process to execute. As soon as time quantum of one process expires, another process begins its execution.

Here also a context switch is occurring but it is so fast that the user is able to interact with each program separately while it is running. But actually only one process/task is executing at a particular instant of time. For example 4 processes and 5 nano seconds.

- With the automatic job sequencing provided by a simple Batch OS, the processor is often idle.
- The problem is that I/O Devices are slow compared to the processor.
- In the below example, the computer spends 96% of its time waiting for I/O

Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
Total	31 μs
Percent CPU Utilization = $\frac{1}{31} = 0.032 = 3.2\%$	

Figure 2.4 System Utilization Example

- Devices to finish transferring data to and from the file.
- This situation, where we have a single program, referred to as uniprogramming.
- The processor spends a certain amount of time executing, until it reaches an I/O instruction, it must then wait until that I/O instruction concludes before proceeding.
- This efficiency is not necessary.
- Suppose that there is room for OS and two user programs.
- When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O.
- Furthermore, we might expand memory to hold three, four or more programs and switch among all of them.
- This approach is known as multiprogramming or multitasking.
- It is the central theme of modern OS.

OPERATING SYSTYEMS UNIT-1

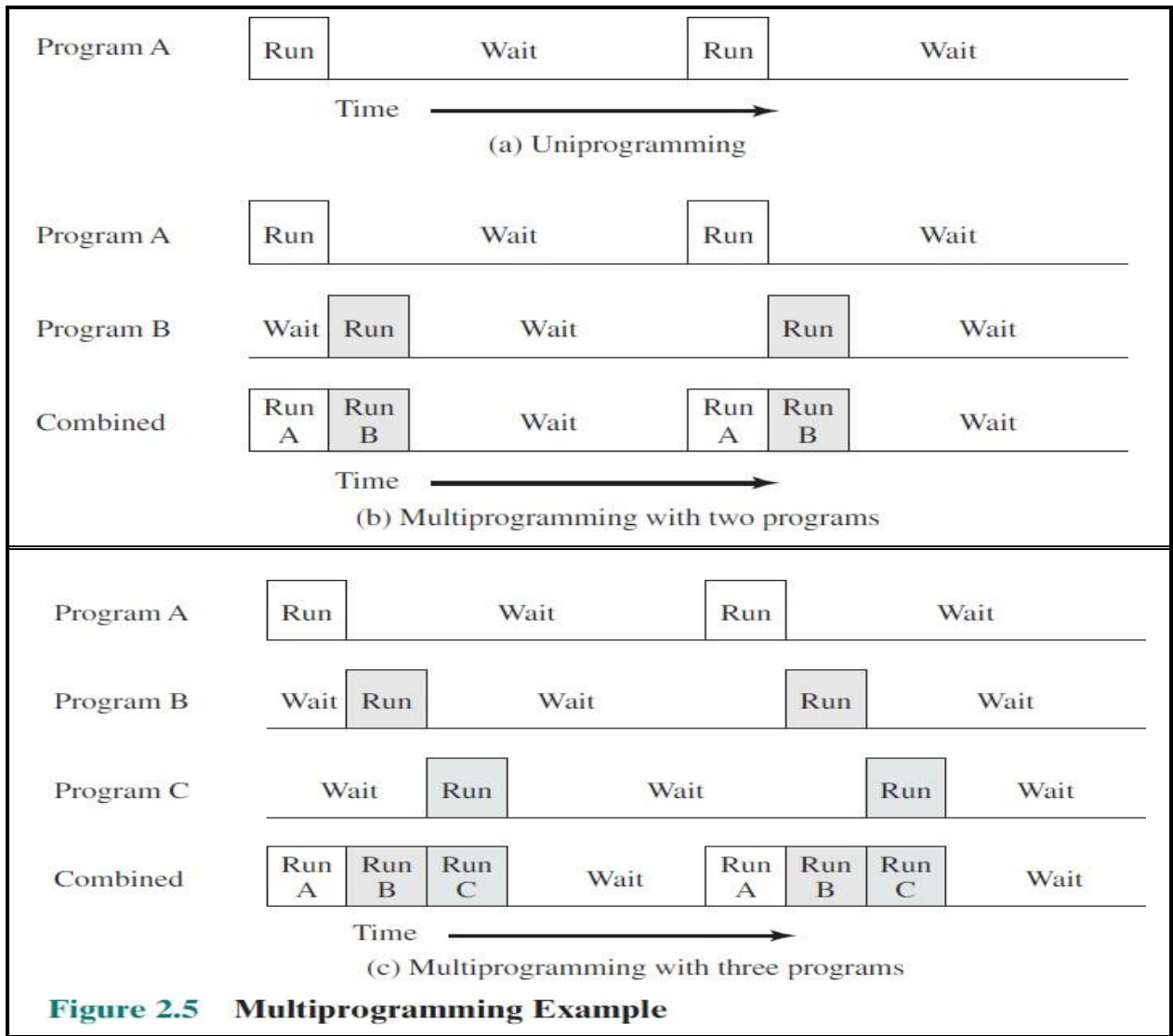


Figure 2.5 Multiprogramming Example

To illustrate the benefit of multiprogramming, we give a simple example. Consider a computer with 250 Mbytes of available memory (not used by the OS), a disk, a terminal, and a printer. Three programs, JOB1, JOB2, and JOB3, are submitted for execution at the same time, with the attributes listed in Table 2.1. We assume minimal processor requirements for JOB2 and JOB3 and continuous disk and printer use by JOB3. For a simple batch environment, these jobs will be executed in sequence. Thus, JOB1 completes in 5 minutes. JOB2 must wait until

OPERATING SYSTEMS UNIT-1

Table 2.1 Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

the 5 minutes are over and then completes 15 minutes after that. JOB3 begins after 20 minutes and completes at 30 minutes from the time it was initially submitted. The average resource utilization, throughput, and response times are shown in the uniprogramming column of Table 2.2. Device-by-device utilization is illustrated in Figure 2.6a. It is evident that there is gross underutilization for all resources when averaged over the required 30-minute time period.

Table 2.2 Effects of Multiprogramming on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

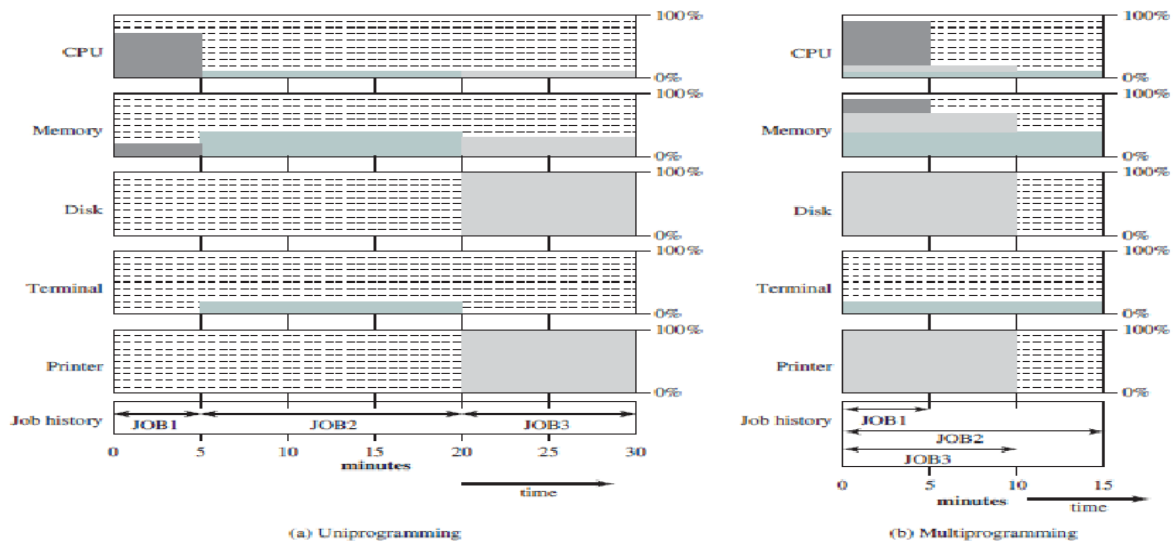


Figure 2.6 Utilization Histograms

Multiprogramming operating systems are fairly sophisticated compared to single-program, or **uniprogramming**, systems. To have several jobs ready to run, they must be kept in main memory, requiring some form of **memory management**. In addition, if several jobs are ready to run, the processor must decide which one to run, this decision requires an algorithm for scheduling. These concepts are discussed later in this chapter.

iii) Time Sharing Systems

- The use of multiprogramming ,Batch processing can be quite efficient.
- However, for many jobs, it is desirable to provide a mode in which the user interacts direct by the computer.
- Indeed, for some jobs , such as transaction processing an interaction mode is essential.
- Multiprogramming allows the processor to handle multiple interactive jobs.
- Later, the technique is referred to as Time Sharing, because processor time is shared among multiple users.
- Time sharing is a logical extension of multiprogramming.
- In a time sharing systems, multiple users simultaneously access the system through terminals.
- With the OS interleaving the execution of each user program in a short burst or quantum of computations.
- In time sharing system, the CPU executes multiple jobs by switching among them but switches occur so frequently, that the users can interact with each program while it is running.
- The user gives instructions to the OS or a program directly using a input device such as keyboard and waits for immediate results on an I/O device.
- Accordingly the response time should be short(less than a second)
- Since each action or command in a time shared system tends to be short only a little CPU time is needed for each user.

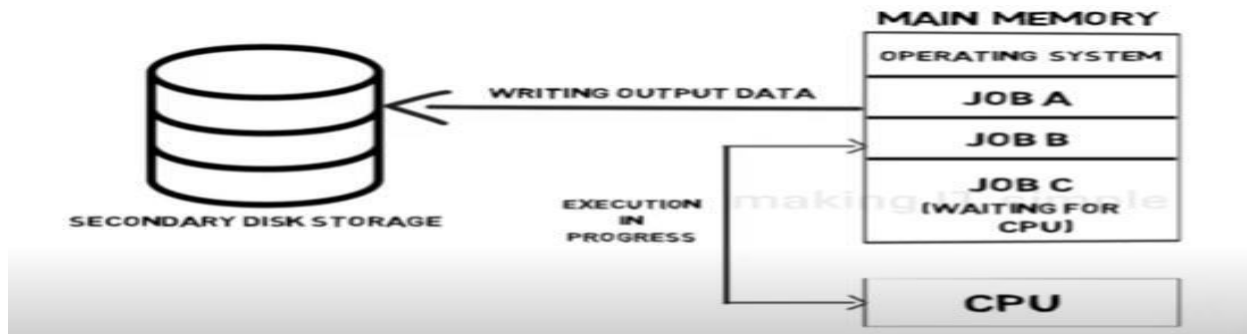
As the system switches rapidly from one user to next, each user is given the impression that the entire computer system is dedicated to his use, even though it is being shared among many users.

- Both batch processing and time sharing are multiprogramming .
- The key differences are listed in Table 2.3

Table 2.3 Batch Multiprogramming versus Time Sharing

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

- The first Time Sharing OS to be developed was the Compatible Time Sharing System (CTSS) developed by MIT by a group known as Project MAC (Multiple Access Computers).
- The system was first developed for IBM 709 in 1961 and later transferred to an IBM 7094.
- A system clock generated interrupts at a rate of approximately one every 0.2 seconds.
- At each clock interrupt, the OS regained control and could assign the processor to another user. This technique is known as Time Slicing.
- Thus, at regular time intervals, the current user would be preempted and another user loaded in.



- To minimize disk traffic, user memory was only written out when the incoming program would rewrite it.
- This principle is illustrated in figure.
- Assume that
 - JOB1: 15,000
 - JOB2: 20,000
 - JOB3: 5000
 - JOB4: 10,000

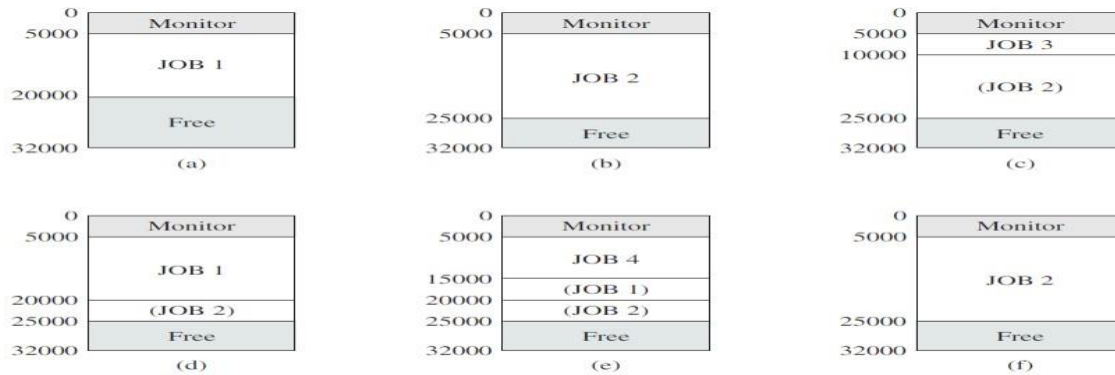


Figure 2.7 CTSS Operation

iv) Personal Computers

- A **personal computer (PC)** is a multi-purpose [computer](#) whose size, capabilities, and price make it feasible for individual use.
- Personal computers are intended to be operated directly by an [end user](#), rather than by a computer expert or [technician](#).



- Most computer users sit in front of a pc, consisting of a monitor, keyboard, mouse and system unit.
- Such system is designed for one user to monopolize its resources.
- The goal is to maximize the work that the user is performing.
- In this case, the OS is designed mostly for ease of use, with some attention paid performance and none paid resource utilization.

v) Parallel Systems

- A system is said to be parallel system(Multiprocessor or Tightly coupled), In which a system have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.
- Parallel systems have three advantages.
 - 1) Increased throughput
 - 2) Economy of scale

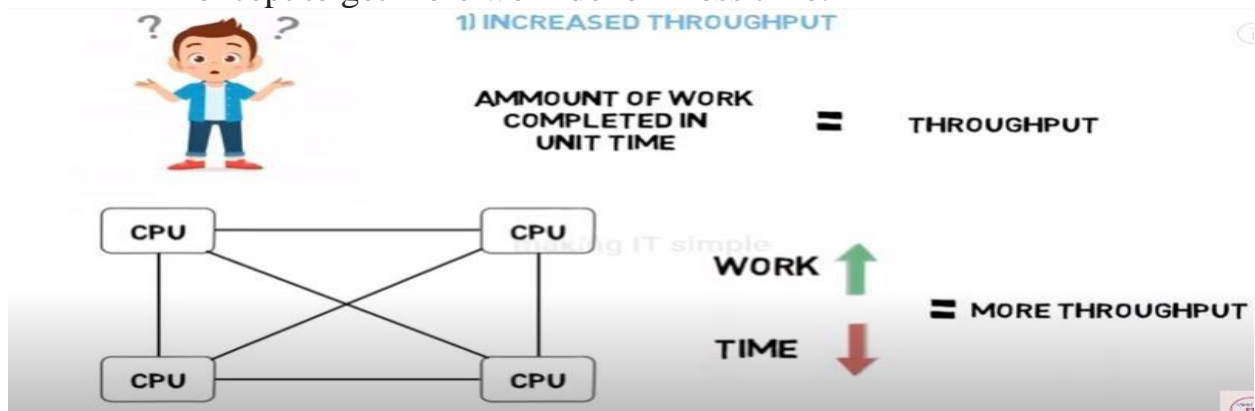
3) Increased Reliability



Multi Processor System



- 1) Increased throughput: By increasing the number of processors, we expect to get more work done in less time.

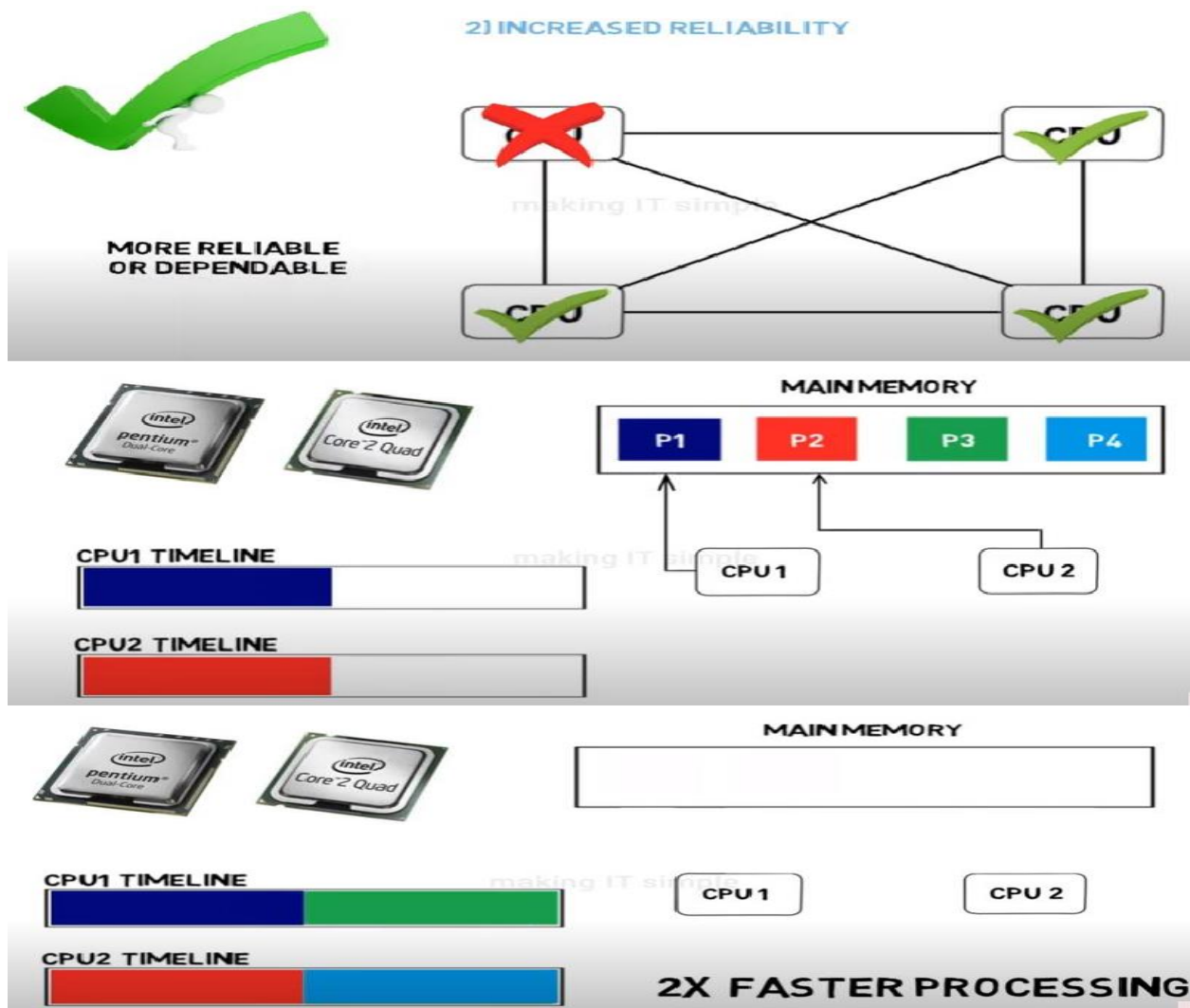


2) ECONOMY OF SCALE:

Multiprocessor systems can cost less than equivalent multiple single processor systems.

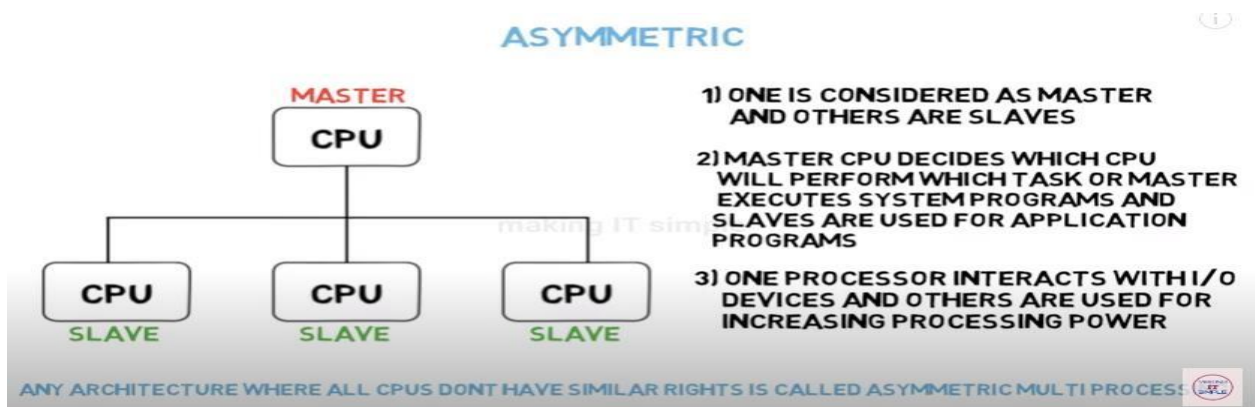
OPERATING SYSTEMS UNIT-1

- 3) **INCREASED RELIABILITY:** If function can be distributed properly among several processors, then the failure of one processor will not halt the system only slow it down.

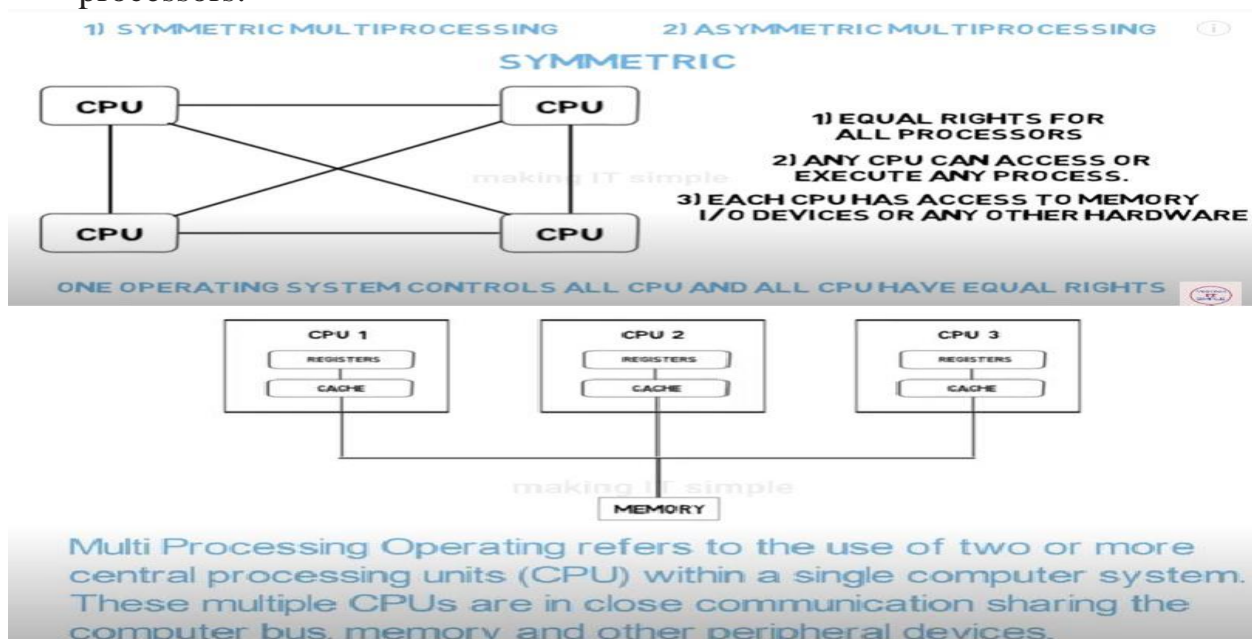


Parallel systems are two types.

- 1) Asymmetric Multiprocessing Systems
 - 2) Symmetric Multiprocessing Systems
- Asymmetric Multiprocessing Systems: In which each processor is assigned a specific task. A master processor controls the system, the other processors either look to the master for instructions or predefined tasks.
 - This scheme defines a master slave relationship.
 - The master processor schedules and allocates work to the slave processors.



- The most common systems use symmetric multiprocessing (SMP),
- **In** which each processor performs all tasks within the operating system.
- SMP means that all processors are peers; no boss–worker relationship exists between processors.



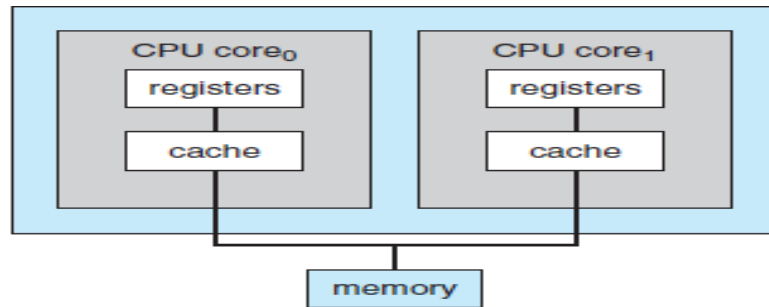


Figure 1.7 A dual-core design with two cores placed on the same chip.

- In Figure 1.7, we show a dual-core design with two cores on the same chip.
- In this design, each core has its own register set as well as its own local cache.

vi) Distributed Systems

- Distributed system is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide users with access to the various resources that the system maintains.
- Access to a shared resource increases computation speed, functionality, data availability, and reliability.

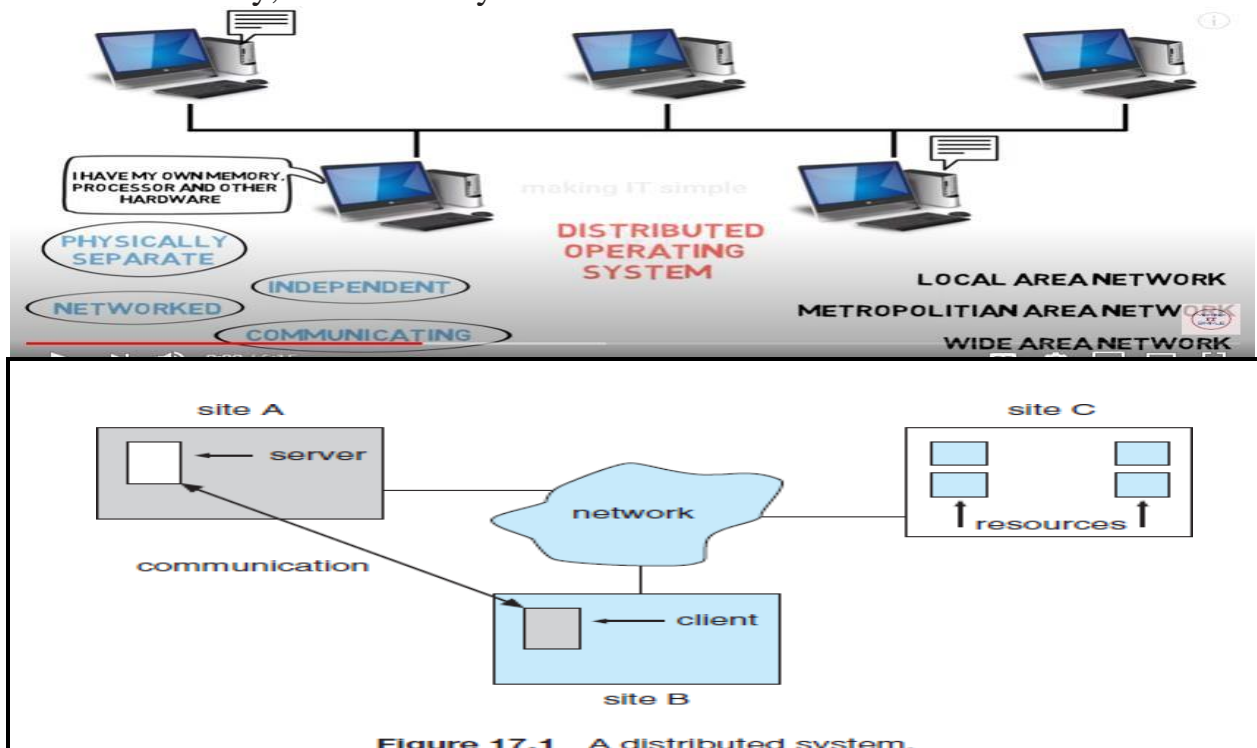
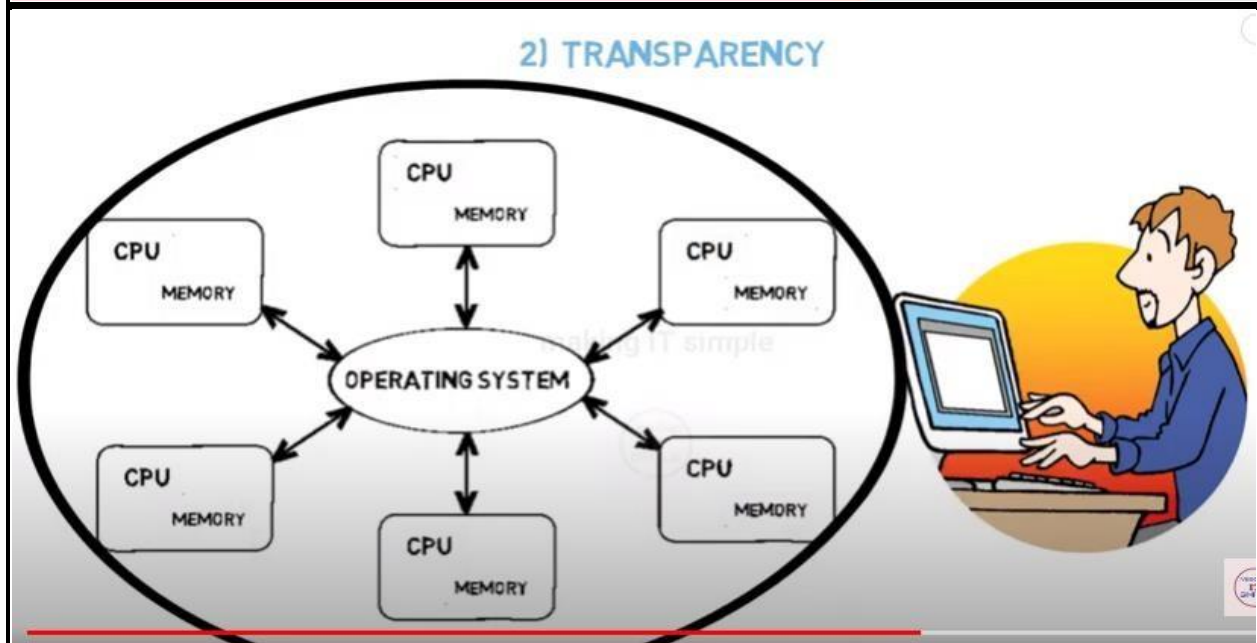
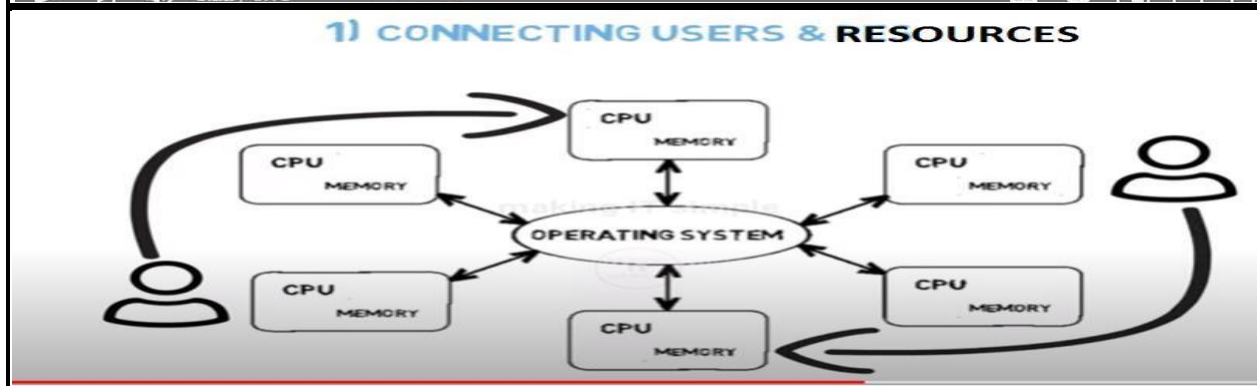
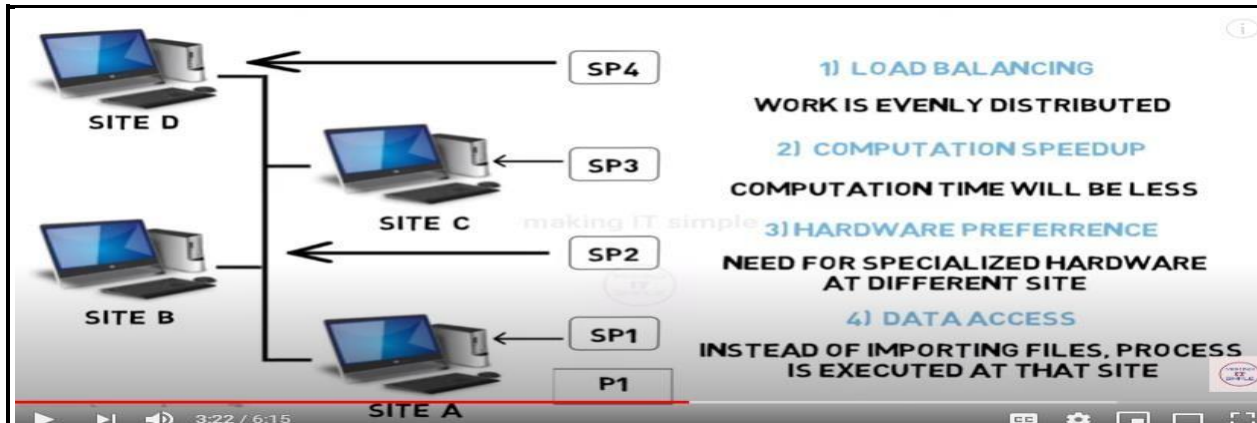


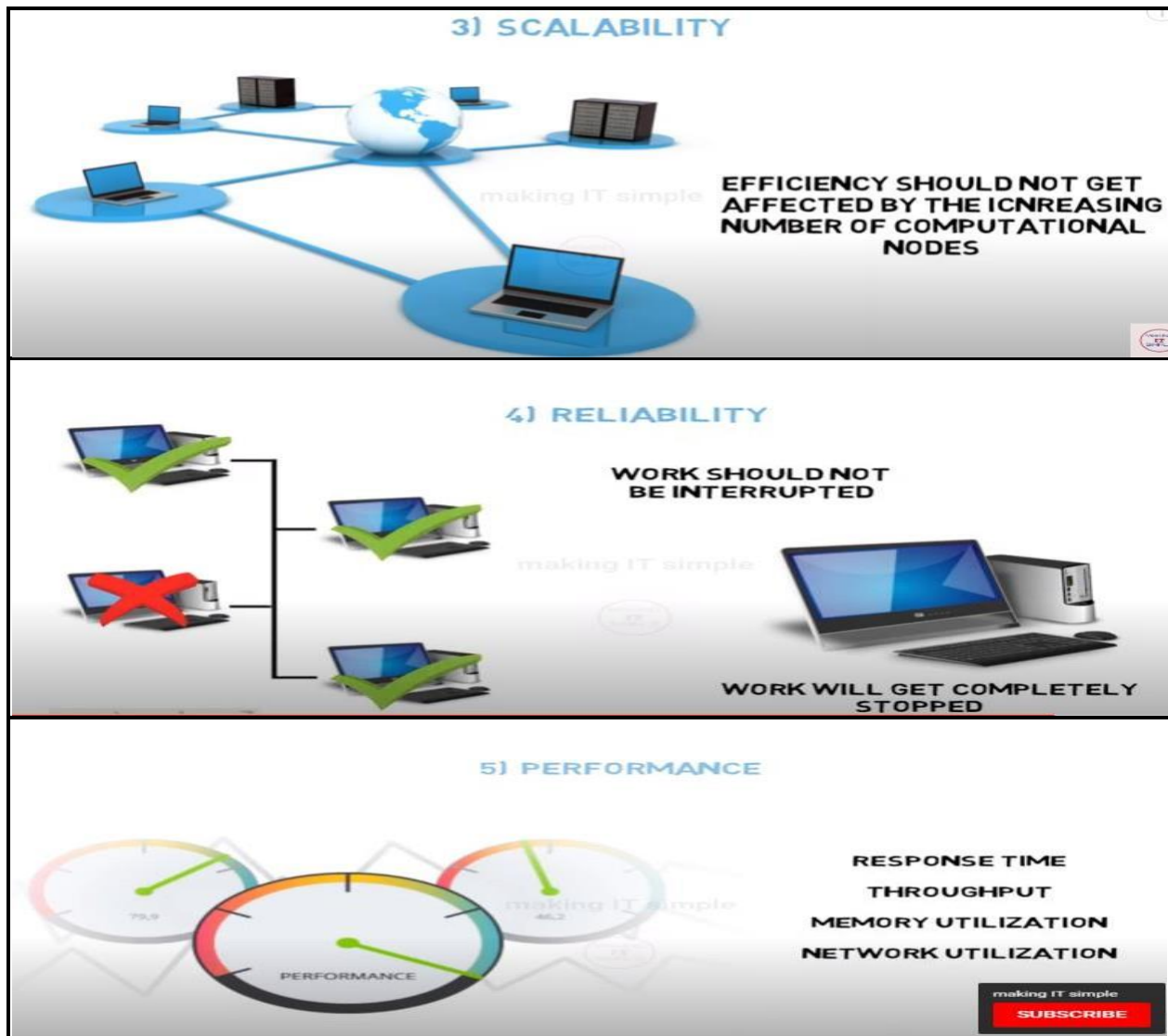
Figure 17.1 A distributed system.

- For example FTP and NFS. The protocols that create a distributed system can greatly affect that system's utility and popularity.
- A **network**, in the simplest terms, is a **communication path between** two or more systems. Distributed systems depend on networking for their functionality.
- Networks are characterized based on the distances between their nodes.

OPERATING SYSTEMS UNIT-1

- A Local-area Network (LAN) connects computers within a room, a building, or a campus.
- A Metropolitan-area Network (MAN) could link buildings within a city.
- A Wide-area Network (WAN) usually links buildings, cities, or countries.
- A Global company may have a WAN to connect its offices worldwide.



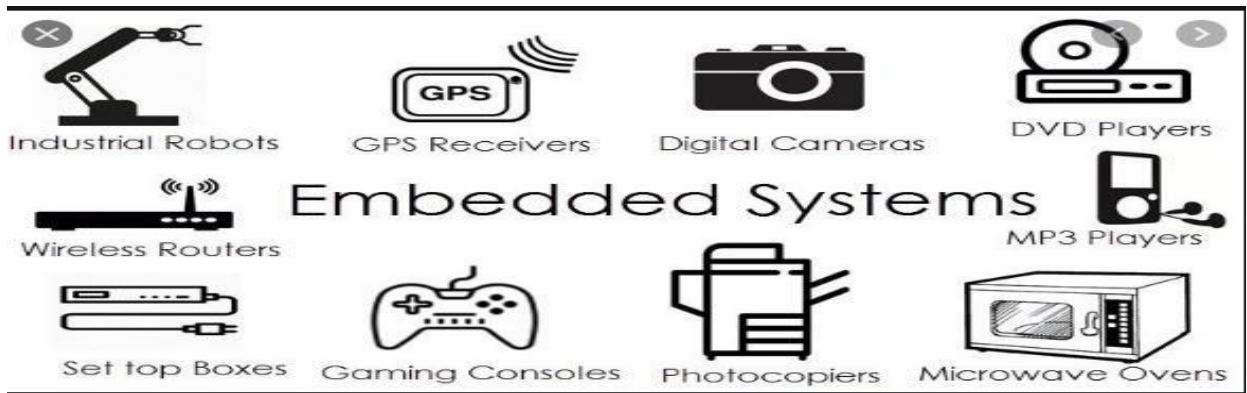


vii) Real Time Systems

- So far we focused mainly general purpose computing systems.
- A Real Time System is a computer system that requires not only that computed results be correct but also that the results be produced within a specified dead line period.
- Real Time systems must produce results within certain time constraints.
- Some Real Time systems are identified as safety critical systems.
- In a safety critical system, incorrect operation – usually due to a missed deadline – results in some sort of “Damage”.
- For example
 - Weapons systems
 - Antilock Brake systems

OPERATING SYSTEMS UNIT-1

- Flight Management systems
- Health related embedded systems
- In these scenarios the Real Time system must respond to events by the specified deadlines, otherwise serious injury might occur.
- Real Time systems executing on traditional computer hardware are used in a wide range applications. Many Real Time systems are embedded in 'specialized devices' such as Home appliances, Consumer devices and Communication devices.



- Real Time systems is of two types
 - i) Hard Real Time Systems
 - ii) Soft Real Time Systems
- Hard Real Time Systems has the most stringent requirements, guaranteeing that critical real time tasks be completed within their deadlines.
- Soft Real Time Systems: A system is less restrictive, simply providing that a critical real time task.



Real time characteristics:

1. Single Purpose
2. Small Size
3. Inexpensively Mass Produced
4. Specific Timing Requirements

- Delivering music on MP3 Player.
- Consider the amount of space available in a Wrist watches or a Microwave Oven.
- They are found in home appliances and consumer devices(Digital Cameras, Microwave Ovens)
- The primary task of real time system is to support the timing requirements of real time tasks.

SPECIAL PURPOSE SYSTEMS.

1. REAL TIME EMBEDDED SYSTEMS
2. MULTIMEDIA SYSTEMS
3. HANDHELD SYSTEMS

1. REAL TIME EMBEDDED SYSTEMS:

These systems are found everywhere, from car engines and manufacturing robots, to DVD's and microwave ovens. They tend to have very specific tasks. The os provide limited features on these systems. They have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotics arms.

The power of these devices both standalone units and as elements of networks and the web is sure to increase as well.

Embedded systems always run real time operating systems.

A real time system is used when time requirements have been placed on the operation of a processor or the flow of data.



2. Multimedia systems:

Most os's are designed to handle conventional data such as text files, programs , word processing documents and spread sheets. However recent technology is the

incorporation of multimedia data into computer systems. Multimedia data consists of audio and video files as well as conventional files. Multimedia describes a wide range of applications in popular use today. These include audio files such as MP3, DVD MOVIES, VIDEO CONFERENCING, SHORT VIDEO CLIPS OF MOVIE PREVIEWS, NEWS STORIES downloaded over the internet.

Multimedia applications may also include live webcasts (broad casting over the WWW) of speeches, sport events etc.

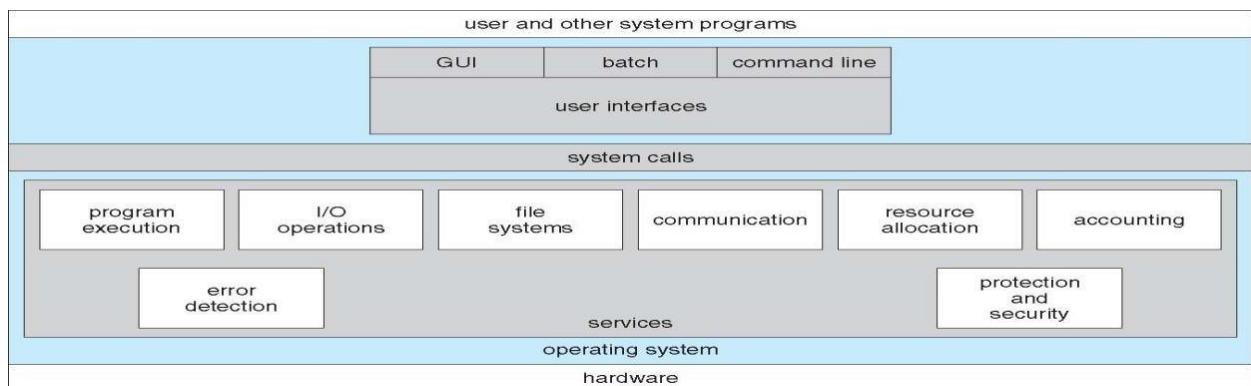
3. Handheld systems.

Handheld systems include PDA's (Personal Digital Assistants) such as Palm and Pocket PC's and Cellular mobile phones. Many of which use special purpose embedded operating systems. Developers of handheld systems and applications face many challenges, most of which are due to the limited size. Most handheld devices have small amounts of memory, slow processors and small display screens. For physical memory somewhere between 1 MB to 1 GB.

We explore virtual memory which allows developers to write programs that behave as if the system has more memory than physically available. A second issue is the speed of the processor used in the devices.

Faster processors requires more power, it requires larger battery and would have to be replaced (or recharged) frequently. A last issue is lack of physical space limits such as small keyboards, small screen based keyboards and small display screen. Some handheld devices use wireless technology such as Bluetooth.

OPERATING SYSTEM SERVICES:



OS Provides services or functions that are helpful to the user.

1. **User Interface:** All OS have a user interface. There are 3 types of user interfaces.
 - a. **CLI (command line interface)** it uses text commands and a method for entering them.
 - b. **Batch interface:** In this, commands and directives to control those commands are entered into files and those files are executed.
 - c. **GUI(Graphical User Interface)** it is window system with pointing device to direct I/O,choose from menus and makes selections and a keyboard to enter text. Some systems provide two or all three variations.
2. **Program Execution:** The system must be able to load a program into memory and to run that program. The program must end with its execution either normally or abnormally.
3. **I/O operations:** a running program may require I/O which may involve a file or an I/O Device.
4. **File System Manipulation:** The programs need to read and write the files and directives. They also need to create, delete and search for a given file.
5. **Communication:** There are many circumstances in which one process reads to exchange info with another process. Communications may be implemented via shared memory and message passing in which packets of information are moved between processes by the os.
6. **Error Detection:** the OS needs to be aware of possible errors. Errors may occur in the CPU, memory H/W, I/O Devices (a connection failure on network, lack of paper in printer) and in the user program(Arithmetic overflow, Illegal memory access). For each type of error the os must take the appropriate action to ensure correct and consistent computing.
7. **Resource Allocation:** When there are multiple users or multiple jobs running at the same time resources must be allocated to each of them. For instance in determining how best to use the cpu.
8. **Accounting:** we want to keep track of which users use how much and what kinds

of computer resources. This record may be used for accounting (so that users can be billed) or to reconfigure the system to improve computing services.

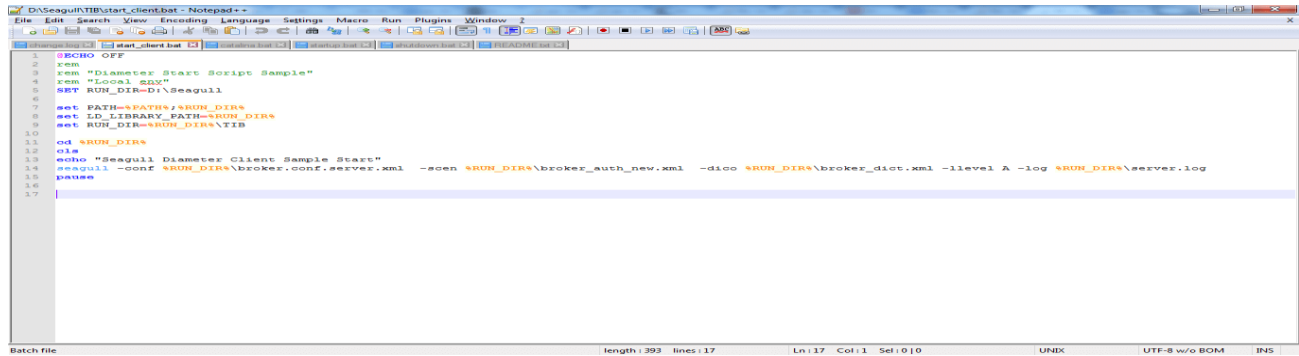
9. Protection and Security: protection involves ensuring that all access to system resources is controlled. Security involves requiring each user to authenticate himself to the system usually by means of a password to gain access to system resources.

```

File Edit View Terminal Tabs Help
fd0      0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0    0
sd0      0.0    0.2    0.0    0.2    0.0    0.0    0.0    0.4    0    0
sd1      0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0    0
extended device statistics
device   r/s    w/s    kr/s    kw/s    wait    actv    svc_t    %w    %b
fd0      0.0    0.0    0.0    0.0    0.0    0.0    0.0    0    0
sd0      0.6    0.0    38.4    0.0    0.0    0.0    8.2    0    0
sd1      0.0    0.0    0.0    0.0    0.0    0.0    0.0    0    0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
- (/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
- (/var/tmp/system-contents/scripts)# uptime
12:53am up 9 min(s), 3 users, load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
- (/var/tmp/system-contents/scripts)# w
4:07pm up 17 day(s), 15:24, 3 users, load average: 0.09, 0.11, 8.66
User      tty      login@ idle      JCPU      PCPU      what
root      console  15Jun07 18days 1          /usr/bin/ssh-agent -- /usr/bi
n/d
root      pts/3    15Jun07 18      4 w
root      pts/4    15Jun07 18days w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
- (/var/tmp/system-contents/scripts)#
  
```



Batch interfaces are non-interactive user **interfaces**, where the user specifies all the details of the **batch** job in advance to **batch** processing, and receives the output when all the processing is done. The computer does not prompt for further input after the processing has started.

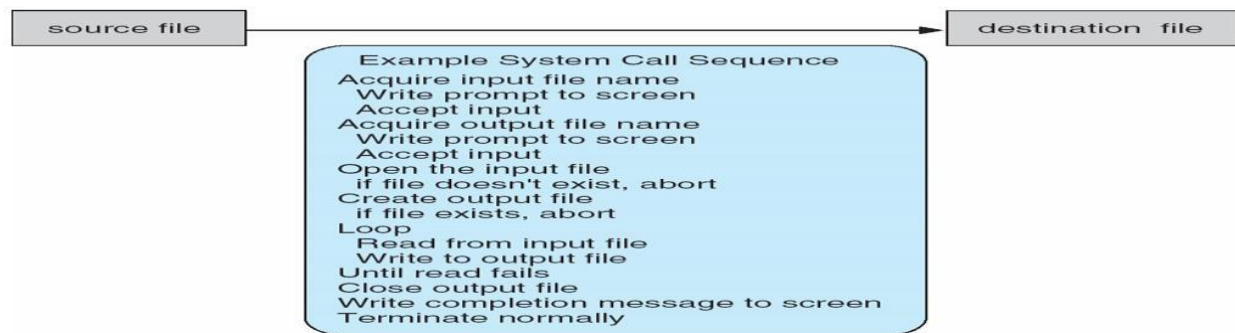


```
1 @ECHO OFF
2 rem
3 rem "Diameter Start Script Sample"
4 rem "Local ENV"
5 SET RUN_DIR=D:\Seagull
6
7 set PATH=%PATH%;%RUN_DIR%
8 set LD_LIBRARY_PATH=%RUN_DIR%
9 set RUN_DIR=%RUN_DIR%\TIB
10
11 cd %RUN_DIR%
12
13
14 echo "Seagull Diameter Client Sample Start"
15 seagull -Conf %RUN_DIR%\broker.conf.server.xml -scen %RUN_DIR%\broker_auth_new.xml -dico %RUN_DIR%\broker_dict.xml -llevel A -log %RUN_DIR%\server.log
16 pause
17
```

Systems calls

- System calls provide an interface to the services made available by an OS.
- These system calls are generally available as routines written in C and C++, although certain low level tasks (hardware must be accessed directly) may be written using Assembly language instructions.
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call used Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM).
- Illustrate how system calls are used, Writing a simple program to read data from file and copy them to another file.
- The first input that the program will need the names of the two files. i.e., input file and output file.
- These names can be specified in many ways, There is one approach for the program to ask the user for the names of two files.
- In another approach (interactive system) will require a sequence of system calls, first to write a prompting a message on the screen and then to read from the keyboard the characters that define the two files.
- On mouse based and icon based systems, the user can use the mouse to select the source name and a window can be opened from the destination name to be specified. This sequence requires many I/O system calls.

Example of how System Calls are executed



1. SIMPLE STRUCTURE

2. LAYERED APPROACH

3. MICRO KERNEL APPROACH

4. MODULE APPROACH

Once the file names obtained, the program must open the input file and create the output file. Each of these operations requires another system call.

There are possible error conditions for each operation. We can see even simple programs may make heavy use of OS. Frequently system executes thousands of System calls per sec. This System call sequence is shown in above figure. Most programmers never see this low level details. However, Application developers design programs according to an API.

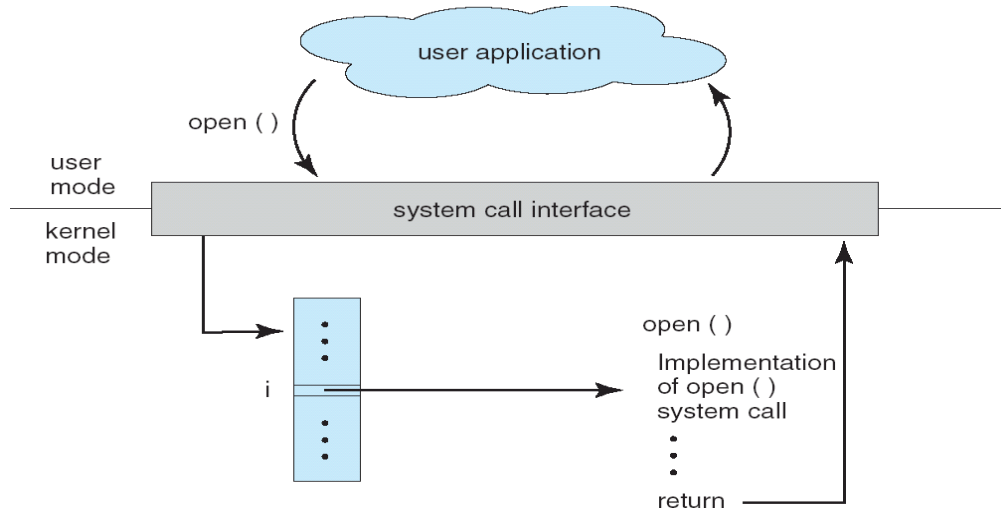
The API specifies a set of functions, that are available to an application programmer, including the parameters that are passed to each function and return values the programmer can expect.

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 intended system call in OS kernel and returns status of the system call and any return values.
- The caller need know nothing about how the system call is implemented.
- Just needs to obey API and understand what OS will do as a result call.

OPERATING SYSTEMS UNIT-1

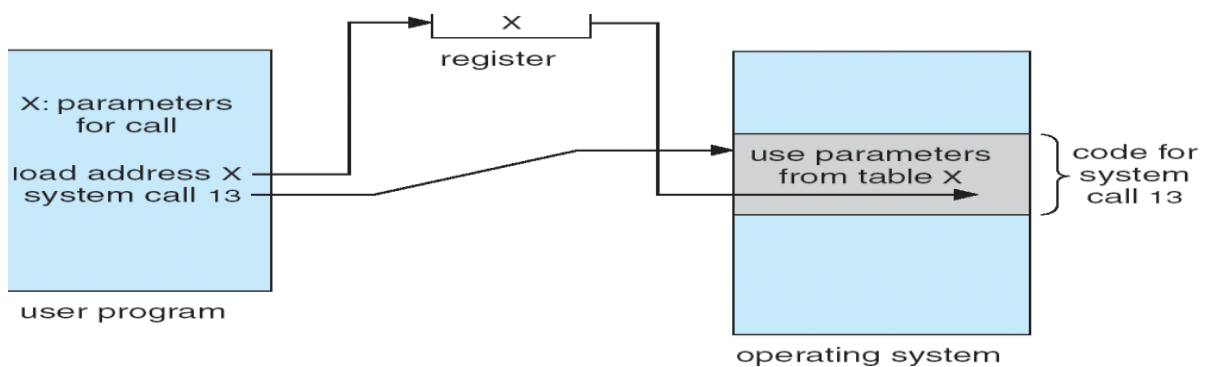
- Most details of OS interface hidden from programmer by API.
- Managed by run-time support library (set of functions built into libraries included with compiler)



There are three general methods are used to pass the parameters to the os.

1. Registers
2. Memory
3. Stack

Parameter Passing via Table



- Simplest: pass the parameters in *registers*
 - In some cases, may be more parameters than registers
- 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.
- Parameters placed, or *pushed*, onto the *stack* by the program and *popped* off the stack by the operating system.
- Block and stack methods do not limit the number or length of parameters being passed

Types of System Calls

System calls can be grouped into six major categories:

1. Process control
2. File management
3. Device management
4. Information maintenance
5. Communications
6. Protection

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

File management

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

Device management

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

Information maintenance

- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

Communications

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

OPERATING SYSTEMS UNIT-1

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() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

1. Process control

A running program needs to be able to halt its execution either normally (end) or abnormally (abort).

If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated. The dump is written to disk and may be examined by a debugger. System program designed to aid the programmer in finding and correcting bugs.

A process or job executing one program may want to load() and execute() another program. We create a new job or process, or perhaps even a set of jobs or processes, we should be able to control its execution.

Created new jobs or processes, we may need to wait for them to finish their execution. We may want to wait for a certain amount of time to pass (wait time). We will want to wait for a specific event to occur (wait event). The jobs or processes should then signal when that event has occurred (signal event).

Two or more processes may share data. To ensure the integrity of the data being shared, operating systems often provide system calls allowing a process *to* lock shared data, thus preventing another process from accessing the data while it is locked. Typically such system calls include acquire lock and release lock. System calls of these types, dealing with the coordination of concurrent processes

2. File management

We first need to be able to create and delete files. Either system call requires the name of the file and perhaps some of the file's attributes. Once the file is created, we need to open it and to use it. We may also read, write, or reposition (rewinding or skipping to the end of the file, for example). Finally, we need to close the file, indicating that we are no longer using it. File attributes include the file name, file type, protection codes, accounting information, and so on. At least two system calls, get file attribute and set file attribute.

3. Device management

A process may need several resources to execute-main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available. A system with multiple users may require us to first request the device, to ensure exclusive use of it. After we are finished with the device, we release it.

Once the device has been requested (and allocated to us), we can read, write, and (possibly) reposition the device, just as we can with files.

4. Information Maintenance

Many system calls exist simply for the purpose of transferring information between the user program and the operating system. For example, most systems have a system call to return the current time and date. Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on.

Another set of system calls is helpful in debugging a program. Many systems provide system calls to dump memory. This provision is useful for debugging. Many operating systems provide a time profile of a program to indicate the amount of time that the program executes at a particular location or set of locations. In addition, the operating system keeps information about all its processes, and system calls are used to access this information. Generally, calls are also used to reset the process information (get process attributes and set process attributes).

5. Communication:

There are two common models of interprocess communication ie., Message passing model and shared memory model.

In the **message-passing model**, the communicating processes exchange messages with one another to transfer information. Messages can be exchanged between the processes either directly or indirectly through a common mailbox. Before communication can take place, a connection must be opened. The name of the other communicator must be known, be it another process on the same system or a process on another computer connected by a communications network.

The source of the communication, known as the **client**, and the receiving daemon, known as a **server**, then exchange messages by using read message() and write message() system calls. The close connection() call terminates the communication.

In the **shared-memory model**, processes use shared memory create() and shared memory attach() system calls to create and gain access to regions of memory owned by other processes.

6. Protection

Protection provides a mechanism for controlling access to the resources provided by a computer system. Typically, system calls providing protection include set permission() and get permission() which manipulate the permission settings of resources such as files and disks. The allow user() and deny user() system calls specify whether particular users can—or cannot—be allowed access to certain resources.

Operating-System Structure

A system as large and complex as a modern operating system must be engineered carefully if it is to function properly and be modified easily. A common approach is to partition the task into small components, or modules, rather than have one **monolithic** system.

1. SIMPLE STRUCTURE.

Many operating systems do not have well-defined structures. Frequently, such Simple structure system started as small, simple and limited. MS-DOS is an example of simple structure. It was originally designed and implemented by few

OPERATING SYSTEMS UNIT-1

people. And it was written to provide most functionality in least space. In MS-DOS, the interfaces and levels of functionality are not separated.

Ex. Application programs are able to access the basic I/O routines to write directly to the display and disk drives.

The disadvantage of this approach is freedom leaves to errant programs causing entire system crashes when user program fails.

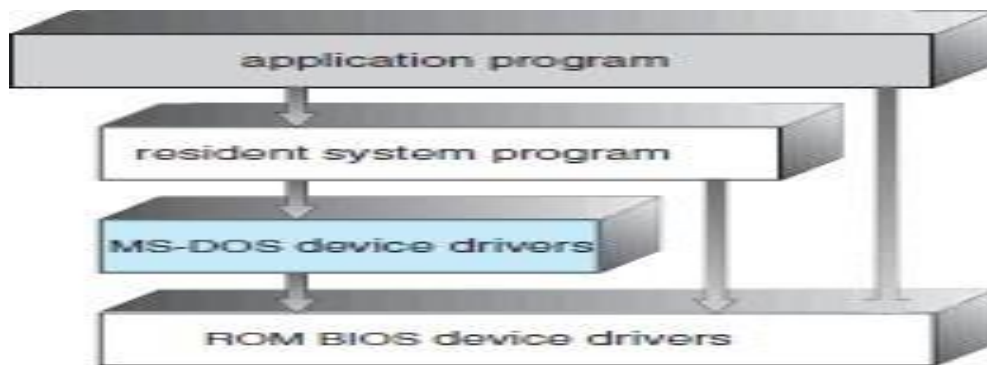


Figure 2.11 MS-DOS layer structure.

Another example of limited structure is the original UNIX OS . It consists of 2 parts ie.. kernel and system programs Everything below the system call interface and above the physical hardware is kernel. The kernel provides the file system, CPU scheduling, Memory management and other functions.

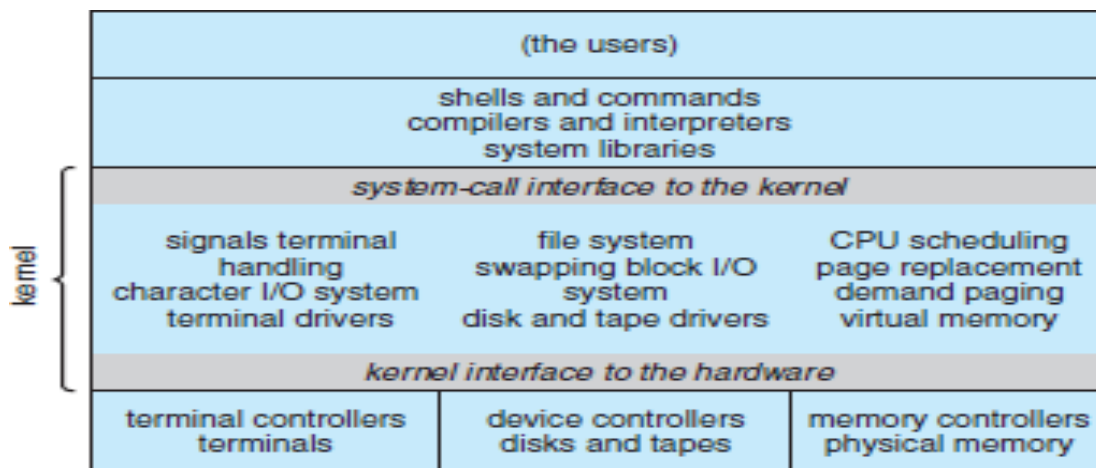


Figure 2.12 Traditional UNIX system structure.

2. Layered Approach.

In this method OS is broken into number of layers or levels. The bottom layer(Layer 0) is the hardware. The highest layer(Layer N) is the user interface.

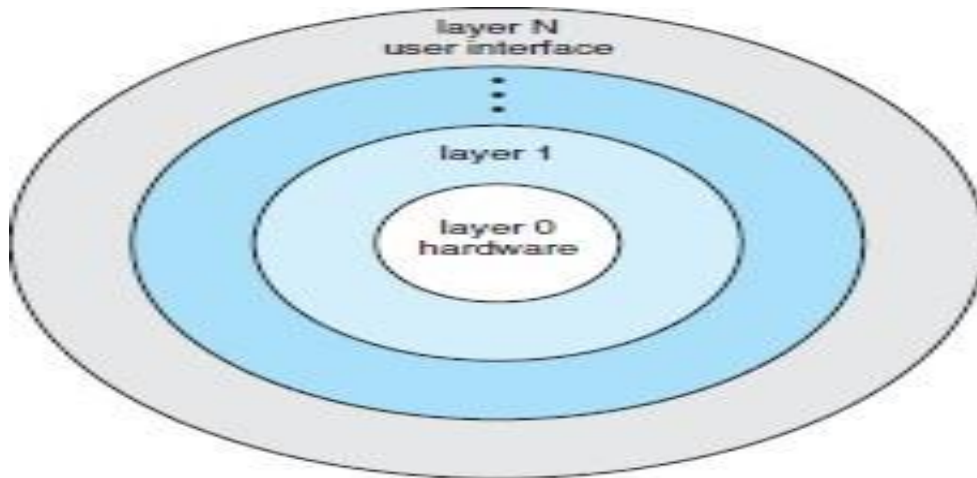


Figure 2.13 A layered operating system.

An OS layer is an implementation of an abstract object made up of data and operations that can manipulate those data. Suppose layer M consists of data structures and a set of routines, that can be invoked by higher level layer. Layer M in turn can invoke operations on lower level layers.

The main advantage of the layered approach is simplicity of construction and debugging.

The layers are selected so that each uses functions(operations) and services of only lower level layers. This approach simplifies debugging and system verification. This approach simplifies debugging and system verification.

The first layer can be debugged without any concern for the rest of the system, because, by definition, it uses only the basic hardware (which is assumed correct) to implement its functions.

Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on. If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged.

Thus, the design and implementation of the system are simplified. The major difficulty involves appropriately defining the various layers.

3. MICROKERNEL APPROACH

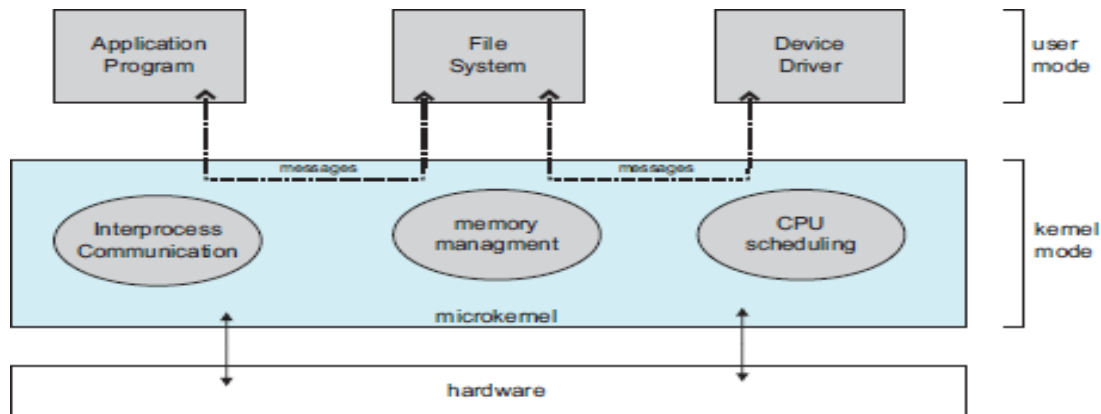


Figure 2.14 Architecture of a typical microkernel.

The UNIX OS has been expanded, the kernel become large and difficult to manages.

In the mid 1980's at Carnegie Mellon University developed an OS called Mach ,that modularized the kernel using the microkernel approach.

In this approach removing all non essential components from the kernel and implementing them as a system and user level programs.

The main function of microkernel is to provide facility between the client programs and the various services that are running in user space.

For example, if the client program wishes to access a file, it must interact with the file server. The client program and service never interact directly. Rather, they communicate indirectly by exchanging messages with microkernel.

One benefit of this approach is easy of extending the OS.All new services are added to user space and it does not require modification of the kernel.

Ex: Tru64 UNIX, Mac OS X kernel (also knows as Darwin).

The micro kernel provides more security and reliability. Since most services are running as user rather than kernel- processes. If service fails, the rest of the OS

remains untouched. Unfortunately, microkernel can suffer from performance decreases due to increased system function overhead. Ex: Windows NT

4. Module approach

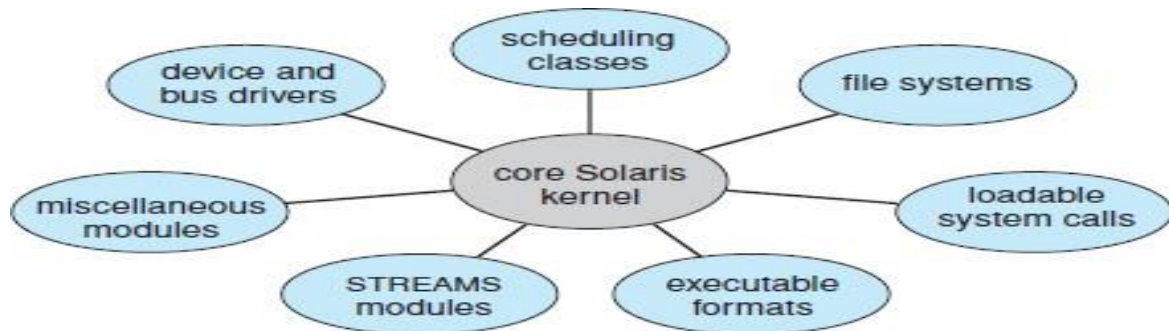


Figure 2.15 Solaris loadable modules.

The best current methodology for OS design involves using OOP techniques to create a modular kernel. Here the kernel has set of core components and links in additional services either during boot time or runtime. Example: Solaris, Linux and Mac OS X.

Solaris loadable modules

1. Scheduling classes
2. File systems
3. Loadable system calls
4. Executable formats
5. STREAMS modules
6. Miscellaneous
7. Device and bus drivers

Such a design allows the kernel to provide core services and also certain features to be implemented dynamically. For example device and bus drivers for specific H/W can be added to the kernel and support different file systems can be added as loadable modules. It's more flexible than layered system in that any module can call any another module.

In the Micro kernel approach in that primary module has only core functions and knowledge of how to load and communicate with other modules but it is more efficient, because modules do not need to invoke message passing in order to communicate.

Ex: Solaris, Linux, Mac os x