Mahatma Gandhi Mission's College of Engineering and Technology
Kamothe, Navi Mumbai

MAHATMA GANDHI MISSION

Assignment -1

Subject-OS Div- A&B Sem-IV Class –SE

Date of Issue: 14/2/2022 Date of Submission: 25/2/2022

Q.No	Question
Q1.Fill in the blanks	
1)	Authentication program runs first after booting the computer and loading the GUI.
2)	To access the services of the operating system, the interface is provided by the system calls.
3)	Command interpreter is also called shell.
4)	Network operating system runs on server.
5)	The bootstrap program initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory, and then starts the operating system
Q2. Choose Correct Options	
1)	1. What is bootstrapping called? a. Cold boot b. Cold hot boot c. Cold hot strap d. Hot boot
2)	2. Which is the Linux operating system? a. Private operating system b. Windows operating system c. Open-source operating system d. None of these
3)	3. What is the mean of the Booting in the operating system? a. Restarting computer b. Install the program c. To scan d. To turn off
4)	4. Which of the following is not an operating system? a. Windows b. Linux c. Oracle d. DOS
5)	5. When was the first operating system developed?

	a. 1948 b. 1949 c. 1950 d. 1951
6)	6. Which of the following is a single-user operating system? a. Windows b. MAC c. Ms-Dos d. None of these
7)	7. Which of the following operating systems does not support more than one program at a time? a. Linux b. Windows c. MAC d. DOS
8)	8. Who provides the interface to access the services of the operating system? a. API b. System call c. Library d. Assembly instruction
9)	9. Which of the following is an example of a Real Time Operating System? a. MAC b. MS-DOS c. Windows 10 d. Process Control
10)	10. Which of the following operating systems do you use for a client-server network? a. MAC b. Linux c. Windows XP d. Windows 2000
Q3. Answer the following questions in brief	

1)	Explain various operating system structures.
Ans:	## Single-tasking and multi-tasking A single-tasking system can only run one program at a time, while a multi-tasking operating system allows more than one program to be running in concurrency. This is achieved by time-sharing, where the available processor time is divided between multiple processes. These processes are each interrupted repeatedly in time slices by a task-scheduling subsystem of the operating system. Multi-tasking may be characterized in preemptive and co-operative types. In preemptive multitasking, the operating system slices the CPU time and dedicates a slot to each of the programs. Unix-like operating systems, such as Linux—as well as non-Unix-like, such as AmigaOS—support preemptive multitasking. Cooperative multitasking is achieved by relying on each process to provide time to the other processes in a defined manner. 16-bit versions of Microsoft Windows used cooperative multi-tasking; 32-bit versions of both Windows NT and Win9x used preemptive multi-tasking. ## Single- and multi-user Single-user operating systems have no facilities to distinguish users, but may allow multiple programs to run in tandem.[8] A multi-user operating system extends the basic concept of multi-tasking with facilities that identify processes and resources, such as disk space, belonging to multiple users, and the system permits multiple users to interact with the system at the same time. Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources to multiple users. ## Distributed A distributed operating system manages a group of distinct, networked computers and makes them appear to be a single computer, as all computations are distributed (divided amongst the constituent computers).[9] ## Templated In the distributed and cloud computing context of an OS, *templating* refers to creating a single virtual machine image as a guest operating system, then saving it as a tool for multiple running virtual machines. The technique is used both in virtualization and cloud computing management, and is common in large server warehouses.[10] ## Embedded Embedded operating systems are designed to be used in embedded computer systems. They are designed to operate on small machines with less autonomy (e.g. PDAs). They are very compact and extremely efficient by design, and are able to operate with a limited amount of resources. Windows CE and Minix 3 are some examples of embedded operating systems. ## Real-time A real-time operating system is an operating system that guarantees to process events or data by a specific moment in time. A real-time operating system may be single- or multi-tasking, but when multitasking, it uses specialized scheduling algorithms so that a deterministic nature of behavior is achieved. Such an event-driven system switches between tasks based on their priorities or external events, whereas time-sharing operating systems switch tasks based on clock interrupts. ## Library A library operating system is one in which the services that a typical operating system provides, such as networking, are provided in the form of libraries and composed with the application and configuration code to construct a unikernel: a specialized, single address space, machine image that can be deployed to cloud or embedded environments.
2)	Explain Evolution of operating system.
Ans:	Serial Processing

With the earliest computers, from the late 1940s to the mid-1950s, the programmer interacted directly with the computer hardware; there was no OS. These computers were run from a console consisting of display lights, toggle switches, some form of input device, and a printer. Programs in machine code were loaded via the input device (e.g., a card reader). If an error halted the program, the error condition was indicated by the lights. If the program proceeded to a normal completion, the output appeared on the printer.

These early systems presented two main problems:

• Scheduling: Most installations used a hardcopy sign-up sheet to reserve computer time. Typically, a user could sign up for a block of time in multiples of a half hour or so. A user might sign up for an hour and finish in 45 minutes; this would result in wasted computer processing time. On the other hand, the user might run into problems, not finish in the allotted time, and be forced to stop before resolving the problem.

• Setup time: A single program, called a job , could involve loading the compiler plus the high-level language program (source program) into memory, saving the compiled program (object program) and then loading and linking together the object program and common functions. Each of these steps could involve mounting or dismounting tapes or setting up card decks. If an error occurred, the hapless user typically had to go back to the beginning of the setup sequence. Thus, a considerable amount of time was spent just in setting up the program to run. This mode of operation could be termed serial processing , reflecting the fact that users have access to the computer in series. Over time, various system software tools were developed to attempt to make serial processing more efficient. These include libraries of common functions, linkers, loaders, debuggers, and I/O driver routines that were available as common software for all users.

Simple Batch Systems

Early computers were very expensive, and therefore it was important to maximize processor utilization. The wasted time due to scheduling and setup time was unacceptable. To improve utilization, the concept of a batch OS was developed. It appears that the first batch OS (and the first OS of any kind) was developed in the mid-1950s by General Motors for use on an IBM 701 [WEIZ81]. The concept was subsequently refined and implemented on the IBM 704 by a number of IBM customers. By the early 1960s, a number of vendors had developed batch operating systems for their computer systems. IBSYS, the IBM OS for the 7090/7094 computers, is particularly notable because of its widespread influence on other systems. The central idea behind the simple batch-processing scheme is the use of a piece of software known as the 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 and places the entire batch on an input device, for use by the monitor. Each program is constructed to branch back to the monitor when it completes processing, at which point the monitor automatically begins loading the next program. To understand how this scheme works, let us look at it from two points of view: that of the monitor and that of the processor.

• Monitor point of view: The monitor controls the sequence of events. For this to be so, much of the monitor must always be in main memory and available for execution (Figure 2.3). That portion is referred to as the 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. The monitor reads in jobs one at a time from the input device(typically a card reader or magnetic tape drive). 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 control to the monitor,

which immediately reads in the next job. The results of each job are sent to an output device, such as a

printer, for delivery to the user.

• Processor point of view: At a certain point, the processor is executing instructions from the portion of main memory containing the monitor. These instructions cause the next job to be read into another portion of main memory. Once a job has been read in, the processor will encounter a branch instruction in the monitor that instructs the processor to continue execution at the start of the user program. The processor will then execute the instructions in the user program until it encounters an ending or error condition. Either event causes the processor to fetch its next instruction from the monitor program. Thus the phrase “control is passed to a job” simply means that the processor is now fetching and executing instructions in a user program, and “control is returned to the monitor” means that the processor is now fetching and executing instructions from the monitor program. The monitor performs a scheduling function: A batch of jobs is queued up, and jobs are executed as rapidly as possible, with no intervening idle time. The monitor improves job setup time as well. With each job, instructions are included in a primitive form of job control language (JCL) . This is a special type of programming

language used to provide instructions to the monitor. A simple example is that of a user submitting a program written in the programming language FORTRAN plus some data to be used by the program. All FORTRAN instructions and data are on a separate punched card or a separate record on tape. In addition to FORTRAN and data lines, the job includes job control instructions, which are denoted by the begin-

ning $. The overall format of the job looks like this:

$JOB

$FTN

•

• ¶ FORTRAN instructions

•

Interrupt

processing

Device

drivers

Job

sequencing

Control language

interpreter

User

program

area

Monitor

Boundary

Figure 2.3 Memory Layout for a

Resident Monitor

$LOAD

$RUN

•

• ¶ Data

•

$END

To execute this job, the monitor reads the $FTN line and loads the appropriate language compiler from its mass storage (usually tape). The compiler translates the user's program into object code, which is stored in memory or mass 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 instruction is required. This instruction is read by the monitor, which regains control after the compile operation. The monitor invokes the loader, which loads the object program into memory (in place of the compiler) and transfers control to it. In this manner, a large segment of main memory can be shared among different subsystems, although only one such subsystem could be executing at a time. 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 input routine checks to make sure that the program does not accidentally read in a JCL line. If this happens, an error occurs and control transfers to the monitor. At the completion of the user job, the monitor will scan the input lines until it encounters the next JCL instruction. Thus, the system is protected against a program with too many or too few data lines. 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 to alternately seize and relinquish control. Certain other hardware features are also

desirable:

• 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. Of course, an OS can be built without these features. But computer vendors quickly learned that the results were chaos, and so even relatively primitive batch operating systems were provided with these hardware features. With a batch OS, processor time alternates between execution of user programs and execution of the monitor. There have been two sacrifices: Some main memory is now given over to the monitor and some processor time is consumed by the monitor. Both of these are forms of overhead. Despite this overhead, the simple batch system improves utilization of the computer.

Multiprogram med Batch Systems

Even 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. Figure 2.4 details a representative calculation. The calculation concerns a program that processes a file of records and performs, on average, 100 machine instructions per record. In this example, the computer spends over 96% of its time waiting for I/O devices to finish transferring data to and from the file. Figure 2.5a illustrates 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 inefficiency is not necessary. We know that there must be enough memory to hold the OS (resident monitor) and one user program. Suppose that there is room for the 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 (Figure 2.5b). Furthermore, we might expand memory to hold three, four, or more programs and switch among all of them (Figure 2.5c). The approach is known as multiprogramming , or multitasking . It is the central theme of modern operating

systems. Figure 2.4 System Utilization Example

Read one record from file 15 ms

Execute 100 instructions 1 ms

Write one record to file 15 ms

Total 31 ms

Percent CPU Utilization = 1

31 = 0.032 = 3.2%

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

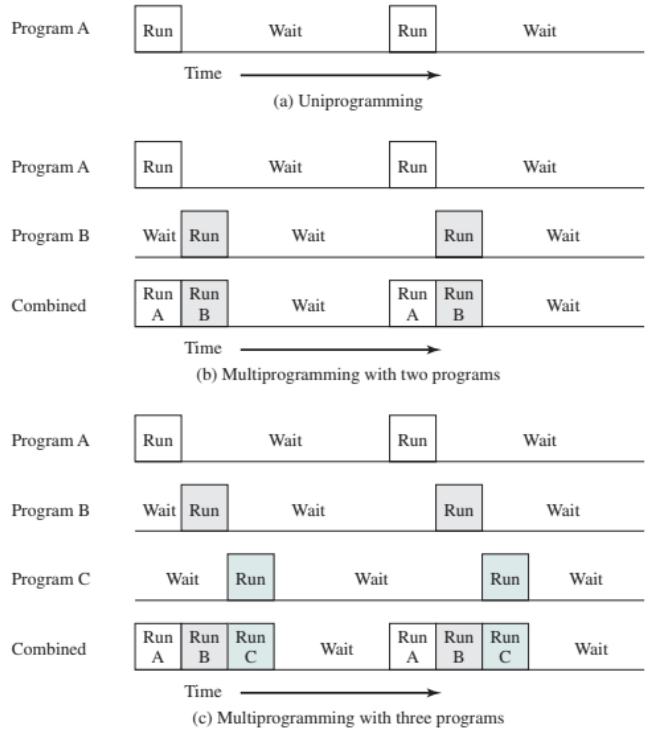


Figure 2.5 **Multiprogramming Example**

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. Now suppose that the jobs are run concurrently under a multiprogramming OS. Because there is little resource contention between the jobs, all three can run in nearly minimum time while coexisting with the others in the computer (assuming that JOB2 and JOB3 are allotted enough processor time to keep their input and

output operations active). JOB1 will still require 5 minutes to complete, but at the end of that time, JOB2 will be one-third finished and JOB3 half finished. All three jobs will have finished within 15 minutes. The improvement is evident when examining the multiprogramming column of Table 2.2 , obtained from the histogram shown in Figure 2.6b . As with a simple batch system, a multiprogramming batch system must rely on certain computer hardware features. The most notable additional feature that is useful for multiprogramming is the hardware that supports I/O interrupts and DMA (direct memory access). With interrupt-driven I/O or DMA, the processor can issue an I/O command for one job and proceed with the execution of another job while the I/O is carried out by the device controller. When the I/O operation is complete, the processor is interrupted and control is passed to an interrupt-handling program in the OS. The OS will then pass control to another job. 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.

Time-Sharing Systems

With 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 directly with the computer. Indeed, for some jobs, such as transaction processing, an interactive mode is essential.

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

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

Table 2.2 Effects of Multiprogramming on Resource Utilization

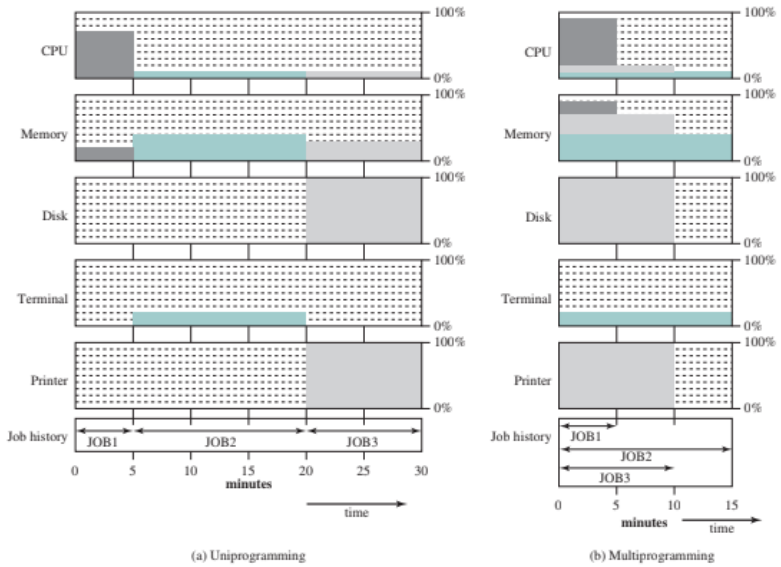


Figure 2.6 **Utilization Histograms**

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

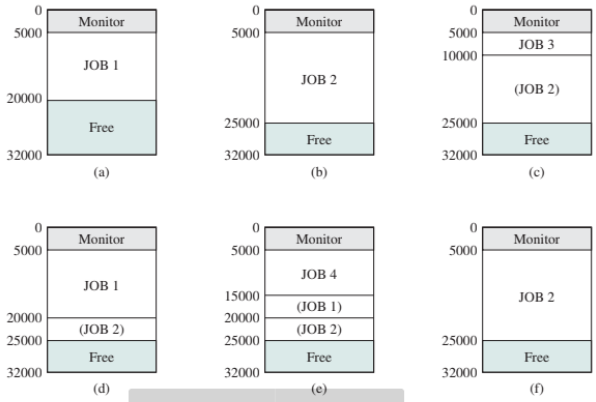


Figure 2.7 **CTSS Operation**

Today, the requirement for an interactive computing facility can be, and often is, met by the use of a dedicated personal computer or workstation. That option was not available in the 1960s, when most computers were big and costly. Instead, time sharing was developed. Just as multiprogramming allows the processor to handle multiple batch jobs at a time, multiprogramming can also be used to handle multiple interactive jobs. In this latter case, the technique is referred to as time sharing , because processor time is shared among multiple users. In a time-sharing system, 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 computation. Thus, if there are n users actively requesting service at one time, each user will only see on the average 1/ n of the effective computer capacity, not counting OS overhead. However, given the relatively slow human reaction time, the response time on a properly

	designed system should be similar to that on a dedicated computer. Both batch processing and time sharing use multiprogramming. The key differences are listed in Table 2.3 . One of the first time-sharing operating systems to be developed was the Compatible Time-Sharing System (CTSS) [CORB62], developed at MIT by a group known as Project MAC (Machine-Aided Cognition, or Multiple-Access Computers). The system was first developed for the IBM 709 in 1961 and later transferred to an IBM 7094. Compared to later systems, CTSS is primitive. The system ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 of that. When control was to be assigned to an interactive user, the user's program and data were loaded into the remaining 27,000 words of main memory. A program was always loaded to start at the location of the 5000th word; this simplified both the monitor and memory management. 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 preserve the old user program status for later resumption, the old user programs and data were written out to disk before the new user programs and data were read in. Subsequently, the old user program code and data were restored in main memory when that program was next given a turn. To minimize disk traffic, user memory was only written out when the incoming program would overwrite it. This principle is illustrated in Figure 2.7 . Assume that there are four interactive users with the following memory requirements, in words: • JOB1: 15,000 • JOB2: 20,000 Table 2.3 Batch Multiprogramming versus Time Sharing to a particular file. The contention for resources, such as printers and mass storage devices, must be handled. These and other problems, with possible solutions, will be encountered throughout this text.
3) **Ans:**	Explain different types of operating systems. Batch operating systems The batch operating system does not have a direct link with the computer. A different system divides and allocates similar tasks into batches for easy processing and faster response. The batch operating system is appropriate for lengthy and time-consuming tasks. To avoid slowing down a device, each user prepares their tasks offline and submits them to an operator. Advantages of using a batch operating system include: • Many users can share batch systems. • There is little idle time for batch operating systems. • It becomes possible to manage large workloads. • It's easy to estimate how long a task will take to be completed.

Some notable disadvantages are:

- Batch operating systems are challenging to debug.
- Any failure of the system creates a backlog.
- It may be costly to install and maintain good batch operating systems.

Batch operating systems are used for tasks such as managing payroll systems, data entry and bank statements.

Time-sharing or multitasking operating systems

The time-sharing operating system, also known as a multitasking OS, works by allocating time to a particular task and switching between tasks frequently. Unlike the batch system, the time-sharing system allows users to complete their work in the system simultaneously. It allows many users to be distributed across various terminals to minimize response time. Potential advantages of time-sharing operating systems include:

- There's a quick response during task performance.
- It minimizes the idle time of the processor.
- All tasks get an equal chance of being accomplished.
- It reduces the chance of software duplication.

Some potential disadvantages of this system are:

- The user's data security might be a problem.
- System failure can lead to widespread failures.
- Problems in data communication may arise.
- The integrity of user programs is not assured.

Examples of time-sharing operating systems include Multics and Unix.

Distributed operating systems

This system is based on autonomous but interconnected computers communicating with each other via communication lines or a shared network. Each autonomous system has its own processor that may differ in size and function. A distributed operating system serves multiple applications and multiple users in real-time. The data processing function is then distributed across the processors. Potential advantages of distributed operating systems are:

- They allow remote working.
- They allow a faster exchange of data among users.
- Failure in one site may not cause much disruption to the system.

- They reduce delays in data processing.
- They minimize the load on the host computer.
- They enhance scalability since more systems can be added to the network.

Potential disadvantages of distributed operating systems include:

- If the primary network fails, the entire system shuts down.
- They're expensive to install.
- They require a high level of expertise to maintain.

Distributed operating systems are used for tasks such as telecommunication networks, airline reservation controls and peer-to-peer networks.

Network operating systems

Network operating systems are installed on a server providing users with the capability to manage data, user groups and applications. This operating system enables users to access and share files and devices such as printers, security software and other applications, mostly in a local area network. Potential advantages of network operating systems are:

- Centralized servers provide high stability.
- Security issues are easier to handle through the servers.
- It's easy to upgrade and integrate new technologies.
- Remote access to the servers is possible.

Potential disadvantages of network operating systems are:

- They require regular updates and maintenance.
- Servers are expensive to buy and maintain.
- Users' reliance on a central server might be detrimental to workflows.

Examples of network operating systems include Microsoft Windows, Linux and Mac OS X.

Real-time operating systems

Real-time operating systems provide support to real-time systems that require observance of strict time requirements. The response time between input, processing and response is tiny, which is beneficial for processes that are highly sensitive and need high precision. These processes include operating missile systems, medical systems or air traffic control systems, where delays may lead to loss of life and property.

Real-time operating systems may either be hard real-time systems or soft real-time systems.

Hard real-time systems are installed in applications with strict time constraints. The system guarantees the completion of sensitive tasks on time. Hard real-time does not have virtual memory. Soft real-time systems do not have equally rigid time requirements. A critical task gets priority over other tasks.

Potential advantages of real-time operating systems include:

- They use device and systems maximally, hence more output.
- They allow fast shifting from one task to another.
- The focus is on current tasks, and less focus is put on the queue.
- They can be used in embedded systems.
- Real-time systems are meticulously programmed, hence free of errors.
- They allow easy allocation of memory.

Potential disadvantages of real-time operating systems are:

- They have a low capacity to run tasks simultaneously.
- They use heavy system resources.
- They run on complex algorithms that are not easy to understand.
- They're unsuitable for thread priority because of the system's inability to switch tasks.

Real-time operating systems are used for tasks such as scientific experiments, medical imaging, robotics and air traffic control operations.

Mobile operating systems

Mobile operating systems run exclusively on small devices such as smartphones, tablets and wearables. The system combines the features of a personal computer with additional features useful for a handheld device. Mobile operating systems start when a device is powered on to provide access to installed applications. Mobile operating systems also manage wireless network connectivity.

Potential advantages of mobile operating systems are:

- Most systems are easy for users to learn and operate.

Potential disadvantages of real-time operating systems are:

- Some mobile OS put a heavy drain on a device’s battery, requiring frequent recharging.

- Some systems are not user-friendly.

Examples of mobile operating systems include Android OS, Apple and Windows mobile OS.

Related: **What Is Mobile Device Management and Why Does It Matter?**

Common operating systems

Here are the most common operating systems in use:

Microsoft Windows

Created by Microsoft, Microsoft Windows is one of the most popular proprietary operating systems for computers in the world. Most personal computers come preloaded with a version of Microsoft Windows. One downside of Windows is that compatibility with mobile phones has been problematic.

Apple iOS

Apple iOS from Apple is used on smartphones and tablets manufactured by the same company. Users of this system have access to hundreds of applications. The operating system offers strong encryption capabilities to control unauthorized access to users' private data.

Google Android

Android from Google is the most popular operating system in the world. It's mainly used on tablets and smartphones. It also runs on devices made by other manufacturers. Users have access to numerous mobile applications available on the Google Play Store.

Apple macOS

Developed by Apple, this proprietary operating system runs on the manufacturer's personal computers and desktops. All Apple and Macintosh computers come equipped with the latest version of macOS, previously known as OS X systems. The ability to prevent bugs and fend off hackers make Apple operating systems popular with their users.

Linux

Created by the Finnish programmer Linus Torvalds, Linux is today developed by programmer collaborators across the world who submit tweaks to the central kernel software. Linux is popular with programmers and corporate servers. It is available for free online.

4) Explain Linux Kernel.

Ans:

The main purpose of a computer is to run a *predefined sequence of instructions*, known as a **program**. A program under execution is often referred to as a **process**. Now, most special purpose computers are meant to run a single process, but in a sophisticated system such a general purpose computer, are intended to run many processes simulteneously. Any kind of process requires hardware resources such are Memory, Processor time, Storage space, etc. In a General Purpose Computer running many processes simulteneously, we need a middle layer to manage the distribution of the hardware resources of the computer efficiently and fairly among all the various processes running on the computer. This middle layer is referred to as the **kernel**. Basically the kernel virtualizes the common hardware resources of the computer to provide each process with its own virtual resources. This makes the process seem as it is the sole process running on the machine. The kernel is also responsible for preventing and mitigating conflicts between different processes.
This schematically represented below:

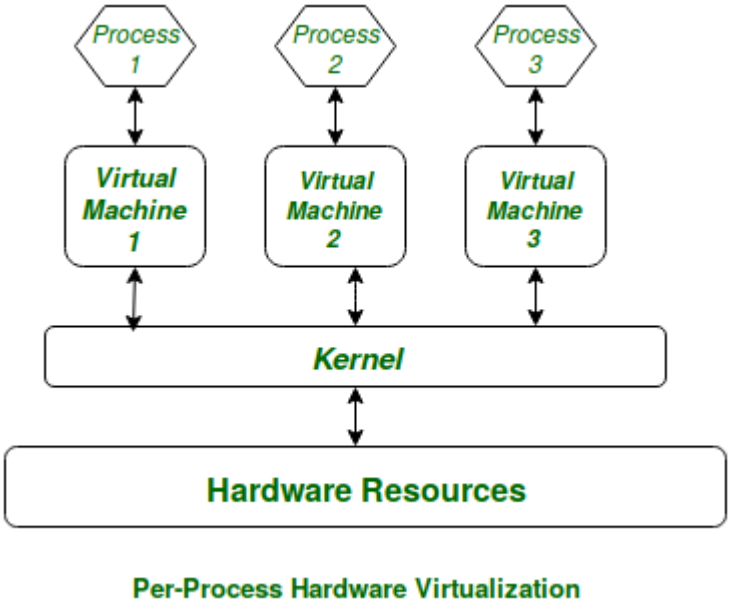


Figure: Virtual Resources for each Process

The **Core Subsystems** of the **Linux Kernel** are as follows:

1. The Process Scheduler
2. The Memory Management Unit (MMU)
3. The Virtual File System (VFS)
4. The Networking Unit
5. Inter-Process Communication Unit

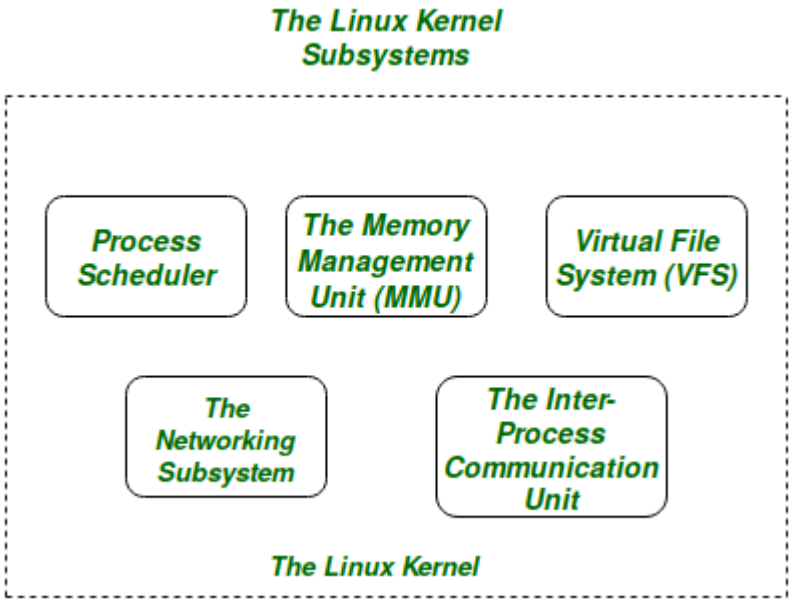


Figure: The Linux Kernel

For the purpose of this article we will only be focussing on the 1st three important subsystems of the Linux Kernel.

The basic functioning of each of the 1st three subsystems is elaborated below:

- **The Process Scheduler:**
  This kernel subsystem is responsible for fairly distributing the CPU time among all the processes running on the system simulteneously.
- **The Memory Management Unit:**
  This kernel sub-unit is responsible for proper distribution of the memory resources among the various processes running on the system. The MMU does more than just simply provide separate virtual address spaces for each of the processes.
- **The Virtual File System:**
  This subsystem is responsible for providing a unified interface to access stored data across different filesystems and physical storage media.

5) Write a note on "Layered System".

Ans:

Layered Structure is a type of system structure in which the different services of the operating system are split into various layers, where each layer has a specific well-defined task to perform. It was created to improve the pre-existing structures like the Monolithic structure (UNIX) and the Simple structure (MS-DOS).

Example – The Windows NT operating system uses this layered approach as a part of it.

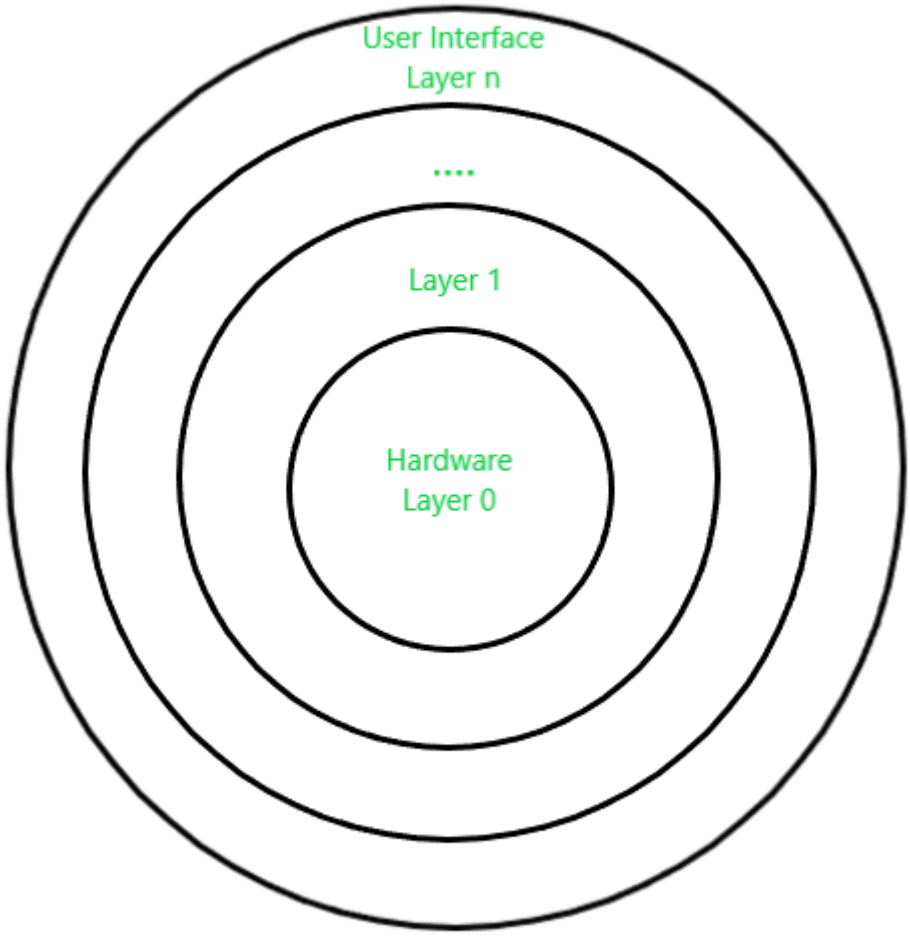
Design Analysis :

The whole Operating System is separated into several layers (from 0 to n) as the diagram shows. Each of the layers must have its own specific function to perform. There are some rules in the implementation of the layers as follows.

1. The outermost layer must be the User Interface layer.
2. The innermost layer must be the Hardware layer.
3. A particular layer can access all the layers present below it but it cannot access the layers present above it. That is layer n-1 can access all the layers from n-2 to 0 but

it cannot access the nth layer.

Thus if the user layer wants to interact with the hardware layer, the response will be traveled through all the layers from n-1 to 1. Each layer must be designed and implemented such that it will need only the services provided by the layers below it.



Layered OS Design

Advantages :

There are several advantages to this design :

1. **Modularity :**
   This design promotes modularity as each layer performs only the tasks it is scheduled to perform.
2. **Easy debugging :**
   As the layers are discrete so it is very easy to debug. Suppose an error occurs in the CPU scheduling layer, so the developer can only search that particular layer to debug, unlike the Monolithic system in which all the services are present together.

3. **Easy update :**
   A modification made in a particular layer will not affect the other layers.
4. **No direct access to hardware :**
   The hardware layer is the innermost layer present in the design. So a user can use the services of hardware but cannot directly modify or access it, unlike the Simple system in which the user had direct access to the hardware.
5. **Abstraction :**
   Every layer is concerned with its own functions. So the functions and implementations of the other layers are abstract to it.

Disadvantages :

Though this system has several advantages over the Monolithic and Simple design, there are also some disadvantages as follows.

1. **Complex and careful implementation :**
   As a layer can access the services of the layers below it, so the arrangement of the layers must be done carefully. For example, the backing storage layer uses the services of the memory management layer. So it must be kept below the memory management layer. Thus with great modularity comes complex implementation.
2. **Slower in execution :**
   If a layer wants to interact with another layer, it sends a request that has to travel through all the layers present in between the two interacting layers. Thus it increases response time, unlike the Monolithic system which is faster than this. Thus an increase in the number of layers may lead to a very inefficient design.