


CMR INSTITUTE OF TECHNOLOGY		USN					<input type="text"/>						
Internal Assessment Test 1 – March 2024													
Sub:	Operating System Concepts								Sub Code:	22MCA12			
Date:	12/03/2024		Duration:	90 min's	Max Marks:	50	Sem:	I	Branch:	MCA			
Note : Answer FIVE FULL Questions, choosing ONE full question from each Module													
PART I									MARKS	OBE			
										CO	RBT		
1	Explain various system structures with diagrams. OR								[10]	CO1	L2		
2	Explain in detail System Calls.								[10]	CO1	L2		
PART II									[10]	CO1	L2		
3	What are the various services provided by the operating system? OR												
4	What are the functions of an operating system?								[10]	CO1	L2		
PART III									[10]	CO1	L2		
5	Explain various types of operating systems with advantages and disadvantages. OR												
6	Explain the dual mode and the timer mode operations of the operating system.								[10]	CO1	L2		
PART IV									[10]	CO1	L2		
7	Explain process states with a neat diagram. OR												
8.	Explain Interprocess Communication.								[10]	CO1	L2		
PART V									[10]	CO5	L2		
9.	Explain FCFS (non preemptive) and SJF (preemptive) scheduling with an example. OR												
10.	Calculate the average waiting time, turn around time for i)SJF ii)Priority Scheduling and iii)Round Robin(tq=2ms) with the								[10]	CO5	L3		

following set of process.

Process	P1	P2	P3	P4	P5
Burst Time	10	1	2	1	5
Priority	3	1	3	4	5

Internal Assessment Test 1 – March 2024

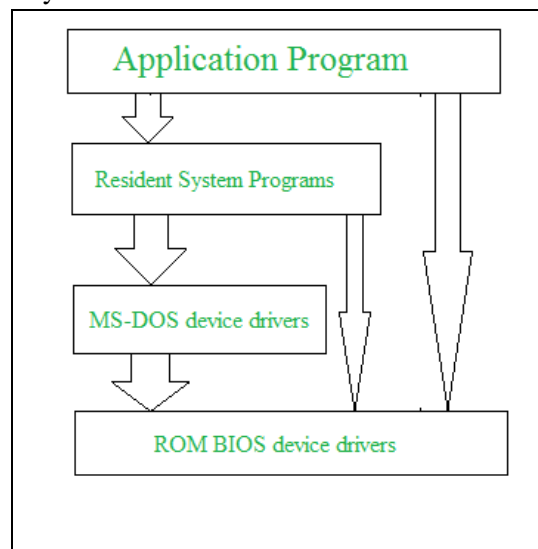
Operating System Concepts						Sub Code:	22MCA12	
12/03/2024	Duration:	90 min's	Max Marks:	50	Sem:	I	Branch:	MCA

PART I

1. Explain Various system structures with diagrams

Simple Structure

Many commercial systems do not have well-defined structures. Frequently, such operating systems started as small, simple, and limited systems and then grew beyond their original scope. MS-DOS is an example of such a system.



It was written to provide the most functionality in the least space, so it was not divided into modules carefully. In MS-DOS, the interfaces and levels of functionality are not well separated. For instance, application programs are able to access the basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS vulnerable to errant (or malicious) programs, causing entire system crashes when user programs fail. Of course, MS-DOS was also limited by the hardware of its era. Another example of limited structuring is the original UNIX operating system. UNIX is another system that initially was limited by hardware functionality.

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 is simplified.

Each layer is implemented with only those operations provided by lower level layers. A layer does not need to know how these operations are implemented; it needs to know only what these operations do. Hence, each layer hides the existence of certain data structures, operations, and hardware from higher-level layers.

The major difficulty with the layered approach involves appropriately defining the various layers. The backing-store driver would normally be above the CPU scheduler, because the driver may need to wait for I/O and the CPU can be rescheduled during this time. A final problem with layered implementations is that they tend to be less efficient than other types. For instance, when a user program executes an I/O operation, it executes a system call that is trapped to the I/O layer, which calls the memory-management layer, which in turn calls the CPU-scheduling layer, which is then passed to the hardware.

Micro kernels

The kernel became large and difficult to manage. In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the kernel using the **microkernel** approach. This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs. The result is a smaller kernel. Micro kernels provide minimal process and memory management, in addition to a communication facility.

The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space. One benefit of the microkernel approach is ease of extending the operating system. All new services are added to user space and consequently do not require modification of the kernel. When the kernel does have to be modified, the changes tend to be fewer, because the microkernel is a smaller kernel.

The resulting operating system is easier to port from one hardware design to another. The microkernel also provides more security and reliability, since most services are running as user rather than kernel processes. If a service fails, the rest of the operating system remains untouched.

Modules

The best current methodology for operating-system design involves using object-oriented programming techniques to create a modular kernel. Here, the kernel has a set of core components and dynamically links in additional services either during boot time or during run time. Such a strategy uses dynamically loadable modules and is common in modern implementations of UNIX, such as Solaris, Linux, and Mac OS X.

A core kernel with seven types of loadable kernel modules:

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

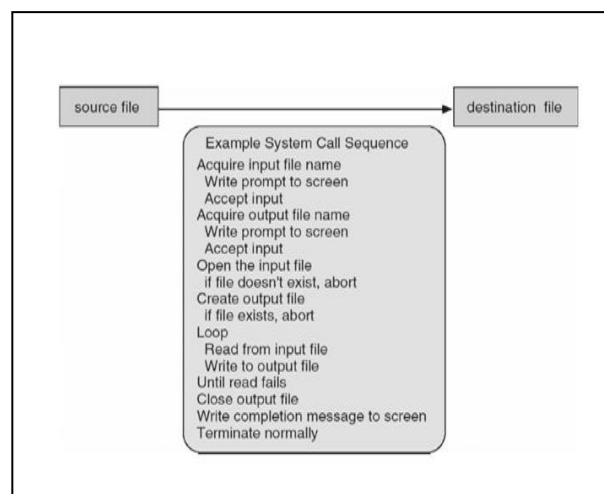
Such a design allows the kernel to provide core services yet also allows certain features to be implemented dynamically. The overall result resembles a layered system in that each kernel section has defined, protected interfaces; but it is more flexible than a layered system in that any module can call any other module. The approach is like the microkernel approach in that the 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.

The Apple Macintosh Mac OS X operating system uses a hybrid structure. Mac OS X (also known as *Darwin*) structures the operating system using a layered technique where one layer consists of the Mach microkernel. The top layers include application environments and a set of services providing a graphical interface to applications. Below these layers is the kernel environment, which consists primarily of the Mach microkernel and the BSD kernel. Mach provides memory management; support for remote procedure calls (RPCs) and inter process communication (IPC) facilities, including message passing; and thread scheduling. The BSD component provides a BSD command line interface, support for networking and file systems, and an implementation of POSIX APIs, including Pthreads.

2. Explain in detail System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use. 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).
- Why use APIs rather than system calls? (Note that the system-call names used throughout this text are generic)

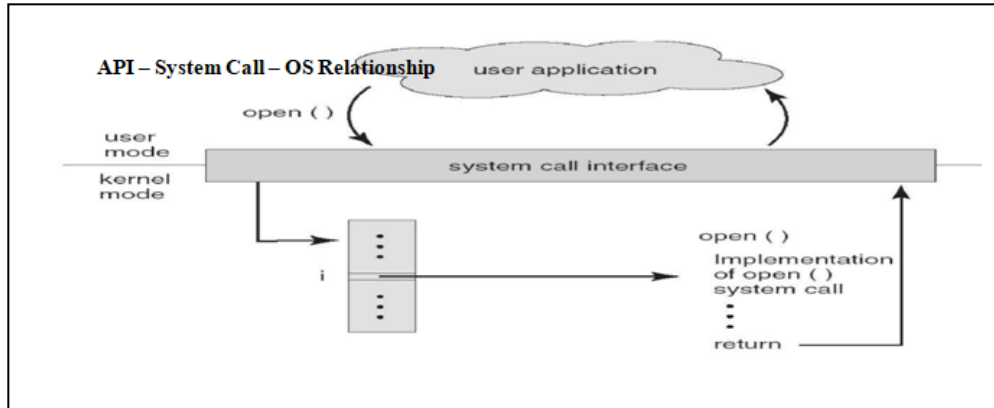
Example of System Calls



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
- 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)
-

API – System Call – OS Relationship



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
- Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
- 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

1.2.2 Types of System Calls

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

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**—a system program

designed to aid the programmer in finding and correcting bugs—to determine the cause of the problem. Under either normal or abnormal circumstances, the operating system must transfer control to the invoking command interpreter. The command interpreter then reads the next command. In an interactive system, the command interpreter simply continues with the next command; it is assumed that the user will issue an appropriate command to respond to any error.

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. We may need these same sets of operations for directories if we have a directory structure for organizing files in the file system. In addition, for either files or directories, we need to be able to determine the values of various attributes and perhaps to reset them if necessary. File attributes include the file name, a file type, protection codes, accounting information, and so on.

At least two system calls, get file attribute and set file attribute, are required for this function. Some operating systems provide many more calls, such as calls for file move and copy.

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. The various resources controlled by the operating system can be thought of as devices. Some of these devices are physical devices (for example, tapes), while others can be thought of as abstract or virtual devices (for example, files). If there are multiple users of the system, the system may require us to first request the device, to ensure exclusive use of it. After we are finished with the device, we release it. These functions are similar to the open and system calls for files.

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.

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).

Communication

There are two common models of inter process communication: the message passing model and the 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. Each computer in a network has a *host name* by which it is commonly known. A host also has a network identifier, such as an IP address. Similarly, each process has a *process name*, and this name is translated into an identifier by which the operating system can refer to the process. The `get host id` and `get processid` system calls do this translation. The identifiers are then passed to the general purpose open and close calls provided by the file system or to specific open connection and close connection system calls, depending on the system's model of communication.

In the shared-memory model, processes use shared memory creates and shared memory attaches system calls to create and gain access to regions of memory owned by other processes. Recall that, normally, the operating system tries to prevent one process from accessing another process's memory. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. The form of the data and the location are determined by the processes and are not under the operating system's control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

1.2.3 System Programs

At the lowest level is hardware. Next are the operating system, then the system programs, and finally the application programs. System programs provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls; others are considerably more complex. They can be divided into these categories:

- **File management.** These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
- **Status information.** Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Others are more complex, providing detailed performance, logging, and debugging information. Typically, these programs format and print the output to the terminal or other output devices or files or display it in a window of the GUI. Some systems also support a registry, which is used to store and retrieve configuration information.
- **File modification.** Several text editors may be available to create and modify the content of files stored on disk or other storage devices. There may also be special commands to search contents of files or perform transformations of the text.
- **Programming-language support.** Compilers, assemblers, debuggers and interpreters for common programming languages (such as C, C++, Java, Visual Basic, and PERL) are often provided to the user with the operating system.
- **Program loading and execution.** Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are needed as well.
- **Communications.** These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. They allow users to send messages to one another's screens, to browse web pages, to send electronic-mail messages, to log in remotely, or to transfer files from one machine to another.

In addition to systems programs, most In addition to systems programs, most operating systems are supplied with programs that are useful in solving common problems or performing common

operations. Such programs include web browsers, word processors and text formatters, spreadsheets, database systems, compilers, plotting and statistical-analysis packages, and games. These programs are known as system utilities or application programs.

PART II

3. What are the various services provided by the operating system?

An operating system (OS) provides a range of essential services to facilitate the management and operation of computer hardware and software. The specific services may vary depending on the type of operating system, but here are some common services provided by most operating systems:

Process Management: Process Scheduling: Allocating CPU time to different processes efficiently. Creation and Termination: Creating, pausing, and terminating processes.

Memory Management: Allocation and Deallocation: Managing the assignment and release of memory space for processes.

Virtual Memory: Providing an abstraction of larger memory space than physically available.

File System Management: File Creation, Deletion, and Manipulation: Handling files and directories.

File Permissions: Controlling access to files and directories.

Device Management: Device Drivers: Communicating with hardware devices like printers, disk drives, and input/output devices.

Interrupt Handling: Managing and responding to hardware interrupts.

Security and Protection:

User Authentication: Verifying the identity of users.

Access Control: Determining who can access resources and perform specific actions.

User Interface: Graphical User Interface (GUI) or Command-Line Interface (CLI): Providing ways for users to interact with the system.

Input/Output Services: Managing input and output operations.

Networking: Network Protocol Implementation: Supporting communication between different computers on a network.

Network Configuration: Managing network settings and connections.

Error Handling: Error Logging and Reporting: Recording and reporting system errors.

Fault Tolerance: Providing mechanisms to recover from system failures.

System Calls: APIs (Application Programming Interfaces): Enabling applications to interact with the operating system.

Utilities:

System Utilities: Tools for system administration and maintenance.

Programming Utilities: Supporting software development.

Task Scheduling:

Batch Processing: Executing tasks in the background at scheduled times.

Real-time Scheduling: Ensuring timely execution of critical tasks.

These services collectively allow the operating system to manage resources efficiently, provide a user-friendly interface, ensure security, and enable seamless interaction between hardware and software components. Different types of operating systems (e.g., Windows, Linux, macOS) may emphasize certain services depending on their intended use and design principles.

4. What are the functions of an operating system?

- Process Management
- Memory Management
- Input and Output Management
- File System Management

Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.
- Process needs resources to accomplish its task
- CPU, memory, I/O, files
- Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
- Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
- Concurrency by multiplexing the CPUs among the processes / threads

Process Management Activities

- The operating system is responsible for the following activities in connection with process management:
- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management

- All data in memory before and after processing

- All instructions in memory in order to execute
- Memory management determines what is in memory when
- Optimizing CPU utilization and computer response to users
- **Memory management activities**
- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes (or parts thereof) and data to move into and out of memory
- Allocating and deal locating memory space as needed

Storage Management

- OS provides uniform, logical view of information storage
- Abstracts physical properties to logical storage unit - **file**
- Each medium is controlled by device (i.e., disk drive, tape drive)
- Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
- Files usually organized into directories
- Access control on most systems to determine who can access what

OS activities include

- Creating and deleting files and directories
- Primitives to manipulate files and dirs
- Mapping files onto secondary storage
- Backup files onto stable (non-volatile) storage media

Input and Output Management - assignment of different output and input

File System Management - Operating system is also responsible for maintenance of a file system, in which the users are allowed to create, delete and move files.

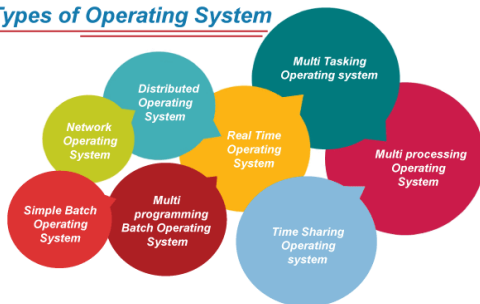
PART III

5. Explain various types of operating systems with advantages and disadvantages.

Types of Operating Systems (OS)

An operating system is a well-organized collection of programs that manages the computer hardware. It is a type of system software that is responsible for the smooth functioning of the computer system.

Types of Operating System

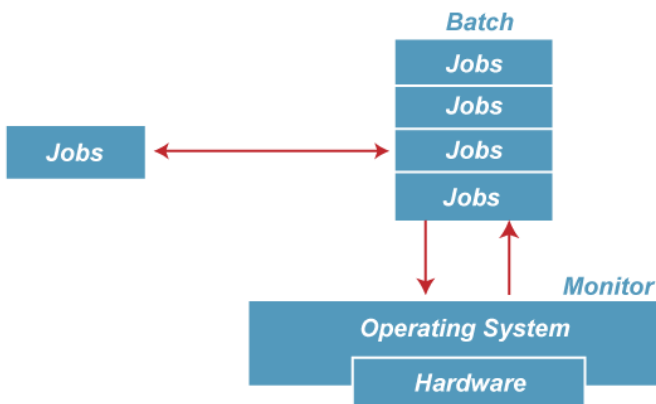


Batch Operating System

In the 1970s, Batch processing was very popular. In this technique, similar types of jobs were batched together and executed in time. People were used to having a single computer which was called a mainframe.

In Batch operating system, access is given to more than one person; they submit their respective jobs to the system for the execution.

The system put all of the jobs in a queue on the basis of first come first serve and then executes the jobs one by one. The users collect their respective output when all the jobs get executed.



The purpose of this operating system was mainly to transfer control from one job to another as soon as the job was completed. It contained a small set of programs called the resident monitor that always resided in one part of the main memory. The remaining part is used for servicing jobs.

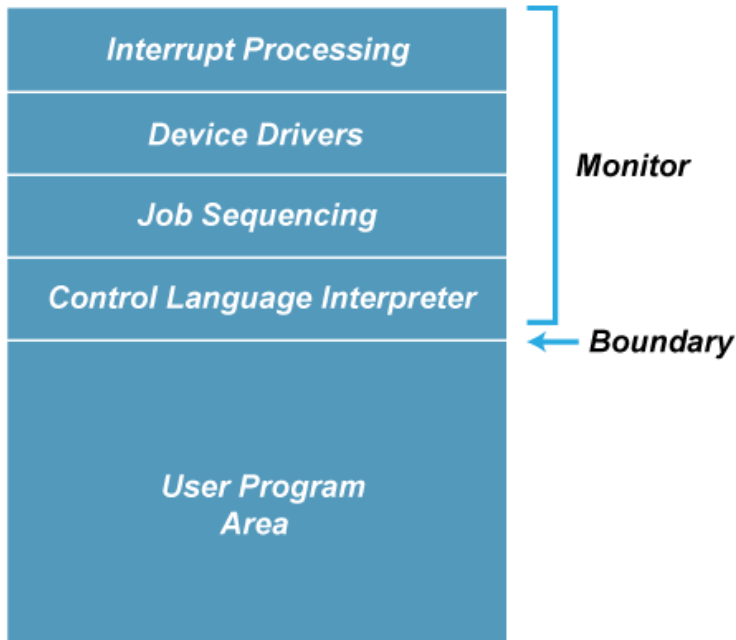


Figure: Memory Layout of the resident monitor

Advantages of Batch OS

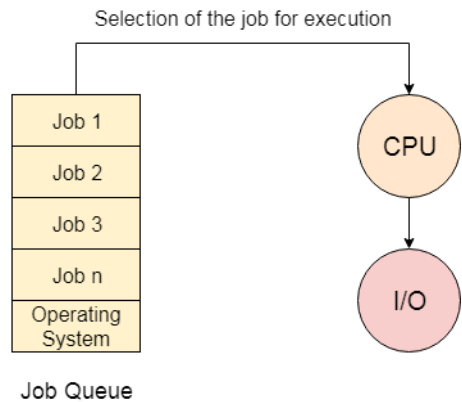
- The use of a resident monitor improves computer efficiency as it eliminates CPU time between two jobs.

Disadvantages of Batch OS

1. Starvation

Batch processing suffers from starvation.

For Example:



There are five jobs J1, J2, J3, J4, and J5, present in the batch. If the execution time of J1 is very high, then the other four jobs will never be executed, or they will have to wait for a very long time. Hence the other processes get starved.

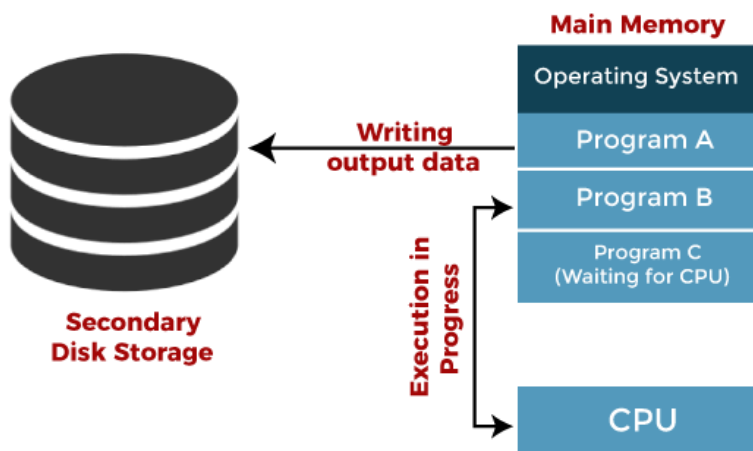
2. Not Interactive

Batch Processing is not suitable for jobs that are dependent on the user's input. If a job requires the input of two numbers from the console, then it will never get it in the batch processing scenario since the user is not present at the time of execution.

Multiprogramming Operating System

Multiprogramming is an extension to batch processing where the CPU is always kept busy. Each process needs two types of system time: CPU time and IO time.

In a multiprogramming environment, when a process does its I/O, The CPU can start the execution of other processes. Therefore, multiprogramming improves the efficiency of the system.



Jobs in multiprogramming system

Advantages of Multiprogramming OS

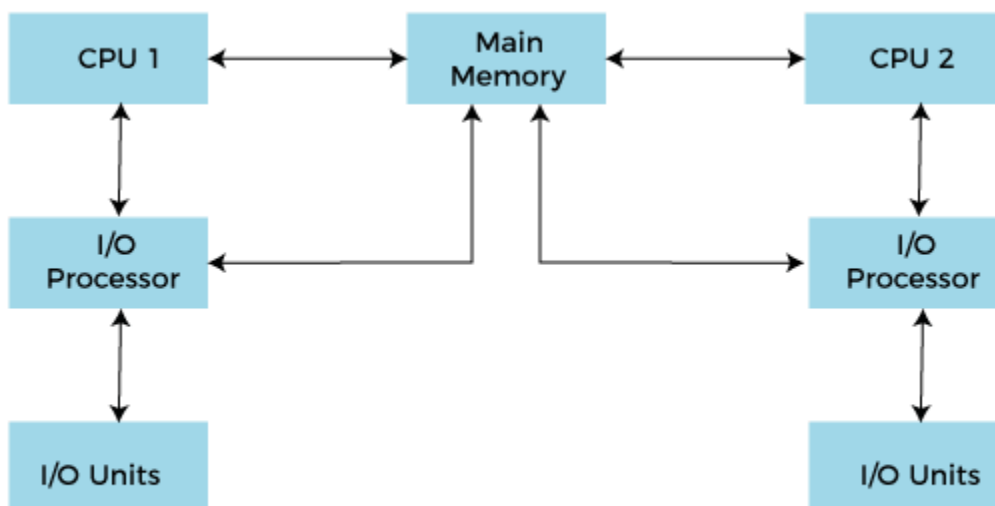
- Throughout the system, it increased as the CPU always had one program to execute.
- Response time can also be reduced.

Disadvantages of Multiprogramming OS

- Multiprogramming systems provide an environment in which various systems resources are used efficiently, but they do not provide any user interaction with the computer system.

Multiprocessing Operating System

In Multiprocessing, Parallel computing is achieved. There are more than one processors present in the system which can execute more than one process at the same time. This will increase the throughput of the system.



Working of Multiprocessor System

In Multiprocessing, Parallel computing is achieved. More than one processor present in the system can execute more than one process simultaneously, which will increase the throughput of the system.

Types of Multiprocessing systems



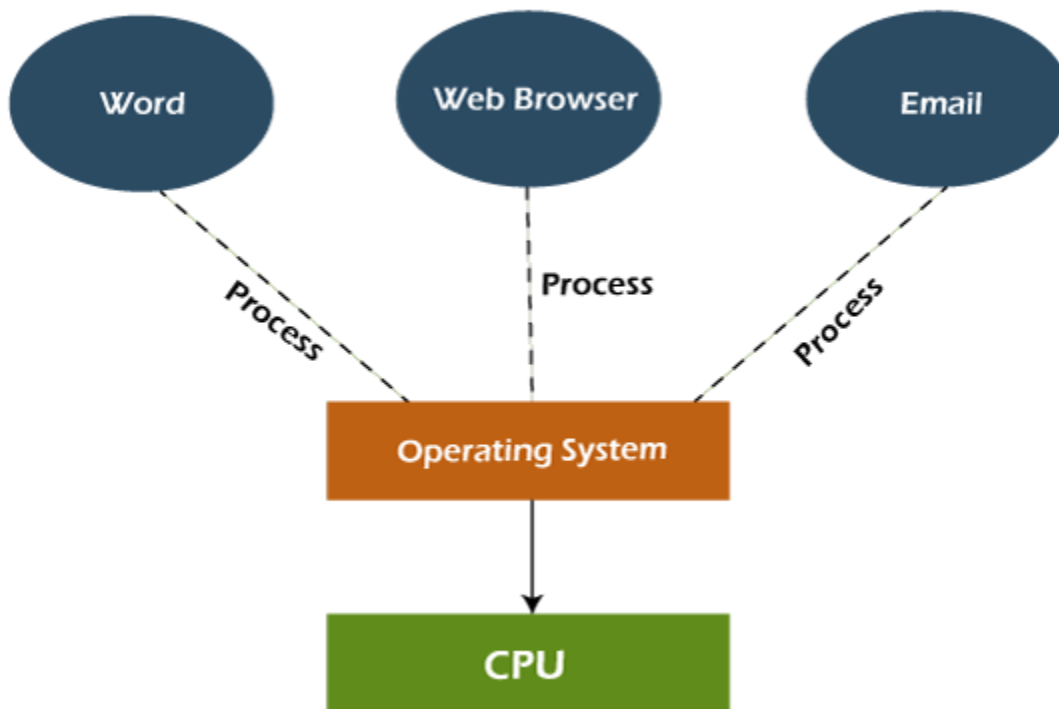
Advantages of Multiprocessing operating system:

- **Increased reliability:** Due to the multiprocessing system, processing tasks can be distributed among several processors. This increases reliability as if one processor fails, the task can be given to another processor for completion.
- **Increased throughput:** As several processors increase, more work can be done in less.

Disadvantages of Multiprocessing operating System

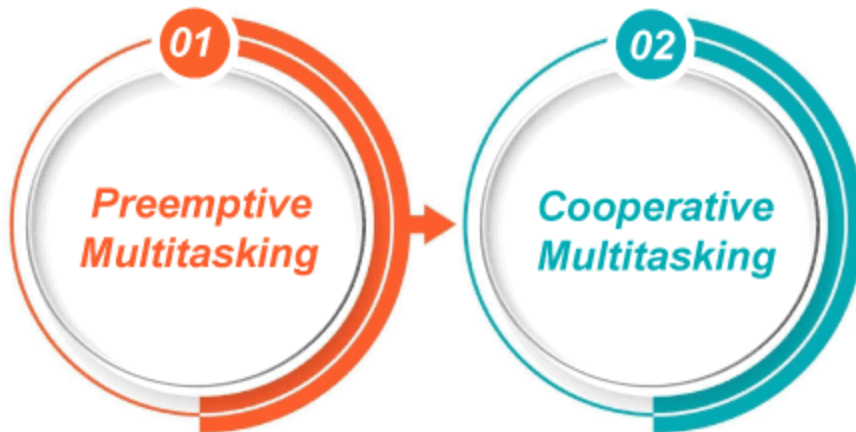
- Multiprocessing operating system is more complex and sophisticated as it takes care of multiple CPUs simultaneously.

Multitasking Operating System



The multitasking operating system is a logical extension of a multiprogramming system that enables **multiple** programs simultaneously. It allows a user to perform more than one computer task at the same time.

Types of Multitasking



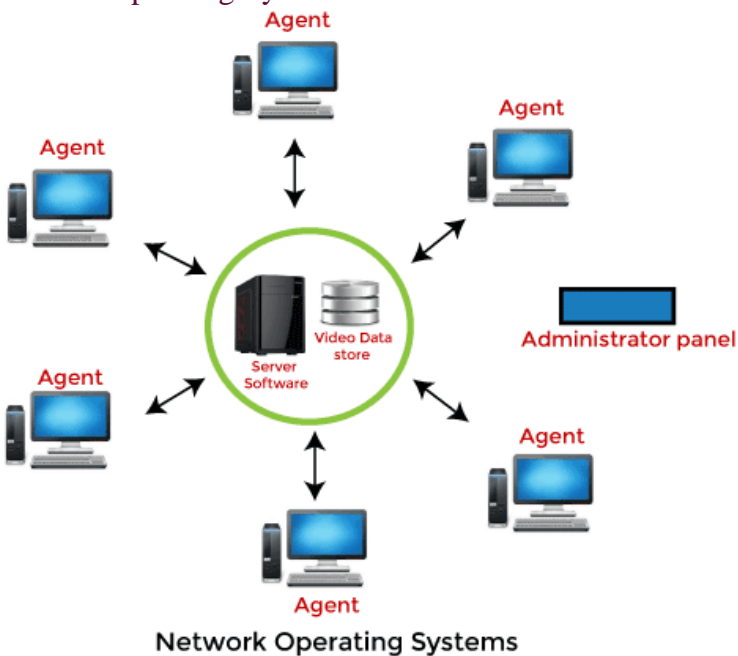
Advantages of Multitasking operating system

- This operating system is more suited to supporting multiple users simultaneously.
- The multitasking operating systems have well-defined memory management.

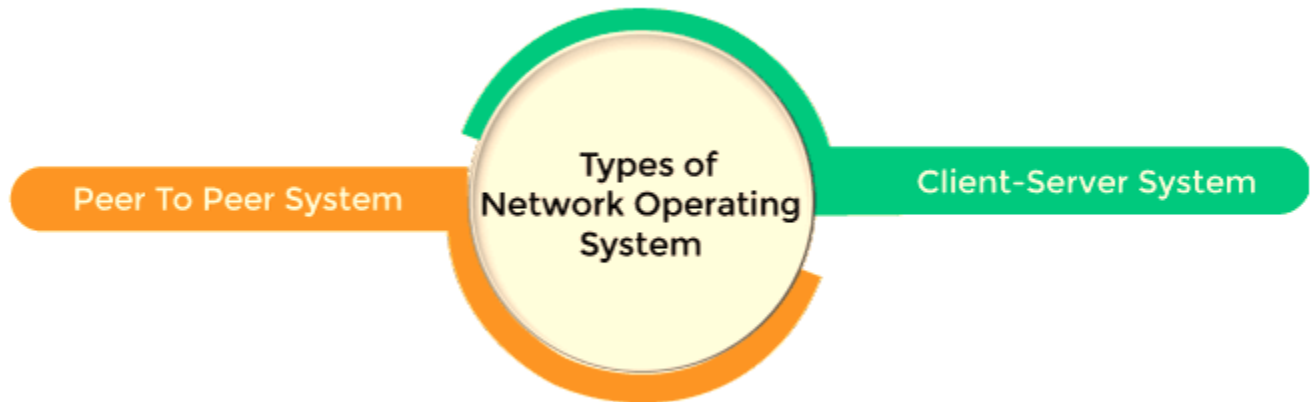
Disadvantages of Multitasking operating system

- The multiple processors are busier at the same time to complete any task in a multitasking environment, so the CPU generates more heat.

Network Operating System



An Operating system, which includes software and associated protocols to communicate with other computers via a network conveniently and cost-effectively, is called Network Operating System.



Advantages of Network Operating System

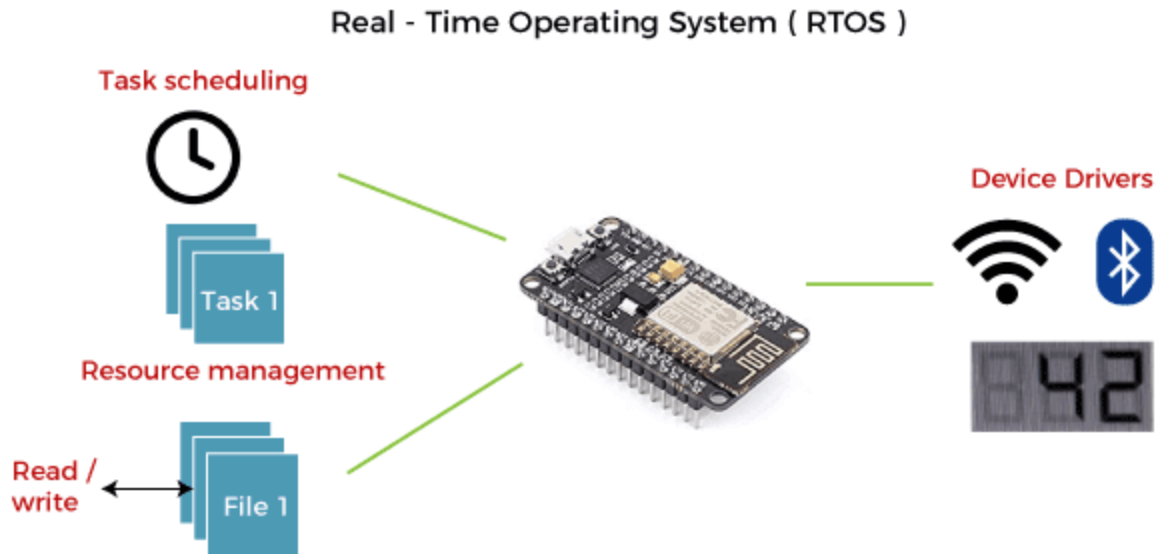
- In this type of operating system, network traffic reduces due to the division between clients and the server.
- This type of system is less expensive to set up and maintain.

Disadvantages of Network Operating System

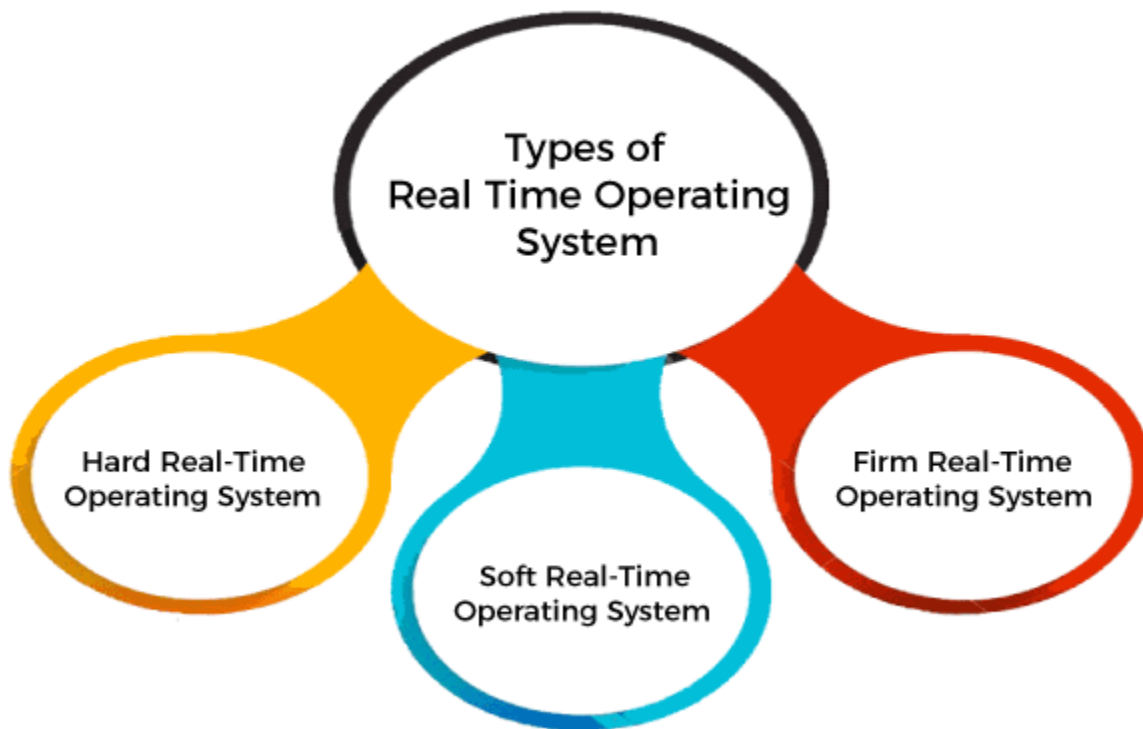
- In this type of operating system, the failure of any node in a system affects the whole system.
- Security and performance are important issues. So trained network administrators are required for network administration.

Real Time Operating System

In Real-Time Systems, each job carries a certain deadline within which the job is supposed to be completed, otherwise, the huge loss will be there, or even if the result is produced, it will be completely useless.



The Application of a Real-Time system exists in the case of military applications, if you want to drop a missile, then the missile is supposed to be dropped with a certain precision.



Advantages of Real-time operating system:

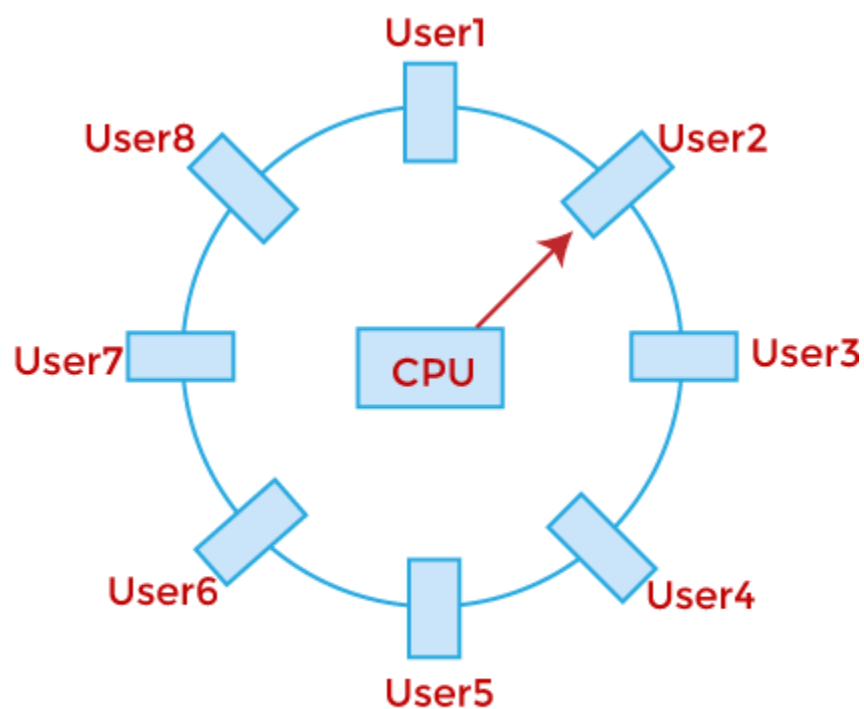
- Easy to layout, develop and execute real-time applications under the real-time operating system.
- In a Real-time operating system, the maximum utilization of devices and systems.

Disadvantages of Real-time operating system:

- Real-time operating systems are very costly to develop.
- Real-time operating systems are very complex and can consume critical CPU cycles.

Time-Sharing Operating System

In the Time Sharing operating system, computer resources are allocated in a time-dependent fashion to several programs simultaneously. Thus it helps to provide a large number of user's direct access to the main computer. It is a logical extension of multiprogramming. In time-sharing, the CPU is switched among multiple programs given by different users on a scheduled basis.



Timesharing in case of 8 users

A time-sharing operating system allows many users to be served simultaneously, so sophisticated CPU scheduling schemes and Input/output management are required.

Time-sharing operating systems are very difficult and expensive to build.

Advantages of Time Sharing Operating System

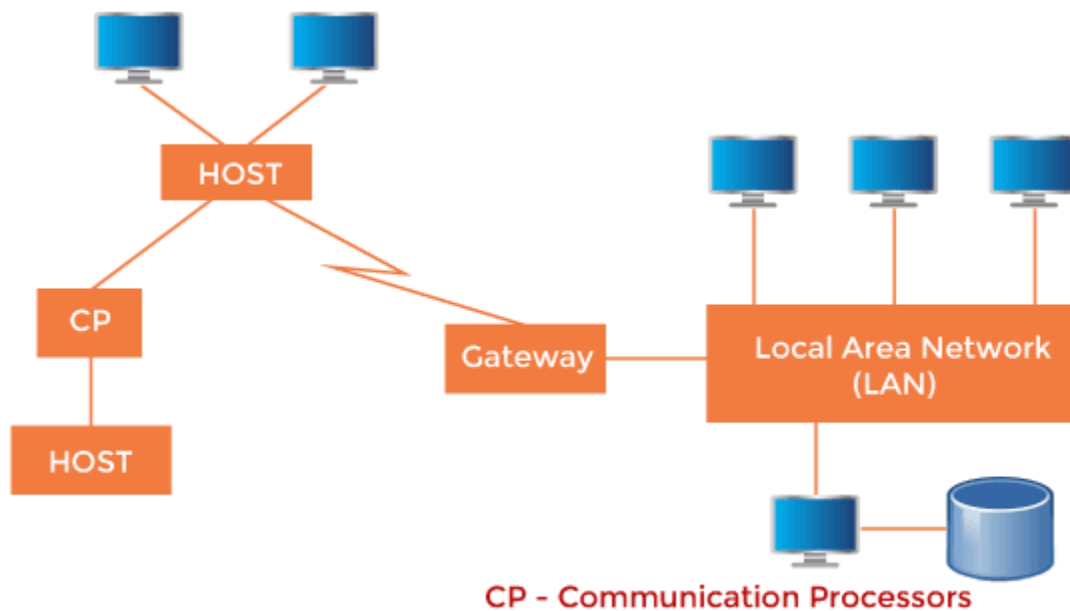
- The time-sharing operating system provides effective utilization and sharing of resources.
- This system reduces CPU idle and response time.

Disadvantages of Time Sharing Operating System

- Data transmission rates are very high in comparison to other methods.
- Security and integrity of user programs loaded in memory and data need to be maintained as many users access the system at the same time.

Distributed Operating System

The Distributed Operating system is not installed on a single machine, it is divided into parts, and these parts are loaded on different machines. A part of the distributed Operating system is installed on each machine to make their communication possible. Distributed Operating systems are much more complex, large, and sophisticated than Network operating systems because they also have to take care of varying networking protocols.



A Typical View of a Distributed System

Advantages of Distributed Operating System

- The distributed operating system provides sharing of resources.
- This type of system is fault-tolerant.

Disadvantages of Distributed Operating System

- Protocol overhead can dominate computation cost.

6. Explain the dual mode and the timer mode operations of the operating system.

Dual Mode (Supervisor Mode and User Mode):

Many modern operating systems operate in dual mode, which involves two distinct privilege levels or modes: Supervisor Mode (also known as Kernel Mode) and User Mode. These modes provide a level of security and protection for the operating system and the applications running on it. The processor's mode bit or register determines which mode is currently active.

Supervisor Mode (Kernel Mode):

In this mode, the operating system has unrestricted access to the hardware and can execute privileged instructions.

It allows direct access to system resources and hardware, enabling the OS to perform critical operations like managing memory, scheduling processes, and handling interrupts.

Only the operating system's core components and authorized device drivers run in supervisor mode.

User Mode:

In user mode, applications and user-level processes run with restricted access to system resources.

User mode provides a level of isolation to prevent applications from interfering with critical system functions.

Privileged instructions that could potentially harm the system or compromise its stability are not allowed in user mode.

The transition between these modes is typically controlled by the operating system in response to certain events, such as system calls, interrupts, or exceptions. This dual-mode architecture enhances system security by preventing user applications from directly accessing or modifying critical system resources.

Timer Mode:

The Timer Mode in an operating system involves the use of a timer or clock to interrupt the processor at regular intervals. This timer interrupt is used for various purposes, including enforcing time-sharing policies, implementing preemptive multitasking, and ensuring system responsiveness. Here's how Timer Mode works:

Preemptive Multitasking:

The timer interrupt allows the operating system to regain control of the CPU periodically, even if a running process has not completed its execution.

This helps in achieving preemptive multitasking, where the OS can switch between multiple tasks to give the appearance of simultaneous execution.

Time-Sharing:

The timer interrupt is used to allocate time slices (quantum) to different processes in a time-sharing system.

Each process runs for a predefined time slice, and when the timer interrupt occurs, the operating system can switch to another process.

System Maintenance and Monitoring:

The timer is used for system maintenance tasks, such as updating system clocks and counters.

It is also employed for performance monitoring and profiling.

Interrupt Handling:

The timer interrupt is one of many types of interrupts that an operating system handles.

When the timer interrupt occurs, the CPU transitions to supervisor mode, and the operating system's interrupt handler is invoked to perform necessary actions.

By using a timer interrupt, the operating system can maintain control over the system's execution, enforce fairness in resource allocation, and ensure that no single process monopolizes

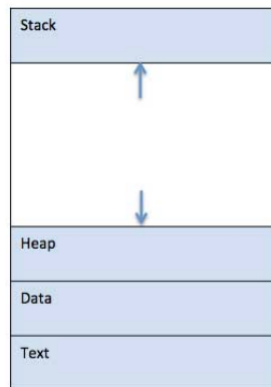
the CPU for an extended period. This mechanism is essential for creating responsive and efficient multitasking environments.

PART IV

7. Explain process states with a neat diagram.

Process : A process is a program in execution. A process is more than the program code, which is sometimes known as the **text section**. It also includes the current activity, as represented by the value of the **program counter** and the contents of the processor's registers. A process generally also includes the process **stack**, which contains temporary data (such as function parameters, return addresses, and local variables), and a **data section**, which contains global variables. A process may also include a **heap**, which is memory that is dynamically allocated during process run time.

Structure of a process



We emphasize that a program by itself is not a process; a program is a *passive* entity, such as a file containing a list of instructions stored on disk (often called an **executable file**), whereas a process is an *active* entity, with a program counter specifying the next instruction to execute and a set of associated resources. A program becomes a process when an executable file is loaded into memory.

Two common techniques for loading executable files are double-clicking an icon representing the executable file and entering the name of the executable file on the command line (as in prog. exe or a. out.)

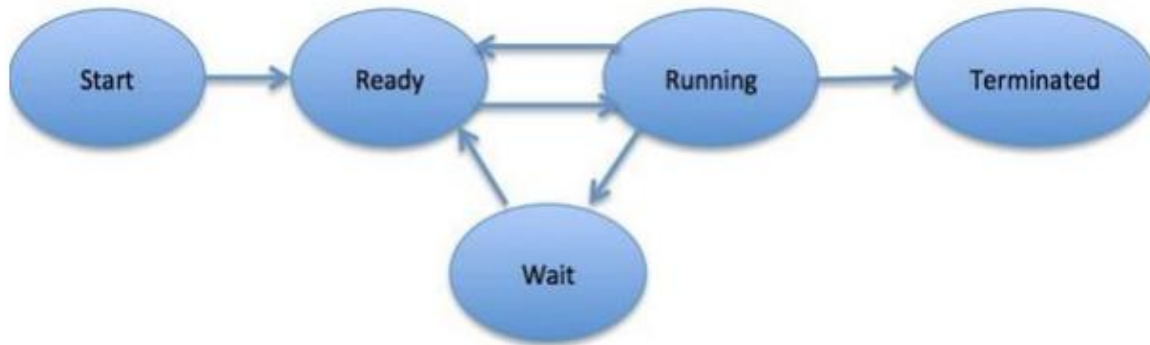
Process State

As a process executes, it changes **state**. The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:

- **New.** The process is being created.
- **Running.** Instructions are being executed.
- **Waiting.** The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- **Ready.** The process is waiting to be assigned to a processor.

- **Terminated.** The process has finished execution.

These names are arbitrary, and they vary across operating systems. The states that they represent are for on all systems, however. Certain operating systems also more finely delineate process states. It is important to realize that only one process can be *running* on any processor at any instant.

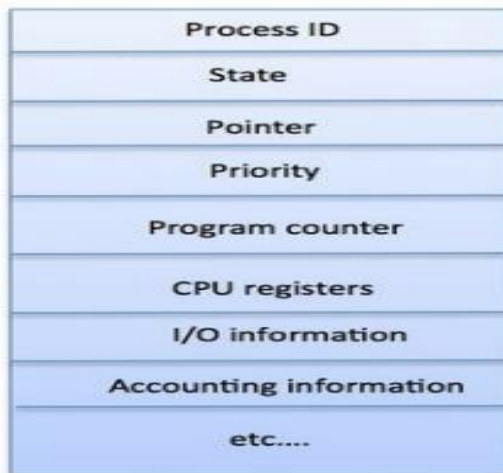


Process State Diagram

Process Control Block

Each process is represented in the operating system by a **process control block (PCB)**—also called a *task control block*.

Process state. The state may be new, ready, running, and waiting, halted, and so on.



Process Control Block (PCB) Diagram

Program counter-The counter indicates the address of the next instruction to be executed for this process.

- **CPU registers-** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information.

CPU-scheduling information- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory-management information- This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system

Accounting information-This information includes the amount of CPU and real time used, time limits, account members, job or process numbers, and so on.

I/O status information-This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

8. Explain Inter process Communication.

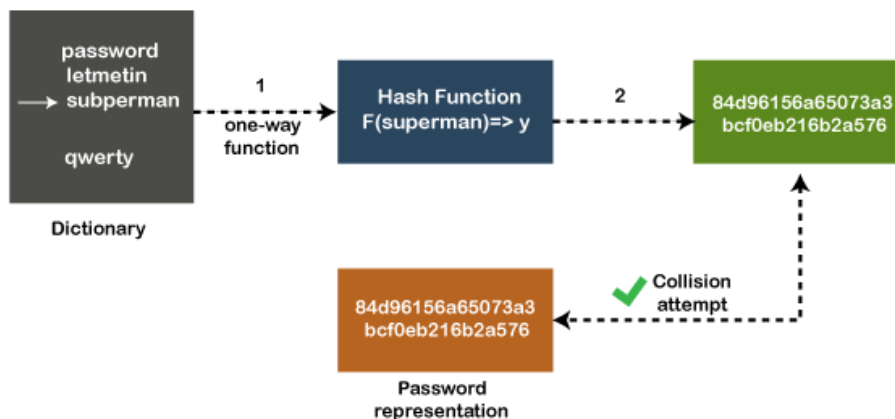
In general, Inter Process Communication is a type of mechanism usually provided by the operating system (or OS). The main aim or goal of this mechanism is to provide communications in between several processes. In short, the intercommunication allows a process letting another process know that some event has occurred.

Let us now look at the general definition of inter-process communication, which will explain the same thing that we have discussed above.

Definition

"Inter-process communication is used for exchanging useful information between numerous threads in one or more processes (or programs)."

To understand inter process communication, you can consider the following given diagram that illustrates the importance of inter-process communication:



Role of Synchronization in Inter Process Communication

It is one of the essential parts of inter process communication. Typically, this is provided by interprocess communication control mechanisms, but sometimes it can also be controlled by communication processes.

These are the following methods that used to provide the synchronization:

1. **Mutual Exclusion**
2. **Semaphore**
3. **Barrier**
4. **Spinlock**

Mutual Exclusion:-

It is generally required that only one process thread can enter the critical section at a time. This also helps in synchronization and creates a stable state to avoid the race condition.

Semaphore:-

Semaphore is a type of variable that usually controls the access to the shared resources by several processes. Semaphore is further divided into two types which are as follows:

1. Binary Semaphore
2. Counting Semaphore

Barrier:-

A barrier typically not allows an individual process to proceed unless all the processes does not reach it. It is used by many parallel languages, and collective routines impose barriers.

Spinlock:-

Spinlock is a type of lock as its name implies. The processes are trying to acquire the spinlock waits or stays in a loop while checking that the lock is available or not. It is known as busy waiting because even though the process active, the process does not perform any functional operation (or task).

Approaches to Interprocess Communication

We will now discuss some different approaches to inter-process communication which are as follows:



These are a few different approaches for Inter- Process Communication:

1. **Pipes**
2. **Shared Memory**
3. **Message Queue**
4. **Direct Communication**
5. **Indirect communication**
6. **Message Passing**
7. **FIFO**

To understand them in more detail, we will discuss each of them individually.

Pipe:-

The pipe is a type of data channel that is unidirectional in nature. It means that the data in this type of data channel can be moved in only a single direction at a time. Still, one can use two-channel of this type, so that he can able to send and receive data in two processes. Typically, it uses the standard methods for input and output. These pipes are used in all types of POSIX systems and in different versions of window operating systems as well.

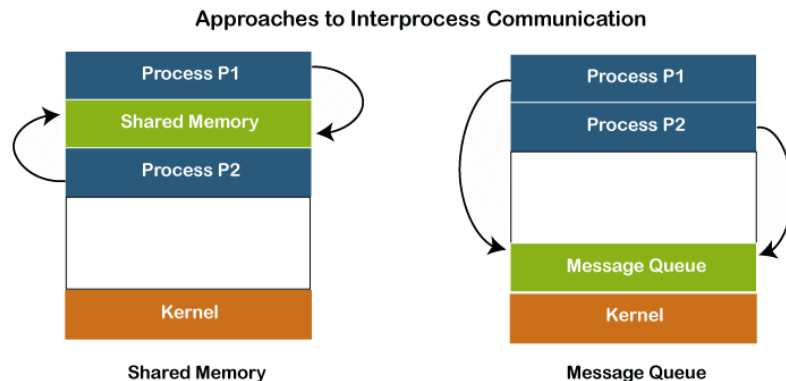
Shared Memory:-

It can be referred to as a type of memory that can be used or accessed by multiple processes simultaneously. It is primarily used so that the processes can communicate with each other. Therefore the shared memory is used by almost all POSIX and Windows operating systems as well.

Message Queue:-

In general, several different messages are allowed to read and write the data to the message queue. In the message queue, the messages are stored or stay in the queue unless their recipients retrieve them. In short, we can also say that the message queue is very helpful in inter-process communication and used by all operating systems.

To understand the concept of Message queue and Shared memory in more detail, let's take a look at its diagram given below:



Message Passing:-

It is a type of mechanism that allows processes to synchronize and communicate with each other. However, by using the message passing, the processes can communicate with each other without restoring the shared variables.

Usually, the inter-process communication mechanism provides two operations that are as follows:

- send (message)
- received (message)

Direct Communication:-

In this type of communication process, usually, a link is created or established between two communicating processes. However, in every pair of communicating processes, only one link can exist.

Indirect Communication

Indirect communication can only exist or be established when processes share a common mailbox, and each pair of these processes shares multiple communication links. These shared links can be unidirectional or bi-directional.

FIFO:-

It is a type of general communication between two unrelated processes. It can also be considered as full-duplex, which means that one process can communicate with another process and vice versa.

Some other different approaches

- **Socket:-**

It acts as a type of endpoint for receiving or sending the data in a network. It is correct for data sent between processes on the same computer or data sent between different computers on the same network. Hence, it is used by several types of operating systems.

- **File:-**

A file is a type of data record or a document stored on the disk and can be acquired on demand by the file server. Another most important thing is that several processes can access that file as required or needed.

- **Signal:-**

As its name implies, they are a type of signal used in inter process communication in a minimal way. Typically, they are the messages of systems that are sent by one process to another. Therefore, they are not used for sending data but for remote commands between multiple processes.

Usually, they are not used to send the data but to remote commands in between several processes.

Why we need interprocess communication?

There are numerous reasons to use inter-process communication for sharing the data. Here are some of the most important reasons that are given below:

- It helps to speedup modularity
- Computational
- Privilege separation
- Convenience
- Helps operating system to communicate with each other and synchronize their actions as well.

PART V

9. Explain FCFS (non preemptive) and SJF (preemptive) scheduling with an example.

First Come First Serve

First Come First Serve CPU Scheduling Algorithm shortly known as FCFS is the first algorithm of CPU Process Scheduling Algorithm. In First Come First Serve Algorithm what we do is to allow the process to execute in linear manner.

This means that whichever process enters process enters the ready queue first is executed first. This shows that First Come First Serve Algorithm follows First In First Out (FIFO) principle.

The First Come First Serve Algorithm can be executed in Pre Emptive and Non Pre Emptive manner. Before, going into examples, let us understand what is Pre Emptive and Non Pre Emptive Approach in CPU Process Scheduling.

Pre Emptive Approach

In this instance of Pre Emptive Process Scheduling, the OS allots the resources to a Process for a predetermined period of time. The process transitions from running state to ready state or from waiting state to ready state during resource allocation. This switching happens because the CPU may assign other processes precedence and substitute the currently active process for the higher priority process.

Non Pre Emptive Approach

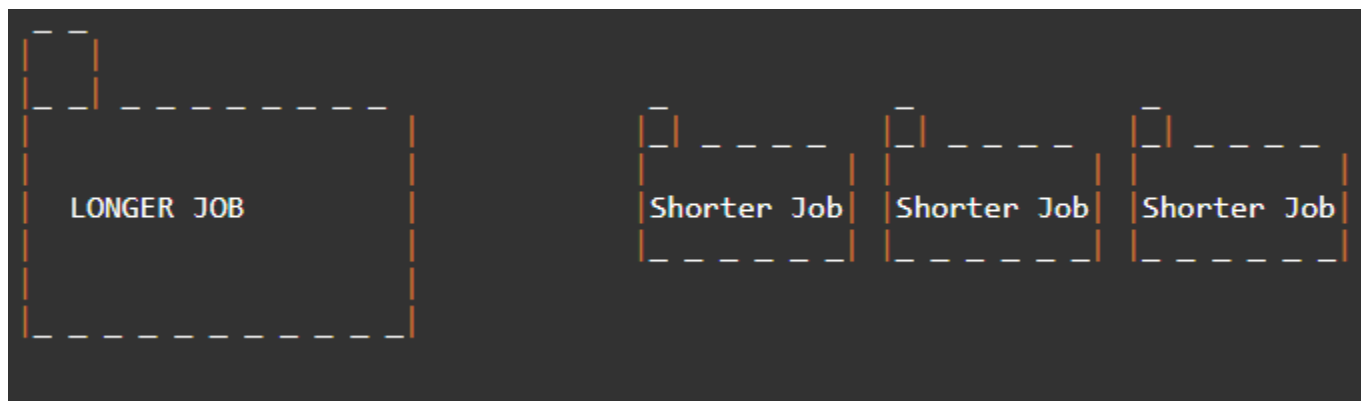
In this case of Non Pre Emptive Process Scheduling, the resource cannot be withdrawn from a process before the process has finished running. When a running process finishes and transitions to the waiting state, resources are switched.

Convoy Effect In First Come First Serve (FCFS)

Convoy Effect is a phenomenon which occurs in the Scheduling Algorithm named First Come First Serve (FCFS). The First Come First Serve Scheduling Algorithm occurs in a way of non preemptive way.

The Non preemptive way means that if a process or job is started execution, then the operating system must complete its process or job. Until, the process or job is zero the new or next process or job does not start its execution. The definition of Non Preemptive Scheduling in terms of Operating System means that the Central Processing Unit (CPU) will be completely dedicated till the end of the process or job started first and the new process or job is executed only after finishing of the older process or job.

There may be a few cases, which might cause the Central Processing Unit (CPU) to allot a too much time. This is because in the First Come First Serve Scheduling Algorithm Non Preemptive Approach, the Processes or the jobs are chosen in serial order. Due, to this shorter jobs or processes behind the larger processes or jobs takes too much time to complete its execution. Due, to this the Waiting Time, Turn Around Time, Completion Time is very high.



So, here as the first process is large or completion time is too high, then this Convoy effect in the First Come First Serve Algorithm is occurred.

Let us assume that Longer Job takes infinite time to complete. Then, the remaining processes have to wait for the same infinite time. Due to this Convoy Effect created by the Longer Job the Starvation of the waiting processes increases very rapidly. This is the biggest disadvantage of FCFS CPU Process Scheduling.

Characteristics of FCFS CPU Process Scheduling

The characteristics of FCFS CPU Process Scheduling are:

1. Implementation is simple.
2. Does not cause any causalities while using
3. It adopts a non preemptive and preemptive strategy.
4. It runs each procedure in the order that they are received.
5. Arrival time is used as a selection criterion for procedures.

Advantages of FCFS CPU Process Scheduling

The advantages of FCFS CPU Process Scheduling are:

1. In order to allocate processes, it uses the First In First Out queue.
2. The FCFS CPU Scheduling Process is straight forward and easy to implement.
3. In the FCFS situation pre-emptive scheduling, there is no chance of process starving.
4. As there is no consideration of process priority, it is an equitable algorithm.

Disadvantages of FCFS CPU Process Scheduling

The disadvantages of FCFS CPU Process Scheduling are:

- FCFS CPU Scheduling Algorithm has Long Waiting Time
- FCFS CPU Scheduling favors CPU over Input or Output operations
- In FCFS there is a chance of occurrence of Convoy Effect
- Because FCFS is so straight forward, it often isn't very effective. Extended waiting periods go hand in hand with this. All other orders are left idle if the CPU is busy processing one time-consuming order.

Problems in the First Come First Serve CPU Scheduling Algorithm

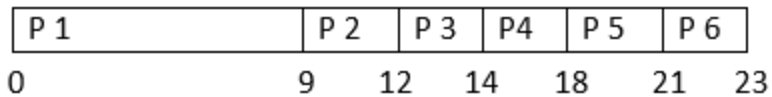
Example

1. S. No	Process ID	Process Name	Arrival Time	Burst Time
2. ---	-----	-----	-----	-----
3. 1	P 1	A	0	9
4. 2	P 2	B	1	3
5. 3	P 3	C	1	2
6. 4	P 4	D	1	4
7. 5	P 5	E	2	3
8. 6	P 6	F	3	2

Non Pre-emptive Approach

Now, let us solve this problem with the help of the Scheduling Algorithm named First Come First Serve in a Non-Preemptive Approach.

Gantt chart for the above Example 1 is:



Turn Around Time = Completion Time - Arrival Time

Waiting Time = Turn Around Time - Burst Time

Solution to the Above Question Example 1

S. No	Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	
1	P1	A	0	9	9	9	0
2	P2	B	1	3	12	11	8
3	P3	C	1	2	14	13	11
4	P4	D	1	4	18	17	13
5	P5	E	2	3	21	19	16
6	P6	F	3	2	23	20	18

The Average Completion Time is:

$$\text{Average CT} = (9 + 12 + 14 + 18 + 21 + 23) / 6$$

$$\text{Average CT} = 97 / 6$$

$$\text{Average CT} = 16.16667$$

The Average Waiting Time is:

$$\text{Average WT} = (0 + 8 + 11 + 13 + 16 + 18) / 6$$

$$\text{Average WT} = 66 / 6$$

$$\text{Average WT} = 11$$

The Average Turn Around Time is:

Average TAT = $(9 + 11 + 13 + 17 + 19 + 20) / 6$

Average TAT = $89 / 6$

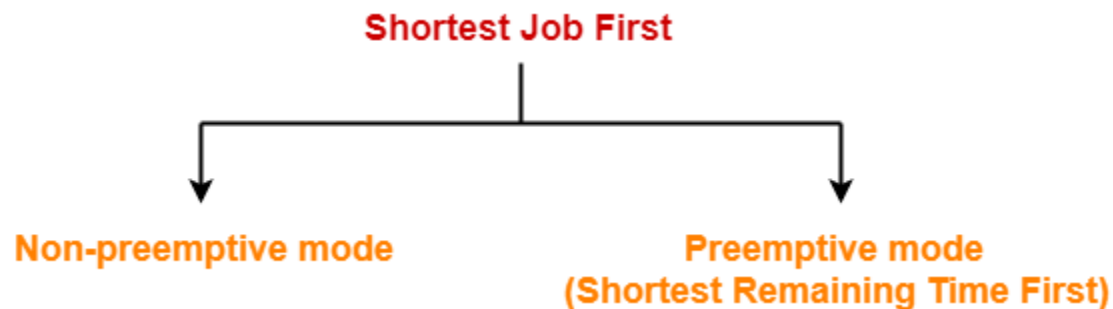
Average TAT = 14.83334

This is how the FCFS is solved in Non Pre Emptive Approach.

Shortest Job First

In SJF Scheduling,

- Out of all the available processes, CPU is assigned to the process having smallest burst time.
- In case of a tie, it is broken by **FCFS Scheduling**.
- SJF Scheduling can be used in both preemptive and non-preemptive mode.
- Preemptive mode of Shortest Job First is called as **Shortest Remaining Time First (SRTF)**.



Advantages-

- SRTF is optimal and guarantees the minimum average waiting time.
- It provides a standard for other algorithms since no other algorithm performs better than it.

Disadvantages-

- It can not be implemented practically since burst time of the processes can not be known in advance.
- It leads to starvation for processes with larger burst time.
- Priorities cannot be set for the processes.

Processes with larger burst time have poor response time.

Problem-01:

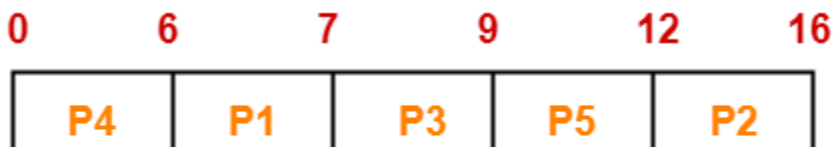
Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

Ⓜ
If the CPU scheduling policy is SJF non-preemptive, calculate the average waiting time and average turn around time.

Solution-

Gantt Chart-



Gantt Chart

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	7	$7 - 3 = 4$	$4 - 1 = 3$
P2	16	$16 - 1 = 15$	$15 - 4 = 11$
P3	9	$9 - 4 = 5$	$5 - 2 = 3$
P4	6	$6 - 0 = 6$	$6 - 6 = 0$
P5	12	$12 - 2 = 10$	$10 - 3 = 7$



Now,

- Average Turn Around time = $(4 + 15 + 5 + 6 + 10) / 5 = 40 / 5 = 8$ unit
- Average waiting time = $(3 + 11 + 3 + 0 + 7) / 5 = 24 / 5 = 4.8$ unit

10. Calculate the average waiting time, turn around time for i)SJF ii)Priority Scheduling and iii)Round Robin($t_q=2ms$) with the following set of process.

Process	P1	P2	P3	P4	P5
Burst Time	10	1	2	1	5
Priority	3	1	3	4	5

Solution:

i) Shortest Job First (SJF):

1. Arrange the processes in ascending order based on burst time.

Process order: P2, P4, P3, P5, P1

2. Calculate waiting time and turnaround time.

Process	Burst Time	Waiting Time	Turnaround Time
P2	1	0	1
P4	1	1	2
P3	2	2	4
P5	5	4	9
P1	10	9	19

3. Calculate the average waiting time and average turnaround time.

Average Waiting Time = $(0 + 1 + 2 + 4 + 9) / 5 = 16 / 5 = 3.2$ ms

Average Turnaround Time = $(1 + 2 + 4 + 9 + 19) / 5 = 35 / 5 = 7$ ms

ii) Priority Scheduling:

1. Arrange the processes in ascending order based on priority.

Process order: P2, P3, P1, P4, P5

2. Calculate waiting time and turnaround time.

Process	Priority	Waiting Time	Turnaround Time
P2	1	0	1
P3	3	1	3
P1	3	3	13
P4	4	13	14
P5	5	14	19

3. Calculate the average waiting time and average turnaround time.

Average Waiting Time = $(0 + 1 + 3 + 13 + 14) / 5 = 31 / 5 = 6.2$ ms

Average Turnaround Time = $(1 + 3 + 13 + 14 + 19) / 5 = 50 / 5 = 10$ ms

ii) Round Robin (with time quantum = 2 ms):

1) Execute processes in a round-robin manner with a time quantum of 2 ms.

Process	Time
P1	2
P2	3
P3	5
P4	6
P5	8
P1	10
P5	12
P1	14
P5	15
P1	19

2) Calculate waiting time and turnaround time.

Process	Waiting Time	Turn Around Time
P1	9	19
P2	2	3
P3	3	5
P4	5	6
P5	10	13

3) Calculate the average waiting time and average turnaround time.

$$\text{Average Waiting Time} = (9 + 2 + 3 + 5 + 10) / 5 = 48 / 5 = 9.6 \text{ ms}$$

$$\text{Average Turnaround Time} = (19 + 3 + 5 + 6 + 13) / 5 = 46 / 5 = 9.2 \text{ ms}$$