UNIT-5

REAL TIME OPERATING SYSTEM

1. Operating System basics :-

* The operating system acts as a bridge b/w the user applications/ tasks and the underlying system resources through a set of system functionalities and services.
* The OS manages the systems resources and makes them available to the user applications/ tasks on a need basis.
* The primary functions of an operating systems is
  - → Make the systems convenient to use.
  - → Organise and manage the system resources efficiently and correctly.
* The figure shows the basic components of an operating system and their interfaces with rest of the world.

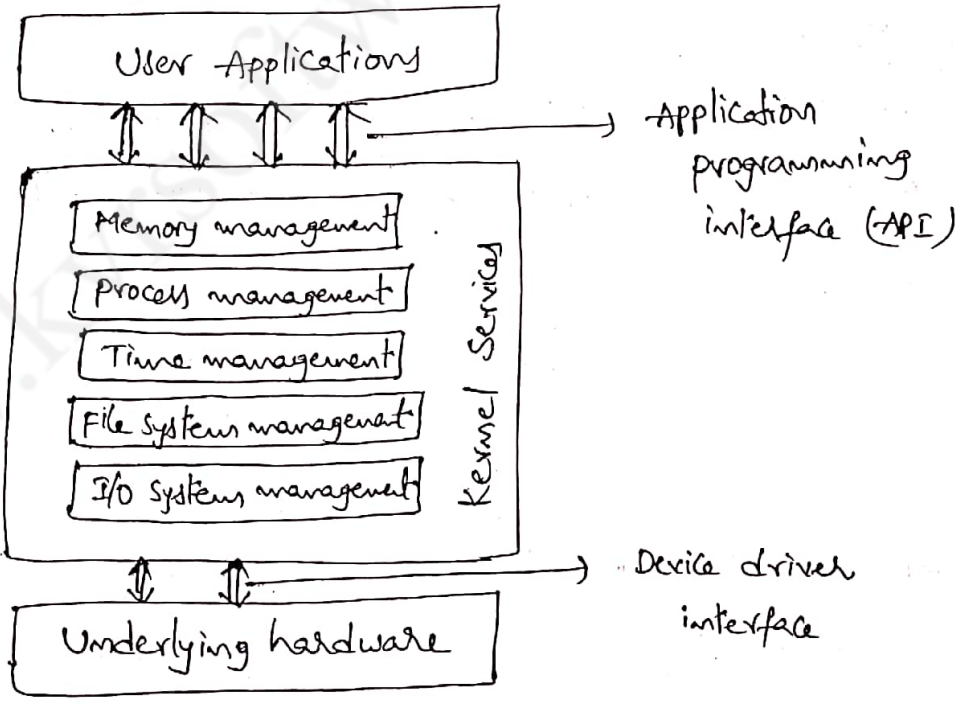


Fig:- The operating systems Architecture.

The Kernel :-

* The Kernel is the core of the operating system and is responsible for managing the system resources and the communication among the hardware and other system services.
* Kernel acts as the abstraction layer b/w system resources and user applicatio
* Kernel Contains a set of system libraries and services.
* For a general purpose OS, the Kernel Contains different services for handling the following.

Process Management :-

* process management deals with managing the process/tasks.
* It includes setting up the memory space for the process, loading the process' Code into the memory space, allocating system resources, scheduling and managing the execution of the process, setting up and managing the process control block(PCB), process termination/deletion, etc.

Memory Management :-

Primary Memory Management : The terms primary memory refers to the volatile - memory (RAM). The Memory management unit (MMU) of the kernel is responsible for

* Keeping track of which part of the memory area is currently used by which process.
* Allocating and De-allocating memory space on a need basis (Dynamic memory allocation).

Secondary storage Management : Secondary memory is used as backup medium for programs and data since the main memory is volatile. The secondary storage management service of kernel deals with

* Disk storage allocation
* Disk scheduling (Time interval at which the disk is activated to back up data).
* Free Disk space management

File System Management :-

* File is a collection of related information.
* A File could be a program (source code or executable), text files, image files, word documents, audio/ video files, etc.
* This service of kernel is responsible for
  - → The creation, deletion and alteration of files.
  - → Creation, deletion and alteration of directories.
  - → Saving of files in the secondary storage memory (e.g: Hard disk storage)
  - → Providing a flexible naming convention for the files.

I/O System (Device) Management :-

* Kernel is responsible for routing the I/O requests coming from different User applications to the appropriate I/O device of the system.
* The kernel maintains a list of all the I/O devices of the system.
* Some kernels, dynamically update the list of available devices as and when a new device is installed.
* The service 'Device Manager' of the kernel is responsible for handling all I/O device related operations.
* The Device Manager is responsible for
  - → Loading and unloading of device drivers.
  - → Exchanging information and the system specific control signals to and from the device.

protection Systems :-

* Most of the modern operating systems are designed in such a way to support multiple users with different levels of access permissions.
  (eg: Windows xp with user permissions like 'Administrator', 'Standard', 'Restricted', etc).

1.1 Kernel space and User space :-

* The program code corresponding to the kernel applications/ services are kept in a contiguous area (os dependant) of primary (working) memory and is protected from un-authorised access by user programs/ applications.
* The memory space at which the kernel code is located is known as 'kernel space'.
* All user applications are loaded to a specific area of primary memory (Main memory) and this memory area is referred as 'User space'.
* User space is the memory area where user applications are loaded and executed.

1.2 Monolithic Kernel and Microkernel :-

* Based on the kernel design, kernels can be classified into 'Monolithic' & 'Micro'.

Monolithic Kernel :-

* In monolithic kernel architecture, all kernel services run in the kernel space
* Here all kernel modules run within the same memory space under a single kernel thread.
* The major drawback of monolithic kernel is that any error or failure in any one of the kernel modules leads to the crashing of the entire kernel application.

Ex:- LINUX, SOLARIS, MS-DOS kernels.

* The architecture of monolithic kernel is

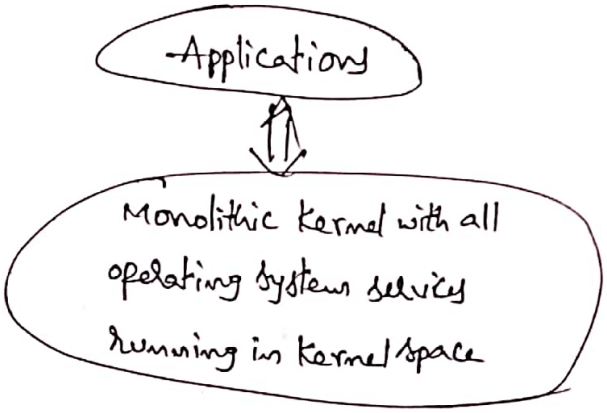


Fig:- The Monolithic Kernel Model.

Microkernel :-

* The microkernel design incorporates only the essential set of operating - system services into the kernel.
* The rest of the operating system services are implemented in programs known as 'servers' which runs in user space.
* Memory management, process management, timer systems and interrupt handlers are the essential services, which forms the part of the microkernel.
* Examples : Mach, QNX, Minix 3 kernels.
* The architecture of microkernel is shown in fig.

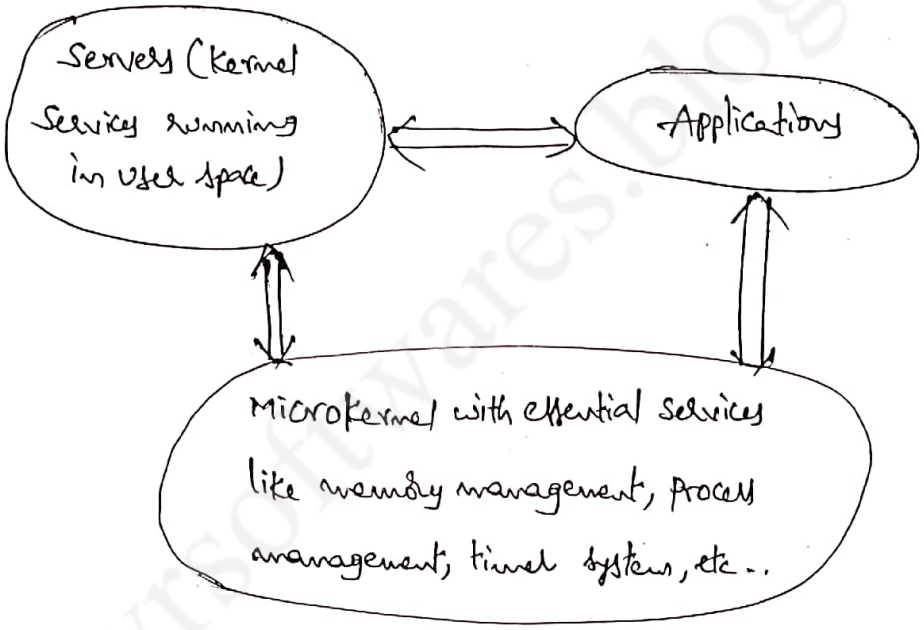


Fig:- The Microkernel model.

* Microkernel based design approach offers the following benfits.

→ Robustness : If a problem is in the services, which runs as 'server' application the same can be reconfigured and re-started without restart the entire OS.

→ Configurability : Any services, which runs as 'server' application can be changed without the need to restart the whole system. This makes the system dynamically configurable.

2. Types of operating systems :-

* Depending on the type of Kernel and Kernel services, purpose and type of Computing systems where the OS is deployed and the responsiveness to applications, operating systems are classified into different types.

(i) General purpose operating systems (GPOS) :-

* The operating systems, which are deployed in general computing systems, are referred as General purpose operating systems (GPOS).
* The Kernel of such an OS is more generalised and it contains all kinds of services required for executing generic applications.
* Their services can inject random delays into application software and may cause slow responsiveness of an application at unexpected times.
* Windows xp/MS-DOS etc are examply of GPOS.

(ii) Real-Time operating system (RTOS) :-

* 'Real-Time' implies deterministic timing behaviour. Deterministic timing behaviour in RTOS context means the OS services consumes only known and expected amounts of time regardless the number of services.
* A Real-Time operating system or RTOS implements policies and rules Concern-ing time-critical allocation of a systems resources.
* The RTOS decides which applications should run in which order and how much time needs to be allocated for each application.
* Windows CE, QNX, VxWorks, Micro C/ OS-II, etc. are examples of Real-Time operating Systems (RTOS).

a) The Real-Time Kernel :-

* The Real-Time Kernel is highly specialised and it contains only the minimum set of services required for running the user applications/ tasks.

* The basic functions of a Real-Time Kernel are listed below:

→ Task/ process management
→ Task/ process Scheduling
→ Task/ process Synchronisation
→ Error/ Exception handling
→ Memory management
→ Interrupt handling
→ Time management.

Task/ process management :- Deals with setting up the memory space for the tasks, loading the task's code into the memory space, allocating system resources, setting up a Task Control Block (TCB) for the task and task/ process termination/ deletion.

* TCB usually contains the following set of information.

TaskID: Task Identification Number.
Task State: The current state of the task
Task Type: Indicates what is the type for this task. The task can be a hard real time or soft real time or background task.
Task priority: Task priority
Task Context pointer: Context pointer. Pointer for context saving.
Task Memory pointer: pointers to the Code memory, data memory and stack memory for the task.
Task System Resource pointers: pointers to system resources used by the task.
Task pointers: pointers to other TCBs (TCB's for preceding, next and waiting tasks)

* Task management service utilises the TCB of a task in the following way

→ Creates a TCB → Reads the TCB → Modify the TCB.
→ Delete/ Remove the TCB → Update the TCB

Task/process Scheduling:- Deals with sharing the CPU among various tasks/process.

* Scheduler is nothing but an algorithm implementation, which performs the efficient and optimal scheduling of tasks to provide a deterministic behaviour.

Task/process Synchronisation:- Deals with synchronising the concurrent access of a resource, which is shared across multiple tasks and the communication between various tasks.

Error/Exception Handling:- Deals with registering and handling the errors occurred/ exceptions raised during the execution of tasks.

* Insufficient memory, timeouts, deadlocks, deadline missing, bus error, divide by zero, unknown instruction execution, etc.. are examples of errors/exceptions

Memory Management:- The memory allocation time increases depending on the size of the block of memory needs to be allocated and the state of the allocated memory block. RTOS kernel uses blocks of fixed size of dynamic memory and the block is allocated for a task on a need basis. The blocks are stored in a 'Free Buffer Queue'.

Interrupt Handling:- Interrupts informs the processor that an external device or an associated task requires immediate attention of the CPU.

Interrupts can be either Synchronous or Asynchronous.

* Interrupts which occurs in sync with the currently executing task is known as Synchronous interrupt. Usually the software interrupts fall under the Synchronous.

* Asynchronous interrupts are interrupts, which occurs at any point of execution of any task, and are not in sync with the currently executing task.

The interrupts generated by external devices connected to the processor/controller, timer overflow interrupts, serial data reception/transmission etc.. are examples for asynchronous interrupts.

Time Management :- Accurate time management is essential for providing precise time reference for all applications. The time reference to kernel is provided by a high-resolution Real-Time Clock (RTC) hardware chip. The hardware timer is programmed to interrupt the processor/controller at a fixed rate. This timer interrupt is referred as 'Timer tick'.

b) Hard Real-Time :-

* A Hard Real-Time System must meet the deadlines for a task without any slippage. Missing any deadline may produce catastrophic results for Hard Real-Time Systems, including permanent data loss and irrecoverable damages to the system/users.
* Hard Real-Time Systems emphasize the principle 'A late answer is a wrong answer'.
* Air bag control systems and Anti-lock Break Systems (ABS) of vehicles are typical examples for Hard Real-Time Systems.
* The Air bag Control System should be into action and deploy the air bags when the vehicle meets a severe accident. Any delay in the deployment of the air bags make the life of the passengers under threat.
* When the air bag deployment task is triggered, the currently executing task must be pre-empted, the air bag deployment task should be brought into execution, and the necessary I/O System should be made readily available for the air bag deployment task.
* Hard Real-Time Systems does not implement the virtual memory model for handling memory. This eliminates the delay in swapping in and out the code corresponding to the task to and from the primary memory.

c) Soft Real-Time :-

* Real-Time Operating System that does not guarantee meeting deadlines, but offer the best effort to meet the deadline are referred as 'Soft Real-Time' OS
* A Soft Real-Time System emphasises the principle 'a late answer is an acceptable answer, but it could have done bit faster'.
* Automatic Teller Machine (ATM) is a typical example for Soft Real-Time System.
* If the ATM takes a few seconds more than the ideal operation time, nothing fatal happens.
* An audio-video playback system is another example for Soft Real-Time Systems. No potential damage arises if a sample comes late by fraction of a second, for play

---

3. TASKS, PROCESS AND THREADS :-

* The terms 'task' refers to something that needs to be done.
* Task is also known as 'Job' in the operating system context.
* A program or part of it in execution is also called a 'process'.
* The terms 'task', 'Job' and 'process' refers to the same entity in the operating system context and most often they are used interchangably

a) PROCESS :-

* A 'process' is a program, or a part of it, in execution.
* process is also known as an instance of a program in execution.
* A process is sequential in execution.
* It requires CPU for executing the process, memory for storing the code, I/o devices for exchange informations, etc..

(i) The structure of a process :-

* The concept of 'process' leads to concurrent execution (pseudo parallelism) of tasks and thereby the efficient utilisation of the CPU and other systems resources

* The structure of a process is

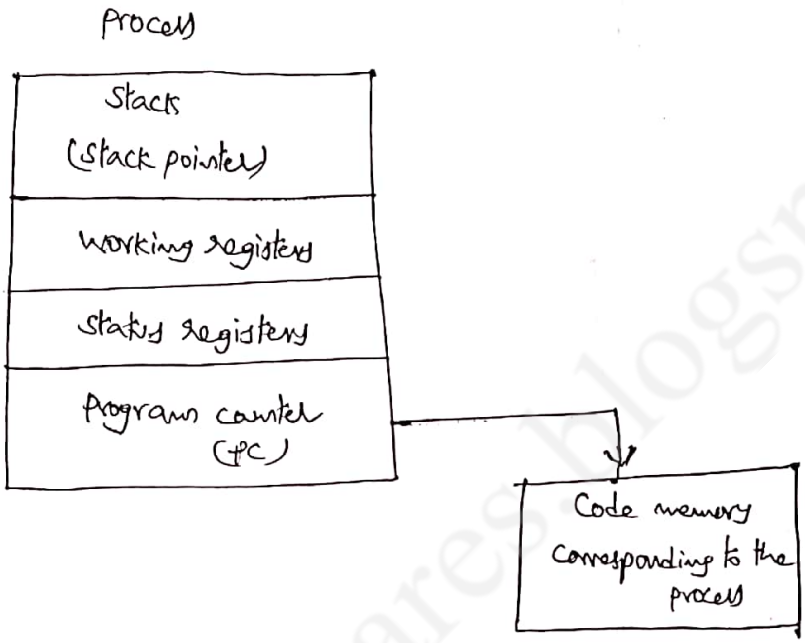


Fig:- structure of a process

* The memory occupied by the process is segregated into three regions, namely, Stack memory, Data memory and Code memory.

* The 'stack' memory holds all temporary data such as variables local to the process.

* Data memory holds all global data for the process.

* The Code memory contains the program code (instructions) corresponding to the process.

Process states and state Transition :-

* The creation of a process to its termination is not a single step operation. The process traverses through a series of states during its transition from the newly created state to the terminated state.

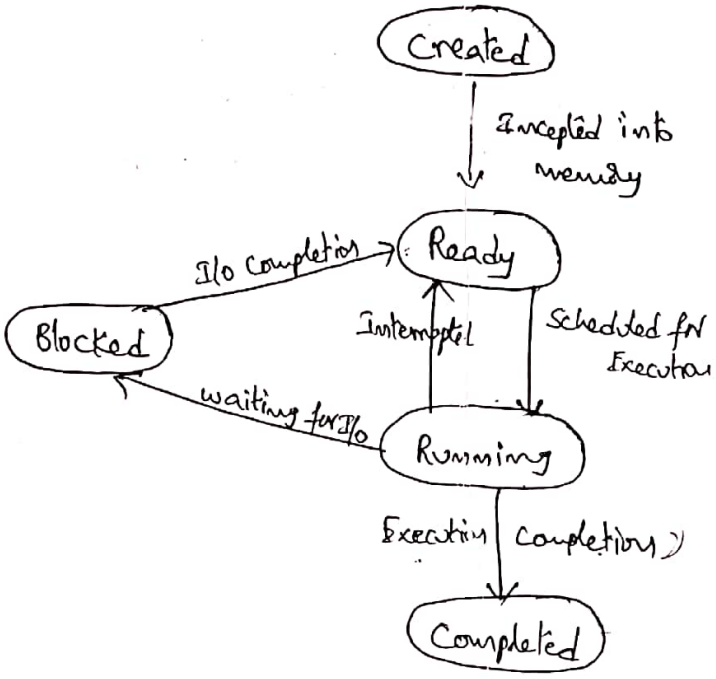


FIG:- process states and state transition representation.

* The state at which a process is being created is referred as 'created state'.
* The state, where a process is accepted into the memory and awaiting the processor time for execution, is known as 'Ready state'.
* The state where in the source code instructions corresponding to the process is being executed is called 'Running state'.
* 'Blocked state/wait state' refers to a state where a running process is temporarily suspended from execution and does not have immediate access to resources.
* A state where the process completes its execution is known as 'Completed state'.
* The transition of a process from one state to another is known as 'state transition'.

Threads :-

* A Thread is the primitive that can execute code.
* A Thread is a single sequential flow of control with in a process.
* Thread is also known as light weight process.
* A process can have many threads of execution.

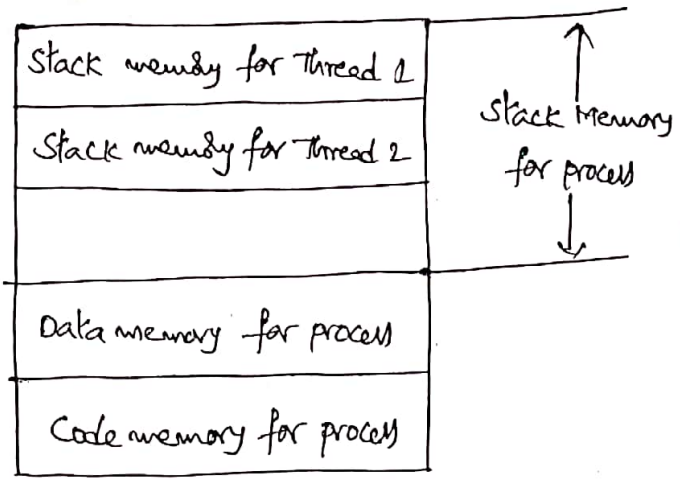


Fig:- Memory organisation of a process and its associated Threads.

⇒ The Concept of Multithreading :-

* A process/task in embedded application may all the subfunctions of a task are executed in sequence, the CPU utilization may not be efficient.
* Instead of this single sequential execution of the whole process, if the task/process is split into different threads carrying out the different subfunctionalities of the process, the CPU can be effectively utilised.
* When the thread corresponding to the I/o operation enters the wait state, another threads which do not require the I/o event for their operation can be switched into execution.

Advantages :-

* Better memory utilisation. Multiple threads of the same process share the address space for data memory.
* The speeds up the execution of the process.
* Efficient CPU utilisation. The CPU is engaged all time.

⇒ Thread Standards :- Thread standards deal with the different standards available for thread creation and management. It is a set of thread class libraries. The commonly available thread class libraries are explained below.

POSIX Threads :- POSIX stands for Portable Operating System Interface.

* 'Pthreads' library defines the set of POSIX thread creation and management functions in 'C' language.

Win 32 Threads :- Win32 threads are the threads supported by various flavours of windows operating systems. Win32 threads are created with the API (Application Programming Interface).

Java Threads :- Java threads are the threads supported by Java programming language. The java thread class 'Thread' is defined in the package "java.lan

⇒ Thread Pre-emption :- Thread pre-emption is the act of pre-empting the currently running thread (stopping the currently running thread temporarily).

* The execution switching among threads are known as 'Thread Context switching'
* When we say 'Thread', it falls into any one of the following types.

↳ User Level Thread :-

* User level threads do not have kernel/operating system support and they exist solely in the running process.

* Even if a process containing multiple user level threads, the OS treats it as single thread and will not switch the execution among the different threads of it.

↳ Kernel / System Level Thread :-

* Kernel level threads are individual units of execution, which the OS treats as separate threads.

* The OS interrupts the execution of the currently running kernel thread and switches the execution to another kernel thread based on the scheduling policies implemented by the OS.

* The following section gives an overview of various thread binding models.

↳ Many-to-one Model :- Many user level threads are mapped to a single Kernel thread, the Kernel treats all user level threads as single thread and the execution switching among the user level thread.

+ Solaris Green threads and GNU portable Threads are examples.

↳ One-to-one Model :- Each User level thread is bonded to a Kernel/System level thread. windows XP/NT/2000 and Linux threads are examples.

↳ Many-to-Many Model :- Many user level threads are allowed to be mapped to many Kernel threads. windows NT/2000 with Thread Fibre package is an example

⇒ Thread Vs Process :-

Thread	Process
(i) Thread is a single unit of execution and is part of process.	(i) process is a program in execution and contains one or more threads.
(ii) A Thread does not have its own data memory and heap memory. It shares the data memory and heap memory with other threads of the same process.	(ii) process has its own Code memory, data memory and stack memory.
(iii) A Thread cannot live independently; it lives with in the process	(iii) A process Contains at least one thread.
(iv) There can be multiple threads in a process	(iv) Each thread holds separate memory area for stack. Shares the total stack memory of the process
(v) Threads are inexpentive to create	(v) process are very expentive to create.

4. Multiprocessing And Multitasking :-

* In the operating system context multiprocessing describes the ability to execute multiple processes simultaneously.
* Multiprocessor systems posses multiple CPUs and can execute multiple processes simultaneously.
* The ability of an operating system to hold multiple processes in memory and switch the processor (CPU) from executing one process to another process is known as multitasking.
* Multitasking involves the switching of CPU from executing one task to another
* The act of switching CPU among the processes or changing the current execution context is known as 'Context switching'.

Types of Multitasking :-

Multitasking involves the switching of execution among multiple tasks. Depending on how the switching act is implemented, multitasking can be classified into different types.

a) Co-operative Multitasking :- Co-operative multitasking is the most primitive form of multitasking in which task/process gets a chance to execute only when the currently executing task/process voluntarily relinquishes the CPU.
* Any task/process can hold the CPU as much time as it wants.
* This type of implementation involves the mercy of the tasks each other for getting the CPU time for execution, it is known as Co-operative multitasking

b) Preemptive Multitasking :- Preemptive multitasking ensures that every task/process gets a chance to execute.
* The preemptive of task may be based on time slots or task/process priority.

c) Non-preemptive Multitasking:- In non-preemptive multitasking, the process/task, which is currently given the CPU time, is allowed to execute until it terminates or enters the 'Blocked/wait' state, waiting for an I/O or system resource.

* In co-operative multitasking, the currently executing process/task need not relinquish the CPU when it enters the 'Blocked/wait' state, waiting for an I/O, or a shared resources access or an event occur.
* In Non-preemptive multitasking the currently executing task relinquishes the CPU when it waits for an I/O or system resource or an event to occur

5. TASK SCHEDULING:-

* Determining which task/process is to be executed at a given point of time is known as task/process scheduling.
* The kernel service/application, which implements the scheduling algorithm, is known as 'scheduler'.
* The process scheduling decision may take place when a process switches its state to
  (i) 'Ready' state from 'Running' state.
  (ii) 'Blocked/wait' state from 'Running' state
  (iii) 'Ready' state from 'Blocked/wait' state.
  (iv) 'completed' state.

The selection of a scheduling criterion/algorithm should consider the following factors:

CPU utilization:- The scheduling algorithm should always make the CPU utilization high.

Throughput :- This gives an indication of the No of process executed per unit of time

Turnaround Time :- It is the amount of time taken by a process for completing its execution. The turnaround time should be a minimal for a good scheduling

Waiting Time :- It is the amount of time spent by a process in the 'Ready' queue waiting to get the CPU time for execution. It is minimal for a good scheduling

Response Time :- It is the time elapsed b/w the submission of a process and the first response, the response time should be as least as possible

The various queues maintained by OS in association with CPU scheduling are:

Job Queue :- Job queue contains all the processes in the system.

Ready Queue :- which are ready for execution and waiting for CPU to get their turn for execution

Device Queue :- Containing the set of processes, which are waiting for an I/O device

Based on the scheduling algorithms used, the scheduling can be classified into the following categories.

a) Non-preemptive scheduling
   i) First-come-First served (FCFS)/ FIFO scheduling.
   ii) Last-come-First served (LCFS)/ LIFO scheduling.
   iii) Shortest Job First (SJF) scheduling
   iv) priority Based scheduling

b) preemptive scheduling
   i) preemptive SJF scheduling/ Shortest Remaining Time (SRT)
   ii) Round Robin (RR) scheduling.
   iii) priority Based scheduling.

a) Non-preemptive Scheduling :-

* In this scheduling type, the currently executing task/process is allowed to run until it terminates or enters the 'wait' state waiting for an I/O or system resources.
* The various types of non-preemptive scheduling adopted in task/process scheduling are listed below.

(i) First-come-First-Served (FCFS)/FIFO Scheduling :-

As the name indicates, the First-come-First-Served (FCFS) scheduling algorithms allocates CPU time to the processes based on the order in which they enter the 'Ready' queue. The first entered process is serviced first.

Eg:- Ticketing reservation system.

(ii) Last-come-First served (LCFS)/LIFO scheduling :-

The Last-come-First served (LCFS) scheduling algorithm also allocates CPU time to the processes based on the order in which they are entered in the 'Ready' queue. The Last entered process is serviced first. LCFS scheduling is also known as Last In First out (LIFO).

(iii) Shortest Job First (SJF) Scheduling :-

Shortest Job First (SJF) scheduling algorithm 'sorts the 'Ready' queue' each time a process relinquishes the CPU to pick the process with shortest (least) eliminated Completion/run time. In SJF, the process with the shortest estimated run time is scheduled first, followed by the next shortest process, and so on.

(iv) priority Based Scheduling :-

* The term Around Time (TAT) and waiting time for processes in non-preemptive scheduling varies with the type of scheduling algorithm.
* priority based non-preemptive scheduling algorithm ensures that a process with high priority is serviced at the earlist compared to other low priority processes in the 'Read..' Queue

b) Preemptive Scheduling :-

In preemptive scheduling, every task in the 'Ready' queue gets a chance to execute. When and how often each process gets a chance to execute is dependent on the type of preemptive scheduling algorithm used for scheduling the processes.

The various types of preemptive scheduling adopted in task/process scheduling are explained below.

(a) Preemptive SJF Scheduling/Shortest Remaining Time (SRT) :-

The non-preemptive SJF scheduling algorithm sorts the 'Ready' queue only after completing the execution of the current process or when the process enters 'wait' state, whereas the preemptive SJF scheduling algorithm sorts the 'Ready' queue when a new process enters the 'Ready' queue and checks whether the execution time of the new process is shorter than the remaining of the total estimated time for the currently ~~time~~ executing process.

(b) Round Robin (RR) Scheduling :-

In this process scheduling, each process in the 'Ready' queue is executed for a pre-defined time slot. The execution starts with picking up the first process in the 'Ready' queue. It is executed for a pre-defined time and when the pre-defined time elapses or the process completes, the next process in the 'Ready' queue is selected for execution. This is repeated for all the processes in the 'Ready' queue.

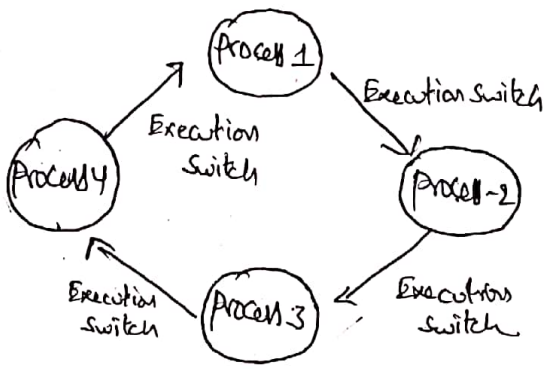


Fig :- Round Robin Scheduling.

(iii) Priority Based Scheduling :- In preemptive scheduling, an high priority process entering the 'Ready' queue is immediately scheduling for execution whereas in the non-preemptive scheduling any high priority process entering the 'Ready' queue is scheduled only after the currently executing process completes its execution or only when it voluntarily relinquishes the CPU.

7. TASK COMMUNICATION :-

* In a multitasking system, multiple tasks / processes run concurrently (in pseudo parallelism) and each process may or may not interact between.
* Based on degree of interaction, the processes running on an OS are classified as.

Co-operating processes :- In the co-operating interaction model one process requires the inputs from other processes to complete its execution.

Competing processes :- The competing processes do not share anything among themselves but they share the system resources. Such as file, display device, etc

Co-opertaing processes exchanges information and communicate through the following methods.

Co-operation through sharing :- The co-operating process exchange data through some shared resources.

Co-operation through Communication :- No data is shared b/w the processes. But they communicate for synchronisation.

The mechanism through which processes / tasks communicate each other is known as Inter process / Task Communication (IPC). IPC is essential for process co-ordination.

Some of the important IPC mechanisms adopted by various kernels are explained below.

(i) Shared Memory :-

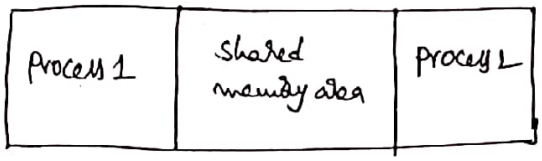


Fig :- Concept of shared Memory

* process share some area of the memory to communicate among them.
* Information to be communicated by the process is written to the shared memory area. Other processes which require this information can read the same from the shared memory area.

Ex :- Notice Board

Different mechanisms are adopted by different kernels for implementing this. A few among them are :

a) Pipes :- 'Pipe' is a section of the shared memory used by processes for communicating. Pipes follow the client-server architecture.

* A process which creates a pipe is known as a pipe server and a process which connects to a pipe is known as pipe client.
* It can be Unidirectional, allowing information flow in one direction or bidirectional allowing bi-directional information flow.

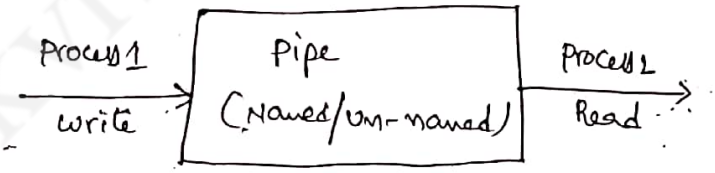


Fig :- Concept of pipe for IPC.

* Microsoft® windows Desktop operating System support two types of 'Pipes' for Inter process communication. They are :

Anonymous Pipes :- These are Unnamed, Unidirectional pipes are used for data transfer b/w two processes.

Named Pipes :- Named pipe is a named, Uni or Bi-direction pipe is used.

b) Memory Mapped objects :- Memory mapped object is a shared memory technique adopted by certain Real-Time Operating Systems for allocating a shared block of memory which can be accessed by multiple processes simultaneously

(ii) Message passing :-

* Message passing is an a/synchronous information exchange mechanism used for Inter process/Thread communication.

* The major difference b/w shared memory and message passing technique is that, through shared memory lots of data can be shared whereas only limited amount of info/data is passed through message passing.

* Message passing is relatively fast and free from the synchronisation overheads compared to shared memory.

* Based on message passing operation b/w the processes, message passing is classified into

a) Message Queue :- Usually the processes which wants to talk to another process posts the message to a First-In-First-out (FIFO) queue called 'message queue', which stores the messages temporarily in a system defined memory object, to pass it to the desire processes.

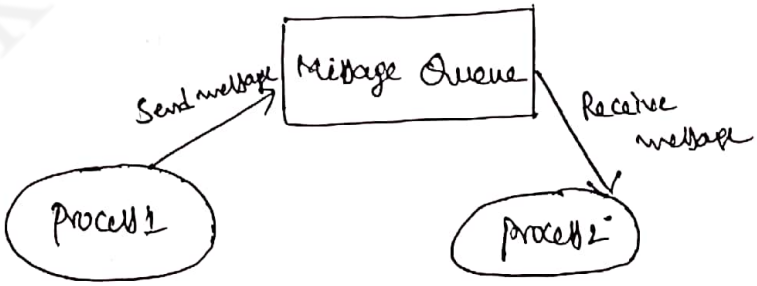


Fig :- Concept of message queue based indirect messaging for IPC.

b) Mailbox :- Mailbox is an alternate forms of 'Message queues' and it is used in certain Real-Time Operating Systems for IPC.

* Mailbox technique for IPC in RTOS is usually used for one way messaging
* The task/thread which wants to send a message to other task/threads creates a mailbox for posting the messages.

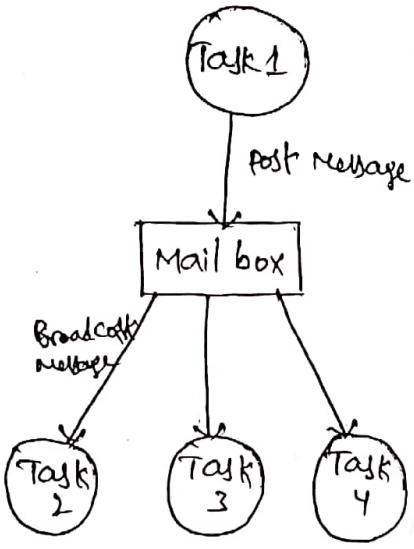


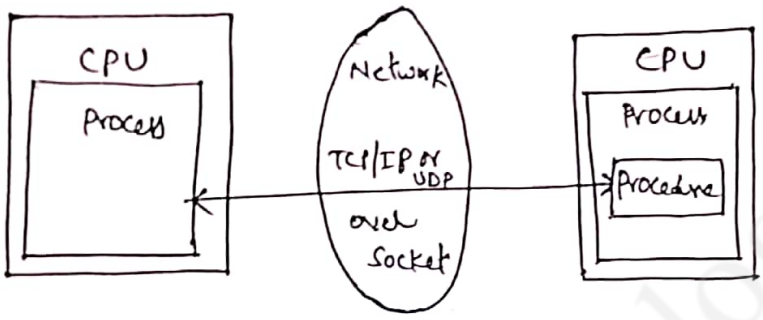
Fig:- Concept of Mailbox based indirect messaging for IPC.

(c) Signalling :- Signalling is a primary way of communication b/w processes/threads. Signals are used for asynchronous notifications where one process/thread fires a signal, indicating the occurrence of a scenario which the other process/thread is waiting.

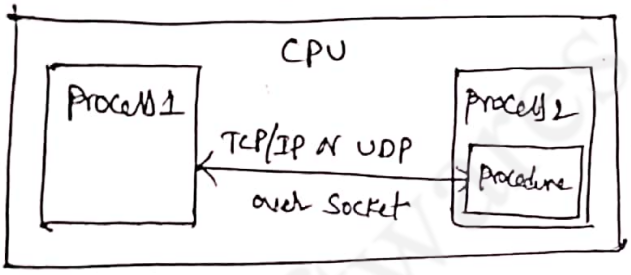
(iii) Remote Procedure Call (RPC) and Sockets :-

* Remote Procedure Call or RPC is the Inter Process Communication (IPC) mechanism used by a process to call a procedure of another process running on the same CPU or on a different CPU which is interconnected in a Network.
* The RPC communication can be either Synchronous (Blocking) or Asynchronous (Non-blocking).

* In the synchronous communication, the process which calls the remote procedure is blocked until it receives a response back from the other process.
* In asynchronous RPC calls, the calling process continues its execution while the remote process performs the execution of the procedure., The result from the remote procedure is returned back to the caller through mechanisms like Callback functions.



Processes running on different CPUs which are networked



processes running on same CPU

* Sockets are used for RPC communication. Socket is a logical endpoint in a two-way communication link between two applications running on a N/w
* A port number is associated with a socket so that the network layer of the communication channel can deliver the data to the designated application
* Sockets are of different types, namely Internet sockets (INET), UNIX Socket etc
* INET Sockets are classified into:
  1. Stream Sockets — connection oriented and they use TCP to establish a reliable connection.
  2. Datagram sockets — rely on UDP for establishing a connection.

: Task Synchronisation :-

In a multitasking environment, multiple processes run concurrently (in Pseudo parallelism and share the system resources. The act of making process aware of the access of a shared resources by each process to avoid conflicts is known as 'Task/Process Synchronisation'.

The various synchronisation issues may arise in a multitasking environment if processes are not synchronised properly. The following section describe the major task communication synchronisation issues observed in multitasking and the commonly adopted synchronisation techniques to overcome these issues.

I) Task Communication/synchronisation Issues :-

a) Racing :- This is the situation in which multiple processes compete (race) each other to access and manipulate shared data concurrently.

b) Deadlock :- A race condition produces incorrect results where as a deadlock condition creates a situation where none of the processes are able to make any progress in their execution, resulting in a set of deadlock process

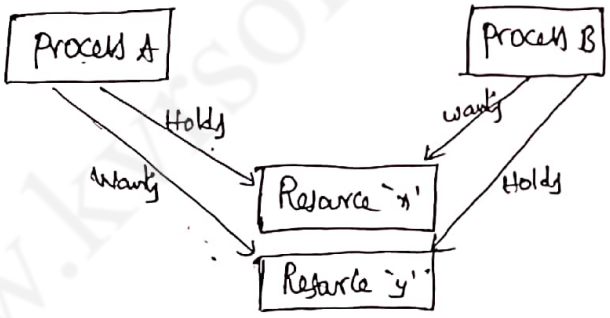


Fig :- Scenarios leading to deadlock.

In its simplest forms 'deadlock' is the condition in which a process is waiting for a resource held by another process which is waiting for a resource held by the process.

The different conditions favouring a deadlock situation are listed below.

Mutual Exclusion!- The criteria that only one process can hold a resource at a time. Meaning processes should access shared resources with mutual exclusion.

Hold and wait!- The condition in which a process holds a shared resource by acquiring the lock controlling the shared access and waiting for additional resources held by other processes.

No Resource preemption!- The criteria that operating systems cannot take back a resource from a process which is currently holding it and the resource can only be released voluntarily by the process holding it.

Circular wait!- A process is waiting for a resource which is currently held by another process which in turn is waiting for a resource held by the first process.

These conditions are also known as Coffman conditions.

↳ Livelock!- The livelock condition is similar to the deadlock condition except that a process in livelock condition changes its state with time.

↳ Starvation!- In the multitasking context, starvation is the condition in which a process does not get the resources required to continue its execution for a long time.

DEVICE DRIVERS :-

* Device driver is a piece of software that acts as a bridge between the operating system and the hardware.
* The architecture of the OS kernel will not allow direct device access from the user application.
* OS provides interfaces in the form of Application Programming Interface (API) for accessing the hardware.

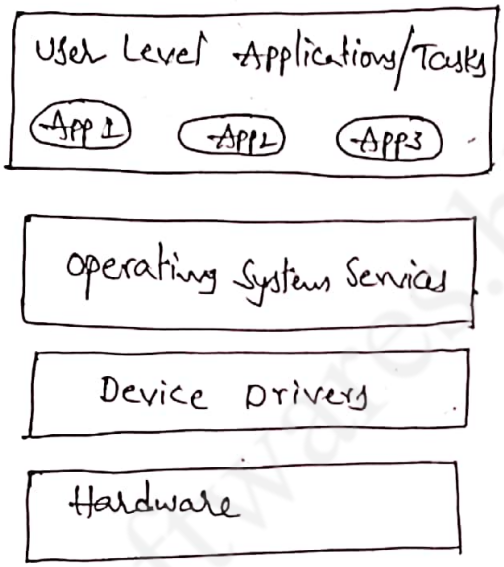


Fig:- Role of Device driver in Embedded OS based products.

* Device drivers are responsible for initiating and managing the communication with the hardware peripherals.
* They are responsible for establishing the connectivity, initialising the hardware and transferring data.
* Device driver implements the following:
  1. Device (Hardware) Initialisation and Interrupt Configuration.
  2. Interrupt handling and processing.
  3. Client interfacing (Interfacing with user applications)

:: HOW TO CHOOSE AN RTOS :-

A lot of factors needs to be analysed carefully before making a decision on the selection of an RTOS. These factors can be either functional or non-functional.

I) Functional Requirements :-

(i) Processor support :- It is not necessary that all RTOS's support all kinds of process architecture. It is essential to ensure the processor support by the RTOS.

(ii) Memory Requirements :- The OS requires ROM memory for holding the OS files and it is normally stored in a non-volatile memory like FLASH.

OS also requires working memory RAM for loading the OS services.

(iii) Real-time Capabilities :- It is not mandatory that the operating systems for all embedded systems need to be Real-time and all embedded operating systems are 'Real-time' in behaviour.

(iv) Kernel and Interrupt Latency :- The kernel of the OS may disable interrupts while executing certain services and it may lead to interrupt latency.

(v) Inter process communication and Task synchronisation :- The implementation of Inter process communication and synchronisation is OS kernel dependent.

certain kernels may provide a bunch of options whereas others provide very limited options.

(vi) Modularisation support :- It is very useful if the OS supports modularisation where in which the developer can choose the essential modules and re-compile the OS image for functioning.

(vii) Support for Networking and communication :- The OS kernel may provide stack implementation and driver support for a bunch of communication interfaces and Networking.

(viii) Development Language Support :- Certain operating systems include the run time libraries required for running applications written in languages like Java and C#.

II) Non-functional Requirements :-

(i) Custom Developed or Off the shelf :- Depending on the OS requirement, it is possible to go for the complete development of an operating system suiting the embedded system needs or use an off the shelf, readily available operating system, which is either a commercial product or an open source product, which is in close match with the system requirements.

Cost :- The total cost for developing or buying the OS and maintaining it in terms of commercial product and custom build needs to be evaluated before taking a decision on the selection of OS.

Development and Debugging Tools Availability :- The availability of development and debugging tools is a critical decision making factor in the selection of an OS for embedded design.

Ease of use :- How easy it is to use a commercial RTOS is another important feature that needs to be considered in the RTOS selection

After sales :- For a commercial embedded RTOS, after sales in the form of email, on-call services, etc. for bug fixes, critical path updates and support for production issues, etc. should be analysed throughly.