

UNIT-IV

RTOS Based Embedded system Design

OPERATING SYSTEM BASICS

An operating system is a program or a collection of programs that control the computer hardware and acts as an interface between the users and hardware. The OS manages the system resources and makes them available to the user applications/tasks on a need basis.

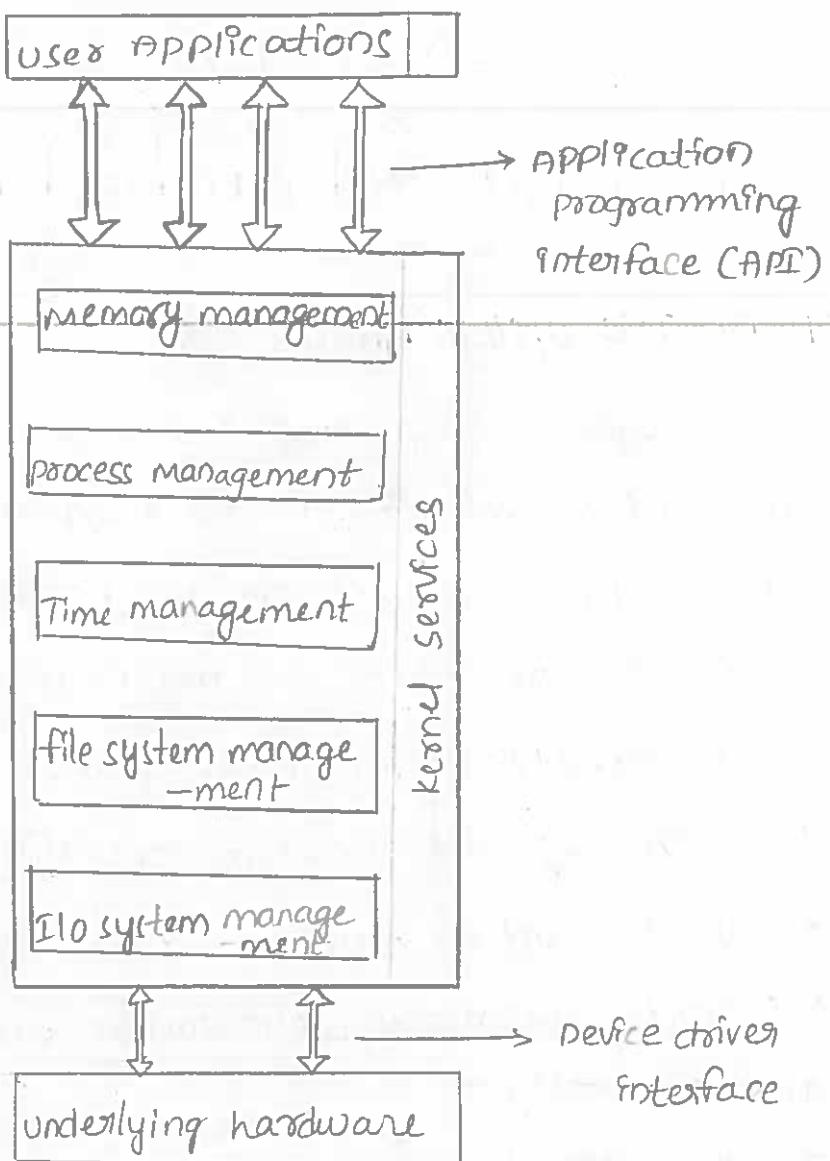
The primary functions of an operating system is:

- * make the system convenient to use
- * organise and manage the system resources efficiently and correctly.

Functions Handled by a General purpose kernel

The kernel is the core of the operating system and it is responsible for managing the system resources and the communication among the hardware and other system service. Kernel acts as the abstraction layer between system resources and user applications. Kernel contains a set of system libraries and services.

For a general purpose OS, the kernel contains different for handling the following



1. process management

process management deals with managing the process tasks.

It is responsible to manage following functions.

- i) Memory space set up for the process.
- ii) Allocates system resources.
- iii) Loads the process code into memory
- iv) scheduling and managing process execution
- v) Interprocess communication and synchronization.
- vi) managing the process control block (PCB)
- vii) deletes or terminates the process.

2. file system management

file is a collection of related information. A file could be program (source code or executable), text files, image files, word documents, audio/video files etc.

- i) The creation, deletion and alteration of files and directories.
- ii) saves file in secondary storage.
- iii) automatically allocates the files in the available free space.
- iv) provides flexible naming conventions to the files

3. primary memory management

primary memory is a volatile memory in which the processes are loaded and data is stored. The primary management of the kernel does the following functions.

- i) It monitors the memory area used by the process
- ii) dynamic memory allocation (allocates and de-allocates memory space).

4. protection system management

protection implies implementation of security policies to restrict the access of user and system by other users or process. In multiuser operating systems, one user can not access other user's information. This facility is provided by protection services running inside the kernel.

5. Input or output system management

kernel is responsible for routing the I/O requests coming from different user applications to the

appropriate I/O devices of the system by using a set of application, programming interfaces (API). The device manager handles all I/O operations and also does the following functions.

- i) It loads and unloads the device drivers.
- ii) It exchanges information and system specific control signals to and from the device.

6. secondary storage management

The secondary storage management of kernel manages the secondary storage devices connected to a system and also backups the system data. This does the following services.

- i) Allocates the disk storage
- ii) schedules the disk
- iii) manages free disk space.

7) Interrupt Handler

kernel is responsible to handle all the system generated external or internal interrupts.

TYPES OF OPERATING SYSTEMS;

The operating systems are classified into various types based on the different applications. The two main operating systems are:

1. general purpose operating system (GPOS)
2. Real Time operating system (RTOS)

1. General purpose operating system

A GPOS is a non-deterministic operating system that supports the general computing system. Its kernel is more generalized and provides services for execution of general applications. GPOS are often quite non-deterministic in behaviour. GPOS are employed in computing systems where deterministic behaviour or time constraints doesn't matter.

Examples: Windows XP, MS-DOS etc.

2. Real Time operating system (RTOS)

An RTOS is an operating system that supports the RTS systems. It can be defined as a multitasking as for the applications that need to be finished with in the dead lines and time critical allocation. Real-time implies deterministic timing behaviour. A RTOS implements policies and rules concerning time-critical allocation of a system resources. RTOS decides the order of applications to be executed and also assigns the time for each application. The predictable performance is achieved RTOS by imposing rules and policies.

Examples: Windows CE, QNX, Vx Works etc.

THE REAL-TIME KERNEL:

Real time kernel is called as the kernel of RTOS, which uses minimum set of services to run a user application. The basic functions expected from the kernel of RTOS are as follows.

1. Task/ process management

Task or process management deals with

- * Allocating memory space to the task
- * Loading task code into memory space
- ~~* Allocation of system resources.~~
- * Task control block (TCB) is set
- * Deletion of task / process.

A task control block (TCB) holds the following set of information of a particular task.

- i) Task ID : Identification no. of a particular task.
- ii) Task state: Indicates the current state of the task.
- iii) Task type: Indicates hard real time or soft real time or background task.
- iv) Task priority: It indicates the priority of the task.
- v) Task context pointer: It encodes the pointer for saving the context.
- vi) Task memory pointers: Pointers to the code memory, stack memory and data memory.
- vii) Task system resources pointers: Pointers to the system resources such as semaphores, mutex etc--- used by the task.
- viii) Task pointers: Pointers are used to other TCB i.e., for preceding, next, waiting tasks.

2. Task/ process scheduling:

Task scheduling refers to implementation of an algorithm to perform the optimal scheduling of tasks efficiently in order to achieve its deterministic

behavior.

3. Task / process synchronization:

In this, the resources and communication of multiple tasks are synchronized.

4. Error / exception Handling:

In this, the errors such as timeouts, deadlocks, divide by zero, bus error etc., that occur at kernel level or task level are registered and handled. OS kernel alerts the the error by using a system call i.e., API. When a task is waiting infinitely for a external device which is not responding, it generates hand-up. This can be avoided by using watchdog timer that is loaded with a maximum wait time.

5. memory management:

When a task is created, the RTOS memory management allocates a memory space by using block based memory allocation technique. This technique achieves deterministic behaviour due to small size and sub optimal usage of memory. The blocks are stored in a free buffer queue and are accessed without protection to avoid timing overheads/ memory fragmentation issues, garbage collection overhead and achieve predictable timings.

6. Interrupt Handling:

It is used to handle various interrupts which decide the behaviour of the RTOS. RTOS kernel employs nested interrupt architecture to provide priority levels of interrupt service routine.

unit-4, pg - 7/22

Interrupts are of two types:

i) Synchronous interrupt:

Synchronous interrupts are those which occurs in synchrony with the current executing task. In this interrupt handler exists within the interrupting task.

Ex: software interrupt, divide by zero, memory segmentation etc.

7. Time management: It is very essential part in all applications. Kernel is provided with Real Time Clock (RTC) hardware chip which is programmed to allow interrupts at a fixed rate. This timer interrupt is referred as 'Timer tick'. This timer tick is taken as the timing reference by the kernel.

Timer tick interrupt is handled by timer interrupt handler of kernel

- * Current context is saved.
- * System time gets incremented. It gets resetted when a timing error occurs if timer tickcount is above the maximum.
- * Kernel timers are updated.
- * The idle state periodic task are activated.
- * Scheduler is activated.
- * The terminated tasks and its data structures are deleted.

5 TASK AND PROCESS

Task: A task is a processor or job in the RTOS that defines the program execution. It maintains the related information on the program. A particular task in an embedded can be in any one of three states.

1. running state
2. blocked state
3. finished state

A task structure comprises data, objects, resources and control block.

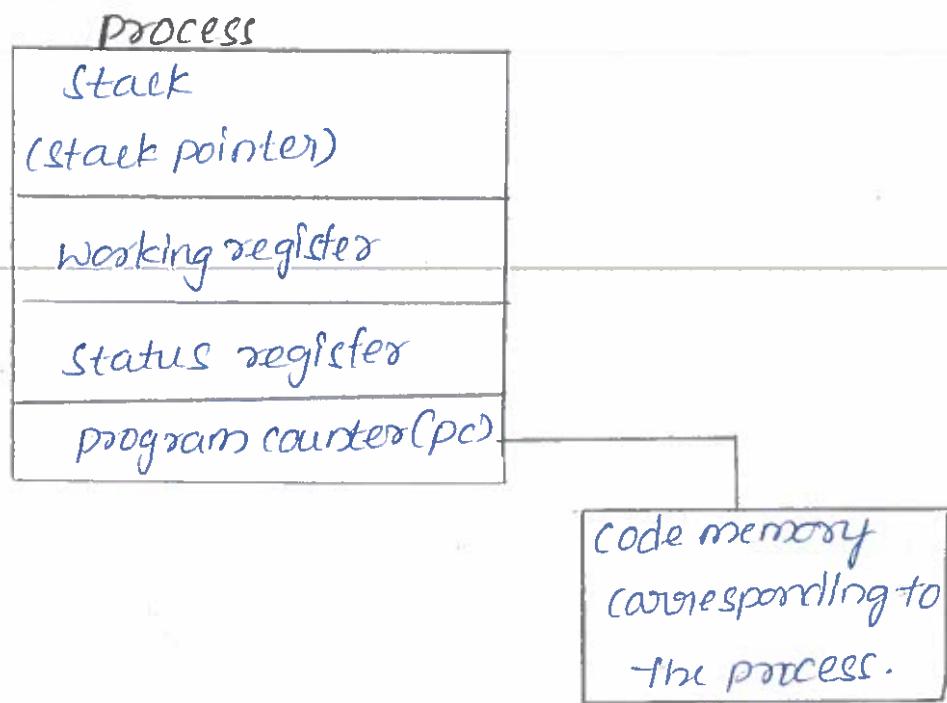
process of operating system:

A process is a program, or part of it, in execution. A process is also known as an instance of a program in execution. Multiple instances of the same program can execute simultaneously. A process requires various system resources like CPU for executing the process, memory for storing the code corresponding to the process and associated variables, I/O devices for information exchange etc. A process is sequential in execution.

structure of process

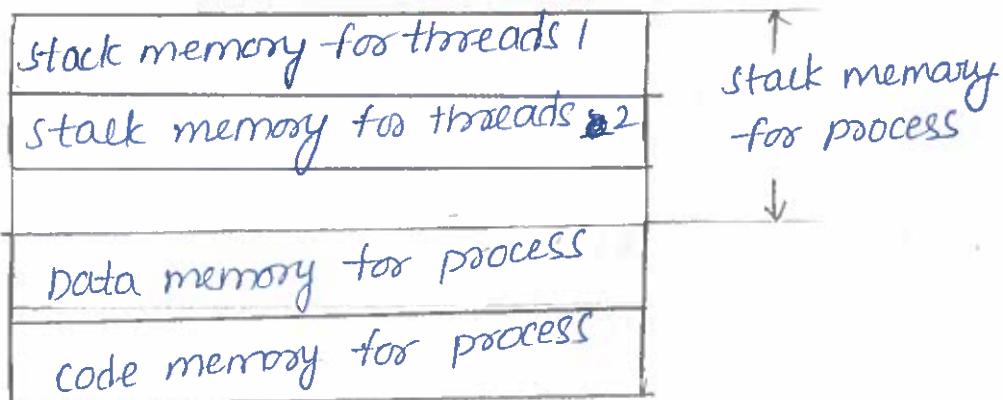
The concurrent execution of tasks leads to efficient utilization of the CPU and other system resources. A process consists of a set of registers, process status, a program counter (PC) to point the next instruction of a process.

The structure of a process is shown in figure.



THEADS:

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



MEMORY ORGANIZATION OF A PROCESS
AND ITS ASSOCIATED THREADS

concept of Multi-Threading:

- * A process /task in Embedded application may be a complex or lengthy one and it may contain various sub-operations like getting input from I/O devices connected to the processor, performing some internal calculations/operations, updating some I/O devices etc.
- * If all the sub-functions of a task are executed in sequence, the CPU utilization may not be efficient.
- * For example, if the process is waiting for a user input, the CPU enters the wait state for the event, and the process execution also enters a wait state.
- * Instead of this single sequential execution of the whole process, if the task/ process is split into different threads carrying out the different sub-functionalities of the process, the CPU on the effectively utilised and when the thread corresponding to the I/O operation enters the wait state, another threads which do not require the I/O went for their operation can be switched into execution.

This leads to more speedy execution of the process and the efficient utilization of the process over time and resources.

TASK / PROCESS

code memory		
data memory		
stack	stack	stack
Registers	Registers	Registers
Thread 1	Thread 2	Thread 3

- * If the process is split into multiple threads which executes a portion of the process, there will be a main threads and rest of the threads will be created within the main threads.
- * Use of multiple threads to execute a process brings the following advantage.

TASK SCHEDULING:

Determining which task / process is to be executed at a given point of time is known as task or process scheduling.

Task scheduling forms the basis of multitasking.

The scheduling processes are implemented in an algorithm and it run by the kernel as a service.

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

- * Ready state from Running state.
- * Blocked/Wait state from running state.
- * Ready state from Blocked/Wait state.
- * 'Completed' state

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

Non-preemptive scheduling:

- * It is employed in systems, which implement non-preemptive multi-tasking model.
- * In this scheduling type, the currently executing task or process is allowed to run until it terminates or enters the 'wait' state waiting for an I/O or system resource.

Preemptive scheduling:

- * It is employed in systems, which implements preemptive multi-tasking model.
- * In preemptive scheduling, every task in the 'Ready' queue gets a chance to execute.
- * In this, the scheduler can preempt (stop temporarily) the currently executing task or process and select another task from the 'Ready' queue

for execution.

- * A task which is preempted by the scheduler is moved to the 'Ready' queue.
- * The act of moving a 'running' process /task into the 'Ready' queue by the scheduler without the processes requesting for it is known as 'preem -ption'.
- * The two important approaches adopted in preem -ptive scheduling are time-based preemption and priority based preemption.

MULTIPROCESSING AND MULTITASKING

Multiprocessing

The process of executing multiple processes simultaneously is referred as "multiprocessing" and the system which performs the "multiproces -sing" is known as "multiprocessor system".

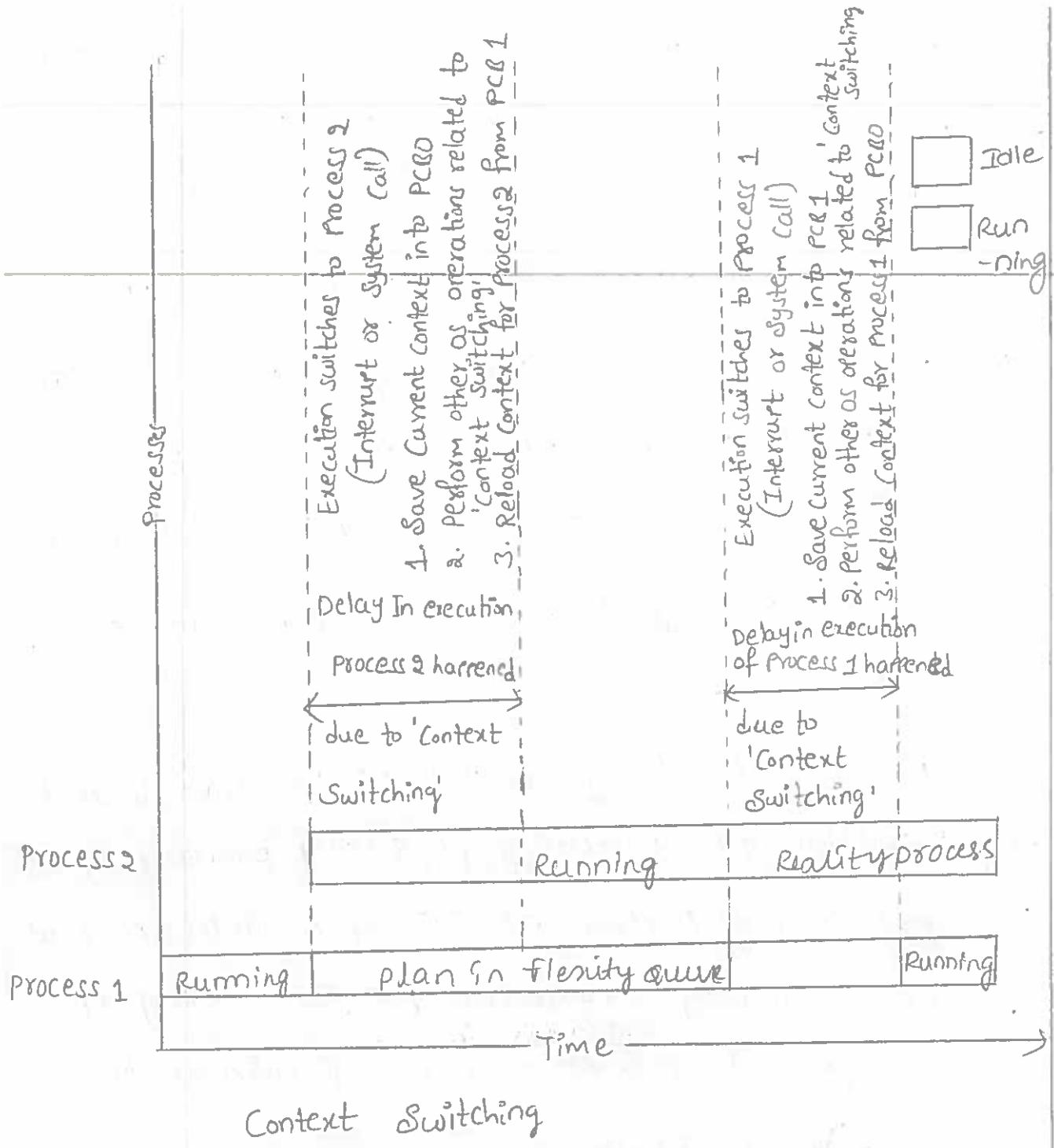
Multi-tasking

It is defined as the ability of an operating system to hold multiple processes in memory and switching the CPU from executing one process to another process. Thus, parallel execution of task occurs in multitasking.

In multitasking, the task/process of occurin

-re of switching will transform the virtual process properties into physical processor into physical properties, which is controlled by the schedules of the os kernel. whenever switching of cpu occurs, the context of currently executing process is saved, which can be retrieved at later stages of execution. Thus, context saving and context retrieval is important, for resuming a process exactly from the point where it was stopped.

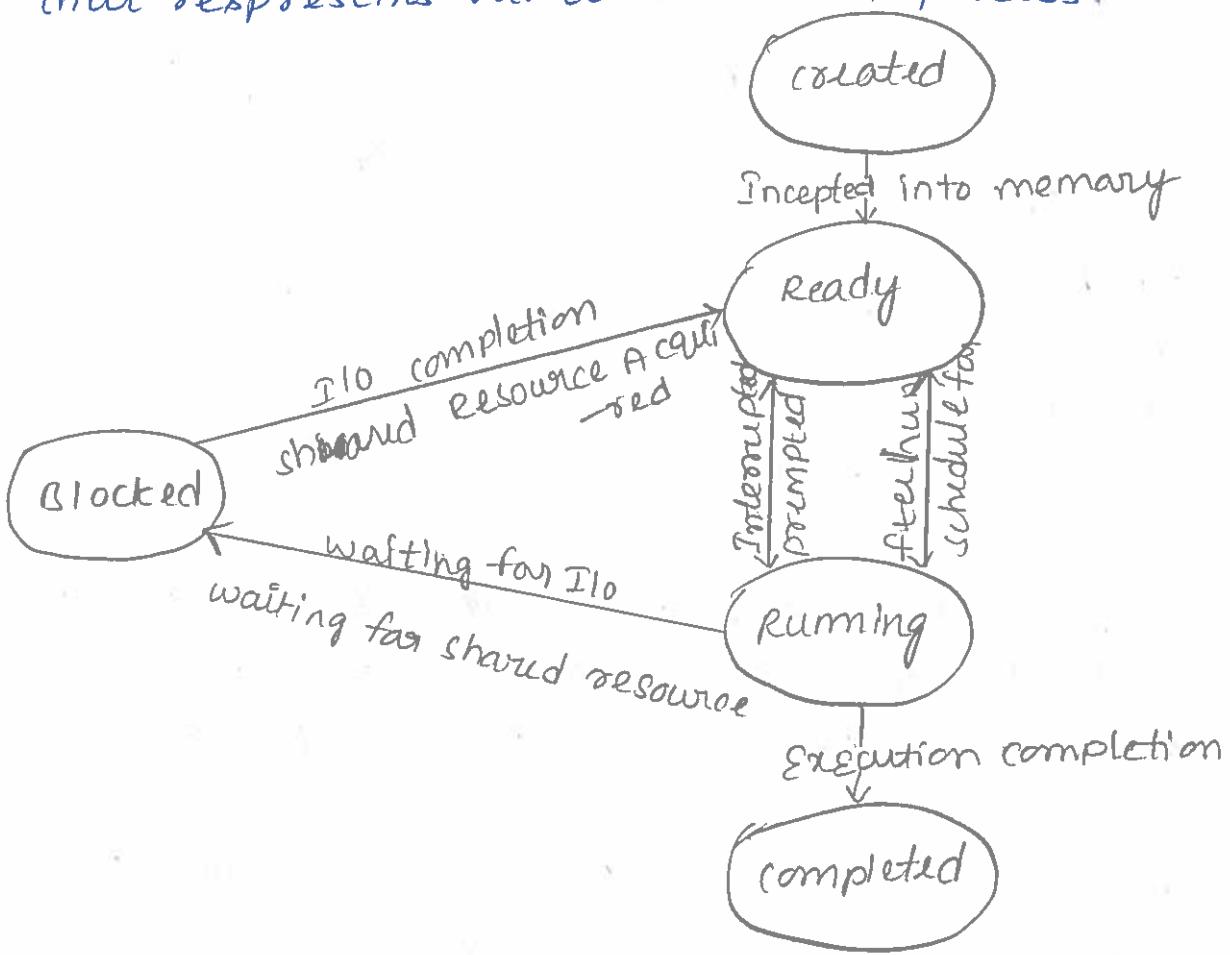
The multitasking consists of 'context switching' (action of switching cpu among processes), 'context saving' (action of saving current context of the currently executing process during cpu switching) and context retrieval (retrieval the saved context details for a process). The context switching is as shown in figure.



PROCESS LIFE CYCLE

process life cycle refers to a process when undergoes a series of states during transition from newly created state to terminated state. The execution of the process is controlled by the operating system and is also responsible for allocating resources to the process.

This figure shows a five-state process model that represents various states of process.



when the number of processes to be executed is large, then those processes are moved to a queue and are operated in a Round Robin fashion.

1. created / New:

A process is said to be in created state, if a process is being created by operating system without allocating any resources.

2. Running:

A process is said to be in running state, if it is being executed by the process using the source code instructions.

3. Ready

A process is said to be in ready state if all the ready processes are placed in the memory and are waiting for an opportunity to be executed.

4. Blocked

In this, the process waits for the occurrence of an event, in order to be executed until that event is completed. It cannot proceed further.

5. Exit

A process is said to be in exit state, if it is aborted or halted due to some reason. An exit process must be freed from the pool of executable processes by the operating system.

As shown in the figure, the process can change their states according to the situations.

1. New ready
2. Ready - running
3. Running - exit
4. Running - ready
5. Running - blocked
6. Blocked - ready
7. Ready - exit

KERNEL SPACE

Kernel space refers to a memory space where the kernel code is placed in a protected primary memory against the accessing of the unauthorized users. In OS, kernel applications are kept in main memory and cannot be swapped between primary and secondary memory.

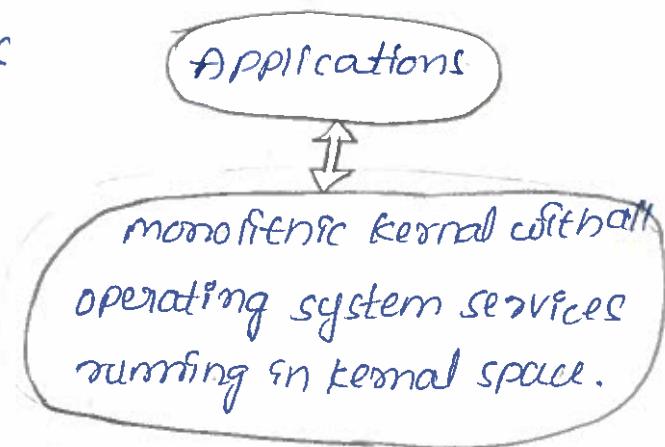
User space

User space refers to a memory space where the user applications are loaded and executed in primary - memory, it supports swapping of user applications between primary and secondary memory.

monolithic kernel:

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

Ex:- LINUX, SOLARIS, MS-DOS



Microkernel :

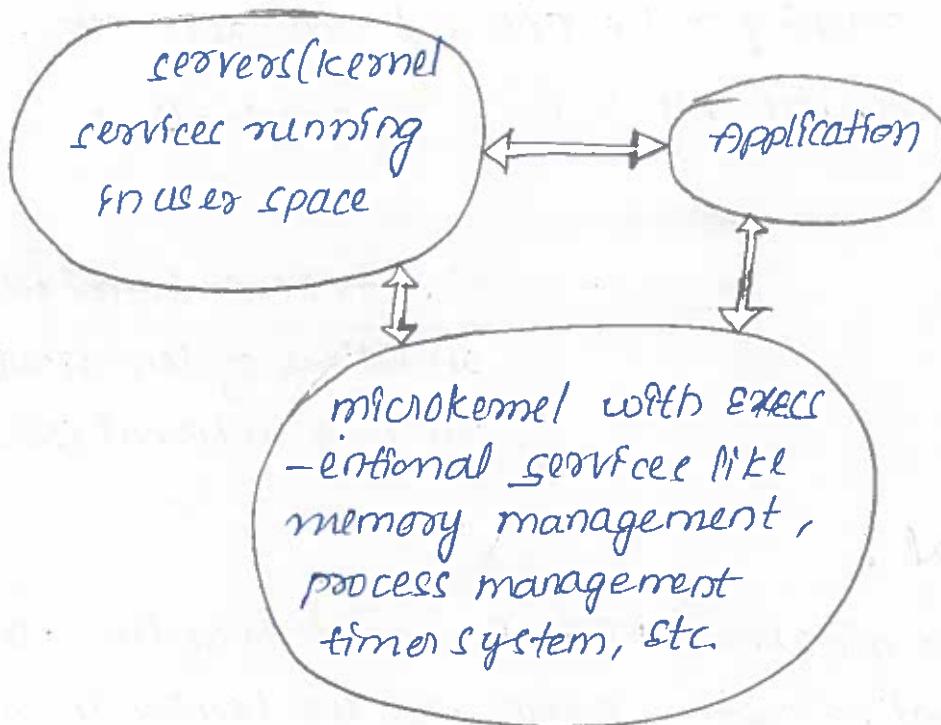
The microkernel design incorporate only the essential set of operating system service into the kernel. The rest of the operating system services are implemented in program known as "servers" which runs in user space. Memory management, process management, timer systems and interrupt handlers are the essential services, which forms the part of the microkernel. Mach, QNX, minix 3 kernel are examples for microkernel is shown below. micro kernel based design approach offers the following benefits.

Robustness:

If a problem is encountered in any of the services which runs are server application, the same can be reconfigured and reshared without the need for re-starting the entire OS. Thus, this approach is highly useful for systems, which demands high "availability".

Configurability:-

Any services, which run an "server" application can be changed without the need to restart the whole system. This makes the system dynamically configurable.



MULTI TASKING:

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 multi-tasking.

Multi-tasking creates the illusion of multiple tasks executing in parallel.

Multi-tasking involves three operations while functioning. They are:

- * context switching
- * context saving
- * context retrieval

context switching :

The act of switching CPU among the process or changing the current execution context is known as context switching.

context saving:

The act of saving the current context which contains the context details for the current running process at the time of CPU switching is known as context saving.

context retrieval

The process of retrieving the saved context details for a process, which is going to be executed due to CPU switching, is known as 'context retrieval'.

Types of multi-tasking:

There are 3 types of multi-tasking.

- * co-operative multi-tasking
- * preemptive multi-tasking
- * non preemptive multi-tasking

co-operative multi-tasking:

- It is a primitive form of multi-tasking.
- In this, a task or process gets a chance to execute only when the currently executing task or process voluntarily relinquishes the CPU.
- In this method, any task / process can hold the CPU as much time as it wants.
- Since this type of implementation involves the meay to the tasks each other for getting the CPU time for execution, it is known as co-operative multi-tasking.

preemptive multi-tasking:

- It ensures that every task or process gets a chance to execute.
- When and how much time a process gets is dependent on the implementation of the preemptive scheduling.
- In this, the currently running task or process is preempted to give a chance to other tasks or process to execute.
- The preemption of task may be based on time slots or task/process priority.

non-preemptive multi-tasking:

- In this, the process or task, which is currently given the CPU time, is allowed to execute until it terminates or enters into 'Blocked / wait' state, waiting for an I/O or system resource.
- The co-operative multi-tasking and non-preemptive multi-tasking differs in their behaviour when they are in the Blocked / wait state.

In co-operative multi-tasking, the currently executing process/task need not relinquish the CPU when it enters the 'Blocked / wait' state waiting for I/O, or a shared resource access or an event to occur.

Whereas in non-preemptive multi-tasking, the currently executing task relinquishes the CPU when it waits for an I/O or system resource or an event to occur.