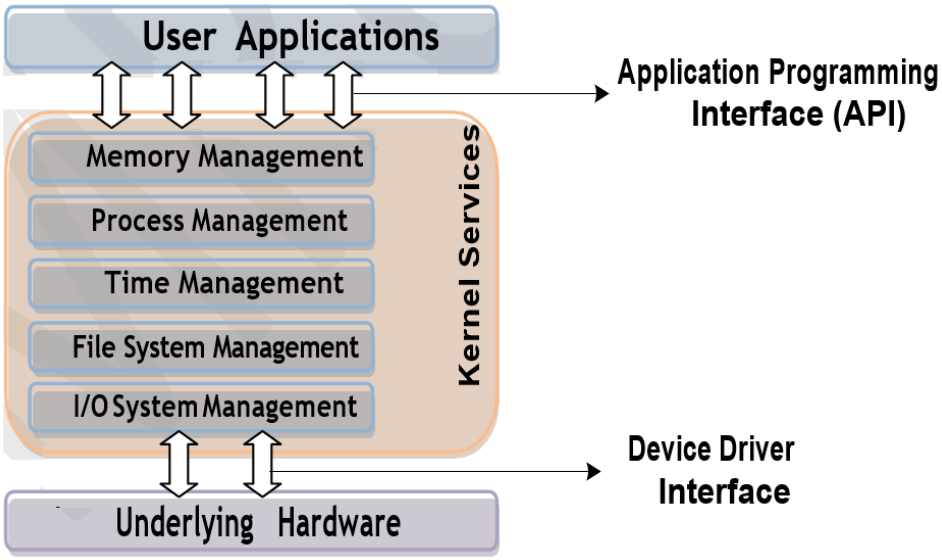
UNIT-5
REAL-TIME OPERATING SYSTEM (RTOS)

1.1. Operating System – Functions and Services

- An **Operating System (OS)** is a software that acts as an interface between computer hardware and the user. The OS acts as a bridge between the user applications/tasks and underlying system resources through a set of system functionalities and services.
- The OS manages the system resources and makes them available to the user applications/tasks on a need basis.
- The primary functions of an OS are
  - Make the system convenient to use
  - Organise and manage the system resources efficiently and correctly



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 between system resources and user applications. Kernel contains a set of system libraries and services.

The Kernel services in operating system are given below:

✓ Memory management:

It performs the task of allocation and de-allocation of memory space to processes. When a process is created the memory management unit allocates the memory addresses (blocks) to it by mapping the process address space. Threads of a process share the memory space of the process.

The Memory management unit of the Kernel is responsible for

- Keeping the 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)

✓ Process management:

It deals with managing the processes/tasks and helps OS to create and delete processes. Process management includes

- create and delete processes
- setting-up 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 PCB
- Inter process communication and synchronization

✓ Time management:

Accurate time management is essential for providing precise time reference for all applications. The time reference to the Kernel is provided by a high-resolution Real Time Clock (RTC).

The hardware timer is programmed to interrupt the Processor at a fixed rate. This timer interrupt is called as '*Timer tick*'. The timer tick interrupt is handled by the Timer interrupt handler of the Kernel.

The 'Timer tick' interrupt can be utilised for implementing the following:

- Save the current context
- Update the timers implemented in Kernel
- Activate the periodic tasks
- Schedule the tasks based on the scheduling algorithm
- Delete the terminated tasks and their associated TCS
- Load the context for the first task in the ready queue.

✓ File system management:

It manages all the file-related activities such as organization storage, retrieval, naming, sharing, and protection of files. File is a named entity on system memory, which may contain data, characters, texts, images, audio, video and mix of these.

The file management service of Kernel is responsible for

- creation, deletion and alteration of files
- saving of files in secondary storage memory (hard disk)
- providing allocation of file space based on free space available
- providing flexible naming convention for the files

✓ I/O System (Device) Management:

The Kernel is responsible for routing the I/O requests coming from different user applications to the appropriate I/O devices of the system. The Kernel maintains a list of all I/O devices of the system and updates the list of available devices as and when a new device is installed.

The device manager of the Kernel is responsible for handling all I/O device related operations. The Kernel talks to the I/O device through a set of low-level system calls, which are implemented in a service, called device drivers. The device drivers are specific to a device or a class of devices.

Device management keeps tracks of all devices and it responsible for

- Loading and unloading of device drivers
- Exchanging information and control signals to/from the device

Application Programming Interface (API) provides the interface between application software and system software so that it is able to run on the processor using the given system software.

Device driver Interface acts as a bridge between the operating system and the hardware. A device driver is a special kind of software program that controls a specific hardware device attached to a computer system**.** They are responsible for establishing the connectivity, initializing the hardware (setting up various registers of the hardware device) and transferring data.

1.2. Types of Operating Systems

- An **Operating System (OS)** is a software that acts as an interface between computer hardware and the user.
- The OS acts as a bridge between the user applications/tasks and underlying system resources through a set of system functionalities and services.
- The OS manages the system resources and makes them available to the user applications/tasks on a need basis.

Operating systems are classified into different types, depends on the type of Kernel services, purpose and type of computing system where the OS is deployed and the responsiveness to applications.

General Purpose OS:

1. Operating systems which are deployed in general computing systems, are referred to as 'General purpose operating system (GPOS)'
2. The Kernel is more generalised and it contains all kinds of services required for executing generic applications.
3. Need not be deterministic in execution behaviour.
4. Focuses on computing throughput
5. Response time is not of great importance
6. Personal Computers /Desktop systems are typical examples for systems where GPOSs are deployed.

Example → Windows XP/ MS-DOS

Real Time Operating System (RTOS):

1. Operating systems which are deployed in embedded systems demanding real-time response, are referred to as 'Real-Time Operating System (RTOS)'
2. An RTOS is an operating system for the systems having the real timing constraints and deadlines on the tasks, ISTs and ISRs.
3. *Real-Time'* implies deterministic timing behaviour, which means that the OS services consumes only known and expected amounts of time regardless the number of services.
4. Focuses on very fast response time
5. Strict timing constraints
6. RTOS implements policies and rules concerning time-critical allocation of system resources. The RTOS decides which application should run in which order and how much time needs to be allocated for each application.

Example → VxWorks, µCOS-II, QNX, VTRX, Windows CE

Types of RTOS:

(1) In-House Developed RTOS:

In-house RTOS has the codes written for the specific need and applications. Small level application developers uses the in house RTOS.

(2) Broad-based Commercial RTOS:

The readily available broad-based commercial RTOS package offers the following advantages

- Provides thoroughly tested and debugged RTOS functions
- Provides several development tools (testing, simulating & debugging)
- Support to many processor architectures
- Support to development of GUIs in the system
- Support to many devices, graphics, network protocols and file systems
- Support to device software optimization
- Provides error and exception handling functions
- Saves maintenance cost
- Saves development time

(3) General Purposes OS with RTOS:

The general purpose OS can be used in combination with the RTOS functions

Ex: Linux→ RT Linux
Windows XP → Windows XP Embedded

(4) Special Focus RTOS:

These are used with specific processors like ARM or 8051 or DSP

Ex: OSEK for auto motives
Symbian OS for Mobile Phones.

Based on the execution behaviour, there are hard real-time and soft real-time systems

Hard Real-Time systems

- A real-time system that strictly adhere to the timing constraints for a task is referred to as 'Hard-Real time system'.
- A hard real-time system must meet the timing deadlines without any delay. Missing the deadline would cause serious failure, including permanent data lose and irrecoverable damages to the system or user. For ex., in a nuclear plant safety system, missing a deadline may cause loss of life/damage to property.
- Hard real-time systems emphasise the principle –

"A late answer is always a wrong answer"

Examples: Airbag control system
Antilock braking system

Soft Real-Time systems:

- The real time system which does not guarantee meeting deadlines, but offer the best effort to meet the deadline are called as 'Soft Real-Time systems'
- Missing deadlines for tasks are acceptable for soft real-time systems, but the frequency of deadlines missing should be within the compliance limit of the Quality of Service (QoS).
- Soft real-time systems emphasise the principle –

  "A late answer is an acceptable answer", but it could have been done a bit faster.

  Example: ATM is a typical example of a soft real-time system. While withdrawing money from ATM, if it takes a few seconds more than the normal operating time, it may not cause any serious problem.

1.3. RTOS – FUNCTIONS & FEATURES

1. A real-time operating system (RTOS) is an operating system with two key features:
   Predictable performance and deterministic behaviour
2. Operating systems which are deployed in embedded systems demanding real-time response, are referred to as 'Real-Time Operating System (RTOS)'
3. An RTOS is an operating system for the systems having the real timing constraints and deadlines on the tasks, ISTs and ISRs.
4. *Real-Time'* implies deterministic timing behaviour, which means that the OS services consumes only known and expected amounts of time regardless the number of services.
5. Focuses on very fast response time
6. Strict timing constraints
7. RTOS implements policies and rules concerning time-critical allocation of system resources. The RTOS decides which application should run in which order and how much time needs to be allocated for each application.

Example → VxWorks, µCOS-II, QNX, VTRX, Windows CE

Basic functions in RTOS

1. Basic Kernel functions and Scheduling
2. OS initiate and start functions
3. Service and system Clock functions
4. Device driver, Network stack, send and receive functions
5. Time and Delay functions and Watchdog Timers
6. Error Handling functions
7. Task state switching functions
8. ISR and IST functions
9. Memory Functions (Create, Query, Get, Put)
10. IPC functions (Signal, Semaphore, Queue, Mailbox, Pipe, Socket, RPC)

Characteristics / Features of RTOS

An RTOS is small, fast, responsive, and deterministic. This means that it will execute tasks quickly and efficiently, responding as expected every time. Due to the significance of its host device, it is more secure and less likely to crash or fail.

- **Determinism** : Repeating an input will result in the same output.
- **High performance** : RTOS systems are typically fast and responsive.
- **Safety and security** : RTOSes are frequently used in critical systems and they must have higher security standards and more reliable safety features.
- **Priority-based scheduling:** Priority scheduling means that actions assigned a high priority are executed first, and those with lower priority come after. This means that an RTOS will always execute the most important task.
- **Small footprint** : Compared to general OSes, real-time operating systems are lightweight. For example, Windows 10, with post-install updates, takes up approximately 20 GB. VxWorks, on the other hand, is about few MB.
- **Time Constraints** : Time constraints related with real-time systems simply means that time interval allotted for the response of the ongoing program. This deadline means that the task should be completed within this time interval. Real-time system is responsible for the completion of all tasks within their time intervals.

Real-Time System Examples

RTOSes can be found in many embedded products. The embedded products with VxWorks alone, powering more than two billion devices. Systems from car engines to deep-space telescopes to helicopter guidance systems to the Mars rovers use embedded systems that run a real-time operating system.

Domain	Examples
Aircraft	Flight display control Drones
Telecom	5G Modem Satellite Modem
Transportation	Airbag control system Antilock braking system
Medical	Magnetic Resonance Imaging (MRI) Surgery Equipment, Ventilators
Manufacturing	Factory robotics Systems Safety Systems - Oil & Gas Vibration monitors

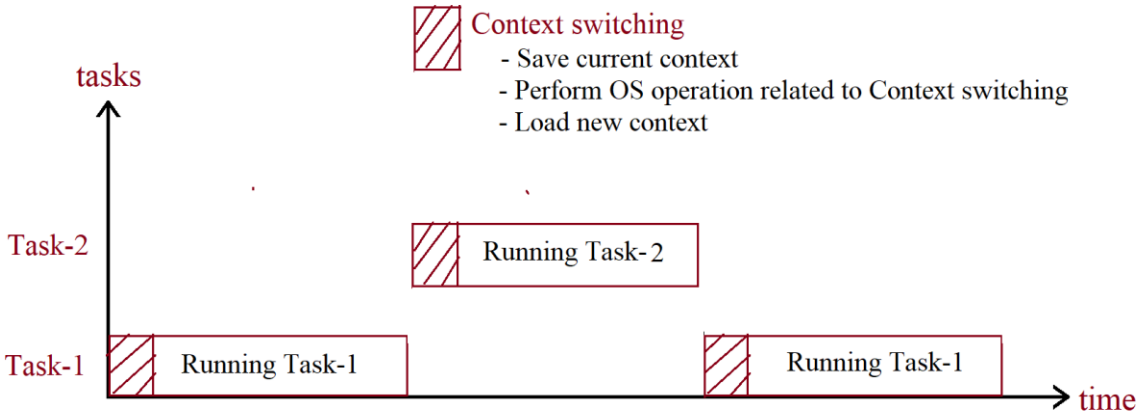
1.4. Multiprocessing & Multitasking:

Multiprocessing describes the ability to execute multiple processes simultaneously. The systems which are capable of performing multiprocessing are knows as multiprocessor systems. Multiprocessor systems have multiple CPUs and can execute multiple processes simultaneously.

Multitasking is the ability of the operating system to hold multiple processes in memory and switch the CPU from executing one process to another process. Multitasking involves the switching of execution among multiple tasks, which is controlled by the scheduler of the OS kernel.

Context switching:

- Multitasking involves the switching of execution among multiple tasks, which is controlled by the scheduler of the OS kernel. The act of switching CPU among the processes or changing the current execution context is known as '**Context switching**'



- Whenever a CPU switching happens, the current context of execution *(Program Counter, Status word, Stack pointer, CPU registers)* should be saved. The act of saving the current context which contains the context details for the currently running process at the time of CPU switching is called as '***Context saving***'
- 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***'
- The context saving and retrieval is essential for resuming a process exactly from the point where it was interrupted due to CPU switching.

Types of Multitasking:

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

Co-operative Multitasking:

- In Cooperative multitasking, a task/process gets a chance to execute only when the currently executing task/process voluntarily relinquishes the CPU.
- In this method, any task/process can avail the CPU as much time as it wants.
- Since 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.

Preemptive Multitasking:

- In Preemptive multitasking, every task/process gets a chance to execute. When and how much time a process gets is dependent on the implementation of the preemptive scheduling.
- As the name indicates, the currently running task/process is pre-empted (temporarily stopped) to give a chance to other task/process from 'Ready' queue for execution.
- The preemption of task may be based on time slots or task/process priority

Non-preemptive Multitasking:

- In non-preemptive multitasking, the currently executing process/task is allowed to run until it terminates (enters the 'Completed' state) or enters the 'Blocked/Wait' state, waiting for an I/O or system resource.
- The co- operative and non-preemptive multitasking differs in their behavior when they are in the 'Blocked/Wait' state. In co-operative multitasking, the currently executing process/task need not relinquish the CPU when it enters the 'Blocked/Wait' sate, whereas in non-preemptive multitasking the currently executing task relinquishes the CPU when it waits for an I/O.

1.5. TASK SCHEDULING:

- Multitasking involves the switching of execution among multiple tasks. In a multitasking system, there should be some mechanism in place to share the CPU among the different tasks and to decide which process/task is to be executed at a given point of time

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

- The scheduling policies are implemented in an algorithm and it is run by the kernel as a service. The kernel service/application, which implements the scheduling algorithm, is known as '*Scheduler*'

- Depending on the scheduling policy the process scheduling decision may take place when a process switches its state from
  - Ready state to Running state
  - Running state to Blocked/ Wait state
  - Blocked/Wait state to Ready state
  - Completed state
  -
- The task scheduling policy can be *pre-emptive*, *non-preemptive* or *co- operative*

- The following are the common Task scheduling models used by schedulers

(1) Cooperative Scheduling
(2) Non-preemptive scheduling
    - First Come - First Serve (FCFS) / FIFO scheduling
    - Last Come – First Serve (LCFS) / LIFO scheduling
    - Shortest Job First (SJF) scheduling
    - Non-preemptive - Priority based scheduling
(3) Preemptive scheduling
    - Shortest Remaining Time (SRT) scheduling
    - Round Robin (RR) scheduling
    - Preemptive - Priority based scheduling

(1) Cooperative Scheduling:

- In Cooperative scheduling, a task/process gets a chance to execute only when the currently executing task/process voluntarily relinquishes the CPU.
- In this method, any task/process can avail the CPU as much time as it wants.
- The scheduler inserts the ready tasks for the **sequential execution** in cooperative model. Cooperative means that each ready task cooperates to let a running one finish.
- Cooperative scheduling is used in applications that require a fixed order of execution (Example – an automatic washing machine)

(2) Non-preemptive scheduling:

- In non-preemptive scheduling, the currently executing process/task is allowed to run until it terminates (enters the 'Completed' state) or enters the 'Blocked/Wait' state, waiting for an I/O or system resource.

(i) First Come - First Serve (FCFS) / FIFO scheduling:

- Allocates CPU time to the processes based on the order in which they enters the '*Ready*' queue.
- The **FIRST** entered process is serviced first
- It is same as any real world application where queue systems are used. Example – Ticketing reservation system.
- Disadvantages
  - Favours monopoly of process
  - Poor device utilization.
  - The average waiting time is not minimal

(ii) Last Come - First Serve (LCFS) / LIFO scheduling:

- Allocates CPU time to the processes based on the order in which they enters the '*Ready*' queue.
- The **LAST** entered process is serviced first
- LCFS scheduling is also known as Last In First Out (LIFO) where the process, which is put last into the '*Ready*' queue, is serviced first
- Disadvantages
  - Favours monopoly of process
  - Poor device utilization.
  - The average waiting time is not minimal

(iii) Shortest Job First (SJF) scheduling:

- Allocates CPU time to the processes based on the execution time for tasks
- In SJF scheduling, the process with the shortest estimated run time is scheduled first, followed by next shortest process and so on.
- The average waiting time for a given set of processes is minimal
- Optimal compared to other non-preemptive scheduling like FCFS
- Disadvantages
  - A process whose estimated execution time is high may not get a chance to execute, if more and more processes with least estimated execution time enters the '*Ready*' queue.
  - May lead to the 'Starvation' of processes with high estimated completion time
  - Difficult to know in advance the next shortest process in the '*Ready*' queue for scheduling since new processes with different estimated execution time keep entering the '*Ready*' queue at any point of time.

(iv) Non-preemptive - Priority based scheduling:

- Allocates CPU time to the processes based on the priority of the tasks
- In Priority based Non-preemptive scheduling, the process with high priority is serviced at the earliest compared to other low priority processes in the 'Ready' queue.
- Disadvantages
  - Similar to SJF scheduling algorithm, non-preemptive priority based algorithm also possess the drawback of '*Starvation*' where a process whose priority is low may not get a chance to execute if more and more processes with higher priorities enter the '*Ready*' queue.
  - '*Starvation*' can be effectively tackled in priority based non-preemptive scheduling by dynamically raising the priority of the low priority task/process which is under starvation (waiting in the ready queue for a longer time for getting the CPU time)
  - The technique of gradually raising the priority of processes which are waiting in the 'Ready' queue as time progresses, for preventing '*Starvation*', is known as '*Aging*'.

(3) Preemptive scheduling:

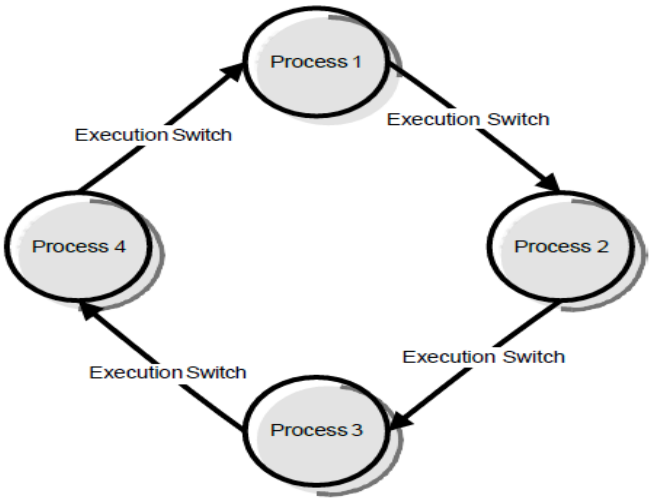
- In Preemptive scheduling, every task/process gets a chance to execute.
- As the name indicates, the currently running task/process is pre-empted (temporarily stopped) to give a chance to other task/process from 'Ready' queue for execution.
- A task which is preempted by the scheduler is moved to the '*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 '*Preemption*'
- When to pre-empt a task and which task is to be picked up from the '*Ready*' queue for execution after preempting the current task is purely dependent on the scheduling algorithm.
- Time-based preemption and priority-based preemption are the two important approaches adopted in preemptive scheduling

(i) Preemptive SJF Scheduling/ Shortest Remaining Time (SRT):

- The *non preemptive SJF* scheduling algorithm sorts the 'Ready' queue only after the current process completes execution or 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 execution time of the currently executing process.
- If the execution time of the new process is less, the currently executing process is preempted and the new process is scheduled for execution.
- Always compares the execution time of a new process entered the 'Ready' queue with the remaining time for completion of the currently executing process and schedules the process with shortest remaining time for execution.
- Preemptive SJF scheduling is also called as Shortest Remaining Time (SRT) scheduling.

(ii) Round Robin (RR) 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
- When the pre-defined time elapses or the process completes (before the pre-defined time slice), the next process in the 'Ready' queue is selected for execution.



Round Robin Scheduling

- This is repeated for all the processes in the 'Ready' queue
- Once each process in the 'Ready' queue is executed for the pre-defined time period, the scheduler comes back and picks the first process in the 'Ready' queue again for execution.

(iii) Preemptive - Priority based Scheduling

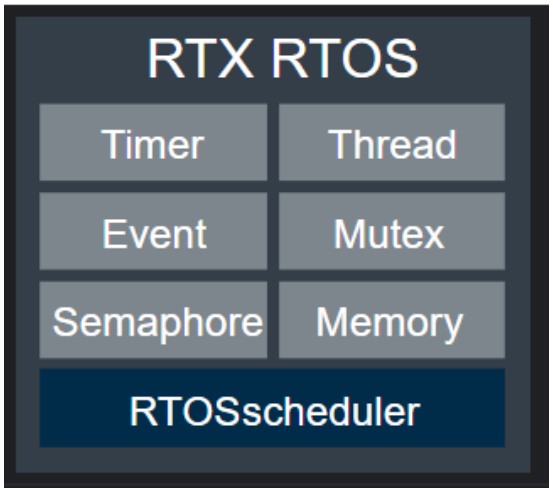
- Same as that of the ***non-preemptive priority*** based scheduling except for the switching of execution between tasks
- In ***preemptive priority*** based scheduling, any high priority process entering the 'Ready' queue is immediately scheduled for execution whereas in the ***non-preemptive*** scheduling any high priority process entering the 'Ready' queue is scheduled only after the currently executing process completes its execution or only when it voluntarily releases the CPU
- The priority of a task/process in preemptive priority based scheduling is indicated in the same way as that of the mechanisms adopted for non-preemptive multitasking.

1.6. Kiel RTX RTOS – Features and Benefits :

The Keil RTX is a royalty-free, deterministic Real-Time Operating System designed for ARM and Cortex-M devices. It allows you to create programs that simultaneously perform multiple functions and helps to create applications which are better structured and more easily maintained.

Modern microcontroller applications must often serve several concurrent activities. RTX manages the switching between the activities. Each activity gets a separate thread which executes a specific task and to simplify the program structure. Keil RTX is scalable and more threads can easily be added later. Threads have a priority allowing faster execution of time-critical parts of user applications.

Keil RTX offers services needed in many real-time applications, such as threads, timers, memory and object management, and message exchange. The following image shows the services of RTX RTOS



Features

- ✓ Royalty-free, deterministic RTOS with source code
- ✓ **Flexible Scheduling**: round-robin, pre-emptive, and cooperative
- ✓ High-Speed real-time operation with low interrupt latency
- ✓ Small footprint for resource constrained systems
- ✓ Unlimited number of tasks each with 254 priority levels
- ✓ Unlimited number of mailboxes, semaphores, Mutex, and Timers
- ✓ Support for multithreading and thread-safe operation
- ✓ Kernel aware debug support in MDK-ARM
- ✓ Dialog-based setup using µVision Configuration Wizard
- ✓ Easy to use, Safe and secure

Benefits

The Keil RTX RTOS delivers following benefits:

1. **Task scheduling** - determining which task/process is to be executed at a given point of time. Tasks are called when needed, ensuring better program flow and event response.

2. **Multitasking** - the ability of the operating system to hold multiple processes in memory and switch the CPU from executing one process to another process. Multitasking involves the switching of execution among multiple tasks, which is controlled by the scheduler of the OS kernel. It gives the illusion of executing a number of tasks simultaneously.

3. **Deterministic behaviour** - Events and interrupts are handled within a defined time (deadline)

4. **Shorter ISRs** - enables more deterministic interrupt behaviour

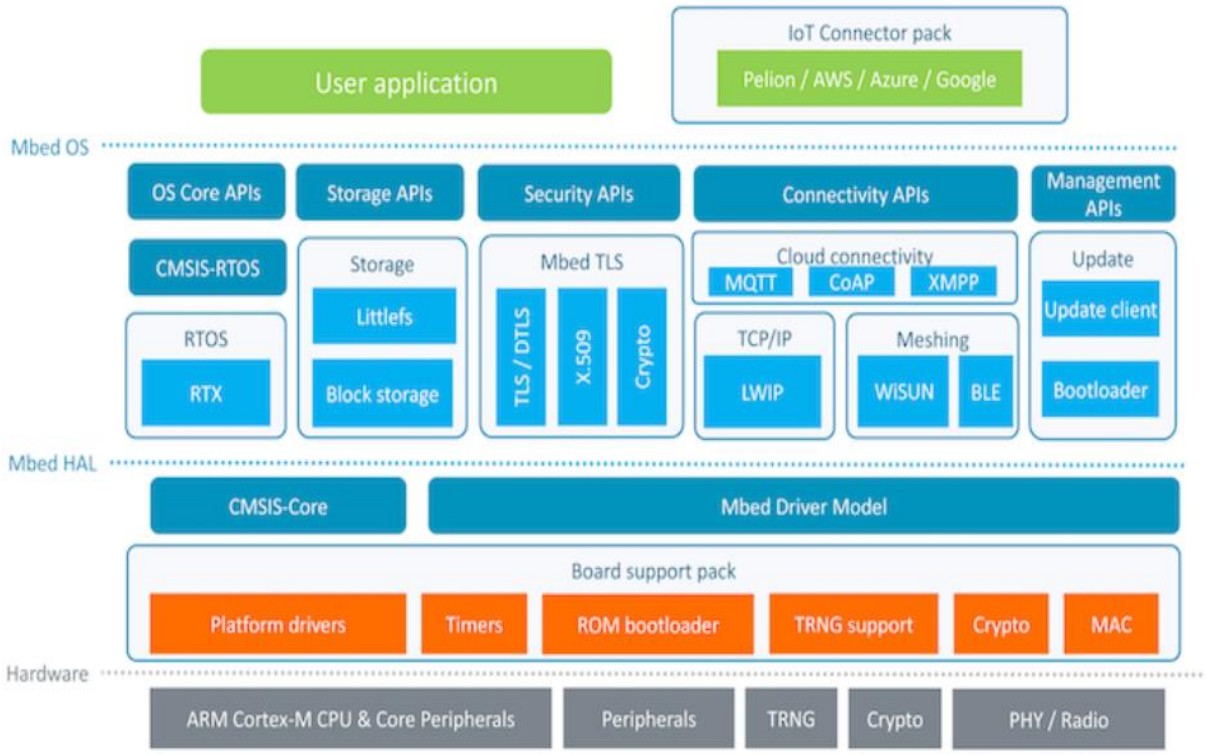
5. **Inter process/task communication** - manages the sharing of data, memory, and hardware resources among multiple tasks

6. **Defined stack usage** - each task is allocated a defined stack space, enabling predictable memory usage

7. **System management** - allows you to focus on application development rather than resource management (housekeeping)

1.7. RTOS on Mbed Platform:

- Mbed OS is an open-source operating system for Internet of Things (IoT) Cortex-M boards: low-powered, constrained and connected.
- Mbed OS provides an abstraction layer for the microcontrollers it runs on, so that developers can write C/C++ applications that run on any Mbed-enabled board.
- The **full profile** of Mbed OS is an RTOS which includes Keil RTX and all RTOS APIs, so it supports deterministic, multithreaded, real-time software execution.
- The RTOS primitives are always available, allowing drivers and applications to rely on threads, semaphores, mutex and other RTOS features. It also includes all **API**s by default, although you can remove unused ones at build time.



- The Mbed OS RTOS capabilities include managing objects such as threads, synchronization objects and timers. It also provides interfaces for attaching an application-specific idle hook function, reads the OS tick count and implements functionality to report RTOS errors.
- Mbed OS uses a hardware abstraction layer (HAL) to support the most common parts of a microcontroller, such as timers. This foundation facilitates writing applications against a common set of application programming interfaces (APIs); your device automatically includes necessary libraries and driver support for standard MCU peripherals, such as I2C, Bus and SPI.

- The HAL also serves as the starting point when adding support for new targets or features to existing targets.
- The structure of Mbed OS enables matching applications and storage systems.

Features and benefits of Mbed OS

Device and Component support

- With support for Mbed OS on a wide range of Arm Cortex-M based devices, developers can prototype IoT applications quickly on low-cost development boards. Simple USB drag and drop programming allows you to rapidly prototype without the need for expensive debug hardware.

Real Time Software Execution

- With an RTOS core based on the widely used open-source CMSIS-RTOS RTX, Mbed OS supports deterministic, multithreaded real time software execution.
- The RTOS primatives are always available, allowing drivers and applications to rely on features such as threads, semaphores and mutexes.

Open Source

- Released under an Apache 2.0 licence, you can use Mbed OS in commercial and personal projects with confidence.

Ease of Use

- With a modular libary structure, the necessary underlying support for your application will be automatically included on your device.
- By using the Mbed OS API, your application code can remain clean, portable and simple, whilst taking advantage of security and communications.

End to End Security

- It addresses security in device hardware, software, communication and in the lifecycle of the device itself:
- **Hardware Enforced Security** The combination of PSA Certified systems, hardware-enforced isolation with TrustZone technology and reference software from the Trusted Firmware-M project
- **Communications Security** The SSL, TLS, and standard protocols for securing communications on the internet can be included in Mbed project with a simple API.

Drivers and Support Libraries

- Driver support for a wide range of standard MCU peripherals is included in Mbed OS. This includes digital and analog IO, interrupts, port and bus IO, PWM, I2C, SPI and serial.
- The C libraries of each supported toolchain are also integrated into Mbed OS, including implementation of thread safety support.