

INDEX

1. BASIC NOMENCLATURE	2
2. OS	2
2.1 OS services	3
3. GPOS(general purpose operating system)	4
4. RTOS	5
4.1 Real-time systems	5
5. RTOS characteristics	6
5.1 Multi-tasking	6
5.2 Kernel	7
5.3 Deterministic time	7
5.4 User control	7
5.5 Reliability	7
5.6 Scheduling	7
5.6.1 Process Scheduling	8
5.6.2 Context Switching	9
6. RTOS Features	10

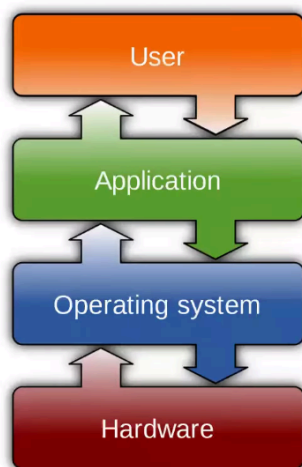
1. BASIC NOMENCLATURE

- **ALGORITHM** : logical procedure of a problem
- **PROGRAM** : real implement of algorithm
- **PROCESS** : instance of program(get required inputs to achieve set of outputs)

2. OS

- A program that provides abstraction of physical machine
 - o Interface to machine
 - o Each part of interface (service : provided by APIs)
 - o Access to physical resource of computing machine
 - o Provide abstract resources like
 - Physical resources : camera, phone
 - Virtual resources : files, pages in memory
- OS has many components
 - o APIs are of 2 types
 - System API
 - Library API (eg : printf(), read())
 - o Virtual memory , virtual file system
 - o Scheduler : The scheduler is a crucial component responsible for managing the execution of processes by determining which process runs at any given time.
 - o Device drivers

What is an Operating System



- It is a software layer between hardware and the application software , which
 - Has Direct Access to Hardware
 - Manage the Hardware according to predefined rules and policies
 - Hide hardware complexity from the Application Software
 - Offers Multitasking / Resource & Data Management

- OS use the device drivers to access hardware to avoid corruption of data

- Device drivers are used to perform same task (it provide all sequence of commands)

2.1 OS services

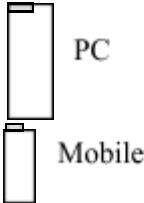
1. **Concurrency** : it means sequence manner of execution
 - Here we assume that all tasks are executing at same time but only one task is at running state at given tick time
 - Each task is a process
2. **Process management** : the registers , stack pointer , program counter details are maintained before context switching of tasks
3. **Memory protection** : one task is not allowed to access other task's memory
 - Example :
 - let us think there is one task performing $a+b$, and other task doing some other job
 - task t1 stores the value of variables a,b in some RAM location
 - so when task t2 with high priority comes in t1 is sent to blocked state at this the t2 comes to running state so at this time t2 must not allowed to access saved memory of t1 and corrupt it
4. **Multi-Threading** : a process can have multiple threads
5. **Context switching** : When the scheduler switches the CPU from executing one process to executing another, it saves the state of the current process and loads the state of the next process. This is known as a context switch.
6. **Synchronization** : the mechanism to know the way to send data from one task to other
 - Inter Process Communication : if there are 2 tasks and one task need the variable from other, but the tasks cannot access each other memory so using this communication we will create a link between them to make tasks get access.
7. **Resource Management** : involves managing system resources like CPU time, memory, and peripheral devices to ensure that tasks are executed efficiently and meet their deadlines
8. **File system** : how files are stored, disks, drivers
 - A hardware disk has blocks structures such that each block may have defined size eg : 1kB
 - So, we cannot read half of block from disk
 - If application asks for 2Mb storage in file system then it stores it 1kB wise blocks named from eg : let us consider blocks columns are named as A, B, C,... and rows as 1, 2, 3,... then it may check for availability of memory and assign these blocks 5A to 6A
 - Pendrive : in this the blocks has limited life cycle; each block can write only 1000 times atleast

- File system implements in such a way that it uses all required memory in blocks

9. **Communication** : It is a real-world communication to OS

3. GPOS(general purpose operating system)

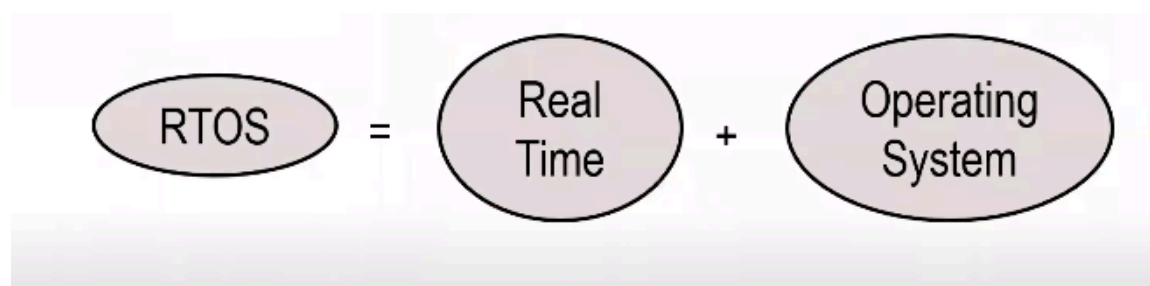
- It is a (software) intermediate layer between application software and hardware
- It takes all the hardware complexities away from the hardware
- It provide APIs used by application software to control hardware
- Example:

- o Windows
 - o Linux
 - o Mac OS
 - o Android
 - o IOS
 - o Embedded OS
- embedded systems
- 

Definition : an abstract layer between hardware and application software, that hides the hardware complexities from application software

4. RTOS

- It is designed to run the application with
 - o Very precise timing
 - o High degree of reliability
- Occupies less memory
- Uses defined APIs to interact with hardware through device drivers
- Supports variety of microprocessors
- It has many debugging tools (gdb)
- a specialized operating system designed to manage hardware resources and execute tasks in real-time. Unlike general-purpose operating systems (like Windows or Linux), an RTOS is optimized for applications where precise timing and predictable responses are critical. Examples include embedded systems, industrial automation, robotics, automotive systems, and telecommunications.



- **Real time** : it should be operated with time constraint means it must have a deadline
- Responds to external events in a timely fashion
- The need to meet deadlines
- Does not mean faster performance

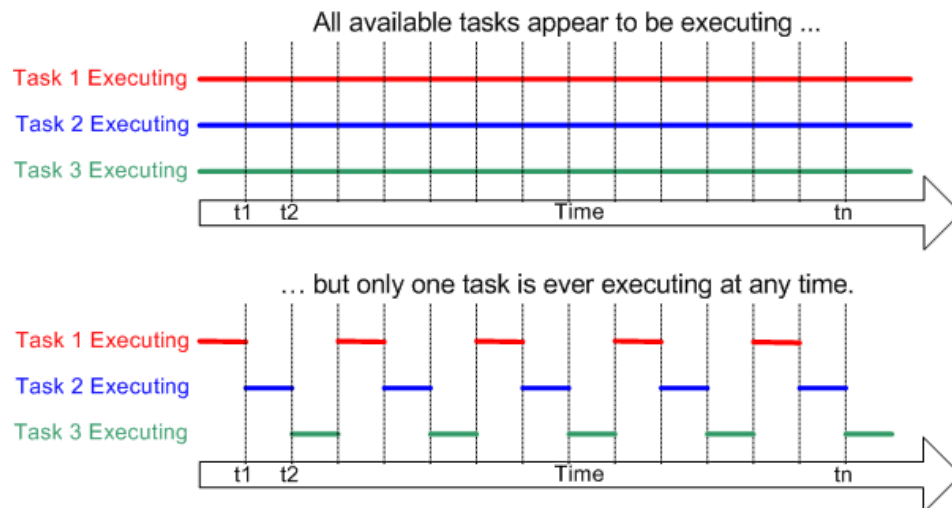
4.1 Real-time systems

1. **Hard** : meets deadline all the time (100%) . missing deadline even single time cause fatal consequences
Eg : air bag systems
2. **Soft** : does not result fatal consequences. It may lower the quality of service
Eg : DVD player
3. **Firm** : If a deadline is missed occasionally, the system does not fail. Results produced by a task after deadline are ignored

5. RTOS characteristics

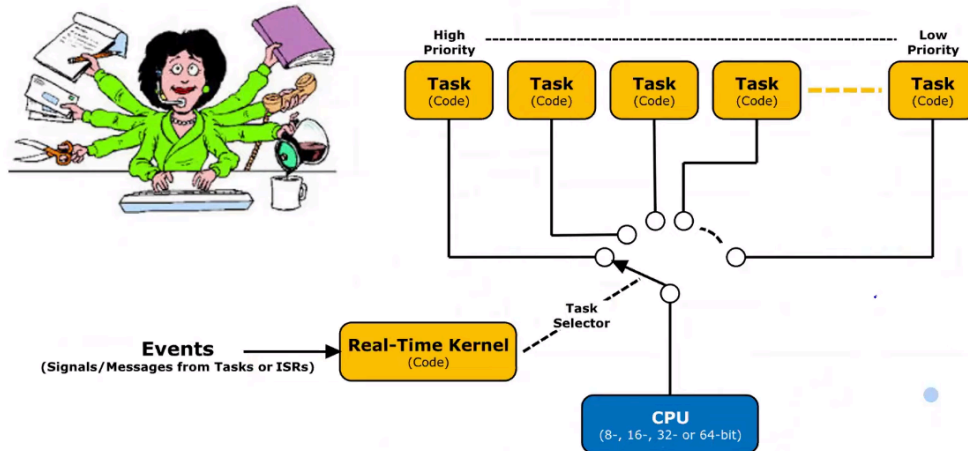
5.1 Multi-tasking

- Multiple super loops while(1) { }
- It allows execution of multiple tasks on single CPU
- All tasks execution as if they completely own entire CPU
- Operating systems such as Linux employ kernels that allow users access to the computer seemingly simultaneously. Multiple users can execute multiple programs apparently concurrently.
- Each executing program is a **task** (or thread) under control of the operating system. If an operating system can execute multiple tasks in this manner it is said to be **multitasking**.
- A multitasking operating system can create the illusion of concurrent execution by rapidly switching between tasks, even though a single core processor can only execute one task at a time. This is depicted by the diagram below which shows the execution pattern of three tasks with respect to time. The task names are color coded and written down the left hand. Time moves from left to right, with the coloured lines showing which task is executing at any particular time. The upper diagram demonstrates the perceived concurrent execution pattern, and the lower the actual multitasking execution pattern.



Multitasking Systems

In a multitasking system, the CPU time is distributed amongst different tasks.



5.2 Kernel

- Share processor time to tasks in predetermined manner / priority
- **Kernel** : It is central component of an operating system that manages operations of computer and hardware. It basically manages operations of memory and CPU time.

- o Kernel loads first into memory when an operating system is loaded and remains into memory until operating system is shut down again. It is responsible for various tasks such as disk management, task management, and memory management.
- o Kernel has a process table that keeps track of all active processes. Process table contains a per process region table whose entry points to entries in region table.
- o Kernel loads an executable file into memory during 'exec' system call'.
- o **Objectives of Kernel :**
 - ☐ To establish communication between user level application and hardware.
 - ☐ To decide state of incoming processes.
 - ☐ To control disk management.
 - ☐ To control memory management.
 - ☐ To control task management.

5.3 Deterministic time

- responds within the mentioned deadline time

5.4 User control

- we can mention the priority of tasks

5.5 Reliability

- long journey tasks runs with same speed even after many number of executions

5.6 Scheduling

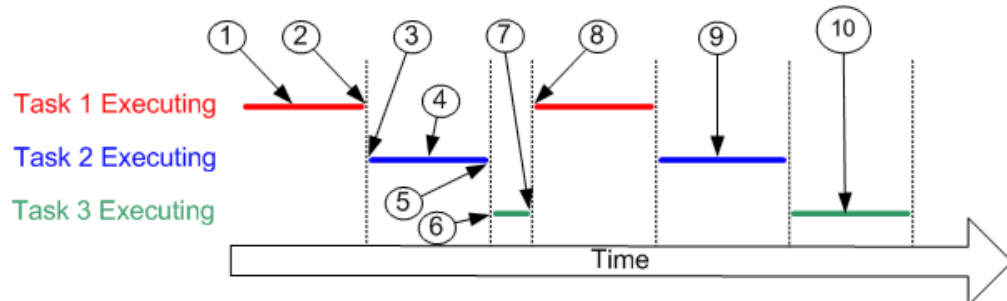
- **Scheduler :** The scheduler is a crucial component responsible for managing the execution of processes by determining which process runs at any given time.

5.6.1 Process Scheduling

The process of deciding which process gets CPU time and for how long. This involves managing the CPU's time among various processes to ensure fair and efficient utilization.

The **scheduling policy** is the algorithm used by the scheduler to decide which task to execute at any point in time. The policy of a (non real time) multi user system will most likely allow each task a "fair" proportion of processor time..

In addition to being suspended involuntarily by the kernel a task can choose to suspend itself. It will do this if it either wants to delay (**sleep**) for a fixed period, or wait (**block**) for a resource to become available (eg a serial port) or an event to occur (eg a key press). A blocked or sleeping task is not able to execute, and will not be allocated any processing time.

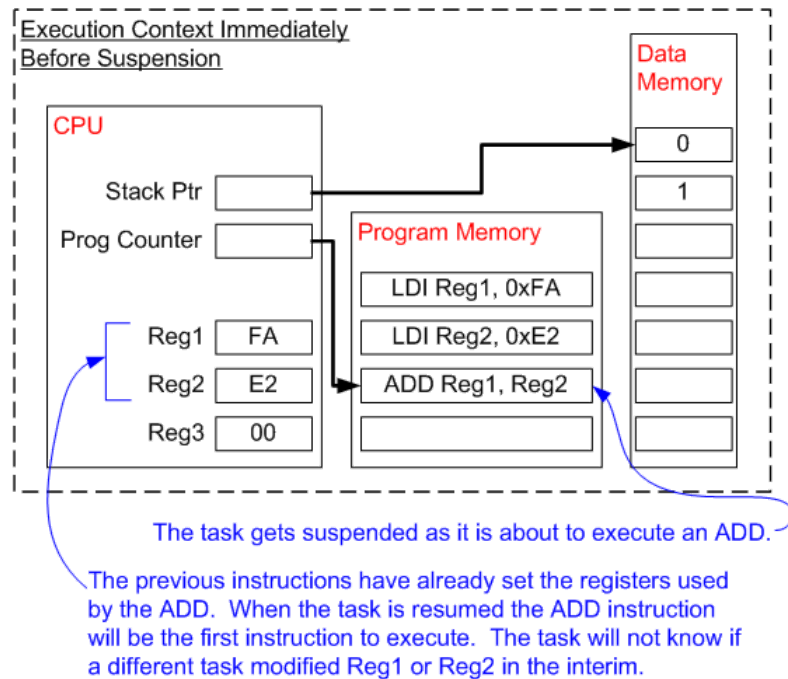


Referring to the numbers in the diagram above:

- At (1) task 1 is executing.
- At (2) the kernel suspends (swaps out) task 1 ...
- ... and at (3) resumes task 2.
- While task 2 is executing (4), it locks a processor peripheral for its own exclusive access.
- At (5) the kernel suspends task 2 ...
- ... and at (6) resumes task 3.
- Task 3 tries to access the same processor peripheral, finding it locked task 3 cannot continue so suspends itself at (7).
- At (8) the kernel resumes task 1.
- The next time task 2 is executing (9) it finishes with the processor peripheral and unlocks it.
- The next time task 3 is executing (10) it finds it can now access the processor peripheral and this time executes until suspended by the kernel.

5.6.2 Context Switching

- When the scheduler switches the CPU from executing one process to executing another, it saves the state of the current process and loads the state of the next process. This is known as a context switch.
- As a task executes it utilizes the processor / microcontroller registers and accesses RAM and ROM just as any other program. These resources together (the processor registers, stack, etc.) comprise the task execution **context**.



- o A task is a sequential piece of code - it does not know when it is going to get suspended (swapped out or switched out) or resumed (swapped in or switched in) by the kernel and does not even know when this has happened.
- o Consider the example of a task being suspended immediately before executing an instruction that sums the values contained within two processor registers. While the task is suspended other tasks will execute and may modify the processor register values. Upon resumption the task will not know that the processor registers have been altered – if it used the modified values the summation would result in an incorrect value.
- o To prevent this type of error it is essential that upon resumption a task has a context identical to that immediately prior to its suspension. The operating system kernel is responsible for ensuring this is the case – and does so by saving the context of a task as it is suspended. When the task is resumed its saved context is restored by the operating system kernel prior to its execution. The process of saving the context of a task being suspended and restoring the context of a task being resumed is called **context switching**.

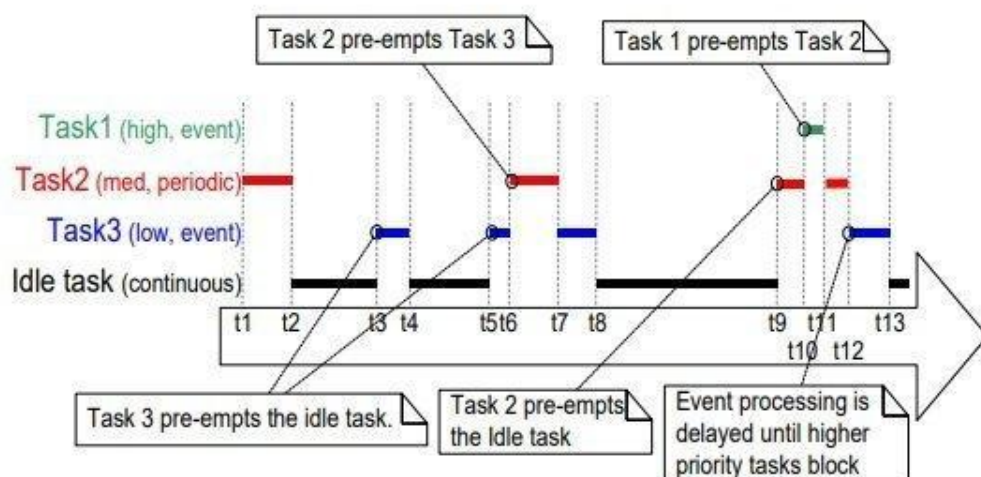
6. RTOS Features

1. Pre-emptive scheduling

- Scheduling is the software deciding which task to run at what time. FreeRTOS has two modes of operation when it comes to handling task priorities: With and without preemption. Which mode to use can be set in the configuration file. When using the mode without preemption, also called cooperative mode, it is up to the developer to make tasks

that yield the CPU through the use of blocking functions and the **taskYIELD()** function.

- When using a preemptive scheduler, a task will automatically yield the CPU when a task of higher priority becomes unblocked. However, there is one exception: When a higher priority task blocks from an ISR, the **taskYIELD_FROM_ISR()** function has to be called at the end of the ISR for a task switch to occur.
- If **configUSE_TIME_SLICING** is set to 1, the scheduler will also preempt tasks of equal priority at each time the tick occurs. Time slicing is not available in cooperative mode.
- To allow pre-emptive scheduling we must set **configUSE_PREEMPTION** to 1
- In both modes, the scheduler will always switch to the highest priority unblocked task. If there are multiple tasks unblocked with the same priority, the scheduler will choose to execute each task in turn. This is commonly referred to as round robin scheduling.
- In the preemptive mode, higher priority tasks are immediately switched to when they get unblocked. In the cooperative mode, the release of a semaphore might unblock a higher priority task, but the actual task switch will only happen when the currently executing task calls the **taskYIELD()** function or enters a blocking state.



2. Fast context switching (one task to other)
3. Small size : an executable programmable file creates a object file
 - Size of object file is your program size
 - So that it accommodate in less memory
4. Ability to respond to external interrupts quickly
5. Multi-tasking with inter process communication such as semaphore
6. Minimization of interval during which interrupts are disabled
7. **Tick time** : the minimum time given to application by OS before next decision

- Example : $t = 30\text{ms}$, at every 30ms the scheduler checks for the tasks to make decision on scheduling tasks