What Are Real Time Operating Systems?

- **RTOS** is an operating system that supports real-time applications by providing logically correct result within the ***deadline*** required.
- Basic Structure is similar to regular OS but, in addition, it provides mechanisms to allow real time scheduling of tasks

category of RTOS

1. Hard Real Time Systems

- Hard real time operating systems can never miss their deadline, otherwise if they do, they may result with calamitous consequences. If the function doesn't meet its deadline, the system will be considered as a failure. The usefulness of a result produced by a hard real time system decreases significantly if it is delayed in its delivery.
- An example of hard real time systems includes flight controls and medical critical care systems.

2. Firm Real Time Systems

- These types of systems must also follow a deadline, however in the case they do miss it, the overall impact may not be disastrous, but it could cause undesired effects to the system, for instance, like a huge reduction in quality of a product.
- Example of this would be various types of multimedia applications.

3. Soft Real Time Systems

- With soft real time systems, the deadline is not extremely strict to the millisecond like hard real time systems.
- These systems can occasionally miss the deadline with some acceptably low probability. Furthermore, if the deadline is missed with these systems, there is no disastrous consequence that will result from it.
- However as with hard real time systems, the usefulness of the result decreases over time as the delay increases.
- An example of soft real time systems includes telephone switches, and online transaction system

Functionality and features of RTOS

- To be extremely focused on the environment.
- Requiring special/complex algorithms for speedier processing.
- RTOS can respond and interact with analog (ADC).
- RTOS are considered to be “reactive” in a harsh environment.
- RTOS are business compatible and user friendly.

Parameters for selecting IoT OS

- Footprint: Since devices are constraint, we expect OS to have low memory, power and processing requirements. The overhead due to the OS should be minimal.
- Scalability: OS must be scalable for any type of device. This means developers and integrators need to be familiar with only one OS for both nodes and gateways.
- Portability: OS isolates applications from the specifics of the hardware. Usually, OS is ported to different hardware platforms and interfaces to the board support package (BSP) in a standard way, such as using POSIX calls.
- Modularity: OS has a kernel core that's mandatory. All other functionality can be included as add-ons if so required by the application.
- Connectivity: OS supports different connectivity protocols, such as Ethernet, Wi-Fi, BLE, IEEE 802.15.4, and more.
- Security: OS has add-ons that bring security to the device by way of secure boot, SSL support, components and drivers for encryption.
- Reliability: This is essential for mission-critical systems. Often devices are at remote locations and have to work for years without failure. Reliability also implies OS should fulfil certifications for certain applications.

"Friends Should Play Minecraft, Creating Realistic Structures"

"Super Pigs Make Cool Sounds, Really Satisfying"

Real Time Operating System Architecture

- There are 6 main components in RTOS, and they are:
  - Scheduler
  - Symmetric Multiprocessing
  - Function Library
  - Memory Management
  - Fast Dispatch Latency
  - User-Defined Data Objects and Classes

PRO CONS

Advantages

- lowers Maximum energy Consumption/ Improved Efficiency
- Task Shifting
- Focus on Application ; not bloatware
- Error Free(not possible)
- 24/7 systems

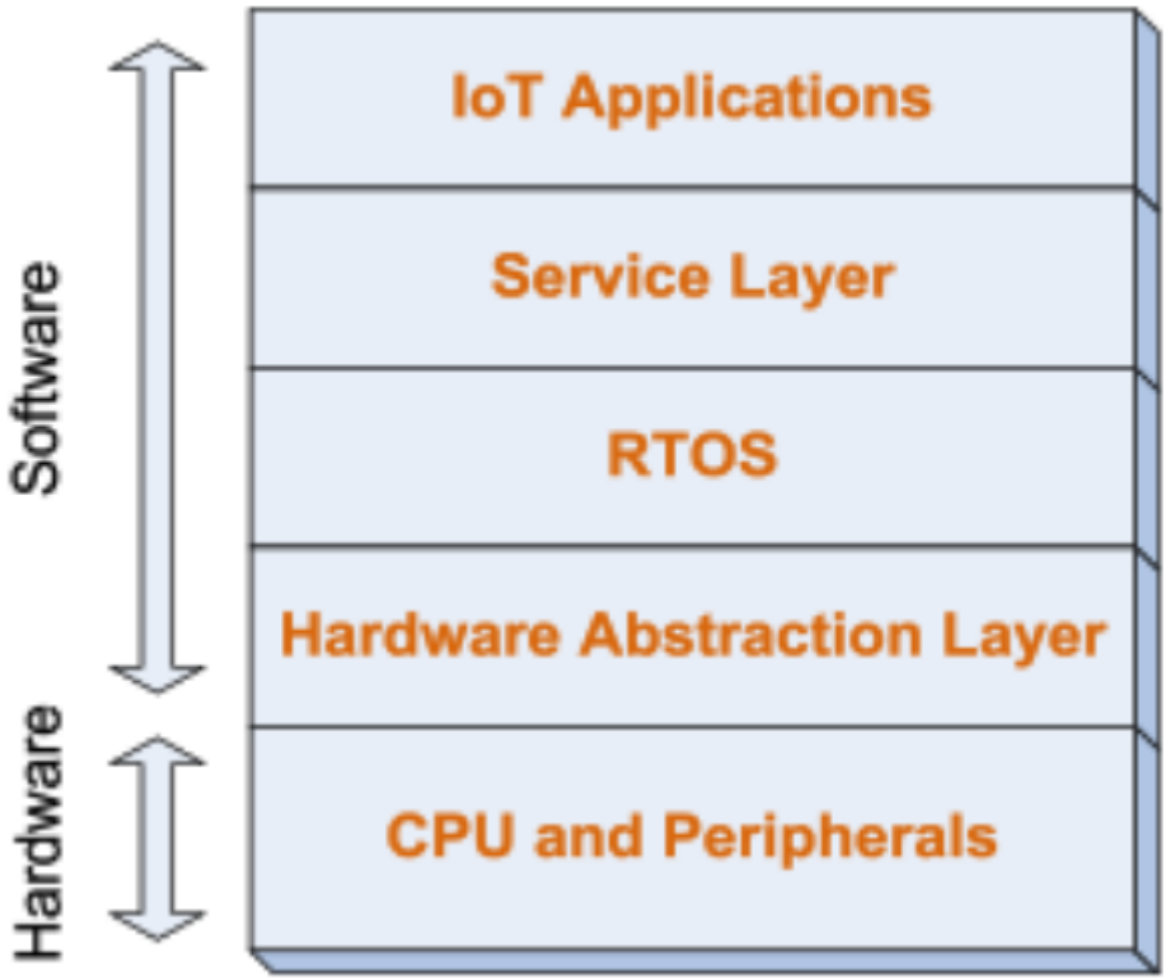
- Priority Based Scheduling
- Modularity
- Easier Testing
- Code Reuse

Disadvantages

- Limited Tasks
- Uses Heavy System Resources
- Low Amount of Multi-Tasking
- Complex Algorithms
- Complex Structure
- Expensive

General Architecture of The RTOS for the IoT

- A general RTOS architecture is shown in Fig., where the IoT applications are the top layer, they use RTOS services in development.
- The hardware abstraction layer (HAL) is the hardware related drivers



Major RTOS used

- Contiki
- TinyOS

- RIOT
- Nano-RK
- LiteOS
- MantisOS
- SOS OS
- Resilient, Expandable, and Threaded OS (RETOS)

LiteOS

- was developed in 2008 and since then has been fighting for popularity in this area because it has a unique modular architecture and kernel; it uses a programming language known as LiteC++
- It implements both event and threads in its programming model, however, the scheduling of tasks is “run to completion”.
- It is an open source OS for IoTs and can be considered for various applications in the IoTs field.

Mantis OS

- Mantis OS is another OS released in 2005 and licensed under BSD.
- Mantis OS has a different layered architecture and implements threads as its programming model system.
- It is written in C language.
- It has a method called “comm” for networking support in IoTs.
- It implements resource sharing as semaphores.