
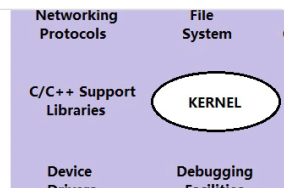


# RTOS

## Understanding Real Time Operating System (RTOS) and How to use it for your next Embedded Design

Real time operating system popularly known as RTOS provides controller with the ability to respond to input and complete tasks within a specific period of time based on priority

 <https://circuitdigest.com/article/understanding-rtos-and-how-to-use-it-for-embedded-systems>



Real-time operating system (RTOS) is an operating system that is specifically designed to respond to events or data within a specific timeframe, which is known as real-time computing. Unlike a general-purpose operating system, an RTOS has a deterministic behavior and can execute tasks and processes within strict deadlines.

An RTOS is built on the architecture of an operating system and provides features such as multitasking, multithreading, memory management, and device drivers. It enables multiple programs to execute concurrently, which can lead to improved system performance and better resource utilization.

The key advantage of using an RTOS is its ability to manage the execution of time-critical tasks. In many embedded systems, tasks need to be completed within a specific time frame to prevent system failure or data loss. An RTOS provides a way to manage the execution of these tasks by prioritizing them and ensuring that they are completed within their respective deadlines.

One of the main features of an RTOS is its scheduler, which decides which task to execute next based on its priority and the available resources. The scheduler uses advanced algorithms to manage the execution of tasks, and it can switch between tasks quickly to meet their deadlines.

Another advantage of an RTOS is its flexibility in adding new features or modifying existing ones. The design of an RTOS allows for the addition of new drivers or modules without disrupting the existing system. This flexibility enables developers to add new features and improve the performance of their systems over time.

RTOS is widely used in various applications such as automotive, aerospace, medical devices, and industrial automation. In these applications, an RTOS can manage the execution of critical tasks such as engine control, flight control, patient monitoring, and process control.

In summary, an RTOS is a specialized operating system that provides real-time computing capabilities and manages the execution of time-critical tasks. It offers features such as multitasking, scheduling, and memory management, and it is widely used in various applications that require deterministic behavior and reliable performance.

S.No	Operating System	Real time System
1	Time sharing is the basis of execution of processes in operating system	Processes are executed on the basis of the <b>order of their priority</b>
2	Operating system acts as an interface between the hardware and software of a system	Real time system is designed to have its execution for the <b>real world problems</b>
3	Managing memory is not a critical issue when it comes to execution of operating system	Memory management is difficult as based on the real time issue memory is allocated , which itself is critical
4	Applications: Office , Data centers , System for home etc	Applications: Controlling aircraft or nuclear reactor , scientific research equipments
5	Examples : Microsoft Windows , Linux ,OS	Examples: Vx Works ,QNX , Windows CE

## TYPES OF RTOS

There are three types of real-time operating systems (RTOS): hard real-time operating system, soft real-time operating system, and firm real-time operating system.

### 1. Hard Real-Time Operating System:

A hard real-time operating system is used in applications where tasks have to be completed within a specific time frame, and failure to meet the deadline could be catastrophic (danger). In such systems, the operating system must guarantee that the task will be completed within the given time frame. A common example of a hard real-time system is a flight control system, where every task must be completed on time to ensure the safety of the flight.

Features of Hard Real-Time Operating System:

- Tasks must be completed on time.
- Failure to meet the deadline is fatal.
- Guaranteed worst-case response time.
- Can lead to system failure if tasks are not completed on time.

### 2. Soft Real-Time Operating System:

A soft real-time operating system is used in applications where tasks need to be completed quickly but missing the deadline is not catastrophic. In such systems, the operating system tries to complete the task as fast as possible, but if it misses the deadline, it is not considered a system failure. An example of a soft real-time system is an online database where data needs to be retrieved quickly, but it is not critical if it takes a few extra seconds.

Features of Soft Real-Time Operating System:

- Tasks should be performed as fast as possible.
- Late completion of tasks is undesirable but not fatal.
- There is a possibility of performance degradation.
- Cannot lead to system failure.

3. Firm Real-Time Operating System:

A firm real-time operating system is used in applications where tasks need to be completed within a specific time frame, but missing the deadline is tolerable to some extent. In such systems, the operating system tries to complete the task within the given time frame, but if it misses the deadline, it is still acceptable as long as it does not significantly affect the system's performance. An example of a firm real-time system is a robot arm used to pick objects, where the arm must pick up the object within a reasonable time frame, but a slight delay is acceptable.

Features of Firm Real-Time Operating System:

- Tasks should be completed within a specific time frame.
- Delayed completion of tasks is tolerable to some extent.
- There is a possibility of performance degradation.
- Cannot lead to system failure.



Firewalls can be implemented as hardware devices or software programs running on a server or client machine. They are designed to prevent unauthorized access and malicious traffic from entering or leaving a network.

## BENEFITS OF RTOS

1. No firewall issues: When using an RTOS, the system is designed to have limited external connections and communications, which helps to mitigate potential security vulnerabilities that could be exploited through a network connection. This reduces the need for complex firewall configurations and reduces the risk of cyber-attacks.
2. Low bandwidth for enhanced performance: RTOS typically have a smaller memory footprint and require fewer processing resources than general-purpose operating systems. This means they can operate on low-power, low-bandwidth devices without compromising on performance, making them ideal for applications such as embedded systems.
3. Improved security and privacy: By using a real-time operating system, it is possible to design systems that provide enhanced security and privacy features. These include secure booting, memory protection, and access controls, all of which help to prevent unauthorized access to the system or sensitive data.
4. Low cost, due to reduction in hardware and software components used for development: By utilizing a real-time operating system, it is possible to reduce the complexity of the overall system design, which can help to reduce costs associated with hardware and software development. Additionally, by utilizing an RTOS with a wide range of pre-existing software modules and libraries, developers can avoid the need to write code from scratch, further reducing development costs.

## MAJOR ISSUES

Real-Time Operating Systems (RTOS) are designed to provide guaranteed response times and high reliability in embedded systems, where timing and predictability are critical. However, like any other software system, RTOS has its own set of issues and limitations.

One of the major issues with RTOS is related to interrupts. Interrupts are a mechanism used by programs to halt the currently executing code and divert the control flow to handle a higher-priority event. In an RTOS, quick response time is essential, and therefore **it is recommended to disable interrupts for the shortest possible duration**. This can be a challenge in systems with a large number of interrupts or where the processing of interrupts requires a significant amount of time.

Another issue with RTOS is related to the size of the kernel. The kernel of an RTOS is responsible for managing system resources, scheduling tasks, and providing real-time response. Since the kernel needs to respond to various events in real-time, it is essential to keep its size small so that it can fit properly within the available ROM (Read-Only Memory). This can limit the number of features and functionalities that can be included in the system.

Finally, RTOS lacks the concept of virtual memory. Virtual memory is a mechanism that enables the operating system to use more memory than is physically available by temporarily transferring pages of data from RAM to disk storage. However, since RTOS is designed to operate in resource-constrained environments, it does not have the luxury of virtual memory. As a result, **sophisticated features that require virtual memory, such as memory management and dynamic loading of code, need to be removed or implemented in a limited way.**

In conclusion, RTOS is a specialized type of operating system that offers many benefits, such as guaranteed response times, high reliability, and low overhead. However, it also has its own set of limitations, such as issues related to interrupts, kernel size, and lack of virtual memory support. As with any technology, it is important to carefully evaluate the benefits and limitations of RTOS before deciding to use it in a particular application.



**The kernel is the central component of an operating system that manages the system's resources and provides a layer of abstraction between the hardware and software. In an operating system, the kernel is responsible for managing the CPU, memory, I/O devices, and other system resources. It also provides services such as process management, memory management, and file system management.**



**In an RTOS, the kernel is designed specifically for real-time applications and provides features such as deterministic scheduling, interrupt handling, and message passing that enable the system to meet strict timing requirements. The kernel in an RTOS is typically designed to be lightweight and efficient, with minimal overhead, to ensure that it can respond quickly to real-time events.**



**In both OS and RTOS, the kernel is essential for managing system resources and providing a layer of abstraction that allows applications to interact with the hardware without having to deal with the low-level details of hardware access.**

## **HOW TO USE RTOS**

### **TORNADO (JUST AN INFO NOT REQD TO KNW ABT IT IN DETAIL)**

Tornado is an integrated environment for developing real-time embedded RTOS applications, primarily based on the VxWorks operating system. It provides a complete development environment for building and debugging real-time systems and applications. Tornado includes three main elements: VxWorks, application building tools, and an integrated development environment (IDE).

1. VxWorks is a real-time operating system designed for embedded systems, which provides deterministic and predictable performance. VxWorks is a popular RTOS used in a wide range of applications, including aerospace, defense, industrial automation, and telecommunications.

2. The application building tools in Tornado include a compiler, linker, and associated programs for building applications for VxWorks. The tools support a range of programming languages, including C, C++, and assembly language.
3. The integrated development environment (IDE) in Tornado provides a complete set of tools for managing, debugging, and monitoring VxWorks applications. The IDE includes a graphical user interface (GUI) for managing projects, editing source code, and debugging applications. It also provides tools for monitoring and analyzing system performance and identifying and resolving issues.

Overall, Tornado provides a complete and integrated development environment for building real-time embedded applications using VxWorks. It is a popular choice for developers working on applications that require real-time performance and determinism, such as aerospace and defense systems, telecommunications equipment, and industrial automation systems.

## **FREE RTOS**

FreeRTOS is a popular open-source real-time operating system kernel that can be used for microcontrollers and small microprocessors. It is designed to be small, portable, and easy to use. FreeRTOS provides a range of features, including task scheduling, inter-task communication, and synchronization primitives. It also includes a heap memory management scheme, which enables dynamic memory allocation and deallocation.

In addition to the kernel, FreeRTOS also provides a variety of libraries and utilities, including a TCP/IP stack, USB stack, and file system. It can be integrated with many development environments, including the Eclipse IDE.

One of the main features of FreeRTOS is its task scheduler, which allows developers to schedule multiple tasks to run on a single processor. The scheduler is based on a preemptive priority-based algorithm, which ensures that higher-priority tasks are executed before lower-priority ones. Developers can also specify the task stack size and priority, allowing them to optimize the system's performance and responsiveness.

FreeRTOS also provides inter-task communication and synchronization mechanisms, which enable tasks to share data and coordinate their activities. This includes mechanisms such as semaphores, mutexes, and message queues, which ensure that tasks operate in a synchronized manner and avoid race conditions and other issues that can arise in concurrent systems.

In addition to task management and synchronization, FreeRTOS also includes a heap memory management scheme, which enables dynamic memory allocation and deallocation. This is useful for applications that require dynamic memory allocation, such as those that work with variable-sized data structures or network protocols.

FreeRTOS also includes a variety of libraries and utilities, including a TCP/IP stack, USB stack, and file system. These libraries and utilities are designed to be lightweight and portable, making them ideal for use in embedded systems and other resource-constrained environments.

FreeRTOS is also designed to be highly portable, with support for a wide range of processors and microcontrollers. It can be easily integrated with many different development environments, including the Eclipse IDE, which provides a powerful and flexible development environment for embedded systems and other real-time applications.

FreeRTOS can be used with a wide range of microcontrollers and development boards, making it highly versatile and adaptable to various hardware platforms. It also supports a variety of development environments and tools, including the Eclipse IDE, Keil  $\mu$ Vision, and IAR Embedded Workbench. This means that developers can choose the development environment that they are most comfortable with and that best suits their project needs.

The ability to integrate with different development environments also makes it easier for developers to create and debug their applications. FreeRTOS provides a range of debugging features, including stack overflow protection, memory leak detection, and run-time error checking. These features can be easily accessed through the integrated development environment, making it easier to identify and resolve any issues that may arise during development.

Furthermore, FreeRTOS also provides a range of libraries and utilities that can be used to enhance the functionality of the operating system. These include a TCP/IP stack, USB stack, file system, and more. These libraries can be easily

integrated with the development environment, **allowing developers to quickly and easily add new features to their applications without having to write all the code from scratch.**

Overall, the ability to easily integrate with a range of development environments and tools makes FreeRTOS a flexible and powerful real-time operating system for embedded systems development.

## **FREE RTOS IN DETAIL**



A microcontroller is a small computer on a single integrated circuit chip that is designed to control specific devices or perform specific tasks. It contains a processor core, memory, and input/output peripherals, all on a single chip. Microcontrollers are widely used in embedded systems such as automobiles, consumer electronics, medical devices, and industrial control systems. They are designed to be low power, cost-effective, and easy to program, and they typically run at slower clock speeds than desktop or laptop computers. Some examples of popular microcontroller families include the AVR, PIC, and ARM Cortex-M series.

FreeRTOS is a popular open-source real-time operating system kernel that can be used for microcontrollers and small microprocessors. It is developed by Richard Barry and the FreeRTOS team and is owned by Real Time Engineers Ltd. One of the biggest advantages of FreeRTOS is its platform-independent behavior in terms of hardware. This means that the C code used to execute an operating system can run on various platforms with different architectures. Thus, irrespective of whether one is using an 8051 microcontroller or a latest ARM microcontroller, the code and its execution process will be similar for both.

FreeRTOS provides many other benefits over other RTOS operating tools, such as easier testing, code reusability, lesser idle time, and easy maintainability. Additionally, the basic Kernel present within FreeRTOS makes it accessible to use for various applications. One can easily attach expanded modules to the operating system to get more applications, making FreeRTOS even more powerful.

### **EXAMPLE: One example of using FreeRTOS is combining it with Nabto**

Nabto is a platform that allows for secure remote access to devices over the internet. It enables the device to be accessed from a web browser or mobile app, and allows for data to be transmitted securely.

By combining FreeRTOS with Nabto, it is possible to create a real-time system that can be remotely accessed and controlled. For example, let's consider a home automation system that is using FreeRTOS to manage the various tasks and processes. With Nabto integrated into the system, the user can remotely access the home automation system from their web browser or mobile app and perform various actions such as turning on lights, adjusting the temperature, or even locking the doors.

The FreeRTOS kernel provides the necessary real-time capabilities for managing the different tasks and processes involved in the home automation system. With Nabto, the user can securely access and control the system remotely, without compromising its real-time capabilities.

Overall, the combination of FreeRTOS and Nabto enables the creation of powerful and flexible real-time systems that can be accessed and controlled remotely. This can have a wide range of applications, from home automation systems to industrial control systems, and beyond.