2.3 Need For RTOS:

What is an RTOS?

Simply put, an RTOS is a piece of software designed to efficiently manage the time of a central processing unit (CPU). This is especially relevant for embedded systems when time is critical.

The key difference between an operating system such as Windows and an RTOS often found in embedded systems is the response time to external events. An ordinary OS provides a non-deterministic response to events with no guarantee with respect to when they will be processed, albeit while trying to stay responsive. The user perceiving the OS to be responsive is more important than handling underlying tasks. On the other hand, an RTOS' goal is fast and more deterministic reaction.

Developers used to OS's such as Windows or Linux will be quite familiar with the characteristics of an embedded RTOS. They are designed to run in systems with limited memory, and to operate indefinitely without the need to be reset.

Because an RTOS is designed to respond to events quickly and perform under heavy loads, it can be slower at big tasks when compared to another OS.

RTOS scheduling

An RTOS is valued for how quickly it can respond and in that, the advanced scheduling algorithm is the key component.

The time-criticality of embedded systems vary from soft-real time washing machine control systems through hard-real time aircraft safety systems. In situations like the latter, the fundamental demand to meet real-time requirements can only be made if the OS scheduler's behavior can be accurately predicted.

Many operating systems give the impression of executing multiple programs at once, but this multi-tasking is something of an illusion. A single processor core can only run a single thread of execution at any one time. An operating system's scheduler decides which program, or thread, to run when. By rapidly switching between threads, it provides the illusion of simultaneous multitasking.

The flexibility of an RTOS scheduler enables a broad approach to process priorities, although an RTOS is more commonly focused on a very narrow set of applications. An RTOS scheduler should give minimal interrupt latency and minimal thread switching overhead. This is what makes an RTOS so relevant for time-critical embedded systems.

The use of RTOS in embedded designs

Many embedded programmers shy away from using an RTOS because they suspect that it adds too much complexity to their application, or it is simply unknown territory. An RTOS typically requires anything up to 5% of the CPU’s resources to perform its duties. While there will always be some resource penalties, an RTOS can make up for it in areas such as simplified determinism, ease of use though HW abstraction, reduced development time and easier debugging.

Using an RTOS means you can run multiple tasks concurrently, bringing in the basic connectivity, privacy, security, and so on as and when you need them. An RTOS allows you to create an optimized solution for the specific requirements of your project.

Additional benefits

In addition to the above, an RTOS such as Zephyr comes with other helpful features too.

There is a powerful logging system, capable of outputting to multiple backends such as UART or RTT. Logs have different priorities, are timestamped and can easily be filtered by module or criticality level.

There are numerous tools to ease debugging, such as CPU usage monitor, stack sentinel, error handlers and support for PC tools which lets the user view thread execution and scheduling, for example.

Of course, Zephyr also supports established OS features such as mutexes, mem allocation services as well as protection, thread synchronization and data passing. Of course, having an array of ready-made drivers for a selection of components is also a great feature. It lets developers focus on the application at hand instead of making and testing low level drivers.

After some acclimatization, I strongly believe that most embedded developers will prefer using an RTOS over bare metal programming.