Most complex real-time systems require a number of tasks (or programs) to be processed almost at the same time. For example, consider an extremely simple real-time system which is required to flash an LED at required intervals, and at the same time monitor for a key input from a keyboard. One solution would be to scan the keyboard in a loop at regular intervals while flashing the LED at the same time. Although this approach may work for a simple example, in most complex real-time systems a multitasking approach should be implemented.

The term multitasking means that several tasks (or programs) are processed in parallel on the same CPU. However, it is not possible for several tasks to run on a single CPU at the same time. Therefore, task switching is done where the tasks share the CPU time. In many applications, tasks cannot run independently of each other and they are expected to co-operate in some way. For example, the execution of a task may depend upon the completion of another, or a task may need some data from another. In such circumstances the tasks involved must be synchronized using some form of inter-task communication method.

Real-time systems are time responsive systems where the CPU is never overloaded. In such systems, tasks usually have priorities which are obeyed strictly. A task with a higher priority can grab the CPU from a lower-priority task and then use the CPU exclusively until it releases the CPU. When the higher-priority task completes its processing, or if it is waiting for a resource to become available, then the lower priority task can grab the CPU and resume processing from the point where it was interrupted. Real-time systems are also expected to react to events as quickly as possible. External events are usually processed using external interrupts and the interrupt latency of such systems is expected to be very short so that the interrupt service routine is executed as soon as an interrupt occurs.

- Without a multitasking kernel, multiple tasks can be executed in a loop, but this approach results in very poorly controlled real-time performance where the execution times of the tasks cannot be controlled.
- It is possible to code the various tasks as interrupt service routines. This may work in practice, but if the application has many tasks then the number of interrupts grow, making the code less manageable.
- A multitasking kernel allows new tasks to be added or some of the existing tasks to be removed from the system without any difficulty.
- The testing and debugging of a multitasking system with a multitasking kernel is easy compared to a multitasking system without a kernel.
- Memory is better managed using a multitasking kernel.
- Inter-task communication is easily handled using a multitasking kernel.
- Task synchronization is easily controlled using a multitasking kernel.
- CPU time is easily managed using a multitasking kernel.
- Most multitasking kernels provide memory security where a task cannot access the memory space of another task.

- Most multitasking kernels provide task priority where higher priority tasks can grab the CPU and stop the execution of lower priority tasks. This allows important tasks to run whenever it is required.

An RTOS (Real-Time Operating System) is a program that manages system resources, schedules the execution of various tasks in a system, synchronizes the execution of tasks, manages resource allocations, and provides inter-task communication and messaging between the tasks. Every RTOS consists of a kernel that provides the low-level functions, mainly the scheduling, task creation, inter-task communication, resource management etc. Most complex RTOSs also provide file-handling services, disk read-write operations, interrupt servicing, network management, user management, etc.

A task is an independent thread of execution in a multitasking system, usually with its own local set of data. A multitasking system consists of several tasks, each running its own code, and communicating and synchronizing with each other in order to have access to shared resources. Perhaps the easiest example of a multitasking system is where there are say 3 LEDS and each LED flashes at a different rate. The programming of such a simple system without a multitasking kernel could be a complicated task. We will see in this Appendix how a multitasking operating system such as the FreeRTOS can be used to share the MCU resources.

The simplest RTOS consists of a scheduler that determines the execution order of the tasks in the system. Each task has its own context consisting of the state of the CPU and its associated registers. The scheduler switches from one task to another one by performing a context switching where the context of the running task is stored and the context of the next task is loaded appropriately so that execution can continue properly with the next task. The time taken for the CPU to perform context switching is known as the context switching time and is usually negligible compared to the actual execution times of the tasks.

FreeRTOS is a real-time operating system that runs on many high-end microcontrollers, including the STM32 family. STM32CubeIDE has bundled FreeRTOS software, although we will not be making calls to FreeRTOS directly. ARM has created the CMSIS-RTOS library, which allows us to make calls to an underlying RTOS such as the FreeRTOS, i.e. we will be making calls to CMSIS-RTOS (version 2) in order to control the underlying FreeRTOS.
FreeRTOS and multitasking is a very large topic and requires several books to explain all of its features. There are several books, tutorials and application notes on the internet that may be helpful for the readers who are new to the concepts of multitasking. The following book can be very helpful to understand the concepts of multitasking and learn to use the FreeRTOS in ARM based MCUs:

Book: *ARM-Based-Microcontroller-Multitasking-Projects* — Using the FreeRTOS
Author: Dogan Ibrahim
Amazon Link: https://www.amazon.co.uk/ARM-Based-Microcontroller-Multitasking ProjectsFreeRTOS/dp/0128212276/ref=sr_1_1?dchild=1&keywords=dogan+ibrahim&qid=1602082838&sr=8-1