A Real Time Operating System (RTOS) is an operating system designed to execute tasks in an appropriate order, by a specified deadline. A task is essentially a function that performs a relatively small task and runs independently from other tasks. A deadline may be specified in terms of time (a task must complete within a certain time frame) or events (a task must complete before a certain event occurs within the system). And RTOS may be hard (no toleration of failure to meet deadlines), firm (low toleration of failure to meet deadlines), or soft (high toleration of failure to meet deadlines).

In order to ensure that all tasks meet their deadlines, an RTOS must be able to schedule tasks. Two major types of scheduling are event-driven and time-sharing. In event-driven scheduling, tasks are assigned priorities, and those with higher priorities are executed first. There are various methods of assigning priorities. Two common methods are Rate Monotonic Scheduling (tasks which take less time to complete are assigned higher priority) and Earliest Deadline First Scheduling (the task with the earliest deadline is assigned the highest priority). Time-sharing, also known as round-robin, circulates through a list of tasks waiting to be executed, allowing each to run for a specified amount of time (known as a time slice), then switching to the next task in the list. In an RTOS, tasks with the same priority are often executed using a time sharing method.

An RTOS must be able to coordinate tasks and allocate resources appropriately. In order to coordinate tasks, most RTOSes support transmission of messages from one task to another. Each task is assigned its own mailbox (or queue) for the messages it receives. In a given application, multiple tasks may request to use a single resource. To avoid confusion an RTOS ensures that once a task has accessed a resource, no other task may use it until the first task completes. This is often accomplished through the use of semaphores. A semaphore essentially locks a resource until the task which “owns” it is complete.

Locking resources can lead to a problem known as priority inversion. Priority inversion occurs when a lower-priority task locks a resource, which a subsequently activated higher-priority task needs. The resource cannot be unlocked until the lower priority task completes, and the lower priority task cannot run until the higher priority
task is complete, but the higher priority task cannot complete without access to the
resource. One solution to priority inversion is to temporarily give higher priority to task
that locks a resource so that it may complete. This may cause further complications,
however.

Using an RTOS can significantly simplify the code for a system. Applications may
be broken up into small tasks, which are easier to debug and maintain. The size of an
RTOS may be a concern, and many RTOS vendors provide resources to allocate
minimum space to the system.