Applications

How to build various different applications based on FreeRTOS, and guidance on suggestions and things to avoid.

Handle multiple Queues

APIs to avoid

This article lists FreeRTOS APIs that are discouraged from being used.

[warning: this article is under construction]

Just because an API exists does not mean it should be used. Different programmers have different guidelines sometimes, but ultimately it is up to you to ensure that your RTOS application is performing deterministic operations. Operations such as dynamic memory usage, dynamic task or queue usage encourages the creation of non deterministic RTOS behavior, and becomes a maintenance issue.

Fundamental Guidance

Create or allocate all resources before the RTOS starts, and after the RTOS has started, avoid any API usage that can cause the system to fail.

Avoid Deletion API

`vTaskDelete`

Theoretically, you can create a task after the RTOS is running, and while the RTOS is running, you could delete a task. In practice, however, it is discouraged to create tasks, and then to delete tasks. A task should not be created if it is going to be deleted because there are better options to handle this scenario.

Let us use a practical example. Let us assume that there is a task responsible to play an MP3 song. In place of creating a task to do this work, and then to delete the task when the playback is completed, it is better to consider alternate options:

1. Let the task sleep on a semaphore (or a Queue). Once the tasks' work is done, one can block on the semaphore or queue again
2. Let the creator of the task also process the code that is meant to run by another dynamically

created task

`vQueueDelete`

Similar to why we should not create or delete a task dynamically, a queue should not be created or deleted dynamically. Fundamentally, dynamic operations such as these encourage or create non-deterministic behavior. Furthermore, depending on the RTOS configuration, the queue may utilize dynamic memory, which may exist at one time, but may not exist at another time during the RTOS runtime.

Fundamental Guidance applies here which is to avoid allocating a resource during runtime. We do not want to deal with queue handles being valid, or invalid. We do not want to run into scenarios when a task is blocked on a queue, and then the queue is deleted to abruptly "pull the plug" and to compromise our running program.

Avoid APIs that facilitate indeterministic behavior

`vTaskPrioritySet`

A task priority should be set once while creating the task, and it should never be changed again . When tasks priorities change, it is hard to diagnose issues and figure out which tasks are using the CPU at any given time because the priorities are not constant and could be changing.

Using a mutex could alter task priorities, but that is something we can let the RTOS manage by itself. Using a mutex may cause tasks to inherit and disinherit priority of another task, but the RTOS would do that under certain situations to avoid priority inversion issue. This should be considered outside of a developer's control as it is the behavior of the RTOS that is well tested.

If you run into a situation when a task needs to perform some work at a higher priority level, then you could interface to this piece of code using an RTOS construct (Queue or Semaphore), and dedicate the task for this effort and let that task have constant priority.

Avoid Task Notifications

Avoid Queue Sets