FreeRTOS Interrupt Management

Interrupts are treated by hardware. Number of interrupt types and their priorities depend on the microcontroller used.

Interrupt priorities are different from task priorities. Interrupts can preempt any task.

Special versions of some API functions are available for use within Interrupt Service Routines (ISRs). Only functions and macros with name ending by `FromISR` are safe to be called within an ISR.

Deferred Interrupts

It is preferred to make ISRs as short as possible to reduce their effect on other tasks. ISR may just unblock a handler task that performs the necessary processing.

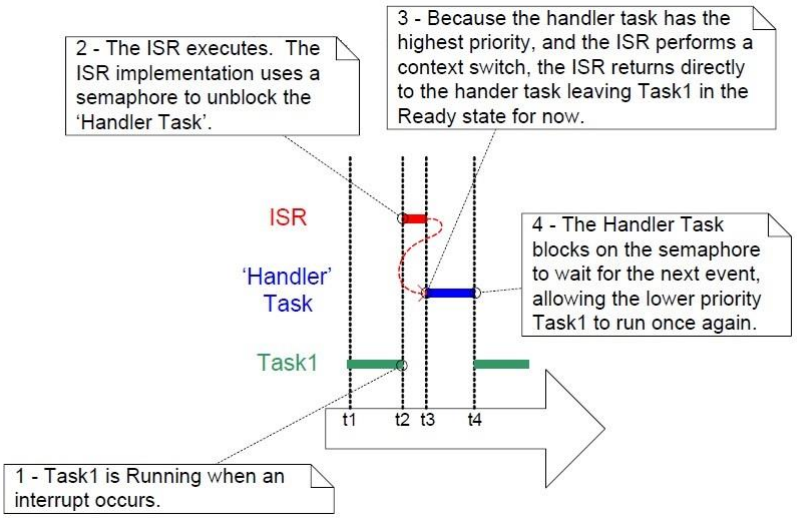
This can be done using the function:

```
xSemaphoreGiveFromISR(handle,&variable)
```

Semaphore handle

This variable returns `pdTRUE` if a task with higher priority than current task was unblocked. In this case, a context switch may be forced.

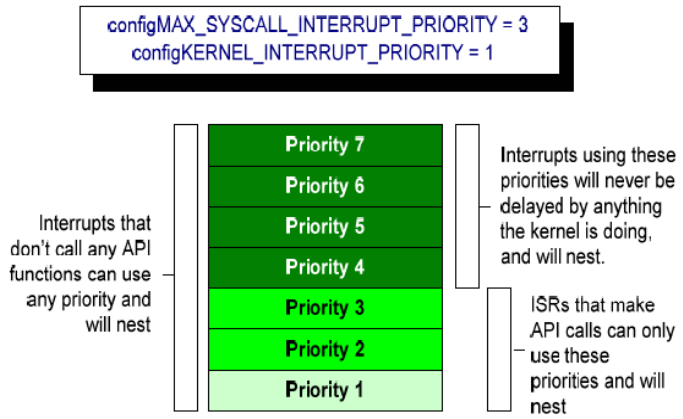
Deferred Interrupts



Deferred Interrupts

By forcing a task context switch (e.g. Using the function `portYIELD()`), ISR and the handler task may respond rapidly to external events if necessary.

FreeRTOS allows to define in the configuration files the interrupt type for the tick timer. It allows also to define a range of interrupt priority levels which always have priority higher than the system. ISR in this range will never be preempted by the system but should not call any API functions.



Interrupts in FreeRTOS Simulator

- Interrupts are simulated in the Windows simulator using high priority Windows threads.
- Functions used in ISR should have a prototype of the form:

```
Uint32_t FunctionName (void)
```

And function should return `pdTRUE` if a context switch need to be forced, and `pdFALSE` otherwise.

Interrupts in FreeRTOS Simulator

- An ISR is set to a given function using:

```
vPortSetInterruptHandler(interrupt number,FunctionName)
```

Where interrupt number is a number between 3 and 31. Levels 0 to 2 are used by the simulator itself.

- Simulated interrupts are triggered using the function

```
vPortGenerateSimulatedInterrupt(interrupt number)
```


Resource Allocation and Deadlocks

Many system resources (I/O devices, communication buses, shared data,…) are assigned according to the following rules:

- Resource should be assigned exclusively to one task.
- Resource assignment cannot be preempted.
- A task that requests a resource assigned to another task is blocked. The requests are queued until the resource becomes free.
- A task may be assigned several resources at the same time.

These rules alone can lead to deadlock.

Resource Allocation and Deadlocks

Simple example:

- Task A requests resource 1 and gets hold of it.
- then task B requests resource 2 and gets hold of it.
- then tasks A requests resource 2.
- then task B requests resource 1.

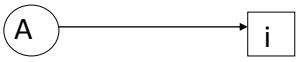
In general:

A set of tasks is said to be in a **deadlock** state if each task in the set is blocked waiting for a condition that only another task in the set can cause.

Resource graphs

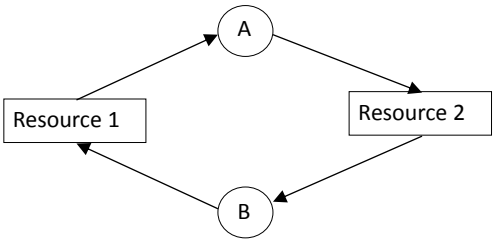
Such situations are better described using graphs.

- Represent each task by a circular node.
- Represent each resource by a square node.
- Represent task *request* of a resource by a directed arc from the task to the resource.



- Represent resource assignment to a task by a directed arc from the resource to the task.

Resource graphs

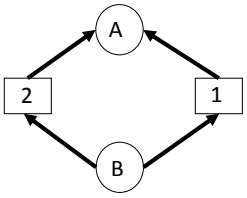


The graph of the simple example contains a *cycle*: a path along directed arcs starting from a node and returning to it.

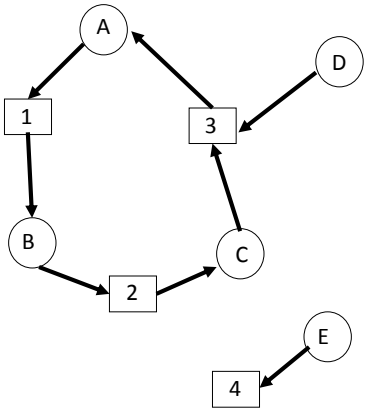
In general, a system exhibits a deadlock *if and only if* its resource graph contains a cycle.

It may be difficult to discover the presence of a cycle in a complex graph.

Resource graphs



No Deadlock



Deadlock. D is also blocked, E is not.

Approaches of handling deadlocks

- Ignore it completely: if the probability of its occurrence is low and its consequences are not severe.
- Deadlock detection and recovery: allow deadlocks to occur, but system must be able to detect a deadlock and recover by killing one of the tasks.
- Deadlock prevention or avoidance : additional assignment rules are used to make deadlocks impossible.

Some Deadlock Prevention Methods

- Task must request all the resources it may need *once at its start*.

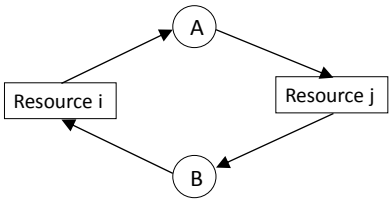
  No "hold and wait" condition.
- Give each resource a unique number. Task may access resources at any time, but *in numerical order*.

  No "circular wait" condition.

These methods may lead to inefficiency in the use of system resources: resources may be allocated but unused for a long period and starvation is possible.

Some Deadlock Prevention Methods

Using this last method, cycle cannot occur in graph



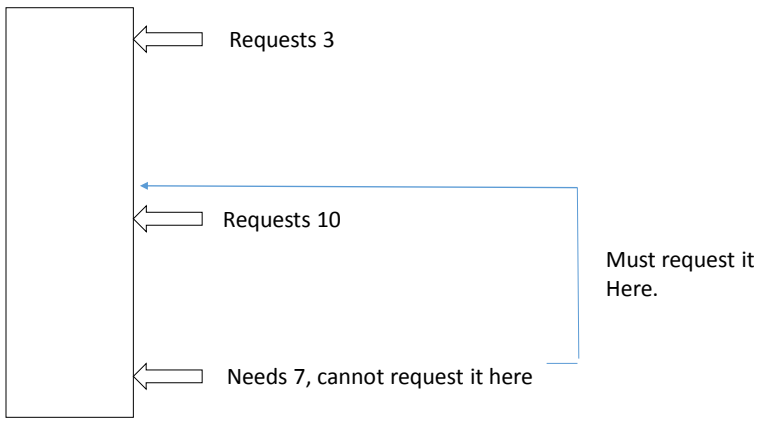
A holds i and requests j $\quad \therefore j > i$

B holds j and requests i $\quad \therefore i > j$

Since numbers are unique, this is impossible.

Some Deadlock Prevention Methods

Task A



Deadlock Avoidance – Banker's Algorithm

In deadlock avoidance methods, the system is given in advance additional information concerning which resources a task will request and use. With this information, the system can decide whether each request can be immediately accepted.

In the following we assume that some resources may be identical (of same type). Task request to a resource of this type can be satisfied by any free instance.

In Banker's algorithm, the system will always be kept in a "safe state": a state (i.e. resource assignments) that cannot lead to deadlock *even in the worst case*.

Deadlock Avoidance – Banker's Algorithm

- Each task declares at its start its maximum requirements of each resource type.

  These will not be assigned to the process until actually needed.

- When a task requests a resource, system will accept the request if:
  - there is a free resource of the required type,
  - maximum declared by the process is not exceeded, and
  - this assignment will keep the system in a safe state.

  Otherwise, the request is denied and the process is blocked.

Deadlock Avoidance – Banker's Algorithm

Example (1): A system that has 5 devices of the same type and runs three tasks. Consider the following state:

Task	Resources held	Declared max.
A	1	3
B	2	3
C	1	3

The above state is a feasible state (Why?)

State is safe since system can still allocate resources to each task (up to its maximum) in some order and avoid a deadlock.

What if process B had declared a maximum of 4?

Deadlock Avoidance – Banker's Algorithm

In general, assume we have n tasks and r resources of the same type, with each task i declaring a maximum usage of C_i resources. If task i is assigned P_i resources, the state ($P_1, P_2, \ldots, P_n$) is feasible if and only if:

$$P_i \leq C_i \leq r \qquad \forall i$$

and $$\sum_{i=1}^{n} P_i \leq r$$

Deadlock Avoidance – Banker's Algorithm

And the state $(P_1, P_2, ..., P_n)$ is safe if and only if we can find an order of task execution $(s_1, s_2, ..., s_n)$ such that:

$$r - \sum_{i=1}^{n} P_i \geq C_{s1} - P_{s1}$$

$$r - \sum_{\substack{i=1 \\ i \neq s1}}^{n} P_i \geq C_{s2} - P_{s2}$$

$$r - \sum_{\substack{i=1 \\ i \neq s1,s2}}^{n} P_i \geq C_{s3} - P_{s3}$$

……..

n conditions

Deadlock Avoidance – Banker's Algorithm

Example (2): A system that has 11 devices of type 1 and 12 devices of type 2, and is running three tasks. Consider the following state:

Task	Type 1 held	Type 1 max	Type 2 held	Type 2 max
A	3	4	4	9
B	2	5	3	9
C	4	6	2	2

Is the above state a feasible state?

Is it a safe state? Will a system using the Banker's algorithm accept being in this state?

Deadlock Avoidance – Banker's Algorithm

In general, assume we have n tasks and m resource types, with r_j resources available from type j. Each task i declares a maximum usage of $C_{i,j}$ resources from type j. If task i is assigned $P_{i,j}$ resources from type j, the resulting state is feasible if and only if:

$$P_{i,j} \leq C_{i,j} \leq r_j \qquad \forall i, j$$

and

$$\sum_{i=1}^{n} P_{i,j} \leq r_j \qquad \forall j$$

Deadlock Avoidance – Banker's Algorithm

And the state is safe if and only if we can find an order of execution task ($s_1, s_2, \ldots, s_n$) such that:

$$r_j - \sum_{i=1}^{n} P_{i,j} \geq C_{s1,j} - P_{s1,j} \qquad \forall j$$

$$r_j - \sum_{\substack{i=1 \\ i \neq s1}}^{n} P_{i,j} \geq C_{s2,j} - P_{s2,j} \qquad \forall j$$

$$r_j - \sum_{\substack{i=1 \\ i \neq s1,s2}}^{n} P_{i,j} \geq C_{s3,j} - P_{s3,j} \qquad \forall j$$

……..

$n \times m$ conditions