FreeRTOS

FreeRTOS kernel directory structure

- The core FreeRTOS kernel source files and demo projects are contained in two sub directories as shown below:
  - Demo
    - Contains the demo application projects.
  - Source
    - Contains the real time kernel source code.
- The core RTOS code is contained in three files, which are called called `tasks.c`, `queue.c` and `list.c`. These three files are in the `FreeRTOS/Source` directory. The same directory contains two optional files called `timers.c` and `croutine.c` which implement software timer and co-routine functionality respectively.

Notes

- 需要一個 cpu timer irq, tick
- 有四種 heap 管理方式可以選
- CPU tick : OS tick 的比例, configTICK_RATE_HZ
  - if CPU = 100MHz, configTICK_RATE_HZ = 1000
    - 1 OS tick = 1000 cpu ticks = 100 ms
- `vTaskStartScheduler` 會創一個優先權最低的 idle thread, priority 0 (tskIDLE_PRIORITY)
- 需要較精準的週期動作時，vTaskDelayUntil 會比 vTaskDelay 更準確
- Watchdog rest 要放在最高優先權的 thread，低優先權有可能永遠執行不到
- Mutex 與 Semaphore 的差異
  - 最大的差異在於 Mutex 只能由上鎖的 thread 解鎖，而 Semaphore 沒有這個限制，可以由原本的 thread 或是另外一個 thread 解開。另外，Mutex 只能讓一個 thread 進入 critical section，Semaphore 的話則可以設定要讓幾個 thread 進入。這讓實際上使用 Mutex 跟 Semaphore 場景有很大的差別。
  - **Mutexes** and **Binary Semaphores** are very similar but have some subtle differences: Mutexes include a priority inheritance mechanism, binary semaphores do not. This makes **binary semaphores the better choice for implementing synchronisation (between tasks or between tasks and an interrupt)**, and **mutexes the better choice for implementing simple mutual exclusion**.
  - Binary semaphore
    - `xSemaphoreCreateBinary` 建完直接 take 是拿不到 key 的，但 Mutex 可以
    - 常用在不同 task 或中斷訊號的同步
    - Semaphore 更常是用在同步兩個 thread 或功能上面，因為 Semaphore 實際上使用的是 signal 的 up 與 down，讓 Semaphore 可以變成是一種 notification 的作用，例如 A thread 執行到某個地方時 B thread 才能繼續下去，就可以使用 Semaphore 來達成這樣的作用
  - Mutex
    - 只有拿到鎖(take) 的 task 才可以釋放鎖(give)

    - 在拿到鎖之後到釋放鎖之間，是可以被切走的，只是別的 task 拿不到 key
    - 當 N 個操作請求時，確保一次只會有一個對共用資源的操作(critical section)

Scheduling

- `configUSE_PREEMPTION`
  - 1: preemptive
  - 0: non-preemptive

Deferred Interrupt Handling

- 可以建立一個最高優先權的 thread 作 event handler，處理從中斷或是各地方來的 event

Real Time Scheduling

- **Time Slicing Scheduling** Policy:
  - This is also known as a round-robin algorithm. In this algorithm, **all equal priority tasks get CPU in equal portions of CPU time**.

Non Preemptive Scheduling

- 如果沒有 call taskDelay or taskDelayUntil, 就會固定在當下的 task 中不會切出去

Preemptive Scheduling

- 在 preemptive scheduling 就算沒有主動讓出執行權 (taskDelay or taskDelayUntil)，一樣會切出去先執行 **大於等於自身優先權**的 task
  - 若有一個動作需要固定頻率精準的執行，就需要採用 preemptive + taskDelayUntil + 最高優先權

```
/* A task being unblocked cannot cause an immediate
context switch if preemption is turned off. */
#if (  configUSE_PREEMPTION == 1 )
{
    /* Preemption is on, but a context switch should
    only be performed if the unblocked task has a
    priority that is equal to or higher than the
    currently executing task. */
    if( pxTCB->uxPriority >= pxCurrentTCB->uxPriority )
    {
        xSwitchRequired = pdTRUE;
    }
    ...
#endif /* configUSE_PREEMPTION */
```

Queues

- When a task attempts to [read from an empty / write to a full] queue the task will be placed into the Blocked state (so it is not consuming any CPU time and other tasks can run) until either [data / space] becomes available on the queue, or the block time expires.
- When the size of a message reaches a point where it is not practical to copy the entire message into the queue byte for byte, define the queue to hold pointers and copy just a pointer to the message into the queue instead. This is exactly how the FreeRTOS+UDP implementation passes large network buffers around the FreeRTOS IP stack.

- If more than one task block on the same queue then the task with the highest priority will be the task that is unblocked first.
- APIs
    - xQueueSend = xQueueSendToBack, FIFO
    - xQueueSendToFront, LIFO
    - xQueuePeek
        - Receive an item from a queue without removing the item from the queue.
    - uxQueueMessagesWaiting
        - Return the number of messages stored in a queue.
    - uxQueueSpacesAvailable
        - Return the number of free spaces in a queue.

Task Control & Utilities

- Task handle (TCB, task control block)
    - store task infomation
        - state
            - ready, blocked, suspended
        - stack point
        - priority
    - 拿到 handle 之後才能做對應之控制
        - change priority
        - suspend & resume
        - delete
- How to obtain a task handle?
    1. 在建立 task 時取得 handle
        - `xTaskCreate, TaskHandle_t *pxCreatedTask`
        - pxCreatedTask can be used to pass out a handle to the task being created. **This handle can then be used to reference the task in API calls that**, for example, **change the task priority, delete or suspend the task**.
    2. 拿到 handle 的另一個方式, 透過 task's name(string)
        - `TaskHandle_t xTaskGetHandle( const char *pcNameToQuery );`
            - Looks up the handle of a task from the task’s name.
- 檢查 CPU 空閒時間
    - `TickType_t xTaskGetIdleRunTimeCounter( void );`
    - Returns the run-time counter for the Idle task. This function can be used to determine how much CPU time the idle task receives.
- Critical sections
    - Critical sections must be kept very short, otherwise they will adversely affect interrupt response times.
    - Method 1
        - `taskENTER_CRITICAL()`, disable all irq (depends on implements)
            - disabling interrupts, either globally, or up to a specific interrupt priority level
            - 因為中斷都停了，OS tick 不會計數
        - `taskEXIT_CRITICAL()`, enable all irq

  - Pros: 簡單直觀暴力，中斷和高優先權的 task 都切不走
  - Cons: overhead 較高，全部等它做完，小心使用
  - FreeRTOS API functions must not be called from within a critical section.
- Method 2
  - `vTaskSuspendAll()`, creating a critical section without disabling interrupts
    - OS tick 會被 pending 不計數
  - `xTaskResumeAll()`
  - Pros: 簡單直觀，高優先權的 task 切不走
  - Cons: 會被中斷切走，所以要注意相同的資源會不會被中斷程序動到
  - FreeRTOS API functions must not be called while the scheduler is suspended.
- Method 3
  - mutex, `SemaphoreHandle_t xSemaphoreCreateMutex( void )`
  - Using a mutex to guard access to a shared resource.
  - Pros: overhead 低
  - Cons: 使用者要自行確認共用的資源確實被保護住，mutex 中會被高優先權的 task 切走，CPU不會全力作完