Time: delay and timers

Time and computer systems

Time in FreeRTOS: System tick 1

- The system tick in a RTOS is the heartbeat.

- In FreeRTOS this is set in the file:
  `FreeRTOSVariant.h` and with the
  `#define configTICK_RATE_HZ`
- Currently this is 66,67 ms (but can be changed)

- At all system ticks the OS checks if it should do a context switch.

Time in FreeRTOS: System tick 2

- A context switch could be triggered because a high priority task has ended a time delay since the last tick. But nothing happens until next system tick.

- The resolution and accuracy of time delays therefor depend on the how often system ticks happens.

- The max waiting time is called latency.

Dealy

In FreeRTOS:

```
vTaskDelay(1000/portTICK_PERIOD_MS);
```

The period (how often the task runs) is

delay + interference from other tasks + execution time + latency

If a task needs to run with a precise period delays are not good

Response

- Often there is no need for precise periods (or periods at all)

- But it is very important that the system response quickly on events

Immediate context switch 1

- In case a task suspends it self the OS will immediately check if some other task wants to run – *it does not wait until next system tick*.

- In case of a resume call the OS will also immediately do a context switch if the resumed task has higher priority then the running.

Immediate context switch 2

- In case of semaphores the OS will also do an immediately context switch if a give and take call makes a higher task ready to run.

- However not if the semaphore call was made from an ISR.
- In that case the ISR must first return and then the context switch happens immediately after.

  Since ISR's ought to be very fast this should add little overhead.

How to use timers 1

In FreeRTOSConfig.h:

```
#define configUSE_TIMERS 1

#define configTIMER_TASK_PRIORITY
( tskIDLE_PRIORITY + 6 )
```

(when used to set up scheduling timer task usually has highest priority)

How to use timers 2

Include:

```
#include <timers.h>
```

Declare global:

```
TimerHandle_t xTimer1;
```

And create:

```
xTimer1 = xTimerCreate("Timer 1",
     (1000/portTICK_PERIOD_MS),
     pdTRUE,
     (void*)0,
     vTimerCallback1);
```

How to use timers 3

```
void vTimerCallback1(
TimerHandle_t pxTimer ){
    xSemaphoreGive(xSemaphore1);
}
```

Callback functions should always be very fast (since they share priority with the timer task and should have the highest priority).

Starting the timer

From main or some other task:

```
xTimerStart(xTimer1, 0);
```

Controlling the task:

In the timer controlled task:

```
while (1) {
    xSemaphoreTake(xSemaphore1,
    portMAX_DELAY);
    task code …
}
```