**Activity 1**

2. Modify the Priority of the new Thread to HIGHPRIO. When running again, has changed the behaviour in any way? Why?

There is no behavior difference. That happens because when the thread with the high priority goes to sleep, the processor is free to execute the other thread.

3. Exchange the code in the new Thread with the code present in the box maintaining HIGHPRIO. Observe the behavior differences and explain the reason for them.

Only the LED of the thread in the High Priority turns ON and OFF. That happens because without a sleep function the thread with high priority never goes idle therefore never allowing the thread with lower priority to be executed.

4. Is there any difference if the Thread returns to NORMALPRIO?

Yes, it returns to its previous behavior (before the insertion of the code in the box).

5. Modify the CH\_TIME\_QUANTUM present in the chconf.h file to 0 and observe again the differences. What’s the reason?

Only one of the LEDs turns ON and OFF. That happens because with the value set to zero the preemption of the threads is disabled, meaning that the cooperative round robin is activated, but it only works with threads of the same priority. In that way only the threads with the highest priorities will be executed.

6. Assign HIGHPRIO to the first Thread and NORMALPRIO to the second one and comment the behaviour.

Yes, it returns to its previous behavior (before setting the CH\_TIME\_QUANTUM to zero).

**Activity 2**

1. What are the differences between the Blinking example based on Threads and timers?

The example with threads generates a concurrency, that’s why we have used some priorities and tested the blinking behavior. With the timers we set a counter to do some task, so we theoretically know when the program will be executed and in which order.

2. Modify the Blinking example based on Timers in order to use two differents timers, one for the led ON and the other for the led OFF

Create a Thread that evaluates wether the led ON virtual timer is active or not, and then switch on/off a led in port GPIO\_18 (Use the chVTIsArmedI function)

- Previous to run the RTOS, what’s the expected behaviour?

We expect that the LED on port 25 goes off for two seconds, after this time, the LED goes on. Then the LED on port 25 goes on and off in 500ms interval, while the LED on port 18 goes on/off every 800ms based on the on/off from the first LED.

- What’s the real behaviour? and why?

The blinking occurs as expected, and this occurs because when the thread initializes, it sets the first timer to set the LED 25 on, with 2 seconds to wait. After, each function called by the timer makes the operation (set the LED 25 on or off) and sets another timer of 500 milliseconds to call the inverse operation. The thread operates verifying if the first LED is on or off and setting the LED 18 according to the result, then sleeps by every 800 milliseconds.

Try to modify the Thread priority and evaluate the differences

The behavior is the same since it’s the only thread. Without concurrency, the low priority will always enter accordingly.

**Activity 3**

1. Evaluate the previous code behavior?

The code works properly. The LED on the port 25 blinks normally and when the second thread is being executed, it blinks faster while the other LED is turned on.

1.1 What happens if the priority of one of the threads is modify to HIGHPRIORITY?

Nothing happens, it works as it did previously

1.2 and modifying the CH\_CFG\_USE\_SEMAPHORES\_PRIORITY to TRUE

The thread with highest priority is executed before the other with lowest priority.

2. Return the CH\_CFG\_USE\_SEMAPHORES\_PRIORITY to FALSE and modify the previous code to use a counting one with counter value N=2 and having both threads NORMALPRIORITY. Is the behaviour different?

Yes, the LED 2 (port 18) remain ON at all times, while LED 1 will alternate from blinking slowly and fast,

2.2 change the priority of one of the threads to HIGHPRIORIY and observe the differences.

There is no observable difference.