Toulouse site

FINAL ASSIGNMENT: Design of an RTOS

BROCHARD Alexandre
Promo: *Aéro 5*
Class: *5TS1*

Course: *Systèmes d'exploitation temps-réel pour l'embarqué* – Professor: *M. Singh*
Submission date: *January 14, 2024*

Contents

Abstract

Real-time systems are computer systems designed to meet stringent timing requirements, where the correctness of the system depends not only on the logical results of computations but also on the time at which those results are produced. These systems are prevalent in critical applications such as avionics, automotive control systems, medical devices, and industrial automation.

One crucial aspect of real-time systems is the Worst Case Execution Time (WCET), which represents the maximum time a task or operation takes to complete under the most adverse conditions. Accurate estimation of WCET is essential for ensuring that tasks meet their deadlines and for providing guarantees on system behavior.

WCET analysis involves identifying the critical paths and sections of code within a real-time application, considering factors such as loops, conditional branches, and resource contention. Tools and methodologies for WCET analysis aid developers in validating system performance, enabling them to make informed decisions about system design, resource allocation, and scheduling.

Understanding and accurately estimating WCET contribute to the development of reliable and predictable real-time systems, ensuring that they respond to stimuli within the required time bounds and meet the demands of safety-critical applications.

Introduction

FreeRTOS is a real-time operating system (RTOS) kernel designed for embedded systems and microcontrollers. It is an open-source, royalty-free real-time kernel that provides a small, efficient, and flexible platform for building embedded applications. FreeRTOS is written in C and is well-suited for resource-constrained devices.

FreeRTOS propose a huge amount of features but do to its complexity thoses features will not be fully used in this project.

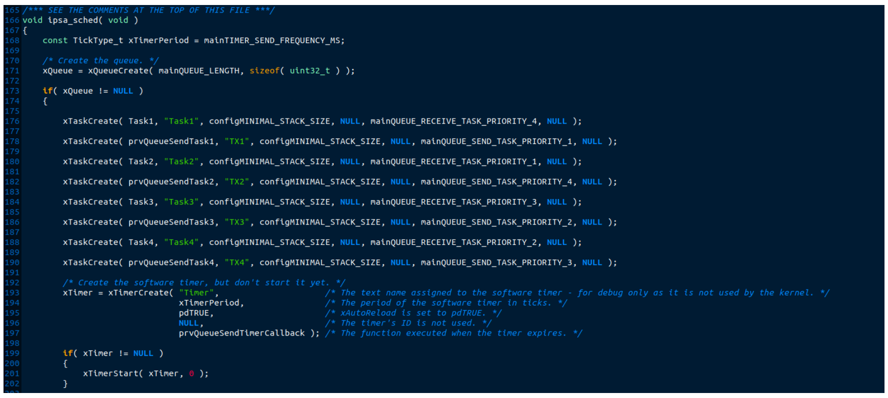
The purpose of this assignment is to implement different tasks in a RTOS system and make an WCET analysis.

These are the following four tasks to implement :

- To print "Working" or some other string which says everything is working as normal
- Convert a fixed Fahrenheit temperature value to degree Celsius
- Define any two long int big numbers and multiply them, print the result
- Binary search a list of 50 elements (fix the list and element to search)

The RTOS is coded on a file that we will call : "ipsa_sched.c"

This file is using multiple subfiles and also different library that allow him to create queue and real-time process.

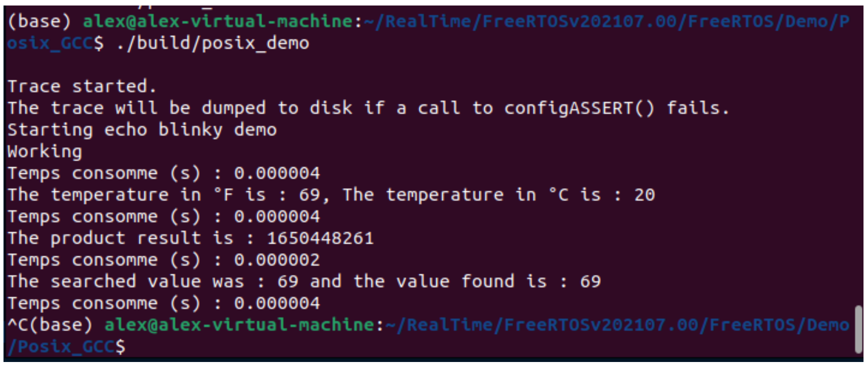


```c
/*** SEE THE COMMENTS AT THE TOP OF THIS FILE ***/
void ipsa_sched( void )
{
    const TickType_t xTimerPeriod = mainTIMER_SEND_FREQUENCY_MS;

    /* Create the queue. */
    xQueue = xQueueCreate( mainQUEUE_LENGTH, sizeof( uint32_t ) );

    if( xQueue != NULL )
    {

        xTaskCreate( Task1, "Task1", configMINIMAL_STACK_SIZE, NULL, mainQUEUE_RECEIVE_TASK_PRIORITY_4, NULL );

        xTaskCreate( prvQueueSendTask1, "TX1", configMINIMAL_STACK_SIZE, NULL, mainQUEUE_SEND_TASK_PRIORITY_1, NULL );

        xTaskCreate( Task2, "Task2", configMINIMAL_STACK_SIZE, NULL, mainQUEUE_RECEIVE_TASK_PRIORITY_1, NULL );

        xTaskCreate( prvQueueSendTask2, "TX2", configMINIMAL_STACK_SIZE, NULL, mainQUEUE_SEND_TASK_PRIORITY_4, NULL );

        xTaskCreate( Task3, "Task3", configMINIMAL_STACK_SIZE, NULL, mainQUEUE_RECEIVE_TASK_PRIORITY_3, NULL );

        xTaskCreate( prvQueueSendTask3, "TX3", configMINIMAL_STACK_SIZE, NULL, mainQUEUE_SEND_TASK_PRIORITY_2, NULL );

        xTaskCreate( Task4, "Task4", configMINIMAL_STACK_SIZE, NULL, mainQUEUE_RECEIVE_TASK_PRIORITY_2, NULL );

        xTaskCreate( prvQueueSendTask4, "TX4", configMINIMAL_STACK_SIZE, NULL, mainQUEUE_SEND_TASK_PRIORITY_3, NULL );

        /* Create the software timer, but don't start it yet. */
        xTimer = xTimerCreate( "Timer",                     /* The text name assigned to the software timer - for debug only as it is not used by the kernel. */
                               xTimerPeriod,                /* The period of the software timer in ticks. */
                               pdTRUE,                      /* xAutoReload is set to pdTRUE. */
                               NULL,                        /* The timer's ID is not used. */
                               prvQueueSendTimerCallback ); /* The function executed when the timer expires. */

        if( xTimer != NULL )
        {
            xTimerStart( xTimer, 0 );
        }
```

Figure 1: First look at the main loop of "ipsa_sched.c".

Once the code is launched, it will do every tasks with a given order of priority and a period that we can define in the file.

Here is the output if every task get the same priority :



```
(base) alex@alex-virtual-machine:~/RealTime/FreeRTOSv202107.00/FreeRTOS/Demo/P
osix_GCC$ ./build/posix_demo

Trace started.
The trace will be dumped to disk if a call to configASSERT() fails.
Starting echo blinky demo
Working
Temps consomme (s) : 0.000004
The temperature in °F is : 69, The temperature in °C is : 20
Temps consomme (s) : 0.000004
The product result is : 1650448261
Temps consomme (s) : 0.000002
The searched value was : 69 and the value found is : 69
Temps consomme (s) : 0.000004
^C(base) alex@alex-virtual-machine:~/RealTime/FreeRTOSv202107.00/FreeRTOS/Demo
/Posix_GCC$
```

Figure 2: First look at output of "ipsa_sched.c".

Basically, we have every output for each task that we define earlier, and we also have the executed time for each task in this run. The information about the time taken by each task is crucial because it will help us create our WCET.

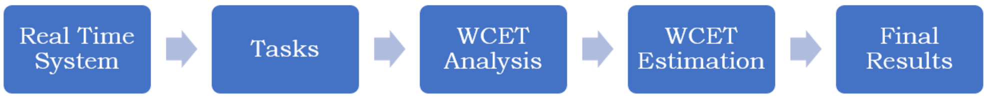


Figure 3: WCET approach for this project

Method

Due to the fact that the code given with FreeRTOS and all the features was really hard to understand and also to implement in the given time, here is how we will approach the WCET :

- For each task, the code we will run multiple times in order to see which is the mean time needed for a task to execute themselves.
- each task is independent so we don't need them to finish them in a specific order to have one of them access an information that another one would be using, but in real projects that might be the case, so we need to fix a priority for each task.
- In order to fix these priorities, we will also make assumptions and try to put them in real situations to have better overview and a more realistic analysis.

Average running time

Here is the average running time for each task :

Task 1	Task 2	Task 3	Task 4
1 µs	4µs	2 µs	4µs
1 µs	3µs	1 µs	3µs
2 µs	4µs	2 µs	4µs
1 µs	4µs	1 µs	4µs
2 µs	3µs	2 µs	3µs
1 µs	3µs	2 µs	3µs
1 µs	4µs	2 µs	4µs
1 µs	5µs	3 µs	4µs
1 µs	4µs	2 µs	4µs

So for each task, the average running time (approximately) is :

- Task 1 : 1µs
- Task 2 : 4µs
- Task 3 : 2µs
- Task 4 : 4µs

Priority of the tasks

Now let's focus on what the tasks are doing and if it's important in our system.

- Task 1 : Simply printing "Working" => not relevant nor important for our system to work.
- Task 2 : Convert the temperature from Fahrenheit to Celsius, but indicates the temperature. So for our analysis we will say that this task is computing the temperature of the system which is an important information.
- Task 3 : Do an operation between two large numbers. This task can be assimilated to a complex computation of a value (weather forecast of a control tower or an aircraft let's assume)
- Task 4 : Binary search in a fixed list. Searching for a relevant information in a list (for example search for the nearest airport).

Every task was a simple case in this assignment but we will turn them into "real" functional tasks that can be on a complex entity (aircraft in our case).

Here's the final ranking :

- 1 : Task 2
  Task 2 takes more time than other tasks (ex aqueo with Task 2) and give an important information (temperature) about the system.
- 2 : Task 4
  Task 4 needs to do multiple operations and even if it takes the same amount of time as Task 2, Task 4 can be interrupted in his process without having is data corrupted (maybe it can be in reality with more complex systems be we assume it's not the case here) also having the temperature first ensure that the system does not cause an overheat which would maybe imply a loss of some data for this task.
- 3 : Task 3
  Do a big operation bewteen two big numbers but its faster than Task 2 and 4, in this context we assume that the weather forecast can also be given by the tower so this task does not really need priority it's just a feature but not necessary in our system.
- 4 : Task 1
  Only print working to ensure that the system is working which can be useful in a very amount of situations but here it's meaningless.

Maximum Execution Time

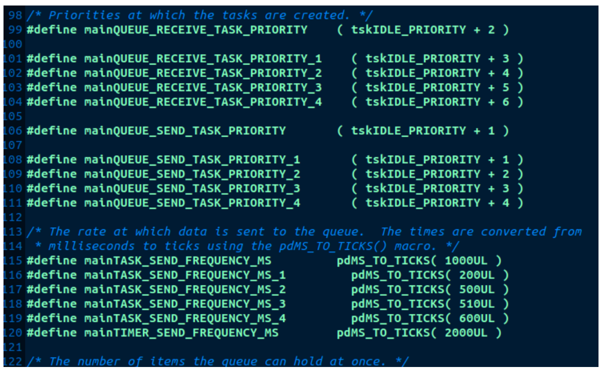
For each task we need now to prevent a "bad execution" and shut them after a reasonable amount of time.
For each task, here's the Maximum Execution Time allowed :

- Task 1 : 7µs
  5µs was the maximum running time observed, so 7µs would be sufficient for this function to work in any case here.
- Task 2 : 12µs
  Same idea here 5µs was the maximum time observed but this information is really important but in the other hand we also need this task to not stop the system and so we are not able to provide too much time for this task.
- Task 3 : 5µs
  This task always take 3µs or less to execute so having +2µs would be sufficient.
- Task 4 : 10µs
  Same idea as for task 2.

In any of those task we don't know how the system will react in each situation so we provide more time than needed for our task to process and shut them if needed. These numbers (+2µs, ...) are chosen based on only hypothesis that this amount would be sufficient but an analysis of the hardware and also the subfunctions of the software might change the maximum time allowed that wa have defined here.

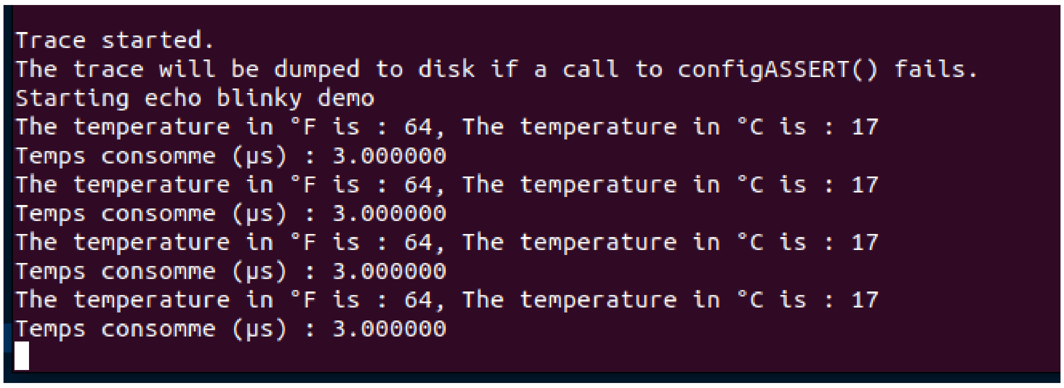
Results

Finally, we have a ranking to implement to have a better code. The problem is I was not able to do a good ranking because I create multiple priority for each task and when I try to run the code only the task with the highest priority was able to run. :



```c
/* Priorities at which the tasks are created. */
#define mainQUEUE_RECEIVE_TASK_PRIORITY    ( tskIDLE_PRIORITY + 2 )

#define mainQUEUE_RECEIVE_TASK_PRIORITY_1    ( tskIDLE_PRIORITY + 3 )
#define mainQUEUE_RECEIVE_TASK_PRIORITY_2    ( tskIDLE_PRIORITY + 4 )
#define mainQUEUE_RECEIVE_TASK_PRIORITY_3    ( tskIDLE_PRIORITY + 5 )
#define mainQUEUE_RECEIVE_TASK_PRIORITY_4    ( tskIDLE_PRIORITY + 6 )

#define mainQUEUE_SEND_TASK_PRIORITY       ( tskIDLE_PRIORITY + 1 )

#define mainQUEUE_SEND_TASK_PRIORITY_1       ( tskIDLE_PRIORITY + 1 )
#define mainQUEUE_SEND_TASK_PRIORITY_2       ( tskIDLE_PRIORITY + 2 )
#define mainQUEUE_SEND_TASK_PRIORITY_3       ( tskIDLE_PRIORITY + 3 )
#define mainQUEUE_SEND_TASK_PRIORITY_4       ( tskIDLE_PRIORITY + 4 )

/* The rate at which data is sent to the queue.  The times are converted from
 * milliseconds to ticks using the pdMS_TO_TICKS() macro. */
#define mainTASK_SEND_FREQUENCY_MS        pdMS_TO_TICKS( 1000UL )
#define mainTASK_SEND_FREQUENCY_MS_1        pdMS_TO_TICKS( 200UL )
#define mainTASK_SEND_FREQUENCY_MS_2        pdMS_TO_TICKS( 500UL )
#define mainTASK_SEND_FREQUENCY_MS_3        pdMS_TO_TICKS( 510UL )
#define mainTASK_SEND_FREQUENCY_MS_4        pdMS_TO_TICKS( 600UL )
#define mainTIMER_SEND_FREQUENCY_MS       pdMS_TO_TICKS( 2000UL )

/* The number of items the queue can hold at once. */
```

Figure 4: Different priority



```
Trace started.
The trace will be dumped to disk if a call to configASSERT() fails.
Starting echo blinky demo
The temperature in °F is : 64, The temperature in °C is : 17
Temps consomme (µs) : 3.000000
The temperature in °F is : 64, The temperature in °C is : 17
Temps consomme (µs) : 3.000000
The temperature in °F is : 64, The temperature in °C is : 17
Temps consomme (µs) : 3.000000
The temperature in °F is : 64, The temperature in °C is : 17
Temps consomme (µs) : 3.000000
```

Figure 5: Task 2 repeating

The intended purpose here was to create a priority to the task due to their WCET and get a code that will run those tasks in the right amount of time and with the right order but I was unable to do so. Another interesting feature would be to have tasks that use shared data with other task to implement other concepts that we learn in this course.

I have successfully implemented the 4 tasks, and I was able to measure the average processing time for each of them and then I create a context for each of these tasks. With the task's behavior, objectives and its average processing time I was able to create a Fixed Priority and also a short WCET in case any of these tasks would be killed for some reason.