FOLLEREAU Juliane

4TS1

IPSA
ÉCOLE D'INGÉNIEURS
DE L'AIR, DE L'ESPACE ET DE LA MOBILITÉ DURABLE

Real-Time Embedded Systems

FINAL ASSIGNMENT

26/04/2024

Content

1. Abstract

In this practical work, we designed a Real-Time Operating System (RTOS) composed of four periodic tasks and one aperiodic task. Periodic tasks have been programmed to perform simple actions such as displaying messages, converting temperature, performing calculations, and binary search. The aperiodic task, for its part, was designed to appear periodically by displaying a predefined message.

Each task was carefully analyzed to determine its longest execution time, i.e. the WCET. Using Python code, we measured the WCET of each task and adjusted the execution periods accordingly.

The results show that the tasks are executed in accordance with our expectations, with consistency between execution periods, priorities and actions carried out. In addition, by adjusting the periods based on the WCET, we ensured the proper functioning of the system.

In conclusion, this work made it possible to successfully design and implement an RTOS, paying particular attention to task planning and execution time management. These results demonstrate the importance of analyzing and adjusting parameters to guarantee the reliability of real-time systems.

2. Introduction

Our mission in this practical work is to design our own RTOS composed of 4 periodic tasks and one aperiodic task. The 4 periodic tasks will perform simple things such as: writing "Working", converting a temperature fixed in Fahrenheit to degrees Celsius and displaying it, multiplying two large integers and displaying it or even doing a binary search of 'one number from a list of 50 already established values. The aperiodic task for its part will simply say that it is there every certain time (we extended it to more than 100 ms otherwise it intervened all the time) by writing “Hello I am the aperiodic task”.

As a reminder, a periodic task is a task which needs to be executed during a certain interval and an aperiodic task is a task which is executed only when it is called so if we say that the aperiodic task must arrive every 100ms it’s will execute every 100 ms.

Firstly we started by downloading the “FreeRTOSv202107.00” folder, then in order to carry out this practical work we placed ourselves in:

FreeRTOSv202107.00 / FreeRTOS / Demo / Posix_GCC

We then worked on a copy of the “main_blinky.c” file which is located directly in the directory where we have just placed ourselves.

We will name this new copy “ipsa_sched.c”, in this file we will take the example of the “main_blinky.c” file in order to design our own RTOS with the 5 periodic and aperiodic tasks that we will see together in the rest of this report.

Many things will have to be taken into account in order to correctly carry out this practical work. We will first have to code the different tasks, analyze them to know their execution time, decide on a WCET using a different code. As a reminder, WCET is the most pessimistic estimate, that is to say the maximum time that a task can take to be executed, we will take the maximum execution time among all the tasks. We will also have to schedule tasks in FP i.e. Fixed-Prority: the task with the highest priority is executed first using FreeRTOS.

3. The different tacks

As mentioned in the introduction, we started by creating our own tasks in the “ipsa_sched.c” file, before seeing what happens in each of the tasks, let’s see in more detail what happens at the start of the code.

It has of course the inclusion of standard and FreeRTOS libraries. Then we defined constants such as the length of the queue used “mainQUEUE_LENGTH”, the periods of the periodic tasks in milliseconds as well as the delay of the aperiodic task. We also set task priorities, with the lowest priority being the highest. We will discuss all these values that we imposed in the rest of this report.

Once all these constants are initialized and before entering the heart of the code we define each task as a function with the return type void. We define them statically which means that they are not accessible outside of the source file. Each task will then contain an infinite loop that periodically executes a specific action and measures the execution time. We also define our list of 50 values and the values that we will search for in task 4 of binary searches.

We finally enter the main code, before arriving at each of our tasks we do different things. First of all there is the creation of the queue with the “xQueueCreate” function with the size previously specified “mainQUEUE_LENGTH”. If this file creation is successful then there is the creation of all periodic tasks and the aperiodic task using “xTaskCreate”. And at the end there is the start of the task scheduler by calling “vTaskStartScheduler()”.

On the other hand, if starting the task scheduler fails, the infinite loop at the end of ipsa_sched is executed.

So to summarize this part of code before the definition of each of the tasks allows you to configure and launch a set of periodic tasks and an aperiodic task using FreeRTOS in addition to creating a queue which will allow communication between the different tasks.

We can now talk about the code of the different tasks, that is to say their behavior, what they do. Initially we set fairly large period values and without thinking too much about it in order to test how our code works, we will come back to the period values in the next part.

a. Periodic Task 1

This first periodic task simply writes "Working" when it is executed. Its period is 1000 milliseconds to start and this value in milliseconds is converted into timer ticks using “pdMS_TO_TICKS”. It is priority +1 so it is the task with the highest priority. It is logical that it has the lowest priority (that is to say that it is a priority) since it is the one which has the shortest period and it is the one which has the least action to do.

b. Periodic Task 2

This second periodic task takes a floating temperature value in fahrenheit then performs the conversion operation for the data in Celsius by displaying it with the two values in the two different units. Its period is 2000 milliseconds, this value is converted into timer ticks in the same way as before. It has priority +2, that is to say it is the second most important in the list.

c. Periodic Task 3

This third periodic task takes two large integers a and b and performs their multiplication then writes the result with the details of the calculation. Its period is 4000 milliseconds and its priority is +3 therefore the third priority task.

d. Periodic Task 4

This last periodic task performs a binary search for a certain number here 25 among the list of 50 numbers that we defined together at the beginning of this code. There is of course a special condition if the desired figure is not found among the values present in the list of 50. Its period is 8000 milliseconds for a priority of +4 which means that this task is the one which will pass into priority. last.

e. Aperiodic Task 1

This task is the only aperiodic task; it simply executes when it is called at each end of its period. So that it does not intervene all the time with a given period in the statement of this practical work of 100ms, we have given it a period of 10000milliseconds so that we can check from time to time that it appears among the others.

4. Test tasks without changing periods

For this first test we leave the periods that we have chosen in the explanation of each of the tasks. The primary objective here is to see if our code works correctly, we will intervene on the periods later.

Our code therefore works well, and with these periods we can clearly observe what we want in the terminal. We have plenty of time to see each of the tasks being executed (thanks to long uax periods). Here is what we can observe:

```
Working
Working
Temperature: 90.000000°F = 32.222221°C
Working
Working
Element 25 found
Multiplication: 1234567 * 9876 = 12192583692
Temperature: 90.000000°F = 32.222221°C
Working
Working
Temperature: 90.000000°F = 32.222221°C
Working
Working
Multiplication: 1234567 * 9876 = 12192583692
Temperature: 90.000000°F = 32.222221°C
Working
Working
Bonjour je suis la tache apériodique
Temperature: 90.000000°F = 32.222221°C
```

We clearly observe that task number 1 which has the shortest period and which has the highest priority is the one which appears the most often! We observe consistency between the execution period, the priority of the tasks and their appearance in the console. In addition we can observe the aperiodic task which occurs every 10000 milliseconds.

5. WCET

Now and using a python code that you can find in "document_principal", we have built a code in C for each of the 4 periodic tasks (you can find each of the 4 codes in the same folder as the previous document). The Python code executes each task 1000 times by calculating its execution time each time. The goal is to determine the worst execution time that each task can take over the 1000 tries. Here are the WCETs collected for each of the 4 tasks:

WCET task 1 : 0,013s = 13 ms

WCET task 2 : 0,012s = 12 ms

WCET task 3 : 0,013s = 13 ms

WCET task 4 : 0,017s = 17 ms

Once we have collected the worst execution time for each of the tasks, that is to say their WCET, we can deduce the overall WCET which is the worst execution time among the 4. It is according to the WCET collected above 17 ms which corresponds to the WCET of task 4.

This WCET will therefore allow us to make modifications to the execution periods. For the execution time of each of the tasks, we will take the WCET to find for each of them. We can therefore lower the periods accordingly and check if it is still functional, here are the new periods chosen according to each of the WCETs and leaving a small margin in addition.

Task 1 : 14ms

Task 2 : 13 ms

Task 3 : 15 ms

Task 4 : 18 ms

I chose to apply these new periods according to the WCET of each of the chosen tasks since the WCET can be seen in our case as the execution time which is the time it takes to execute its task, this task he will do it periodically following a certain period which is therefore his execution period. The execution period must therefore correspond exactly or be a little longer than the execution time in order to allow the task to be carried out correctly. In our case this works since the chosen execution time corresponds to the WCET of each task. So there is a fairly large margin if we differentiate between the average execution time of the task and its period. The difference between the two would be the time between the end of the execution and the deadline.

6. Final result

By applying the new periods in the code (we will also change the period of the aperiodic spot for 50 ms) we do not have time to see if all the spots appear but we can notice by stopping that they are all present..

So in this practical work we were able to develop our own RTOS composed of 4 periodic tasks and 1 aperiodic task. We assigned them priorities and actions to carry out such as displaying "working" or even calculating a temperature conversion. In parallel with this implementation we created a python code allowing the calculation of the WCET for each of the periodic tasks. These 4 WCETs allowed us to deduce the overall WCET which is 17ms. At the end of this work we intervened on the different periods by modifying them.