Clément STOH
Manon THOMAS

Real-Time Embedded Systems

Final assignment

Jasdeep SINGH

26/04/2024

Aim of the project

The aim of this project is to design an RTOS (Real-Time Operating System).
The purpose of an RTOS is to manage the resources of a computer system and to respond to deadlines for the execution of different tasks. If the system is designed correctly, it will be able to meet strict deadlines by guaranteeing a real-time response.

To design our RTOS, we need to code four tasks in C. We will analyse the execution time of these tasks to deduce their WCET.
We will then order the tasks using time constraints. This will give us our RTOS capable of providing real-time response.

The Tasks

We are coding 4 independent tasks. We are working in the C language.

1. The first task displays: 'Working'.
2. The second task converts a temperature in fahrenheit to Celsius.
3. Then it displays: 'The temperature conversion is **F = **C'.
4. The third task multiplies two large numbers and displays: 'The multiplication betwen ** and ** is **'.
5. The fourth task takes an array of 50 numbers and looks for where a certain number is placed. It displays: 'The element ** is found at index **' or 'The element ** was not found'.
6. The fifth task, which is a bonus task, displays 'The task 5 is starting', then waits 0.1s and displays 'The task 5 is completed'.

Now that we have our tasks, we need to see how long they take to be run.

Calcul of the WCET

The WCET (Worst Case Execution Time) is the maximum time our code/task takes to execute. It is important to define this value correctly so that the time constraints are respected by our tasks in the RTOS.

In our case, we will find this value using python code. (git link)

Using Ubuntu, we'll run the python code. This calculates the execution time of a task a thousand times. Then it keeps the worst time. You need to modify the .py code for each task. All you have to do is change the name of the working file.

We use this method on our 5 tasks. The result is the same for all our tasks: 0.01s.

However, we have to take a safety factor. In our case we'll use 0.01s, so WCET1=WCET2=WCET3=WCET4=WCET5=0.02s.

So the WCET of our total code is 0.08s.

Note that this calculation time depends on the computer on which our code is running.

Le RTOS

Before having our tasks scheduled with a RTOS we need to know if it is schedulable.

To know if our tasks are schedulable together, we can do an utilization analysis. It means we need to calculate:

$$\Sigma_{i=1}^{n} = \frac{C_i}{T_i}$$

With C_i the execution time and T_i the period.

Concerning the code where the task are programmed in C, we had to declare them as a “static” function. Indeed, it makes them only visible in this file, allowing encapsulation within the tasks. The scheduler is initialized by the function “ipsa_sched. Then we create the tasks using “xTaskCreate()” that allows us to set the tasks parameters.

Finally the scheduler is started thanks to “vTaskStartScheduler()”. It will manage task scheduling and execution.

```
xTaskCreate(vTask1, "Task_welcome", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY+0, NULL);
xTaskCreate(vTask2, "Task_conversion", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY + 1, NULL);
xTaskCreate(vTask3, "Task_multiplication", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY + 2, NULL);
xTaskCreate(vTask4, "Task_search", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY + 3, NULL);
xTaskCreate(vTask5, "Task_bonus", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY + 4, NULL);
```

For a RTOS environment we need to ensure that the CPU isn’t overloaded by tasks. That’s why we are using the function “Delays(vTaskDelay)” that simulates the period of execution.

Another important part is to choose the priority of the tasks. We decided that the less complex tasks will have the highest priority as they take less time to execute.

In conclusion, we can say that the scheduling worked because the theorical WCET are close to the ones found with the RTOS.