Quentin CHEVALLIER

Real-Time Operating System (RTOS) Design and Implementation

Abstract

This project proposes the design and implementation of a real-time operating system (RTOS) to efficiently manage a set of periodic tasks. The tasks encompass the periodic display of the message "Working," Fahrenheit to Celsius temperature conversion, multiplication of large integer numbers, and binary search in a pre-defined list.

1. Introduction

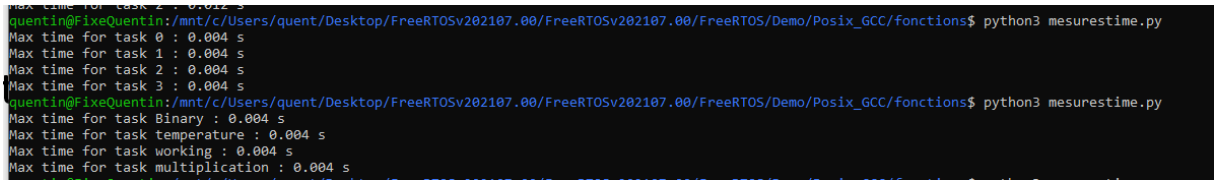
A real-time operating system is crucial for coordinating the execution of critical tasks in various domains such as embedded systems and industrial control. This project focuses on designing an RTOS capable of handling a diverse set of tasks, each with specific requirements.

2. Methodology

2.1 Task Analysis

Each task has undergone a detailed analysis to determine its execution time, with a specific emphasis on the worst-case execution time (WCET). The analysis involved the creation of measuretime.py (found in the git repository) which independently launches the time command in the command prompt, providing the WCET for each task.

After 1000 iterations (launched 2 times), the WCET results are as follows:



Task	Binary	Temperature	Multiplication	Working
WCET (in seconds)	0.004	0.004	0.004	0.004

We need to find a period for each task. We can resolve this equation:

$$\frac{0.004}{0.004 * x} + \frac{0.004}{0.004x} + \frac{0.004}{0.004x} + \frac{0.004}{0.004x} \leq 1$$

$$\frac{1}{x} + \frac{1}{x} + \frac{1}{x} + \frac{1}{x} \leq 1$$

$$\frac{4}{x} \leq 1 \; So \; x \geq 4$$

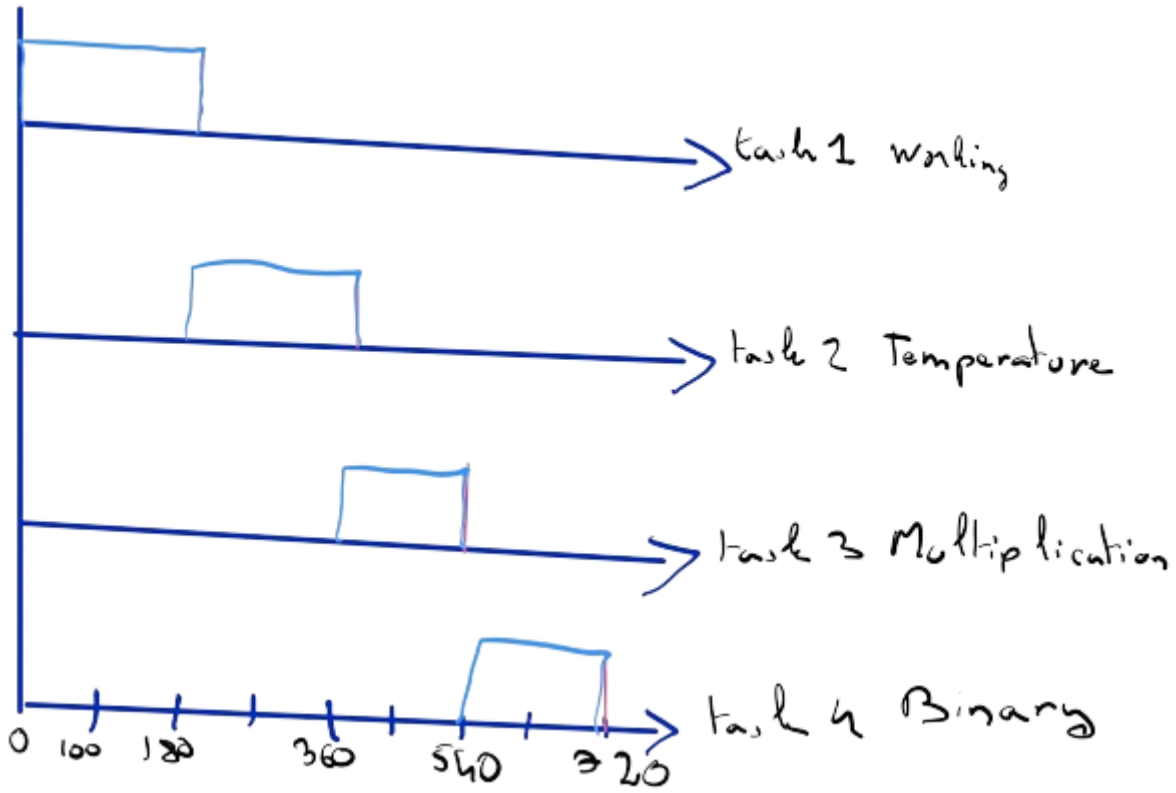
Multiplying by 4.5 for safety, the periods were calculated in seconds and milliseconds.

Task	Binary	Temperature	Multiplication	Working
WCET (in s)	0.004	0.004	0.004	0.004
Period (in s)	0.018	0.018	0.018	0.018
Period(in ms)	180	180	180	180

We can verify: $\frac{0.004}{0.018} + \frac{0.004}{0.018} + \frac{0.004}{0.018} + \frac{0.004}{0.018} = 0.88 \leq 1$

So, tasks should be schedulable and can be verified in FreeRTOS.

We can draw the scheme of task scheduling which will be really simple as the tasks have the same period.



Ordonnance scheme

2.2 System Architecture

The RTOS is based on FreeRTOS and uses a First-Come First-Serve (FCFS) scheduler. Task creation, scheduling, and synchronization are handled in accordance with each task's requirements.

Code is using FreeRTOS (Real-Time Operating System) to create a multitasking system on a Linux environment can be summarized as follows.

Project Configuration:

The project offers two demos: a simple "blinky" version and a more complex application. The current file implements the simple version.

Tasks, a queue, and a software timer are created.

It uses the console for display instead of an LED.

Task Frequencies:

Four distinct tasks (working, temperature, Multiplication, BinarySearch) are created with their own frequencies defined in milliseconds (all the same because of the mesure made before).

Each task uses vTaskDelayUntil to execute its code periodically.

A queue (xQueue) is used for communication between tasks.

Tasks periodically send values to the queue, and a receiving task reads these values.

A software timer (xTimer) is used to generate interrupts at regular intervals.

The main_ipsa_sched function initializes the queue, creates tasks, and starts the task scheduler.

3. Results

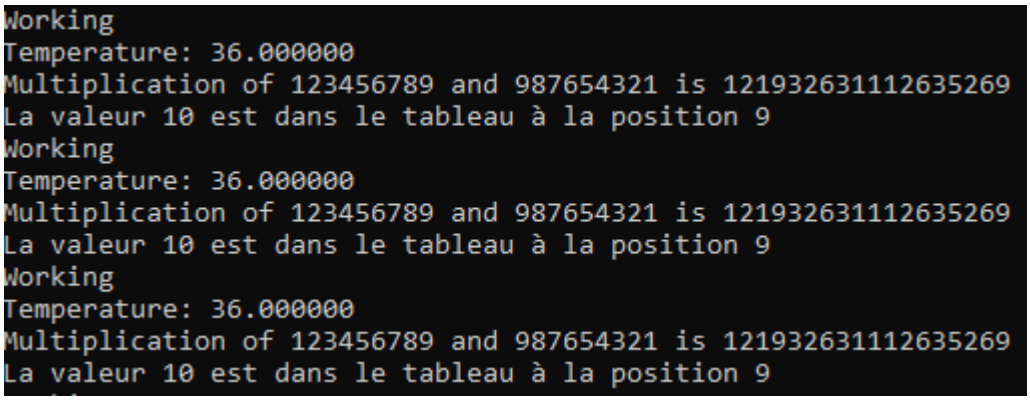
3.1 Task Schedulability Analysis

Calculations show that the entire task set can be successfully scheduled under the defined time constraints.

3.2 Execution Observations

We can observe the scheduling seems to perfectly process in the command prompt.

He is a screenshot of what we obtain.



The tasks appear to repeat as expected without out speed from task to another.

4. Conclusion

This project has demonstrated the feasibility of implementing an RTOS to manage a diversified set of tasks. The obtained results seem to be correct but are hard to measure without impacting the scheduling.

I had some difficulties to make my iterations on each task to measure the WCET, so I did a bash script and a Python script, but the Python script appear to give me more previse measures (in bash all is at 0 as if it round).

Finally, I calculated a period for each task according to the condition for scheduling. Then I implemented the founded period in the c script, and everything is ok and execute respecting the scheduling.