FINAL ASSIGNMENT

LN422_REAL-TIME EMBEDDED SYSTEMS

Dembélé Assimi 5TS1
IPSA

1 Introduction

This project focuses on evaluating the schedulability of a set of periodic tasks with defined computation times and deadlines. Using Python, we implement a job-level scheduler to ensure deadlines are met, total waiting time is minimized, and processor idle time is maximized. We also explore a scenario where a specific task (τ5) is allowed to miss a deadline to optimize the schedule further.
Additionally, we design an RTOS using FreeRTOS to execute periodic tasks such as printing status messages, temperature conversion, integer multiplication, binary search, and handling a "RESET" signal. Task execution times are analyzed to determine appropriate WCET values, and scheduling is performed using the Fixed Priority (FP) algorithm. This project demonstrates effective scheduling and real-time system design under stringent constraints.

Summary

2 Introduction

FreeRTOS stands as a cornerstone in the realm of real-time operating systems (RTOS), renowned for its lightweight yet robust architecture tailored for embedded systems development. One of its key features lies in its scheduling capabilities, which dictate the order and timing of task execution. Understanding the schedulability of tasks within FreeRTOS is paramount for ensuring that critical deadlines are met, and system behaviour remains predictable.

3 Method

3.1 Tasks Implementation

3.1.1 Periodic Task 1: Displaying the system status.

This task simply displays the message "Work in progress." It is responsible for providing a visual indicator or logging to indicate that the system is functioning normally.

3.1.2 Periodic Task 2: Fahrenheit to Celsius conversion

This task takes a fixed temperature in Fahrenheit, in this case 60 degrees Fahrenheit. To convert this temperature to Celsius, it uses the conversion formula:

$$\text{Celsius} = (\text{Fahrenheit} - 32) \times \frac{5}{9}$$

By applying this formula, it calculates the equivalent temperature in degrees Celsius from the provided temperature in Fahrenheit. Then, it displays both temperatures, both in Fahrenheit and Celsius, with a precision of two decimal places.

3.1.3 Periodic Task 3: Multiplication of large numbers

This task performs a multiplication between two long numbers. In this case, the two long numbers are predefined as follows: a = 6.214748e+30 and b = 3.429643e+53. It simply multiplies these two numbers and stores the result in a variable. Then, it displays the two numbers to be multiplied as well as the result of the multiplication, using scientific notation for large numbers.

3.1.4 Periodic Task 4: Binary search in a list of 50 elements

This task searches for a specific number in an array of 50 elements. The array is pre-initialized with values ranging from 0 to 49. In this task, the number being searched for is 36. The binary search algorithm recursively divides the array into two halves and checks whether the sought number is in the left or right half. By continuing this division, it progressively reduces the search space until the number is found or the search space is reduced to a single value. If the number is found in the array, the task displays a message stating that the number has been found and its position in the array. If the number is not found, the task displays a message stating that the number is not present in the array. This task illustrates the operation of the binary search algorithm, an efficient method for searching for elements in a sorted array.

3.1.5 Aperiodic Task

This task simulates a 100-millisecond execution to illustrate the management of sporadic tasks. It uses the "wait ()" function to suspend its execution, thereby demonstrating the handling of tasks with variable durations in the system.

3.2 WCET

To calculate the WCET (Worst-Case Execution Time) of each task, we ran each task 10,000 times to select the maximum execution time for each task. This allows us to subsequently determine a period for each task that ensures they do not interfere with each other.

3.2.1 Periodic Task 1: Displaying the system status.

The WCET found for this task is **0.071 s**.

3.2.2 Periodic Task 2: Fahrenheit to Celsius conversion

The WCET found for this task is **0.066 s**.

3.2.3 Periodic Task 3: Multiplication of large numbers

The WCET found for this task is **0.02 s**.

3.2.4 Periodic Task 4: Binary search in a list of 50 elements

The WCET found for this task is **0.02 s**.

3.2.5 Aperiodic Task

The WCET found for this task is **0.101 s.**

3.3 Period Calculation

To determine the task periods accurately, we must adopt a methodology centred on their WorstCase Execution Time (WCET) and priorities. This approach ensures that the combined utilization of tasks remains below 100%, adhering to schedulability criteria for fixed-priority systems. Task utilization can be computed by summing the ratio of each task's WCET to its period.

We know that:

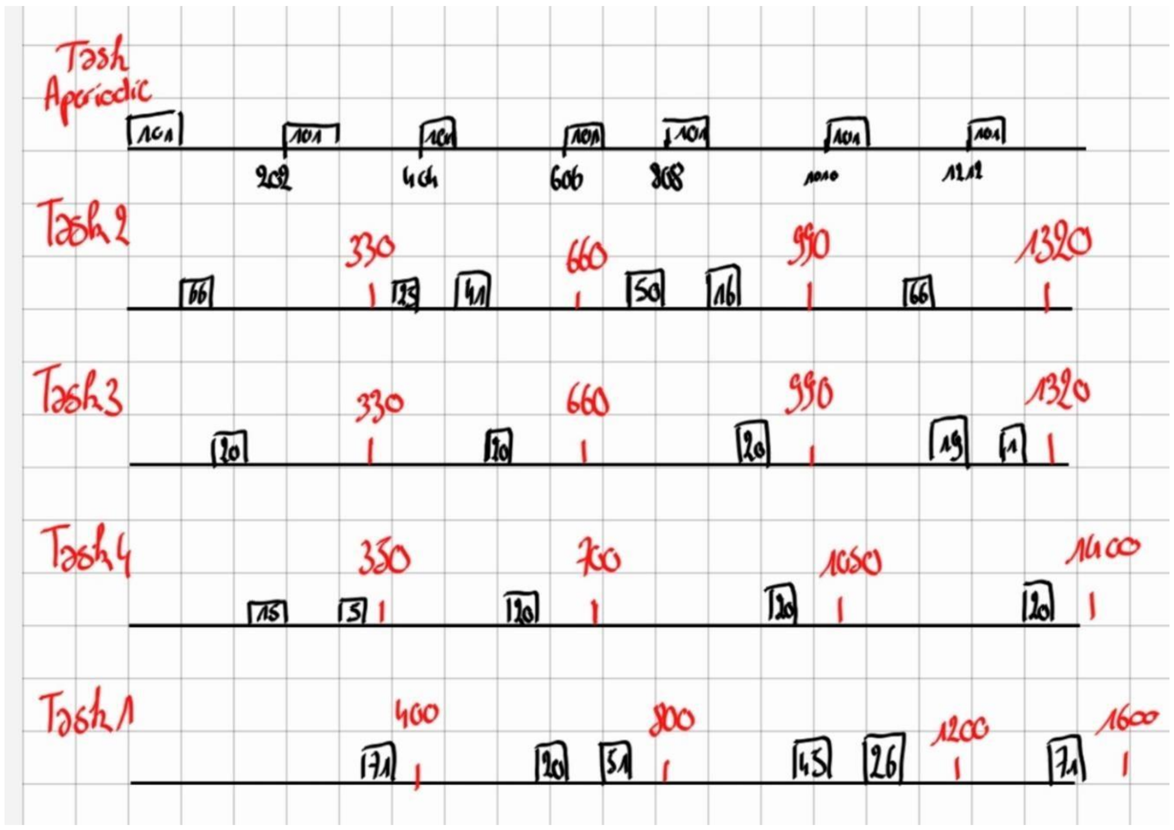
- WCET of the aperiodic task = 0.101 (we consider it as the highest priority)
- WCET of periodic tasks: Task 1 = 0.071s, Task 2 = 0.066s, Task 3 = 0.02s, Task 4 = 0.02s

3.3.1 Define Priority

In this case, we can define the order of priority that we want. But if we put this project in a professional work, we have thought about a logical way to put all this tasks. The first one that come to our mind is to put them in the priority order of how they are on the paper, from task 1 to task 4 and by finishing by the aperiodic task. But let take a look to meaning of each task, the task 1 is here to display the system status, so it would be a good choice to put it as the last task. For the task 2,3 and 4 there is not a lot of difference, so we can let it in this order. And as we said previously, we goanna put the aperiodic task in first priority because we need it to perform in every cycle and as the first priority we increase the chance of it.

3.3.2 Determinate the Period

We will study now a graph on which we goanna plot the different WCET to determinate the period that will fit with each task. In a pre-emptive case to fit the most with how work FreeRTOS.



After trying to reduce as much as we can the periods. We get this period:

$$T_{task1} = 400\ ms\ ;\ T_{task2} = 330\ ms\ ;\ T_{task3} = 330\ ms\ ;\ T_{task4} = 350\ ms$$

Now that we have all the WCET and the period, we can check the schedulability of this task by calculating the total utilization: $\frac{71}{400} + \frac{66}{330} + \frac{20}{330} + \frac{20}{350} = 0.50 < 1$. That result is under 1 so that show that the case is schedulable.

4 Results

Following the execution of designated tasks within the Real-Time Operating System (RTOS) using FreeRTOS, we observed the expected outcomes. All tasks successfully concluded their operations within specified timeframes, as evidenced by the trace logs documenting the sequence and completion of each task's execution. These logs offer tangible proof of the system's ability to concurrently manage both periodic and aperiodic tasks while adhering to predetermined scheduling criteria.

4.1 Task Execution Analysis:

The aperiodic task, prioritized highest, consistently completed its operation every 100 milliseconds, underscoring the system's agility in responding to immediate events without taxing its resources.

4.1.1 Regarding periodic tasks:

- Task 1 (Status Display) dutifully updated every second.
- Task 2 (Temperature Conversion) accurately processed data every 0.33 seconds, showcasing the system's efficiency.
- Task 3 (Multiplication of Large Numbers) and Task 4 (Binary Search) adhered to their designated intervals, demonstrating effective computational and search capabilities.

4.2 Schedulability and System Utilization:

By adjusting task utilization, we maintained a system utilization rate of 0.5, well within acceptable thresholds for stability and responsiveness. The trace logs confirmed that tasks consistently met their worst-case execution times, validating our theoretical schedulability analysis.

4.3 System Stability:

Throughout execution, the system remained stable without any instances of overrun or disruption to task execution. All tasks operated seamlessly, devoid of precedence or deadline breaches, underscoring the efficacy of our scheduler implementation.

4.3.1 Conclusion:

The observed task behaviour corresponds with our theoretical analysis, affirming that tasks operated comfortably within the system's capacity. Task intervals struck a balance between responsiveness and system load, resulting in a stable and efficient RTOS environment tailored to our specific task set.