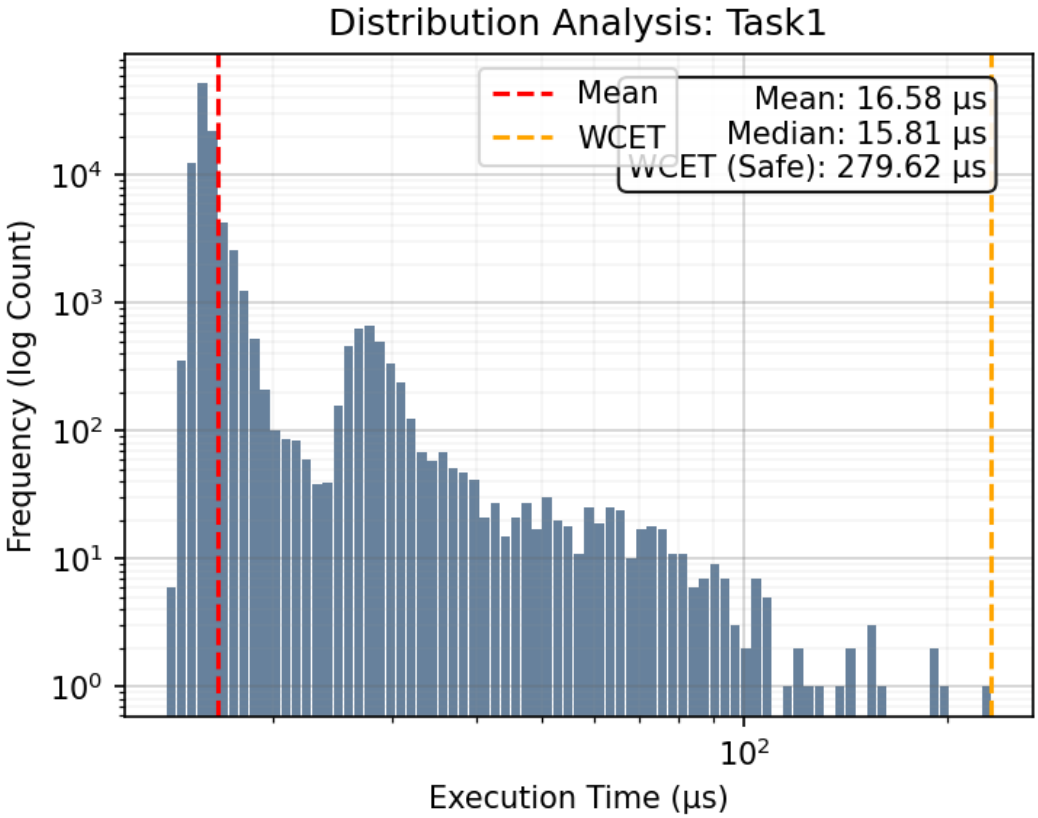
Year 2 assignment : Real-Time Scheduling and Probabilistic Analysis of Multi-Rate Tasks



Produced by : Yoann HOARAU

IPSA - Institut Polytechnique des Sciences Avancées

GitHub project link:
https://github.com/Yawane/IPSA_FreeRTOS

Academic year 2025 - 2026
8 January, 2026

Contents

1 Introduction

1.1 Context and objectives

This Final Project is based on the implementation and timing analysis of a real-time embedded system, developed using FreeRTOS [1]

The primary goal of this assignment is to demonstrate proficiency in Fixed Priority Scheduling (Preemprive and Non-Preemptive), and to perform a statistical analysis of task execution times. Thus, the key objectives are:

- Coding five distinct periodic tasks with varying computational requirements, explained in next subsection 1.2.
- Obtaining a statistical distribution of the execution times for each task with 100,000 data points to characterize their WCET[2].
- Perform schedulability analysis using Non-Preemptive and Preemptive Fixed priority.
- Calculating the probability of deadline misses under restricted timing conditions.

1.2 Task definitions

The system consists of five periodic tasks, as follows:

1. **System Status**: prints a "Working" string in the console to indicate nominal operation. This task has maximum priority.
2. **Thermal conversion**: converts a randomly generated Fahrenheit value to Celsius.
3. **Large Multiplication**: performs multiplication of two randomly generated `long long` numbers.
4. **Sorting**: Generate and sort a random list of 20 elements.
5. **Reset Monitor**: Executes every 20ms to monitor a '`RESET`' input and resets the parameter after detection.

1.3 Contraints

To ensure system compliance with real-time theory, the following constraints are:

- **Frequency ratios**: $f_2 \geq 4f_4 \quad | \quad f_3 \geq 3f_4$.
- **Processos Utilization**: the total CPU utilization (U) must not exceed 69% (Liu and Layland bound for guaranteed scheduability).
- **Deadlines**: the deadline is et at 80% of the measured WCET to evaluate the probability of failure.

[1] I was not able to use FreeRTOS for this project, the reasons will be explained in section 6.
[2] WCET: Worst Case Execution Time.

2 Code explanation

This section will detail the timing behaviour of the five periodic tasks detailed in the previous section 1.2, and the schedulability analysis. The goal is to understand their variance in execution time and measure their WCET.

2.1 Data collection

To accurately characterize the timing behaviour of the tasks, I used a combination of `C++` high resolution timing[3] and Python automation.

2.2 single task timing

Each of the five tasks is implemented in a dedicated `C++` file. To measure the execution time of a single iteration, the `std::chrono` library is utilized to achieve nanosecond precision.
Here is a example `C++` code to measure task execution:

```cpp
#include <iostream>
#include <chrono>

int main() {
    auto start = std::chrono::high_resolution_clock::now();
    // ... task execution
    auto end = std::chrono::high_resolution_clock::now();

    auto duration = std::chrono::duration_cast<chrono::nanoseconds>(end - start);
    std::cout << duration.count() << std::endl;
    return 0;
}
```

The system records start and end time using `std::chrono::high_resolution_clock::now()`, computed the duration and print the result to the console. The print allows the timing data to be "piped" to the Python script.

In fact, if the `C++` file output in the console "Working..." (during the task execution) and "20300" (as the duration of the execution), then the Python can read these standard outputs as a list `['Working...', '20300']`, thus access to the execution time with `int(output_lines[-1])`.

2.3 Automated Python data extraction

To reach the required sample size of 100,000 data points, a python function loop over the execution of the compiled `C++` files. It uses the `subprocess` module to capture console output as explained earlier.

Here is the simplified Python function:

```python
def execute_iterations(self, iterations: int = 100_000) -> None:
    # [...] setup
    for _ in range(iterations): # loop over iterations
        out = subprocess.run() # params for .exe file
```

[3]This is used with the package `<chrono>` and `chrono::high_resolution_clock::now();` command

```
5         output_lines = out.stdout.strip().splitlines()
6         raw_times.append(int(output_lines[-1])) # measured execution time.
```

The script is designed to extract the execution time from the last element of the output array.

2.4 Python schedulability analysis

The last section of the code is to evaluate the schedulability of the real-time system, focusing on the Response Time Analysis (RTA) for both preemptive and non-preemptive environments.

2.4.1 Preemptive schedulability

In a preemptive system, a high-priority task can interrupt a lower-priority task at any time. The code computes the Worst-Case Response Time (R) for each task. It is the time elapsed between the task being ready to run and the moment it finishes execution.

The response time is initially assumed to be to the task's own execution time (C). The code calculates how many times higher-priority tasks will interrupt the current task during its execution. Since these interruptions increase the total response time, the code recalculates the interference based on the new, longer duration. This repeats until the value of R stabilizes (converges) or exceeds the task's period (T). If $R \leq T$ for all tasks, the system is considered schedulable.

2.4.2 Non-preemptive schedulability

In a non-preemptive system, once a task starts executing, it cannot be interrupted by any other task, even if a higher-priority task arrives. This introduces a delay (B).

The response time is calculated as:

$$R = C + B + Interference$$

If the combined time of its own execution, the blocking from a lower task, and the interference from higher tasks is less than its period, the task passes.

3 Results and analysis

3.1 Results n°1

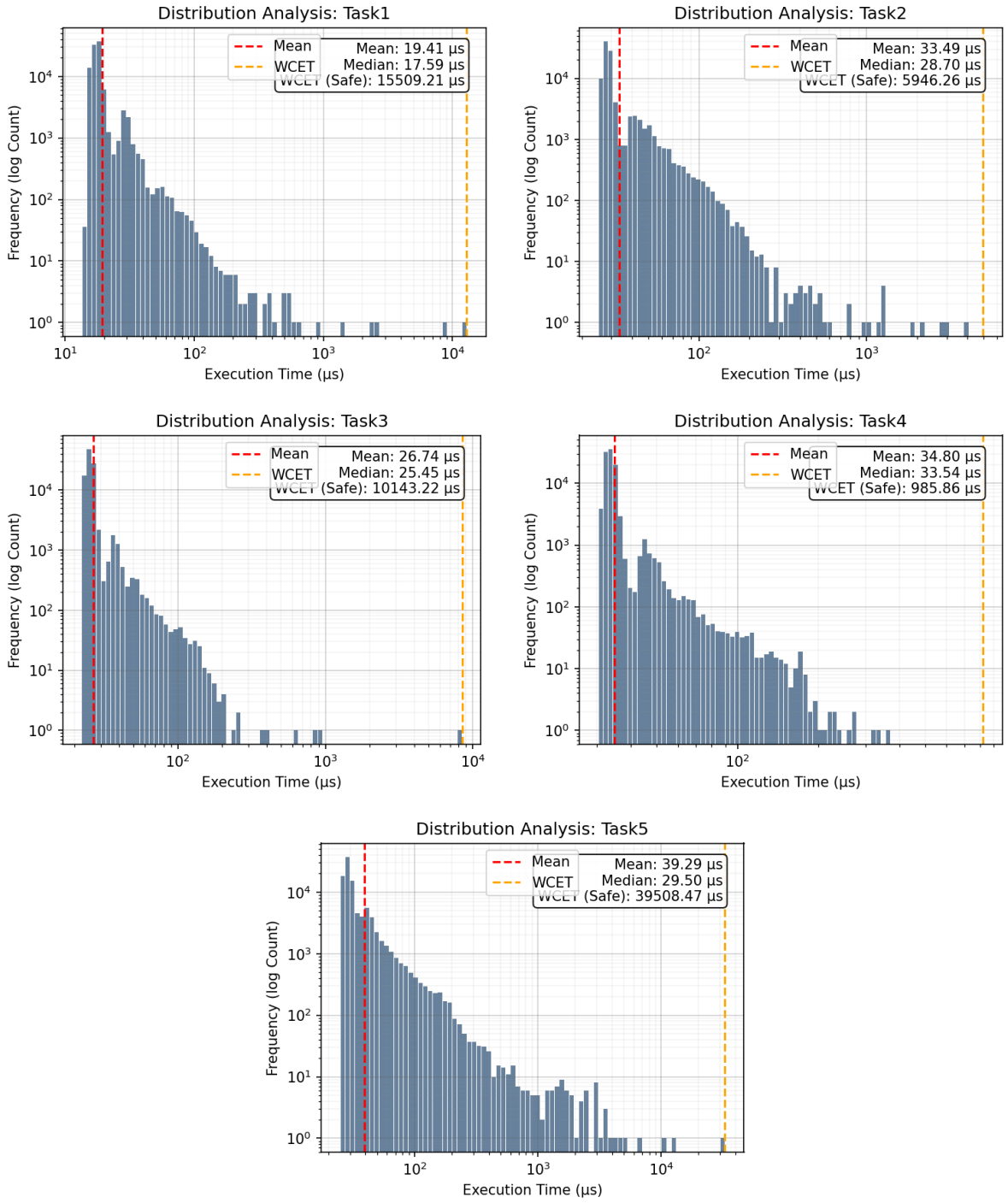


Figure 1: Timing distribution for all five tasks with 100,000 data points.

Each graph shows the distribution of execution times for tasks one to five, with 100,000 data points. The y axis are logarithmic in order to see rare events, such as an execution

time of 10,000 µs for Task 1 even though the mean and median are about 19µs.

3.1.1 WCET determination

In all graphs of **Figure 1**, the yellow vertical line shows the measure WCET, and the legend writes the safe WCET, with 20% margin.

In a deterministic real-time system, this value should be close to the mean. However, the empirical data collected on a Windows OS shows significant outsiders and extreme execution times. As illustrated in the logarithmic distribution plots, while 99.9% of executions occur near median, rare events caused execution spikes up. These spikes are artifacts of the non-deterministic testing environment.

3.1.2 Schedulability analysis

In order to get a PASS on all RTA, we should find better value for the Task 4 period `base_t4`, such that the total utilization if below 69 %.

As the console output message showed, the execution timings are supposed fixed parameters: $C_1 = 15,51\,\text{ms}$; $C_2 = 5,95\,\text{ms}$; $C_3 = 10,14\,\text{ms}$; $C_4 = 0,99\,\text{ms}$; $C_5 = 39,51\,\text{ms}$.

The fixed periods T are: $T_1 = 100\,\text{ms}$; $T_5 = 200\,\text{ms}$.

The total utilization U is the sum of individual utilizations C_i/T_i. Note that $T_2 = T_4/4$ and $T_3 = T_4/3$ is a constraint. The equation is then:

$$U = \underbrace{\left(\frac{C_1}{T_1} + \frac{C_5}{T_5}\right)}_{\text{Fixed utilization}} + \underbrace{\left(\frac{4 \cdot C_2 + 3 \cdot C_3 + C_4}{T_4}\right)}_{\text{Variable utilization}}$$

$$\iff \quad U(T_4) = 0.3527 + \frac{55.21}{T_4}$$

To correctly have a total utilization $U < 0.69$, we find T_4 as:

$$0.69 = 0.3527 + \frac{55.21}{T_4} \Longrightarrow T_4 \approx 163.7\ \text{ms}$$

3.2 Results n°2

Because the execution timings are not fixed parameters, the calculated period for task 4 is not right all the time. So, to be always safe, lets take $T_4 = 450$ ms.

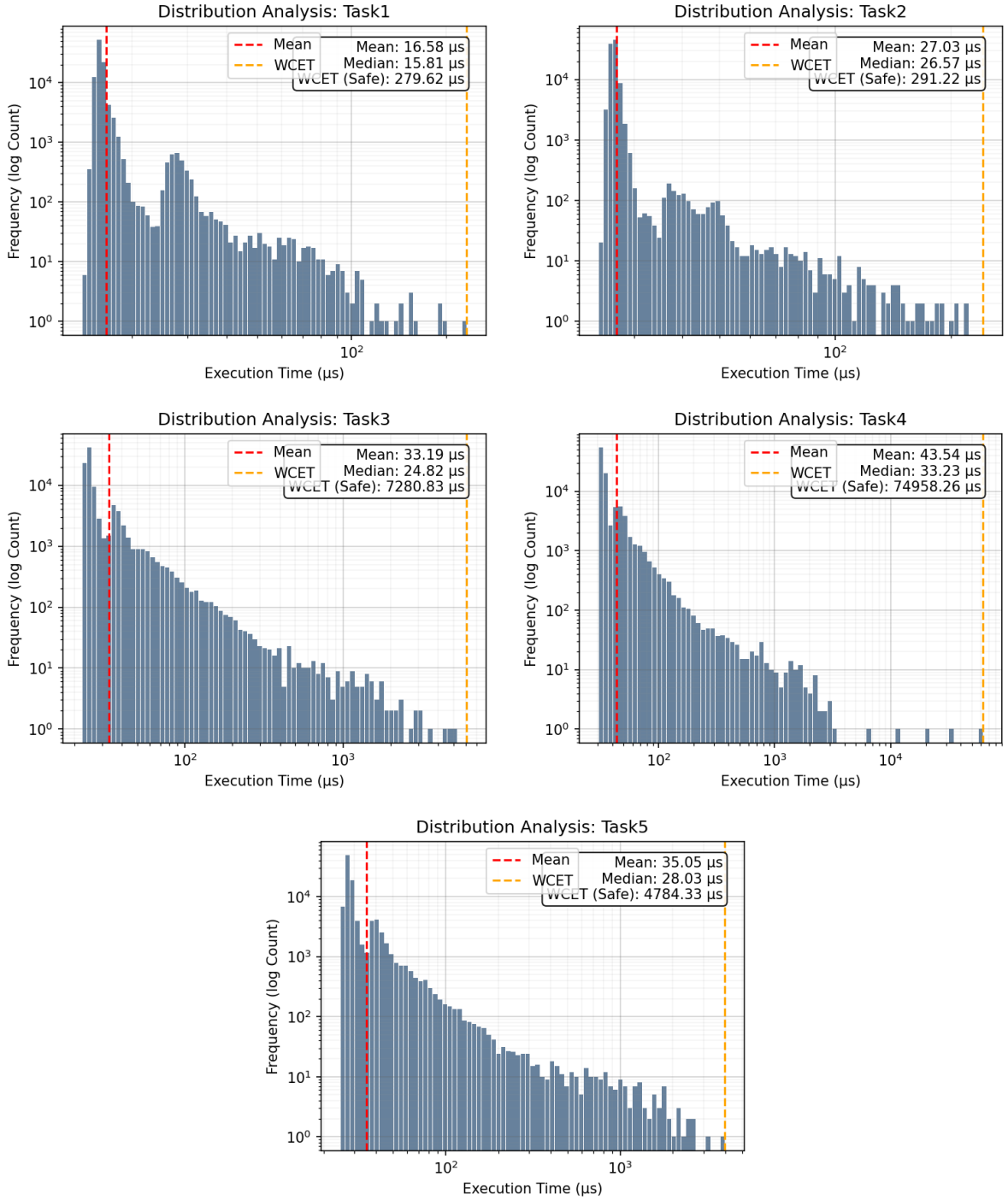


Figure 2: Timing distribution for all five tasks with 100,000 data points.

A comparison of the results reveals that the WCET (Worst-Case Execution Time) values are not identical, causing slight variations in the means due to extreme outliers. However, the median serves as a superior metric for this analysis as it is less sensitive to these anomalies. Across all tasks, the medians remain consistent, reflecting a stable execution behavior regardless of isolated timing spikes.

4 Schedulability Analysis output

Here is the console output of the schedulability analysis:

```
=== SYSTEM CONFIGURATION ===
Task1: Period=100ms, Priority=6
Task2: Period=112.5ms, Priority=4
Task3: Period=150.0ms, Priority=3
Task4: Period=450ms, Priority=1
Task5: Period=200.0ms, Priority=2

=== UTILIZATION ANALYSIS ===
  Task1:   0.28% (C=0.280ms, T=100.0ms)
  Task2:   0.26% (C=0.291ms, T=112.5ms)
  Task3:   4.85% (C=7.281ms, T=150.0ms)
  Task4:  16.66% (C=74.958ms, T=450.0ms)
  Task5:   2.39% (C=4.784ms, T=200.0ms)
  TOTAL:  24.44% / Max: 69%

=== RTA (PREEMPTIVE) ===
  [PASS] Task1: R=0.2796ms <= T=100ms
  [PASS] Task2: R=0.5708ms <= T=112.5ms
  [PASS] Task3: R=7.8517ms <= T=150.0ms
  [PASS] Task4: R=87.5943ms <= T=450ms
  [PASS] Task5: R=12.6360ms <= T=200.0ms

=== RTA (NON-PREEMPTIVE) ===
  [PASS] Task1: R=75.2379ms (B=74.958ms)
  [PASS] Task2: R=75.5291ms (B=74.958ms)
  [PASS] Task3: R=82.8099ms (B=74.958ms)
  [PASS] Task4: R=163.1234ms (B=74.958ms)
  [PASS] Task5: R=87.5943ms (B=74.958ms)

==============================
       FINAL VERDICT
==============================
Utilization Check:      [PASS]
Preemptive Sched:       [PASS]
Non-Preemptive Sched:   [PASS]
==============================
```

As the output says, the total processor demand is 24.44 %, which is well below the 69 % theoretical limit. This indicates that the system is not overloaded and has significant computational headroom.

Under preemptive conditions, where high-priority tasks can interrupt lower-priority ones, all tasks meet their deadlines.

- Highest Priority (Task 1): Has a very short response time ($R = 0.2796$ ms) as it is never interrupted.
- Lowest Priority (Task 4): Experienced the longest response time ($R = 87.5943$ ms) due to frequent interruptions from all other tasks, but still finished well within its 450 ms period.

In the non-preemptive mode, tasks cannot be interrupted once they start. This introduces a Blocking (B) factor, where a high-priority task must wait for a lower-priority task to finish.

- Impact of Task 4: Because Task 4 has the longest execution time ($\approx$ 75 ms), it creates a significant blocking delay for all other tasks.

- Result: Even with this heavy blocking penalty, the response times remain below the required periods. For example, Task 1's response time jumps from 0.28 ms to 75.23 ms, but since its deadline is 100 ms, it still passes.

he system is fully schedulable under both preemptive and non-preemptive disciplines.

5 Probabilistic Deadline Miss Analysis

This section analyzes the probability of a task exceeding a specific temporal threshold, referred to here as the Deadline Risk.

The "Deadline Risk" is calculated by determining the percentage of total iterations where the execution time exceeded 50 % of the measured WCET.

```
=== BENCHMARK PHASE ===

>> Benchmarking Task1 (100000 cycles)...
  -> Stats for Task1:
     Mean: 16584.49 ns | WCET(1.2x): 279620.40 ns
     Deadline Risk (50.0% WCET): 0.0150%

>> Benchmarking Task2 (100000 cycles)...
  -> Stats for Task2:
     Mean: 27034.87 ns | WCET(1.2x): 291220.80 ns
     Deadline Risk (50.0% WCET): 0.0410%

>> Benchmarking Task3 (100000 cycles)...
  -> Stats for Task3:
     Mean: 33185.76 ns | WCET(1.2x): 7280832.00 ns
     Deadline Risk (50.0% WCET): 0.0060%

>> Benchmarking Task4 (100000 cycles)...
  -> Stats for Task4:
     Mean: 43540.71 ns | WCET(1.2x): 74958261.60 ns
     Deadline Risk (50.0% WCET): 0.0020%

>> Benchmarking Task5 (100000 cycles)...
  -> Stats for Task5:
     Mean: 35053.66 ns | WCET(1.2x): 4784328.00 ns
     Deadline Risk (50.0% WCET): 0.0110%
```

As we can see, each deadline risk is negligible. The maximum risk is for Task 2 at 0.041 %, corresponding to 41 miss on a 100,000 dataset. The minimum risk is for Task 4 at 0.002 %, corresponding to 2 deadline missees only.

If we take 80 % as a risky WCET, then the deadline miss will be even more negligible

6 FreeRTOS

Originally, the integration of FreeRTOS was intended to manage task scheduling for this project, as specified in the assignment requirements. Extensive attempts were made to configure the FreeRTOS environment, specifically targeting the `FreeRTOS/Demo/Posix_GCC` simulator. However, several issues prevented a successful compilation: The `make` command generated persistent errors across multiple environments, including Windows Terminal, PowerShell and WSL (Linux subsystem). These errors prevented the creation of a stable

build of the RTOS kernel, making it impossible to produce a working executable.

The project's progress was then complicated by an unexpected hardware change (my PC was stolen, but my work was saved in the cloud). The work on FreeRTOS on a borrowed third-party PC was too complex and sketchy. Consequently, I opted for an alternative route using Python automation and `C++` high-resolution timing to fulfill the statistical and schedulability analysis requirements.

7 Conclusion

This project successfully characterized the timing behavior of five periodic tasks. By collecting 100,000 data points per task, the execution timing distributions were accurately plotted, showing that while rare execution spikes occur to the non-deterministic environment, the median performance remains stable.

In this work, we did the task characterization to provide provide Mean, Median and WCET metrics. To respect the 69 % utilization constraint, Task 4's period was set to 450 ms, resulting in a total CPU load of 24 %. The RTA confirmed that the system is fully schedulabble under both preemptive and non-preemptive scheduling. The risk assessment demonstrated a negligible deadline mess risk (below 0.05 %).

Useful links to GitHub and code

GitHub link:
https://github.com/Yawane/IPSA_FreeRTOS

These are the elements in the repository:

- `new_main.py`: The main python code.
- `Task1.cpp`: C++ file of Task1.
- `Task2.cpp`: C++ file of Task2.
- `Task3.cpp`: C++ file of Task3.
- `Task4.cpp`: C++ file of Task4.
- `Task5.cpp`: C++ file of Task5.
- `Report_HOARAU`: This actual PDF you are reading right now.