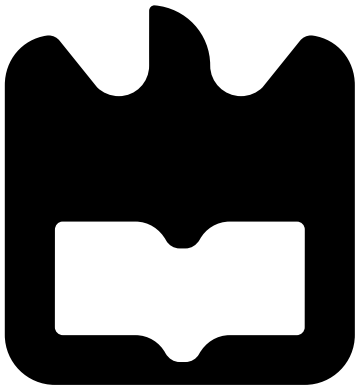

Zephyr RTOS Project Report

107474-Joseane Pereira
109050-Gabriel Costa
Universidade de Aveiro, DETI

December 17, 2024

Contents

1 Introduction

The aim of this project is to apply the Linux Real-Time Services and the Real-Time Model to the the development of a real-life inspired real-time application.

Following the typical structure of embedded software, the project encompasses a set of cooperating tasks, involving synchronization, shared resources, access to a real-time database, etc.

2 Architecture

2.1 System Overview

In this project, the real-time monitoring system is structured around five primary tasks, each with distinct roles that simulate components in an industrial environment:

- **Sound Generation Task**: This task generates sound signals that simulate the acoustic output of a motor. The generated sound data serves as the foundation for speed measurement and fault detection tasks, mimicking real-world operational noise from industrial machinery.
- **Speed Measurement Task**: The purpose of this task is to calculate the motor's rotational speed based on the generated sound signals. The task calculates the most powerful frequency, which corresponds to the frequency of the motor, and then converts it to RPM, representing the motor speed.
- **Fault Detection Task**: This task is responsible for identifying potential issues in the motor, such as bearing faults. It processes the sound data to detect specific low-frequency signals that may indicate these issues. By monitoring frequencies below 200 Hz and comparing their amplitude with the motor frequency, this task can flag conditions that suggest operational concerns.
- **Print RTDB Task**: Outputs from the above tasks are stored in a Cyclic Access Buffer (CAB), which allows the data to be periodically stored and retrieved. This task prints the results at regular intervals and shows the accumulated data from the buffer on a plot, providing a real-time overview of motor speed and any detected faults.
- **Playback**: This task allows for real-time audio playback of the sound generated by the first task. This functionality provides an additional verification layer, letting users monitor the sound data directly to ensure accurate sound generation and processing.

2.2 Data Structures and Access Methods

The primary data structure used in this project is a Cyclic Access Buffer (CAB). The CAB is implemented to manage data flow between the sound generation task and the subsequent processing tasks in a synchronized and efficient manner. This approach is well-suited for real-time applications where multiple tasks need access to the latest data without conflicts or delays.

- **Buffer Allocation**: The CAB allocates dedicated buffers to store sound samples as they are generated. The task searches for an available buffer—one that is not currently in use—before writing data to it. This approach prevents data corruption from concurrent access and ensures that each task operates on a stable, unmodified set of data.

- **Data Access by Processing Tasks**: To read data from the CAB, processing tasks first identify the buffer that was most recently updated. Before accessing this buffer, they increment a counter that tracks the number of tasks currently using it. This counter mechanism prevents new data from being written to the buffer while it is actively in use. Once the tasks have finished reading, they decrement the counter, signaling that the buffer is available for future writes.

2.3 Synchronization Mechanisms

The implementation relies on multiple threads that perform specific tasks concurrently. To ensure proper coordination, various synchronization methods are applied, which are crucial to prevent conflicts and ensure that timing requirements are respected across the tasks. Below are the synchronization techniques identified:

- **Clock-based Periodic Synchronization**: All threads use the function "clock_nanosleep" with the "TIMER_ABSTIME" flag and "CLOCK_MONOTONIC" as the reference clock to maintain periodic execution intervals.

  Each thread calculates the time of its next execution cycle and then waits for this specific absolute time, which is computed using a custom function "TsAdd" to increment the target time with the thread's predefined period.

- **Mutex and Locking in CAB Functions**: The CAB structure uses mutexes to control access to shared resources. This prevents tasks that are reading from the buffer from simultaneously modifying the counter, which tracks the number of tasks currently using the buffer.

sound gen execution time: 4.69ms speed measurement execution time: 5.19ms bearing issues execution time: 2.31ms RTDB print execution time: 0.03ms

3 Discussion of the task priorities, activation rates and system schedulability

Each task in the system is assigned a specific priority and activation rate to ensure timely execution and prevent task starvation. The priorities, activation rates, and periods are configured to meet real-time requirements and ensure system schedulability. The following table outlines the task priorities and activation periods.

Task	Priority	Activation Period	Description
Sound Generation	1	3s + 200ms	Generates sound signals representing the motor's acoustic output.
Speed Measurement	2	3s + 800ms	Analyzes sound data to measure motor speed by detecting the dominant frequency in the spectrum.
Bearing Issues	3	6s + 800ms	Detects potential faults in the motor by checking for low-frequency noise (below 200 Hz).
Playback	4	4s + 900ms	Plays back the generated sound for monitoring and debugging purposes.
Database Print	5	4s + 800ms	Prints the real-time database contents, including speed and fault information.

Table 1: Task Priorities and Activation Periods

3.1 Priority Assignment and Activation Period Rationale

Priority Assignment Rationale: Tasks are prioritized based on their importance in the data processing chain:

- **Sound Generation** has the highest priority (1) because it provides the data required by subsequent tasks.
- **Speed Measurement** (priority 2) and **Bearing Issues** (priority 3) are secondary since they need the generated sound data to operate correctly.
- **Playback** (priority 4) and **Database Print** (priority 5) are dependent on the other tasks, so they have lower priorities.

Activation Period Rationale: The periods are selected to balance the need for fresh data and processing time.

- **Sound Generation** has the shortest period to ensure continuous data flow.
- **Speed Measurement** has a longer period as to allow the generation of sound data before processing.
- **Bearing Issues** has an even longer period to not only allow sound generation but also speed measurement, but as bearing issues are less frequent, it can have a longer period.
- **Playback** and **Database Print** activate slightly less frequently beacuse they are dependent on the other tasks.

3.2 System Schedulability

To confirm that all tasks meet their deadlines, we perform a schedulability analysis using the system's utilization factor.

Utilization Factor (U): Given that each task is independent and periodic, we can calculate the utilization factor U using:

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

where C_i is the computation time and T_i is the period of each task. Assuming each task completes within its assigned period, this calculation helps ensure that the system remains schedulable. Empirical validation was conducted by observing actual execution times to confirm that all tasks meet their deadlines.

For our sistem, the utilization factor is calculated as follows:

$$U = \frac{4.69}{3.2} + \frac{5.19}{3.8} + \frac{2.31}{6.8} + \frac{0.10}{4.8} + \frac{0.03}{4.8} = 2.44$$

The utilization factor is above 1, indicating that the system is overloaded and may not meet all deadlines. However, the empirical validation shows that all tasks are executed within their assigned periods, ensuring that the system remains schedulable. This demonstrates that the system can handle the workload effectively and meet real-time requirements.

And here is the empirical validation of the system schedulability:

Task	**Activation Period**	**Execution Time**	**Deadline**	**Difference**	**Status**
Sound Generation	3.2s	4.69ms	3.2s	0.49ms	Met
Speed Measurement	3.8s	5.19ms	3.8s	1.39ms	Met
Bearing Issues	6.8s	2.31ms	6.8s	0.51ms	Met
Playback	4.8s	0.10ms	4.8s	0.10ms	Met
Database Print	4.8s	0.03ms	4.8s	0.03ms	Met

Table 2: Empirical Validation of Task Schedulability

4 Task Execution Sequence and Relevant Events

The system follows a well-defined sequence to maintain real-time performance. Below is the order of task execution and relevant interactions:

1. **Sound Generation**:
    - Generates sound signals simulating motor noise and stores the wave in the Cyclic Access Buffer (CAB).
2. **Speed Measurement**:
    - Reads wave data from the CAB to calculate the motor's speed based on the dominant frequency.
    - Converts the frequency data to RPM and records it in the Real-Time Database (RTDB).

3. **Bearing Issues**:
    - Checks for low-frequency noise in the generated wave (indicating potential motor faults).
    - Logs the fault frequency and amplitude in the RTDB if an issue is detected.
4. **Playback**:
    - Retrieves the wave from the CAB and plays it back for real-time monitoring.
5. **Database Print**:
    - Periodically prints the contents of the RTDB, including motor frequency, RPM, and detected faults.
    - Every three cycles, it generates a Gnuplot chart showing RPM and fault data over time.

4.1 Relevant Events and Inter-Task Communication

Buffer Operations (CAB): The CAB is critical for data synchronization:

- The **Sound Generation** task writes to the CAB, while **Speed Measurement** and **Bearing Issues** read from it.
- A user counter within the CAB manages concurrent access to prevent data corruption.

Real-Time Database (RTDB): The RTDB serves as a shared repository for task outputs:

- **Speed Measurement** and **Bearing Issues** update the RTDB with speed and fault data.
- **Database Print** accesses the RTDB to present the latest information.

4.2 Task Execution Timeline

Figure **??** illustrates the task activation times, periods, and interactions within a 5-second window.

Figure 1: Gantt chart of the task execution sequence

The chart shows:

- Task Start/Finish Times: Each task's execution times are represented, demonstrating how tasks interact and avoid overlaps.
- Buffer and Database Operations: Annotations illustrate when tasks read from or write to the CAB and RTDB.
- Synchronization Events: Events like CAB writes and RTDB updates are highlighted to clarify data flow.

This diagram and description illustrate a robust execution flow, ensuring that each task performs its designated function within the required time frame, achieving the system's real-time goals.

5 Tests

The Sound Generation task produces a waveform with a frequency range from 2 kHz to 5 kHz.

After five cycles, the Sound Generation task begins to introduce substantial noise to the generated sound.

6 Results

In the Measuring Speed task, the most powerful frequency detected always falls within the expected range, confirming that frequency analysis is functioning as expected.

The noise generated before is promptly detected by the Bearing Issues task, with its presence illustrated in the graph generated by the RTDB printing task and displayed in the terminal. The Playback task further confirms this change, as the audio output shifts from the initial smooth waveform to a noticeably distorted sound, verifying that the noise has been successfully introduced and detected.

7 Analysis

8 Conclusion

In this project, a multi-threaded system was successfully implemented to generate, process, and monitor audio signals in real-time. Through a coordinated set of tasks, each functioned as intended: the Sound Generation task produced audio with variable frequencies, which were accurately detected and analyzed by the Measuring Speed and Bearing Issues tasks. The synchronization mechanisms, including mutexes in the Circular Audio Buffer (CAB), prevented data races and ensured stable inter-thread communication. Additionally, the Playback task provided an audible confirmation of sound changes, which were corroborated by visualizations of signal data in the database printing task.

Overall, this project demonstrated effective use of periodic task scheduling, synchronization, and signal processing to achieve reliable sound generation and real-time monitoring.