Dylan MAGDELEINE

Robin BARSCZUS

4TS1

FINAL ASSIGNMENT

Real time information systems

Jasdeep Singh

April 2024

Table of contents

Abstract

This report outlines the theoretical development and implementation of a Real-Time Operating System (RTOS) using FreeRTOS, designed for managing specific tasks with stringent real-time constraints. Due to incompatibility issues with the VSCode development environment, the practical execution and testing of the system's functionalities could not be conducted. Instead, the project focuses on a detailed conceptual analysis and hypothetical modeling of task execution, priority handling, and worst-case execution time (WCET) estimation, adjusted by safety factors to ensure reliability under adverse conditions.

INTRODUCTION

This project undertakes the design and implementation of a Real-Time Operating System (RTOS) tailored for specific tasks in the domain of Real-Time Information Systems.

The project encompasses the creation of an RTOS capable of handling a set of defined tasks with varying complexities and real-time constraints. These tasks include periodic operations such as status monitoring, data conversion, arithmetic computations, and search algorithms.

Each task is meticulously analyzed to determine its Worst Case Execution Time (WCET) and scheduling requirements. The system's schedulability is evaluated to ensure that all tasks meet their timing constraints without violating system integrity.

Furthermore, the project involves the utilization of FreeRTOS, Through the integration of FreeRTOS, the project aims to demonstrate a practical implementation of real-time systems principles in a tangible software environment.

Method

1. Development Environment and Simulation Tools

- **RTOS Platform**: FreeRTOS was selected for its robust task management features and adaptability in embedded systems.
- **IDE and Tools**: VSCode was used for code development and structuring, albeit with noted compatibility issues for direct RTOS execution. Theoretical development was supported by pseudocode and flowcharts to outline task logic and system behavior.
- **Simulation Strategy**: Given the limitations with direct execution, a detailed hypothetical model was constructed to represent task operations, queue management, and timer functionalities.

1. Task Design and System Configuration

- **Task Prioritization**: Five tasks were conceptualized with varying priorities:
  - **Task 5** (Highest Priority): Handles aperiodic critical events, requiring immediate attention upon trigger.
  - **Task 1** (Lowest Priority): Executes non-critical system status updates at regular intervals.
- **Queue and Timer Setup**: A single queue facilitates communication between tasks. A software timer is incorporated to manage periodic and aperiodic triggers, ensuring tasks execute according to their timing requirements.

2. Execution Time Analysis

- **WCET Estimation**: Lacking empirical data, WCETs were theoretically derived based on typical execution paths for each task. Safety margins were calculated by applying a factor of 1.2 to the estimated times, aiming to cover unforeseen delays and provide a buffer against system overloads.

Results

1. Hypothetical Task Performance and Analysis

- **Task 1 - System Monitoring**: Scheduled every 100 ms to print a status message. Estimated WCET is set at 1 ms, considering only the time to output a string to the console.

- **Task 2 - Temperature Conversion**: Converts temperatures every 500 ms. Given its simplicity, the WCET is estimated at 2 ms.
- **Task 3 - Arithmetic Computation**: Performs multiplications of large integers, chosen for its computational intensity, with a WCET estimated at 10 ms, reflecting the possible processor cycles required.
- **Task 4 - Binary Search**: Utilizes a classic algorithm to search within a static list, triggered manually with a WCET of 6 ms, based on logarithmic complexity.
- **Task 5 - Aperiodic Event Handling**: Designed to react to external events without a fixed period, with its execution speed critical but not empirically tested.

1. Challenges and Observations

- **Incompatibility Issues**: The primary challenge was VSCode's inability to directly run FreeRTOS, necessitating a theoretical approach to system design and validation.
- **Safety Margins and Reliability**: The use of a 1.2x safety factor on estimated WCETs addresses potential deviations and ensures that the system can handle tasks reliably even under peak loads or unexpected delays.

Conclusion

This theoretical exploration underscores the potential of RTOS designs like FreeRTOS in effectively managing tasks with diverse operational and timing requirements. The detailed hypothetical modeling provides valuable insights into task scheduling and prioritization within an RTOS. Future directions should aim at practical implementations and empirical validations to refine the designs and enhance system performance, particularly focusing on real-world environments where FreeRTOS can be executed and tested directly. Further studies could also explore the integration of more complex task interactions and error handling mechanisms to assess system robustness under a variety of operational conditions.