Real-Time Embedded Systems, 7.5 hp Assignments

Assignment 1

Objectives

- Set up and familiarize yourself with the lab environment (e.g., Raspberry Pi (RPi) and ARM Cross Compiler).
- Understand the basic C programming concepts required for the rest of the course.
- Acquire basic skills for Raspberry Pi bare-metal programming.

Part 1 - Bitwise operators in C

Bitwise operators are commonly used to manipulate bit sets and bit masks. The objectives of Part 1 are to learn how to use bitwise operators and encapsulate them into functions, including the required documentation as comments in `iregister.h`. You should also implement the functions, including post- and pre-conditions, in `iregister.c`. Additionally, implement a user interface showing the result of each function.

Part 2 - Bare-metal LED control via the Raspberry Pi GPIO

This part involves controlling an LED using the Raspberry Pi GPIO.

Assignment 2

Objectives

- Strengthen your skills in working with the Raspberry Pi, particularly concerning input and output.
- Experiment with concurrency (without any underlying support) and observe the main issues.

Part 1 - PiFace Control and Display

The PiFace Control and Display allows the display of 16 characters by 2 lines. The objective of Part 1 is to complete the implementation of a C library for the PiFace Control and Display.

Part 2 - Taylor expansion of e^x

In Part 2, you will implement the Taylor expansion of the exponential function e^x. This algorithm has linear time complexity O(n), where n is the input value. The Taylor expansion of e^x will be used in Part 3 to concurrently execute with another task.

Part 3 - Manual Interleaving

This part explores event-driven programming in embedded systems. Students will build kernels that execute two tasks according to an offline scheduler. Tasks are executed in sequence at a fixed rate in a loop, and certain tasks may be limited to every N turns of the loop. The goal is to build a kernel that calculates and displays the result e^n while blinking an LED at a constant rate.

Assignment 3

Objectives

- Strengthen your skills in working with the Raspberry Pi, particularly concerning concurrent programming.
- Understand, use, and modify the internals of a small multi-threading kernel called tinythreads.
- Implement cooperative multitasking (aka non-preemptive multitasking) using a lightweight multi-threading kernel implemented with non-local gotos.

Part 1 - Tinythreads

This part involves understanding the Tinythreads library, a lightweight multi-threading kernel implemented using non-local gotos. It has been modified and ported to RPi 3 (ARMv8) for this assignment.

Part 2 - Testing your understanding of Tinythreads

Part 3 - Cooperative multitasking using TinyThreads

In this part, you will implement cooperative multitasking using TinyThreads. Cooperative multitasking allows multiple tasks to execute concurrently and voluntarily yield control to another task.

Assignment 4

Objectives

- Implement preemptive multitasking using a modified version of tinythreads.
- Add mutual exclusion to the modified version of tinythreads.
- Implement Round Robin, Rate Monotonic, and Earliest Deadline First scheduling algorithms.

Part 1 - Round Robin with Timer Interrupts

This part involves implementing the Round Robin scheduling algorithm. Tasks run for predetermined time slices, and after each interval, the running task is preempted, allowing the next task in line to run.

Part 2 - Rate Monotonic Scheduling

Rate Monotonic (RM) scheduling assigns task priorities based on their periods. Shorter periods result in higher task priorities.

Part 3 - Earliest Deadline First Scheduling

The Earliest Deadline First (EDF) scheduling algorithm calculates task deadlines and adjusts priorities accordingly. Tasks with closer deadlines have higher priorities, but EDF scheduling introduces computational overhead.