Requirement Analysis Document (RDA)

ROSA - Real time Operating System for AVR32

Lamija Hasanagić
Zhaneta Nene
Tin Vidović

November 2019

Contents

1 Introduction

The project aims to extend the functionalities of the Real-time operating system for AVR32 (ROSA). ROSA is a small cooperative operating system for AVR32 microcontrollers and provides sufficient features for small electronic devices [2]. The current version of ROSA has no preemptive scheduling, task priorities, dynamic task handling, and resource protection. For ROSA to support safety-critical systems, like wheel loaders, it is, therefore, necessary to expand the functionalities of the operating system.

2 Current System

The system under development is an extension of an existing real-time operating system by the name of ROSA. It is a small-scale real-time kernel for the AVR32 UC3A processor line. The kernel is aimed at the Atmel EVK1100 platform.[1]

ROSA provides pseudo-parallel execution of user tasks. As the AVR32 UC3A processors are single-core processors ROSA provides mechanisms that enable task switching in order to simulate parallelism by serving the different tasks alternatively. In reality only a single task is executing at any given time instance but the provided mechanisms make it seem as though all of the tasks are being executed in parallel.[1]

The current system supports the creation of tasks before run-time. Structures are provided which contain the currently executing task and its Task Control Block as well as a TCB list of all other tasks. Currently ROSA is a cooperative operating system meaning only a yield function is provided which stops the execution of the currently executing task and starts the execution of the next task.[2]

The ROSA operating system has so far successfully been used, in its current capacity, in small electronic devices including vacuum cleaners, laundry machines and similar devices.[2]

New customers and company expansion now elicit new and improved functionality to the existing one. More specifically, but not limited to, ROSA is meant to be used as Real-time operating system for a series of wheel-loaders which have up to five Electronic Control Units each performing around a dozen tasks. These new use-cases dictate a need for extended functionality and safety mechanisms described in detail in the following section.[2]

3 Proposed System

3.1 Overview

In order to achieve the newly imposed functional and safety requirements an extension to the existing functionalities is required. More specifically, ROSA should now implement a fixed priority preemptive scheduling, with clock ticks to enable time sharing between created tasks. Tasks should also be able to be created on the fly as opposed to before run-time. To enable protection of critical code sections and regulate access to shared resources semaphore handling functionality is also required along with an appropriate synchronization protocol.

In the following subsections functional and nonfunctional requirements are described.

3.2 Functional Requirements

- The system shall provide fixed priority preemptive scheduler
- The system shall provide internal system clock ticks
- The system shall provide absolute delay functionality
- The system shall provide relative delay functionality
- The system shall provide dynamic creation of tasks
- The system shall provide dynamic termination of tasks
- The system shall provide a mechanism for creation of semaphores
- The system shall provide a mechanism for locking/unlocking the semaphores
- The system shall provide a mechanism for deletion of semaphores

3.3 Nonfunctional Requirements

3.3.1 Usability

- The user will be able to create an absolute delay through a single system call
- The user will be able to create a relative delay through a single system call
- The user will be able to create a task dynamically through a single system call
- The user will be able to terminate a task dynamically through a single system call
- The user will be able to create a binary semaphore through a single system call
- The user will be able to lock a binary semaphore through a single system call
- The user will be able to release a binary semaphore through a single system call
- The user will be able to delete a binary semaphore through a single system call

3.3.2 Reliability

- The system will not enter a deadlocked state under any circumstances

3.3.3 Supportability

- The OS will support AVR32UC3A line of processors
- The OS will support up to 12 user created tasks

3.3.4 Implementation

- The project will be implemented in the C programming language
- Atmel Studio 7 will be used as a development tool
- The scheduler will use Fixed Priority Preemptive Scheduling algorithm
- The resolution of system clock ticks will be in the millisecond range
- The Immediate Priority Ceiling Protocol will be used for task synchronization

4 Time and Activity Plan

Following is the proposed time and activity plan for the project.

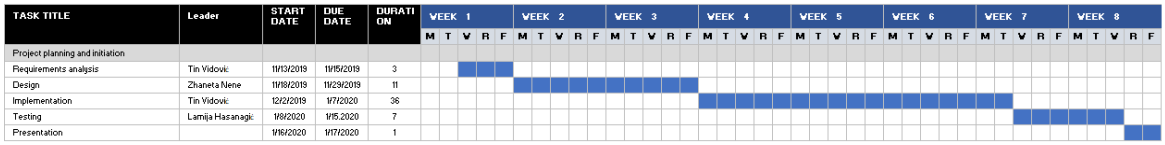

Figure 1: Time and Activity Plan

References

[1] Jansson, Marcus. “ROSA, PDF.” 7 Nov. 2010.

[2] Mäki-Turja, Jukka. “ROSA Operating System, PPT.”