Project Oxide

Real Time Operating System

Design and Approach

Background

While Internet of Things (iOT) technologies have advanced greatly in recent years, the embedded technologies that many of these iOT devices still run on operating systems built using the C programming language.

Our project, OSxide, aims to modernize real time operating systems by using Rust, a modern systems programming language that focuses on speed and security. By using Rust, we hope to mitigate many of the security vulnerabilities that have historically come with systems build using C.

Scope and Objectives

While the motivation for a real time operating system (RTOS) in Rust comes from the need for secure iOT platforms, the scope of this project is narrower, with the main objective being the construction of the most basic RTOS in Rust.

Objectives:

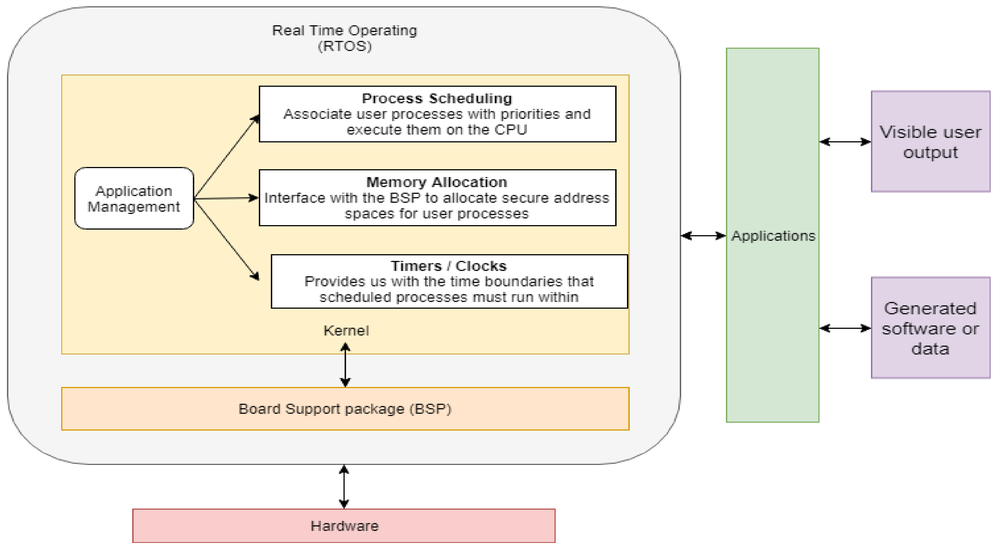
- Explore the capabilities of Rust in an embedded device setting. Particularly, in the context of the NRF51 development kit we used for testing.

- Develop a small and basic real time operating system in Rust. This requires the development of a very simple scheduler.

- Consider the advantages using Rust gives over a similar system built in C. Is it faster? Is it safer?

Initially our approach was to design from the ground up creating everything from the board support package (BSP) to the kernel. We quickly determined that this was avoidable as nordic systems has premade packages available for use; however, these packages are all written in C and we wanted to target Rust. After some digging we came across Jorge Aparicio, an embedded Rust developer, who has been developing tools for embedded Rust development for the last few years. Utilizing his tools we were able to generate some base Rust code which allowed access to our device's peripherals and chip functionality.

Our approach models a typical RTOS setup. The BSP interfaces directly with the hardware and chip registers. The Kernal utilizes this hardware interface to allocate memory, schedule process, and ensure time boundaries. The applications then communicate with these layers to perform tasks on the hardware.

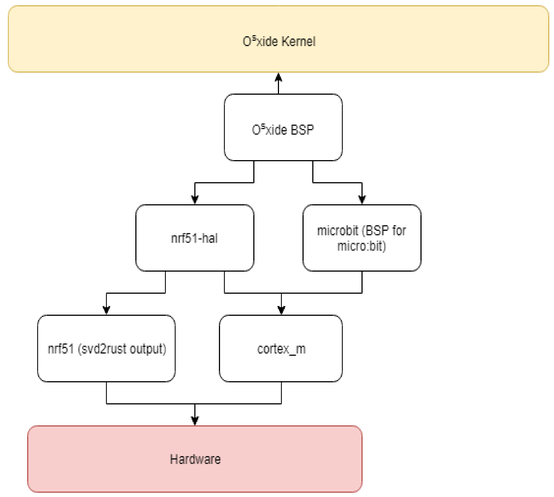


Team Members

Kernel Design

Our Kernal design is simple with process scheduling consisting of a linear running of what applications are placed in memory. Memory Allocation is handled by allowing applications to read and write only to a small section of predetermined memory space.

The current model of our project is as follows. More will be added to fill in the Oxside BSP and Kernel sections.



Future Work

At this time we are still developing the BSP for our nrf51 chip. Once we are done with developing the BSP we will be able to add simple Kernel capabilities like process scheduling.

Our current roadblock is getting program interrupts working. Interrupts will allow us to stop and start different processes on our operating system and give our process scheduling more depth than a linear process schedule allows.