# Bern RTOS - A real-time operating system for microcontrollers written in Rust

Degree programme: Master of Science in Engineering | Specialisation: Industrial Technologies
Thesis advisor: Prof. Roger Weber
Expert: Daniel Kühni (Inetronic AG)

Microcontroller performance and connectivity is ever increasing and so is the complexity of embedded systems. A larger code base naturally leads to more software defects that can cause a fault or can be exploited. As opposed to C/C++ the Rust programming language is memory-safe, eliminating most common run-time bugs by design. Based on Rusts safety principles a fail-safe real-time operating system (RTOS) has been developed in this thesis.

## Embedded Rust ecosystem

Rust is a programming language with memory-safety and thread-safety guaranteed at compile time. By mitigating software defects from runtime to compile time Rust might challenge to role of C/C++ on microcontroller based embedded systems. Especially as the complexity of microcontroller applications increases. The embedded Rust ecosystems already provides community driven hardware abstraction layers (HAL) and a real-time framework (RTIC). However, missing is an RTOS which is fail-safe, can be easily integrated in a project and provides an intuitive application programming interface (API).

## A new RTOS from the ground up

In previous projects the requirements for the kernel were defined based on the analysis of existing RTOS written in Rust or C. The core components of the kernel i.e. scheduler and synchronization primitives were then implemented and tested on a microcontroller. This thesis introduced the concept of threads and processes to microcontrollers. Processes run in isolation from each other preventing software faults from spreading across the entire system. Process memory and stack boundaries are enforced in hardware. A violation of these boundaries results in immediate termination of the thread. The kernel handles message passing between threads in different processes. In addition, usability was increased significantly by adding message queues, memory allocation, system logs and event tracing.

## Espresso machine example application

In the second part of the thesis Bern RTOS was put to the test on a real-world use case: an espresso machine. The goal was to implement an integration test of many kernel components and to evaluate the API usability.
First, the machine was upgraded with custom made electronics including a touch screen and additional sensors to measure water pressure, flow and temperatures at multiple points. The application emulates a typical RTOS use case where many loosely coupled tasks are executed on the same microcontroller. There is a real-time critical aspect with the temperature control and actuators which must run in a deterministic manner. A high background load is caused from updating the graphical user interface with live sensor measurements. A dynamic and unknown system load was created by a TCP/IP connection which interacts with a computer storing machine state and measurements in a database.

## Outlook towards an open source RTOS

The espresso machine application demonstrates that Bern RTOS provides the fundamental features to develop a real-time application. It also shows that Rust can be used effectively for microcontroller applications. The text size of the complete example application is with 286 kB (7 kB Bern RTOS, 219 kB GUI) comparable to an implementation in C/C++ . Yet, guaranteeing memory-safety and thread-safety at compile time. The run-time costs for Bern RTOS are low. In case of the espresso machine the kernel uses 2% of the total CPU execution time.
The Bern RTOS code base is open-source (MIT license) and further development continues as a community project. **bern-rtos.org**

Modified espresso machine used in the example application.

Stefan Lüthi
bern-rtos@luethi.tech