toki: A Build- and Test-Platform for Prototyping and Evaluating Operating System Concepts in Real-Time Environments

Oliver Horst  
fortiss GmbH – Research Institute of the Free State of Bavaria  
Guerickestr. 25, 80805 Munich, Germany  
Email: horst@fortiss.org

Uwe Baumgarten  
Technical University of Munich  
Department of Informatics, Germany  
Email: baumgaru@in.tum.de

Abstract—Typically, even low-level operating system concepts, such as resource sharing strategies and predictability measures, are evaluated with Linux on PC hardware. This leaves a large gap to real industrial applications. Hence, the direct transfer of the results might be difficult. As a solution, we present toki, a prototyping and evaluation platform based on FreeRTOS and several open-source libraries. toki comes with a unified build- and test-environment based on Yocto and QEMU, which makes it well suited for rapid prototyping. With its architecture chosen similar to production industrial systems, toki provides the ground work to implement early prototypes of real-time systems research results, up to technology readiness level 7, with little effort.

I. INTRODUCTION

Currently, most applied real-time systems research prototypes are developed and evaluated on top of Linux on PC hardware. This leaves a large gap between real industrial applications in that field and the prototype. In case of low-level operating system (OS) concepts concerning, e.g., context switch times, resource sharing, intra-node communication, and predictability, the drawn conclusions could even be void due to the completely different nature of the industrial platform. Furthermore, we see a lack in practical examinations of which latency and how much temporal predictability, in the sense of [1], is achievable with certain configurations (e.g., software architectures and predictability measures). Hence, we see the need to ease constructing early prototypical implementations of research results on relevant hardware in relevant environments.

On the one hand, Linux seems to be a good choice here. It is available for a wide variety of hardware platforms, provides excellent third-party library support, and can fulfill real-time requirements to some extent. However, it is a complex task to configure Linux in a way that its influence on low-level performance benchmarks is negligible or at least predictable. Moreover, integrating own concepts into the Linux kernel requires detailed knowledge about the kernel sources and its concepts. Hence, even though specialized distributions, such as LITMUSRT [2], can drastically reduce the configuration effort, the integration complexity remains as main issue.

Industrial-grade real-time operating systems (RTOSs), on the other hand, provide considerably less influence on performance measurements, but at the costs of usability. They are either delivered as minimal systems, like FreeRTOS [3], which lack tooling support and provide not much more than a scheduler, or as sophisticated platforms tailored to specific industrial fields, such as AUTOSAR [4], which come along with royalty fees, tooling, and complex, configurable software stacks.

As an exception, the GENODE OS framework [5] comes with toolchains for several hardware platforms and provides a convenient, production ready RTOS. Unfortunately, GENODE’s overall orientation to safety and security severely hinders rapid prototyping. Its micro-kernel based design requires that all adaption and changes to the OS obey the strict isolation mantra, which is conceptually challenging and time consuming.

Therefore, we see the need for a minimal, yet flexible real-time system framework that provides comfort similar to commercial platforms, but comes without a complex software architecture and strict inherent design concepts. Ideally, the framework should be fully based on open-source software, provide an integrated standard C-library, a target toolchain, configuration tools, and an emulation environment for testing. A plus would be the possibility to safety certify the system.

Accordingly, we present toki, a flexible and configurable OS framework, close to production industrial systems, but without the hassle of complex software architectures.

II. THE BUILD- AND TEST-PLATFORM TOKI
toki, Japanese for “when an action occurs”, is a build- and test-environment constructed around FreeRTOS or SafeRTOS [6], respectively. toki’s main goal is to compose an easy to use rapid prototyping platform to develop and evaluate new operating-system- and intra-node-communication-concepts for cyber-physical systems on commodity microcontrollers. Therefore, toki combines the following open-source projects into the unified architecture and build-system illustrated in Fig. 1: FreeRTOS [3] (v10.0.0), newlib [7] (v3.0.0), libXil/libXilPm [8] (v2019.1), and lwIP [9] (v2.0.3). These base components
are supplemented by the ESFree Scheduling Library [10], HST User Mode Scheduler [11], and TACLe benchmarks [12].

These components were chosen under the following aspects: (i) the simplicity to modify and extend their code base, (ii) their ability to closely mimic the software stacks of comparable industrial systems, (iii) the flexibility of their design regarding rapid prototyping, and (iv) compatible software licenses.

At platform-level, the toki build-system (Fig. 1) utilizes Yocto [13] to provide a self-contained build-environment that builds all components of toki together with their dependencies (e.g., a GCC cross-compiler). At project-level, we ensured proper include prefixes for all components, by restructuring their sources and adding a CMake [14] based build-system, where needed. This reorganization is conducted by a dedicated Python script, to allow nearly automatic upstream pulls.

To allow software developers to benefit from the code insight and debugging capabilities of their favorite IDE, single-core deployments of toki can be built by CMake. Production-ready images of multi-core deployments, on the other hand, can only be built by Yocto. toki deployment configurations, in general, are specified either manually or with assistance of the toki-config tool, and guide Yocto in the build process.

Besides the changes required for the build-environment, we contribute the following new features to FreeRTOS on ARMv8: asymmetric multiprocessing boot support, a newlib syscall interface, a memguard [15] implementation, and software-tracing support via ARM’s system trace macrocell [16].

The toki build-environment is seamlessly integrated with a virtual test-environment. Hence, all images built by Yocto can either be tested on the real hardware or its emulated counterpart. The emulation is handled by a tailored QEMU [17], built by Yocto, which enables a fully virtual development cycle.

Currently, toki is solely tested and ready-to-run on the Xilinx Zynq UltraScale+ MPSoC platform [18], a contemporary ARMv8 multi-core SoC with an integrated FPGA. This platform was selected, because of (i) Xilinx’s extensive software support for it, including a bare-metal driver kit, and (ii) the included FPGA, which allows us to increase the accuracy of our performance evaluations. Nevertheless, toki was designed with portability in mind; hence, it can easily be ported to other target platforms, e.g., the STM32Cube by STMicro [19].

In the future, we plan to conduct measurements to compare the interrupt handling latency of toki with other software stacks and extend toki with FreeRTOS+POSIX [20], a communication middleware, and precision time protocol (PTP) support.

Feel free to try out the latest version of toki, by downloading it from: https://git.fortiss.org/toki

III. DEMONSTRATION

We will demonstrate toki through two showcases:

1) A video of the TAPPS project’s [21] final demonstrator, showing the on-the-fly installation of an application into the real-time critical control path of the throttle control of a production electric motorcycle, realized with toki.

2) A live-demo of toki’s configuration, build, simulation, and deployment cycle on the example of memory benchmarks deployed to distinct cores and regulated by memguard.

The first showcase focuses on the real-time capabilities and applicability to industrial use-cases, and the second on the configuration flexibility and measurement capabilities of toki.

ACKNOWLEDGMENT

We would like to thank all contributors to toki for their work, namely: Martin Jobst, Johannes Wiesböck, Dorel Coman, Ulrich Huber, Mahmoud Rushdi, Tuan Tu Tran, Firas Trimech, Raphael Wild, Andreas Ruhland, and Dhiraj Gulati.

REFERENCES

[1] B. Sun et al., “Definitions of predictability for cyber physical systems,” J. of Syst. Architecture - Embedded Syst. Des., vol. 63, pp. 48–60, 2016.
[2] J. M. Calandrino et al., “LITMUS’RT: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers,” in 27th IEEE Int. Real-Time System. Symp. (RTSS), Dec. 2006, pp. 111–126.
[3] https://www.freertos.org.
[4] AUTOSAR, “Classic platform release 4.3.1,” AUTOSAR Std., Dec. 2017.
[5] N. Feske, GENODE Foundations – Operating System Framework 19.05, GENODE Labs, 2019.
[6] https://www.highintegritysystems.com/safertos/.
[7] https://sourceware.org/newlib/.
[8] https://github.com/Xilinx/embeddedsw.
[9] https://savannah.nongnu.org/projects/lwip/.
[10] R. Kase, “Efficient Scheduling Library for FreeRTOS,” Master’s thesis, KTH Information and Communication Technology, 2016.
[11] F. E. Páez et al., “FreeRTOS user mode scheduler for mixed critical systems,” in 6th Argentine Conf. on Embedded Syst. (CASE), Aug. 2015.
[12] H. Falk et al., “TACLeBench: A Benchmark Collection to Support Worst-Case Execution Time Research,” in 16th Int. Workshop on Worst-Case Execution Time Analysis (WCET), 2016, pp. 2:1–2:10.
[13] https://www.yoctoproject.org.
[14] https://cmake.org.
[15] H. Yun et al., “MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms,” in 19th IEEE Real-Time and Embedded Technol. and Appl. Symp. (RTAS), Apr. 2013.
[16] https://www.genode.org.
[17] https://www.freertos.org.
[18] https://www.xilinx.com/products/silicon-devices/soc/zyynq-ultrascale-mpsoc.html.
[19] https://www.xilinx.com/content/S3_com/en/stm32cubecore-ecosystem.html.
[20] https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_POSIX/.
[21] http://www.tapprojekt.eu.