Improving Privilege Levels and Memory Protection for Teaching in Embedded Xinu

Spencer Christensen
spencer.christensen@marquette.edu
Marquette University
Milwaukee, WI, USA

Spencer O'Brien
spencer.obrien@marquette.edu
Marquette University
Milwaukee, WI, USA

Abstract

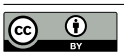
The RISC-V computer architecture has generated a lot of interest within computer science communities and has seen increased usage in recent years with companies like NVIDIA, Huawei, and Google having announced or begun to produce hardware incorporating the RISC-V architecture. Because of this popularity, Embedded Xinu, a simple, education focused operating system, was ported to RISC-V. While Embedded Xinu successfully ran on RISC-V, due to limitations in RISC-V boot loader technology, the kernel was required to break privilege level specifications and configure the hardware directly from machine mode. To remedy this, the kernel and boot sequence needed modifications to run exclusively in supervisor mode, as intended in the RISC-V specifications. These modifications allow for proper hands on instruction regarding privilege levels and user-kernel space distinctions, improving the breadth and quality of operating systems education using Embedded Xinu.

ACM Reference Format:
Spencer Christensen and Spencer O'Brien. 2026. Improving Privilege Levels and Memory Protection for Teaching in Embedded Xinu. In *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.2 (SIGCSE TS 2026), February 18–21, 2026, St. Louis, MO, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3770761.3777159

1 Introduction

1.1 Motivation for Educational OS Development

Many colleges and universities offer operating systems courses where students develop minimalistic, lightweight operating systems to familiarize themselves with computer architecture. This exposure allows students to have a lower-level mental framework of how computers work and facilitates a set of problem-solving skills that students will likely use outside of their coursework. One of the factors that influences the applicability of the coursework is the relevance of the architecture the operating system is built in. For example, there have been many advancements in computer architecture, including the facilitation of parallelism, memory protection, caching, trap handling, etc. [8], and if the underlying architecture does not support these features, then students cannot gain hands-on experience interacting with these concepts. As architectures change, operating systems that students interact with in these courses need to be ported to more modern architectures and incorporate these new design concepts in order to remain educationally relevant.



SIGCSE TS 2026, St. Louis, MO, USA

ACM ISBN 979-8-4007-2255-4/2026/02
https://doi.org/10.1145/3770761.3777159

1.2 Background & Related Work

1.2.1 Xinu History. Embedded Xinu is based on the Xinu operating system developed by Douglas Comer [6] in the 1980s. In the late 2000s Xinu was ported to the MIPS32 architecture under the new name "Embedded Xinu", as part of an experimental lab for teaching operating systems on baremetal hardware [3]. Embedded Xinu is designed to be lightweight and architecture agnostic, allowing it to be ported to numerous architectures and hardwares. In 2013 Embedded Xinu was ported to ARM64 on the newly popularized Raspberry Pi platform [2], this platform was chosen for its low cost and flexible application. Embedded Xinu has been used as a research platform for educational improvements in a wide variety of undergraduate and graduate coursework [1, 4, 5, 9–11].

1.2.2 Embedded Xinu on RISC-V. The most recent port to RISC-V from ARM [7] was completed a few years ago and has successfully implemented both the basic feature set required for coursework, and more advanced, implementation specific features such as paging and virtualization. While this initial port was a great leap forward for Embedded Xinu, some important libraries such as "pthread" and the network stack are still under development. Additionally, some quirks of the Nezha hardware platform were left unresolved. The RISC-V port greatly simplified assignments in the associated operating systems course, allowing students to focus less on the implementation minutia, and more on the important concepts at hand. Paging, for example, was improved as the RISC-V hierarchical page tables have consistent size, allowing for page-table walks to be assigned to students for the first time.

1.3 The Problem

During the initial port of Xinu onto RISC-V, [7] the developers implemented a majority of the features/projects from the ARM version of Xinu onto the RISC-V port. However, with the simplified privilege scheme offered by RISC-V, there was considerable motivation to implement projects involving page tables and memory protection. These projects could be made on Embedded Xinu's previous architectures, but implementing these concepts required a level of complexity that would be indigestible to students. With the goal of introducing these concepts to students, the challenge of designing and implementing a functional, compact privilege level scheme became apparent.

2 Approach & Uniqueness

2.1 Getting Started

Since much of the Embedded Xinu kernel relied on the specific hardware state that the boot loader created before its crash, we were determined to refactor into the proper privilege mode with as few

changes as possible, therefore limiting the damage that restructured hardware permissions could cause. This refactor required two main efforts; firstly we needed to rebuild the boot loader such that it would complete its task and safely hand off to the kernel; secondly we needed to modify the kernel so that it would properly handle its new privilege level and permissions, ideally without disrupting more components than those in the initialization process.

2.2 The Boot Loader

The boot loader used on the Embedded Xinu system had two major flaws. The first flaw, as previously mentioned, was that the boot loader would crash somewhere in the U-boot step, and let the kernel have access to machine mode. The second flaw was that the boot loader was bundled with a 10 gigabyte Linux distribution, which acted as a fallback for the Xinu kernel. In order to address these issues we had to find an updated version of U-boot for RISC-V, and build it from source. While reading into the U-boot documentation, we discovered that much of the hardware configuration done by our network booting script could in fact be hard-coded into the boot loader configuration. This along with disabling scanning for the unused USB devices allowed us to not only correct the boot sequence, but shave off nearly 20 seconds from every run.

2.3 Initialization & Interrupt Refactoring

Once the boot loader was rebuilt, we ran into issues with both the kernel initialization sequence and with interrupt handling. Since the kernel now started in S-mode (as opposed to M-mode), all the initialization around the M-mode control and status registers (CSRs) were disallowed by the hardware. To resolve this we had to root out any assembly calls that modify M-mode privileged registers, and ensure that all the configuration changes originally made by the kernel are made by the rebuilt boot loader. With the kernel now able to boot past the initialization phase, we had to sort out exception trapping. While the existing exception handling infrastructure was viable, the privilege change prevented them from being trapped by the kernel handler. This was resolved with CSR reconfiguration and a deeper look at the RISC-V interrupt and exception structure. Finally we implemented timer control through the platform level interrupt controller (PLIC). The PLIC allows for hardware outside the CPU to raise interrupts that can be trapped and handled like standard kernel level interrupts. In this case we configured the external timer module to raise an interrupt every second, thus allowing the kernel to track time at that interval.

3 Results & Contributions

After changing the code, it is now possible to implement page tables and memory protection without illegal memory calls entering the trap handler in M-mode and then delegating the exception down to S-mode. This simplifies the trap handling sequence so students only need to focus on the relationship between S-mode and U-mode. Now students can implement the trap handler without considering more privilege levels than necessary, and they will be able to successfully test their implementation of page tables.

3.1 Maintainability of Xinu / Hardware

With the kernel boot sequence now in-line with RISC-V design philosophy, Embedded Xinu should be resilient to boot loader updates thus making it stable and maintainable. Since the previous iteration of the kernel only worked due to the soft crash of the boot sequence landing in M-mode, it will be incompatible with any updated boot loader firmware and must be deprecated.

A major improvement made during this research is the discovery that environment information can be configured in each board's boot loader, enabling fixed MAC addresses and hardware configuration. This allows for a simpler, more robust network boot system using only DHCP instead of relying on interruption over the UART busses, thus reducing need for delicate scripting infrastructure. This puts Embedded Xinu in a state where any university wishing to adopt bare-metal RISC-V education can do so with reduced complexity and headache, gaining a stable and maintainable system.

3.2 Future Work

With the interrupt being redesigned, the PLIC is more amenable to accepting a variety of external interrupts. This enables the possibility of introducing students to asynchronous serial drivers. If this project is developed on the RISC-V version of Xinu, students could then use concepts already introduced in class (e.g. buffers, semaphores, messaging, etc.) to implement an asynchronous serial driver. With an asynchronous driver it should also be possible to port the previous version of the Xinu shell onto RISC-V as the code is architecture agnostic. This would complete the port of Xinu onto RISC-V and make the RISC-V port the most modern and comprehensive version of Embedded Xinu. With these major improvements a study can be conducted on future operating systems courses to determine the educational outcomes of the changes.

References

[1] Priya Bansal, Rade Latinovich, et al. 2017. XinuPi3: Teaching Multicore Concepts Using Embedded Xinu. In *CSERC '17* (Helsinki, Finland). ACM, 20–25. doi:10.1145/3162087.3162091

[2] Eric Biggers, Farzeen Harunani, et al. 2013. XinuPi: Porting a lightweight educational operating system to the Raspberry Pi. In *2013 Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education.*

[3] Dennis Brylow. 2008. An experimental laboratory environment for teaching embedded operating systems. *SIGCSE Bull.* 40, 1 (March 2008), 192–196. doi:10.1145/1352322.1352201

[4] Dennis Brylow and Bina Ramamurthy. 2009. Nexos: a next generation embedded systems laboratory. *SIGBED Rev.* 6, 1, Article 7 (Jan. 2009), 10 pages. doi:10.1145/1534480.1534487

[5] Dennis Brylow and Kyle Thurow. 2011. Hands-on networking labs with embedded routers. In *SIGCSE '11.* ACM, 399–404. doi:10.1145/1953163.1953283

[6] D. Comer. 2025. *Operating System Design: The Xinu Approach.* CRC Press. https://books.google.com/books?id=9URUEQAAQBAJ

[7] Alexander Gebhard, Jack Forden, et al. 2024. Using Embedded Xinu to Teach Operating Systems on Baremetal RISC-V. In *SIGCSE '24* (Portland, OR, USA). ACM, 380–386. doi:10.1145/3626252.3630959

[8] John L. Hennessy and David A. Patterson. 2019. A new golden age for computer architecture. *Commun. ACM* 62, 2 (Jan. 2019), 48–60. doi:10.1145/3282307

[9] Benjamin Levandowski, Debbie Perouli, and Dennis Brylow. 2019. Using Embedded Xinu and the Raspberry Pi 3 to Teach Parallel Computing in Assembly Programming. In *IPDPSW '19.* 334–341. doi:10.1109/IPDPSW.2019.00063

[10] Patrick J. McGee, Rade Latinovich, and Dennis Brylow. 2020. Using Embedded Xinu and the Raspberry Pi 3 to Teach Operating Systems . In *IPDPSW '20.* IEEE Computer Society, Los Alamitos, CA, USA, 307–315. doi:10.1109/IPDPSW50202.2020.00063

[11] Matthew H. Netkow and Dennis Brylow. 2010. Xest: an automated framework for regression testing of embedded software. In *WESE '10* (Scottsdale, Arizona). ACM, Article 7, 8 pages. doi:10.1145/1930277.1930284