Work-in-Progress: Unishyper, A Reliable Rust-based Unikernel for Embedded Scenarios

Keyang Hu, Lei Wang, Ce Mo, Bo Jiang
{hky1999,wanglei,moce4917,jiangbo}@buaa.edu.cn
School of Computer Science and Engineering, Beihang University
Beijing, P.R. China

ABSTRACT

Unishyper is a reliable Rust-based unikernel with good performance for embedded scenarios. It relies on Rust *Features* to enable customization at fine granularity. To achieve high reliability, Unishyper makes full use of the Rust language features to reduce memory safety bugs, ensure safe resource management, and achieve fault handling and recovery. Finally, Unishyper achieves high performance through safe multi-threading model as well as the single-privilege-level and single-address-space design.

CCS CONCEPTS

• **Computer systems organization** → **Embedded software**.

KEYWORDS

Unikernel, embedded system, modular system, Rust, reliability

ACM Reference Format:
Keyang Hu, Lei Wang, Ce Mo, Bo Jiang. 2023. Work-in-Progress: Unishyper, A Reliable Rust-based Unikernel for Embedded Scenarios. In *International Conference on Embedded Software (EMSOFT '23), September 17–22, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3607890.3608459

1 INTRODUCTION

The unikernels [4] compile the entire software stack of system libraries, language runtime, and applications into a single bootable VM image running on a standard hypervisor. In this way, unikernels can achieve much smaller footprint and attack surface, which are promising to support embedded scenarios. However, existing unikernels likes OSv[1], Unikraft [2] and RustyHermit[3] are mainly designed for cloud computing scenarios rather than embedded scenarios. Some unikernels have relative weak application development ecosystem [4], which makes it difficult to be widely used in practice. We propose Unishyper[1], a reliable modular unikernel written in Rust designed for embedded scenarios with high reliability and performance. Unishyper proposes a hierarchy of memory safety, resource management safety, and fault handing safety mechanisms to provide high reliability. It can run on both bare-metal embedded devices and hypervisors. Unishyper is highly reliable. It makes full use of the Rust language features to eliminate memory safety bugs and to enable safe resource management. Furthermore, it proposes fault handling and recovery mechanism based on the unwind mechanism of Rust. While retaining the small image size and performance advantage of unikernel based on the single-privilege-level and single-address-space design, it further realizes a lightweight and safe multi-threading model to improve performance. Unishyper explores how to utilize hardware supported mechanism, Intel-MPK, to achieve memory isolation under single address space. Unishyper has good support for Rust ecosystem. It supports the Rust standard library (Rust std), which means that Rust crates can run on Unishyper without modification. Finally, Unishyper can further reduce the image size and its complexity through the *Unilib* design, which enables Unishyper to optionally offload some of its work to the monitor VM (MVM) of the hypervisor.

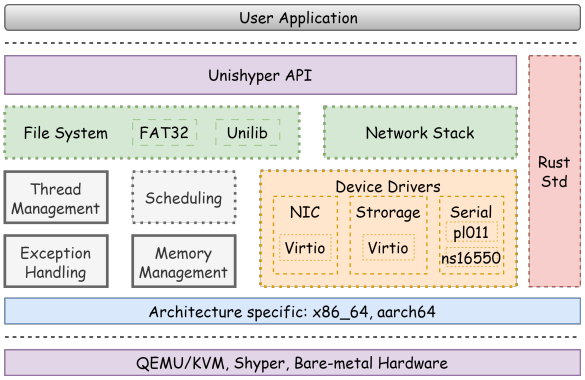

Figure 1: Architecture overview of Unishyper. The dashed rectangles are optional modules.

2 THE DESIGN OF UNISHYPER

2.1 System Architecture

The Figure.1 describes the key components of Unishyper architecture. The main design goals of Unishyper are as follows:

- **Multi-core and multi-thread support**: Unishyper provides a multi-core and multi-thread programming framework.
- **Single address space (SAS) and Single privilege level (SPL)**: All threads share one page table and run under the same privilege level, rely on the Rust ownership and hardware mechanism for memory management and isolation.
- **Modular structure design**: All functional modules of Unishyper are organized by a set of well-designed common calling interfaces. The selection and organization of modules are carried out at compile time.

[1]This work was partially supported by the National Natural Science Foundation of China (No. 62077002).

EMSOFT '23, September 17–22, 2023, Hamburg, Germany

ACM ISBN 979-8-4007-0291-4/23/09.
https://doi.org/10.1145/3607890.3608459

- **Static link**: Unishyper eliminates useless code during compilation, which can reduce generated binary image size.
- **Multi-platform support**: Currently Unishyper supports ARMv8, x86_64 and RISC-V architecture. Unishyper's general interface design abstracts a special platform layer, supporting a variety of virtualization platforms (Shyper/KVM) or running directly on embedded bare-metal devices.

2.2 Modularized Design for Customization

2.2.1 Configurable features. Feature-based customized compilation takes advantage of Rust conditional compilation. Features provided by Unishyper includes target architecture, scheduling algorithm, file system, network stack etc. The flexible conditional compilation of Rust brings Unishyper compile-time code instrumentation configuration, eases the way of source code tailoring and allows programmer to be free to select the required modules.

2.2.2 Application development with std and no_std support. Unishyper provides both `no_std` support and `std` support. Feature "alloc" with `no_std` support means a more compact image size. And `std` support means that a rich Rust community ecosystem can be used, but its compilation requires a modified Rust toolchain. Our team fork our own Rust toolchain and create a target specification for Unishyper, which provides relevant configuration of architecture description and corresponding link script files. We also need to provide our own specific implementation for std library.

2.2.3 Unilib interface. One of Unishyper's outstanding contributions is to propose *"Unilib"*, a set of interfaces based on the idea of offload that spans unikernel and the running hypervisor platform. Based on Unilib, unikernels can conveniently use the rich general-purpose library interfaces provided by general purpose OSs running in other virtualized partitions, and choose to offload part of the protocol stack, further simplifying the unikernel structure. We have implemented the Unilib-fs interface on Unishyper with the cooperation of the Shyper hypervisor[5] and its monitor VM (a Linux VM). Unilib-fs chooses to let the user program on Linux with C library (libc) take over the file operations initiated by Unishyper. The whole fs operation includes a hypercall through the Unilib interface, a trap into the hypervisor, an inter-processor interrupt(IPI) forwarding the request to the target core of the running Linux, and the actual fs operation handled by Linux's user daemon. The operation result is notified to the Unikernel through a similar hypercall-IPI process to realize a complete file operation.

The introduction of Unilib allows unikernel to choose to abandon the complex and huge device drivers and libraries code, which greatly simplifies the code size of Unikernel.

2.3 Design for High Reliability

2.3.1 Fault record and recovery. In order to meet the requirements of embedded scenarios for system reliability and easy traceability, based on the Rust `unwind`, Unishyper implements a fault record and recovery mechanism, including backtracing error call chain, fault processing and resource release, etc. When `panic` or exception happens, the unwind module use the exception handling data to backtrace stack frames and land at landing-pad finally, thus restoring normal operation of the program. Unishyper implements a `thread_wrapper` which can restart the thread after unwinding, meeting the reliability requirement of embedded scenarios.

2.3.2 Memory isolation based on Intel MPK. On x86_64 architecture, Unishyper uses Memory Protection Keys (MPK) from Intel to achieve a lightweight thread memory isolation. According to memory management model of Unishyper, thread can only access the memory space controlled by the `MappedRegion` structures of their own, and each `MappedRegion` structure hold a one-to-one mapping of virtual pages and physical page frames. With MPK, each mapping can be assigned one protection key(record on certain bits on page table entry), from 0 to 15, and the access to certain page is controlled by register, PKRU, which can be modified with the instruction `wrpkru`. Unishyper groups threads into different subset, each has a specific protection key. Threads cannot access memory from other subsets, otherwise it'll trigger a page fault of protection-key violation. Our isolation design is general, and we also consider porting to ARMv8 with Memory Tagging support.

Table 1: Micro-benchmark on TX2 platform (CPU cycles)

OS	thread_yield	spawn	alloc	dealloc
Unishyper	1173	4492	204	857
FreeRTOS	1274	3343	1025	915
Linux-4.9.140	4975	55780	5099	9131

3 EVALUATION

Micro-benchmark evaluations on Nvidia TX2 hardware(evaluated by CPU cycles read from ARM PMU counter) are shown in Table 1. Unishyper has significant performance advantage over Linux, comparable to FreeRTOS. With LTO and opt-level='s' in compilation parameters, Unishyper can achieve a memory footprint around 100KB. Network test on QEMU/KVM shows that Unishyper has obvious advantages in network latency compared with OSv and Linux, especially in latency fluctuation performance, which is very suitable for the requirements of high latency and high reliability in embedded environments. We also simulated two case studies by running HTTP server and ROS client on top of Unishyper, to verifiy its application prospect in the field of embedded control.

4 CONCLUSION

A well designed unikernel for embedded scenario must achieve several goals: customizability, high reliability and good performance. This paper proposes Unishyper, a reliable Rust-based unikernel for embedded scenarios. In general, Unishyper achieves small memory footprint, good performance and high reliability.

REFERENCES

[1] Avi Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov. 2014. OSv—optimizing the operating system for virtual machines. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*. 61–72.

[2] Simon Kuenzer, Vlad-Andrei Bădoiu, Hugo Lefeuvre, Sharan Santhanam, Alexander Jung, Gaulthier Gain, Cyril Soldani, Costin Lupu, Ştefan Teodorescu, Costi Răducanu, et al. 2021. Unikraft: fast, specialized unikernels the easy way. In *Proceedings of the Sixteenth European Conference on Computer Systems*. 376–394.

[3] Stefan Lankes, Jonathan Klimt, Jens Breitbart, and Simon Pickartz. 2020. RustyHermit: A Scalable, Rust-Based Virtual Execution Environment. In *High Performance Computing: ISC High Performance 2020 International Workshops, Frankfurt, Germany, June 21–25, 2020, Revised Selected Papers 35*. Springer, 331–342.

[4] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. 2013. Unikernels: Library Operating Systems for the Cloud. *SIGARCH Comput. Archit. News* 41, 1 (mar 2013), 461–472. https://doi.org/10.1145/2490301.2451167

[5] Yicong Shen, Lei Wang, Yuanzhi Liang, Siran Li, and Bo Jiang. 2022. Shyper: An embedded hypervisor applying hierarchical resource isolation strategies for mixed-criticality systems. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1287–1292.