Towards Highly Specialized, POSIX-compliant Software Stacks with Unikraft: Work-in-Progress

1st Sharan Santhanam, 2nd Simon Kuenzer,
3rd Hugo Lefeuvre, 4th Felipe Huici
NEC Laboratories Europe GmbH
Heidelberg, Germany
firstname.lastname@neclab.eu

5th Alexander Jung
Lancaster University
NEC Laboratories Europe GmbH
Heidelberg, Germany
a.jung@lancs.ac.uk

6th Santiago Pagani
Robert Bosch GmbH
NEC Laboratories Europe GmbH
Heidelberg, Germany
santiago.pagani@de.bosch.com

7th George-Cristian Muraru
University Politehnica of Bucharest
Bucharest, Romania
george.muraru@upb.ro

8th Stefano Stabellini
Xilinx, Inc
San Jose, USA
stefanos@xilinx.com

9th Justin He
ARM Technology (China) Co., Ltd
Shanghai, China
Justin.He@arm.com

10th Jonathan Beri
Mountain View, USA
jmberi@gmail.com

***Abstract*—Increasingly, embedded devices are being equipped with ARM processors. Because of ease-of-use and widespread support for drivers and applications, Linux is often used as the OS of choice, even though it consumes a significant amount of the device's limited resources and its large attack surface presents opportunities for exploits. In this paper we propose Unikraft, a fully librarized operating system and build tool which allows for generating specialized OSes and software stacks targeting specific applications, while removing unneeded functionality. As a proof of concept, we port Unikraft to the Raspberry Pi 3 B+ and to a Xilinx Ultra96-V2. On these boards, Unikraft is able to boot in 88-158 milliseconds, consume only hundreds of KBs of memory when running real-world application such as NGINX, all the while providing visible reductions in power consumption compared to Linux distributions. Unikraft is an open source project and can be found at unikraft.org.**

***Index Terms*—operating systems, embedded systems**

I. Introduction

Traditionally, embedded devices have been centered around specialized, embedded processors and the embedded operating systems running on them (e.g., FreeRTOS [1] and Zephyr [2]). This model has been, and still is, extremely effective in ensuring efficient resource consumption, especially power, but forces developers to port applications to such OSes, since, for the most part, they are not POSIX-compliant.

Increasingly, many embedded devices are being designed around general-purpose processors, especially ARM-based ones. This is a radical shift in the way we think about embedded devices: many so-called "embedded" devices (e.g., in the IoT, edge, gateways and automotive domains) use Linux as default because it is easy to install, is POSIX-compliant and comes with a great array of applications and programming languages, not to mention a friendly, well-known development environment. The great downside is that this monolithic kernel is resource hungry: it is not unusual for a significant amount of a device's resources to be consumed by Linux itself, leaving less for the actual application. Further, Linux is to a large extent a monolithic kernel, making it often hard or time consuming to customize (e.g., completely removing the scheduler if it is not needed, adding a new memory allocator, or trying to trim it down to reduce boot times). Finally, Linux's significant code base (in the order of millions of lines of code) results in a large attack surface and exploits, and makes it expensive to certify in domains where safety is critical.

In order to break the dichotomy between (1) difficult-to-use but resource efficient embedded OSes [1], [2] and (2) power-hungry but user and application friendly general-purpose OSes such as Linux, we introduce a novel micro-library operating system called Unikraft which allows for automatically building highly specialized images for embedded devices. Unlike other approaches, Unikraft bridges the gap between resource efficiency and ease of porting with a micro-library approach, allowing for bottom-up specialization and code elimination while retaining POSIX compatibility. In addition, the extremely lean images it produces are ideal candidates for cheaper certification.

To show feasibility, we add ARM64 support to Unikraft, and further add support for a number of boards, including the Raspberry Pi 3 B+ and a Xilinx Ultra96-V2. On such boards, Unikraft is able to boot in as little as 88-158 milliseconds, consume only hundreds of KBs of memory when running real-world application such as NGINX, while still providing visible reductions in power consumption compared to Linux distributions. Unikraft is an open source project under the BSD license and can be found at [3].

II. Building a True Library OS

Unikraft is a *micro-library* operating system whose components are modular, meaning that they can be easily removed and replaced at compile time. Unlike other library OSes (e.g., OSv [4], MirageOS [5]), Unikraft is **fully** librarized (see fig. 1): OS primitives such as the scheduler, memory allocator and even boot code are libraries. These can be removed or replaced with equivalents via a `Kconfig` menu as long as they comply with a set of well-defined APIs. To enable

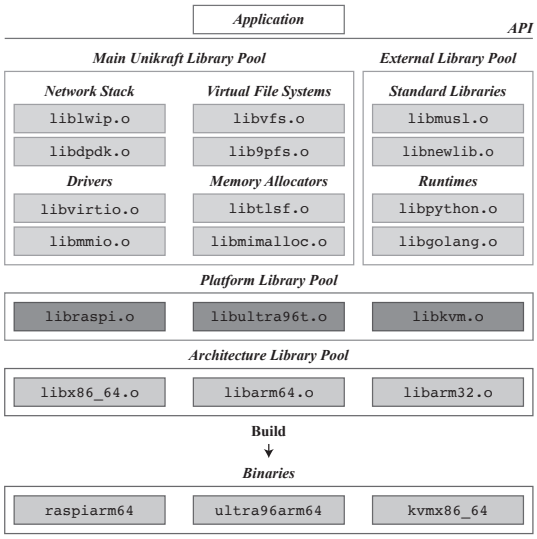

Fig. 1: Unikraft's architecture. All components are micro-libraries. Users select an architecture, a platform, the target application and Unikraft creates the image.

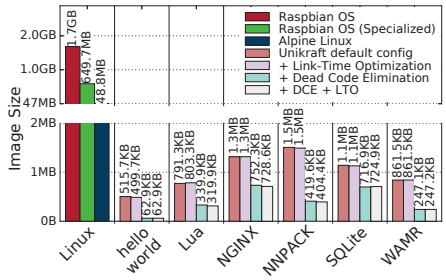

Fig. 2: Unikraft image sizes for a number of applications compared to base image sizes for different Linux distributions.

quick boot times, Unikraft can, for example, allow for the use of a simple but quick memory allocator during boot and initialization, whilst still using a different allocator for the application. Alternatively, a user could entirely remove the scheduler if not needed, and run tasks to completion in an event-driven architecture. This and many other scenarios are easy to implement in Unikraft because of its modular design.

Unikraft enables users to easily build extremely specialized, custom OSes without having to develop any actual code as it provides a POSIX-like interface which allows for running standard, off-the-shelf applications such as databases (e.g., SQLite), web servers (e.g., NGINX), key-value stores (e.g., Redis), machine learning frameworks (e.g., PyTorch and TensorFlow), and runtime language environments (e.g., Web Assembly, Python/Micropython, Lua and Ruby). In addition, Unikraft supports a number of compile-time languages including C/C++, Go, Java and Rust, with the potential for allowing different libraries to be written in different languages and combined together into a single, specialized image.

Security: Unikraft's level of customization means that it can produce very lean images with a minimal Trusted Computing Base (TCB), reducing the number of potential attack vectors. In addition, thanks to its configurability, it is easy to remove exploitable and unneeded services commonly found in general purpose OSes such as the shell or even the network stack. As the code is statically linked, Unikraft can make aggressive use of optimization techniques such as Dead Code Elimination (DCE) and Link Time Optimization (LTO) to drastically reduce the size of the resulting images (see fig. 2). And, unlike other, existing projects [6], Unikraft is actively working to add security mechanisms commonly found in standard OSes such as stack protection, page protection, ASLR and security hardening via fuzzing techniques.

Programming Language Support: The build system recognizes extensions for C, C++, assembly, JAVA, Go and Rust and it is trivial to support new extensions. Interpreters for other languages are external micro-libraries; as of this writing, Unikraft supports Python3, MicroPython, Lua, Web Assembly (WAMR) [18], JavaScript (V8 and Duktape) and Ruby.

Porting Effort: Unikraft supports a range of Cortex-A ARM64 CPUs. In our experience, porting to a new device, assuming complete unfamiliarity with it, takes roughly about 2-3 weeks, and includes mapping of memory regions in the page table, allocation of the heap and stack regions and device initialization (e.g., the Generic Interrupt Controller (GIC)); all other code is common across devices and is already provided by Unikraft.

Dependency Management: One of the challenges in designing a true library OS is to match the functionality required by an application to that provided by Unikraft's micro-libraries. To address this, we rely on the `Kconfig` language to establish these dependencies: an application or library describes the necessary dependent library using either the *select*, the *imply* or the *depends on* keywords. Once the dependencies are established, Unikraft's `make` based build system ensures that application dependencies are satisfied. Further, to automate this process, Unikraft comes with a set of static and run-time tools to identify dependencies, and to match them to the functionality provided by the micro-libraries[1].

III. Evaluation

As a proof of concept, we port Unikraft to the Raspberry Pi 3 B+ and the Xilinx Ultra96-V2. We compare Unikraft with off-the-shelf Alpine Linux and Raspbian OS as well as a specialized, stripped down Raspbian (removing non-essential build tools, system packages and docs).

We begin our evaluation by measuring boot times (fig. 4), where Unikraft can boot the Raspberry Pi (excluding the approximately 3 seconds it takes for the GPU to boot) in 88 ms and the Xilinx board in 158 ms, which is much lower than Linux (though clearly these numbers would go up when initializing devices needed by applications). Memory consumption is also two orders of magnitude lower in Unikraft compared to the Linux distributions (fig. 5). Finally, we evaluate power consumption versus Linux when idle and when running a CPU-intensive calculation of π [7] (fig. 3); as shown, Unikraft

[1]These tools are in the process of being upstreamed as of this writing.

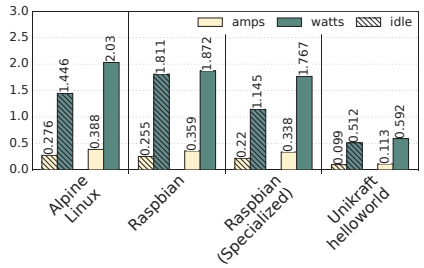

Fig. 3: RPI Unikraft power use vs. Alpine and Raspbian when idle and when calculating π.

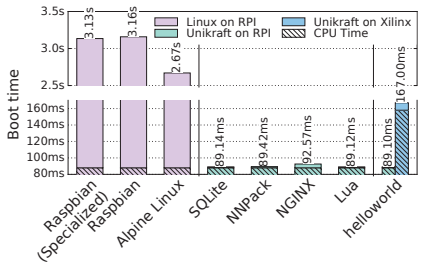

Fig. 4: Boot times of different applications on Unikraft vs. those same applications running on Linux Alpine and Raspbian.

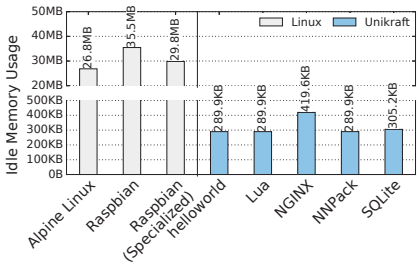

Fig. 5: Idle memory consumption for different applications on Unikraft vs. Alpine and Raspbian.

can provide important power reductions in both scenarios in comparison to Linux (running only on a single core and with networking modules disabled, for fair comparison).

In sum, we obtain low image sizes, boot times, memory consumption, and power usage comparable to that of bespoke embedded OSes, all of the while (1) retaining the advantages of Linux/a POSIX interface in terms of porting and associated costs and (2) providing a wide range of easy customization possibilities.

IV. Related Work

Recent years have seen a large number of library OSes or unikernel projects appear. Most are application-specific or programming language-specific projects: RuntimeJS (JavaScript), IncludeOS (C++), HaLVM (Haskell), LING (Erlang), MirageOS [5] (OCaml), MiniPython (MicroPython), ClickOS [8] (NFV), and MiniCache [9] (content cache). In contrast, Unikraft supports a range of applications and runtimes efficiently.

Rump [10] is a unikernel project that merges the NetBSD kernel with application code in a single memory address space. It provides good compatibility for standard applications, but its reliance on a monolithic kernel means there is not much room for specialization. Unlike Unikraft, HermiTux [11] and OSv [4] are difficult to customize, and their reliance on binary compatibility comes with performance costs.

Finally, Zephyr [2] is a real-time operating system. Like Unikraft, it provides a single memory address space and targets efficiency; unlike Unikraft, it is not POSIX-compliant, consists of a monolithic kernel and primarily targets micro-controllers.

V. Conclusion and Future Work

We have introduced Unikraft, a micro-library operating system and build tool to generate custom software stacks aimed at running on ARM64-based embedded devices. Unikraft's highly specialized images result in important reductions in terms of image size, boot times, memory use and power consumption. As future work, we are looking into adding support for additional boards; the Linaro 96Boards [12] initiative might be a good path to supporting many devices with relatively small effort. In terms of drivers, we are looking into providing a compatibility layer such that unmodified Linux drivers can be used within a Unikraft image.

VI. Acknowledgements

This paper has received funding from the EU's Horizon 2020 research and innovation programme under grant agreement numbers 825377 (UNICORE) and 871793 (ACCORDION).

References

[1] freertos.org, "FreeRTOS," https://www.freertos.org/, 2020.
[2] Zephyr Project, "Zephyr," https://www.zephyrproject.org/, 2020.
[3] unikraft.org, "Unikraft - Extreme Specialization for Security and Performance," http://unikraft.org.
[4] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har'El, D. Marti, and V. Zolotarov, "OSv–Optimizing the Operating System for Virtual Machines," in *Proceedings of the 2014 USENIX Annual Technical Conference*, ser. USENIX ATC'14. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 61–72. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/kivity
[5] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'13)*. ACM, 2013. [Online]. Available: https://doi.org/10.1145%2F2451116.2451167
[6] NCC Group, "Assessing Unikernel Security," https://www.nccgroup.com/globalassets/our-research/us/whitepapers/2019/ncc_group-assessing_unikernel_security.pdf.
[7] CodeCodex, "Calculate digits of pi: C implementation," http://www.codecodex.com/wiki/Calculate_digits_of_pi#C, 2020.
[8] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the Art of Network Function Virtualization," in *11th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI '14. Seattle, WA: USENIX Association, 2014, pp. 459–473. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/martins
[9] S. Kuenzer, A. Ivanov, F. Manco, J. Mendes, Y. Volchkov, F. Schmidt, K. Yãsukata, M. Honda, and F. Huici, "Unikernels Everywhere: The Case for Elastic CDNs," in *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '17. New York, NY, USA: ACM, 2017, pp. 15–29. [Online]. Available: http://doi.acm.org/10.1145/3050748.3050757
[10] "Rump Kernels," http://rumpkernel.org/.
[11] P. Olivier, D. Chiba, S. Lankes, C. Min, and B. Ravindran, "A binary-compatible unikernel," in *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE 2019. New York, NY, USA: ACM, 2019, pp. 59–73. [Online]. Available: http://doi.acm.org/10.1145/3313808.3313817
[12] Linaro, "96boards," https://www.96boards.org/, 2020.