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

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, while still 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
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 runtime 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 \(\pi\) [7] (fig. 3); as shown, Unikraft

\(^1\)These tools are in the process of being upstreamed as of this writing.
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


