Unikraft: Fast, Specialized Unikernels the Easy Way

Caleb FONYUY-ASHERI SUUYNYUY

Plan

I. Introduction

II. Problem

III. Proposed Solution

IV. Properties and Architecture

V. Application Porting

VI. Evaluation and Results

VII. Conclusion

Introduction

Introduction

A new library that

1. Provides modular OS to compile unikernel with only relevant components
2. Provides a performance-oriented and composable API for easy development.

Problem

Problem with current OS

1. Application-kernel switches
2. Multiple address spaces
3. Unused functionalities
4. Unrequired networking layers
5. Memory access
6. Memory allocation

Problem

1. Strong dependencies between system libraries
2. Most applications do not need all these libraries
3. Kernel size
4. Resource usage
5. Performance
6. POSIX compliance

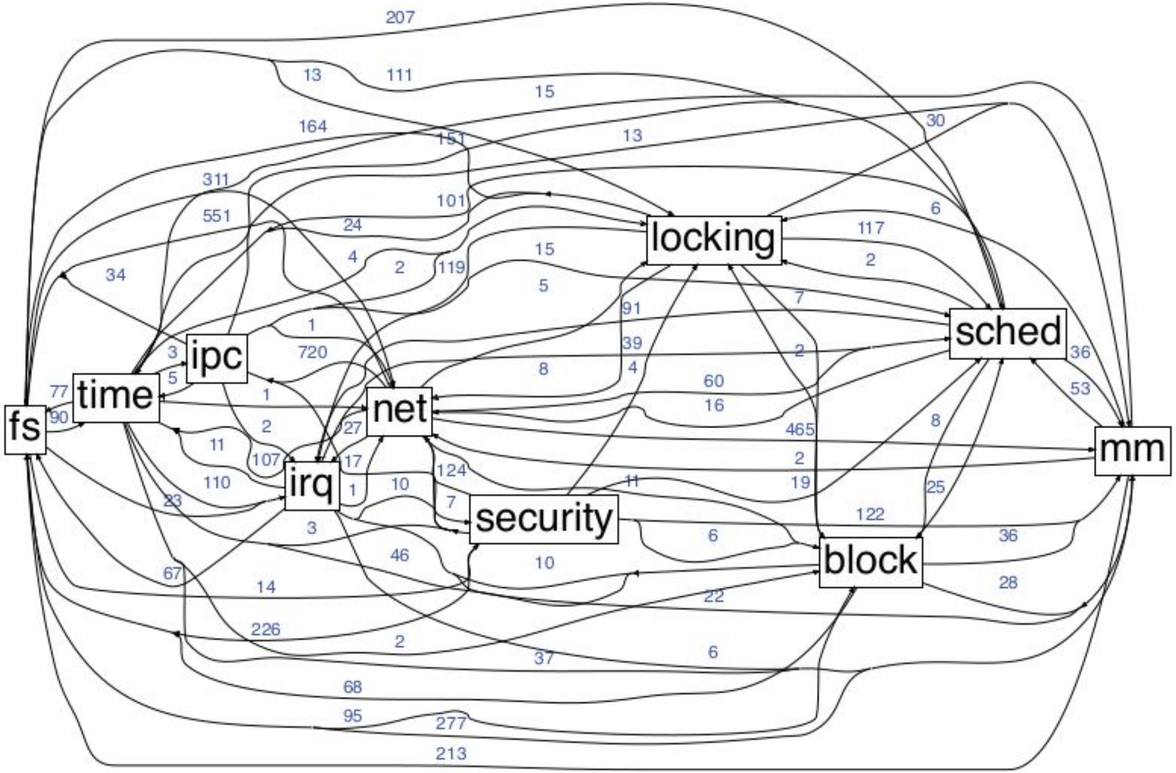


Figure 1. Linux kernel components have strong inter-dependencies, making it difficult to remove or replace them.

Proposed Solution

Proposed Solution

1. A fully modular kernel which is fully and easily customizable
2. Performance-minded, well-defined APIs from kernel, easily selected and composed for application needs

Dependency Graph on Unikraft

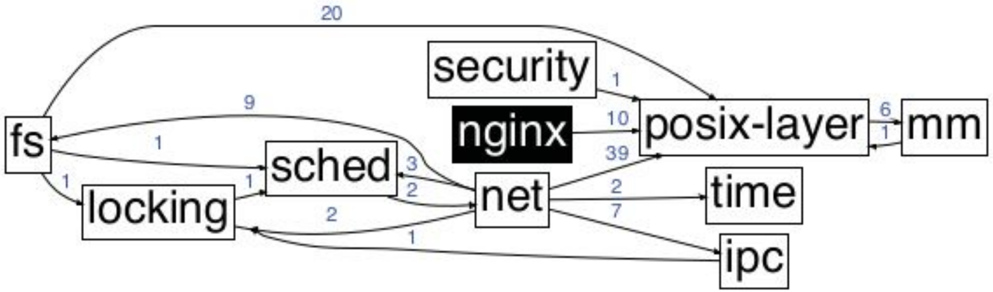


Figure 2. Nginx Unikraft dependency graph

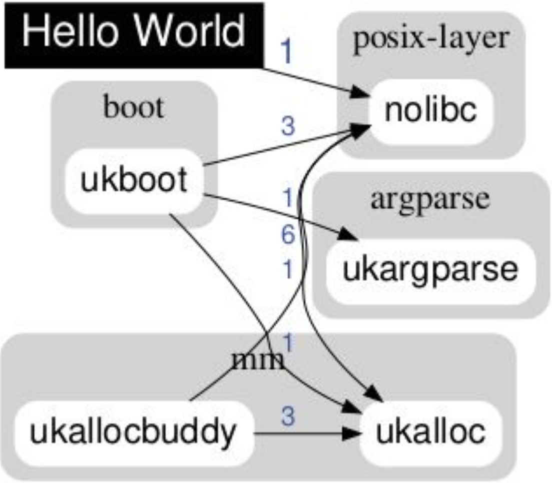


Figure 3. Helloworld Unikraft dependency graph

Properties and Architecture

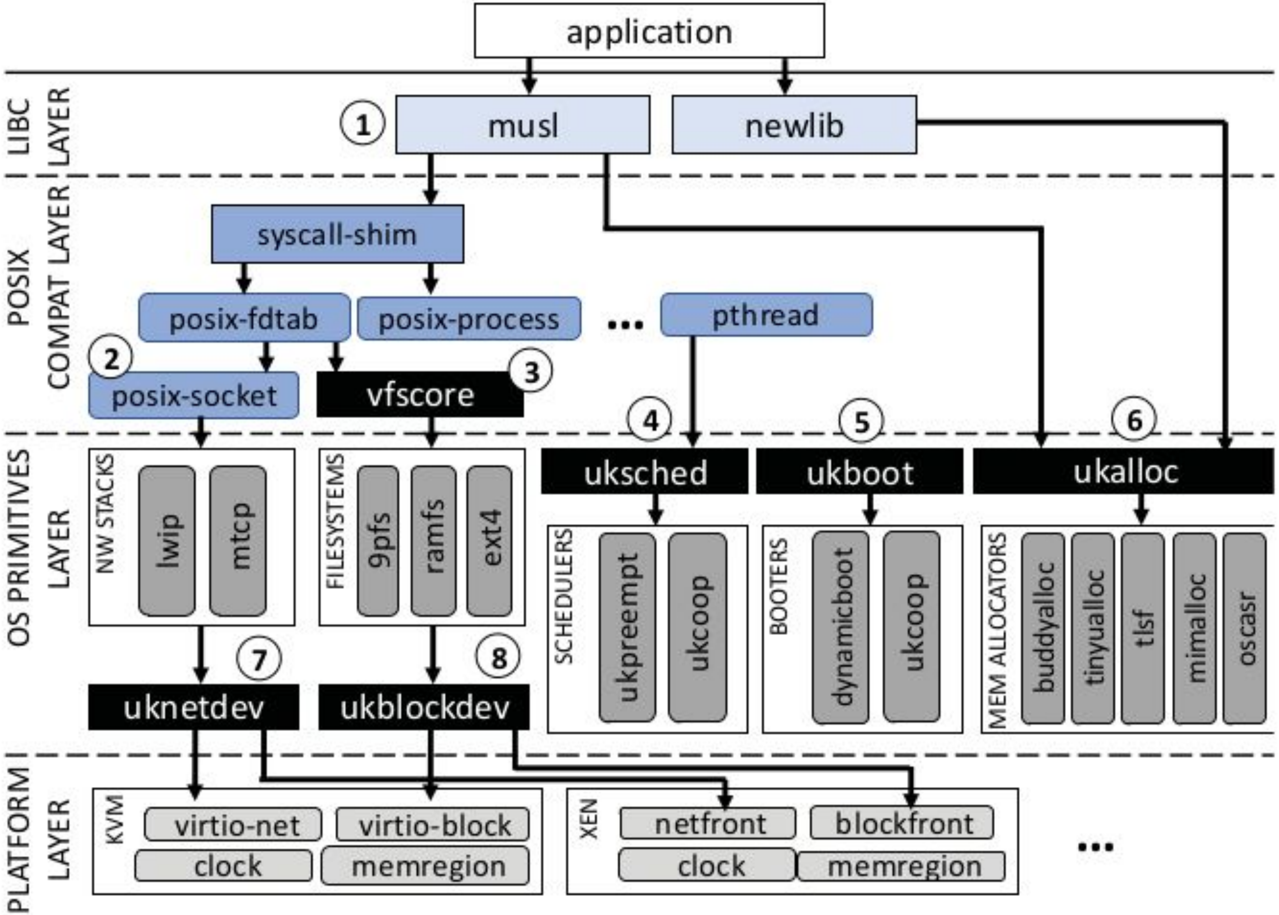
Properties

1. Single address space
2. Fully modular system
3. Single protection level
4. Static linking
5. POSIX support
6. Platform abstraction

Architecture

Defined APIs

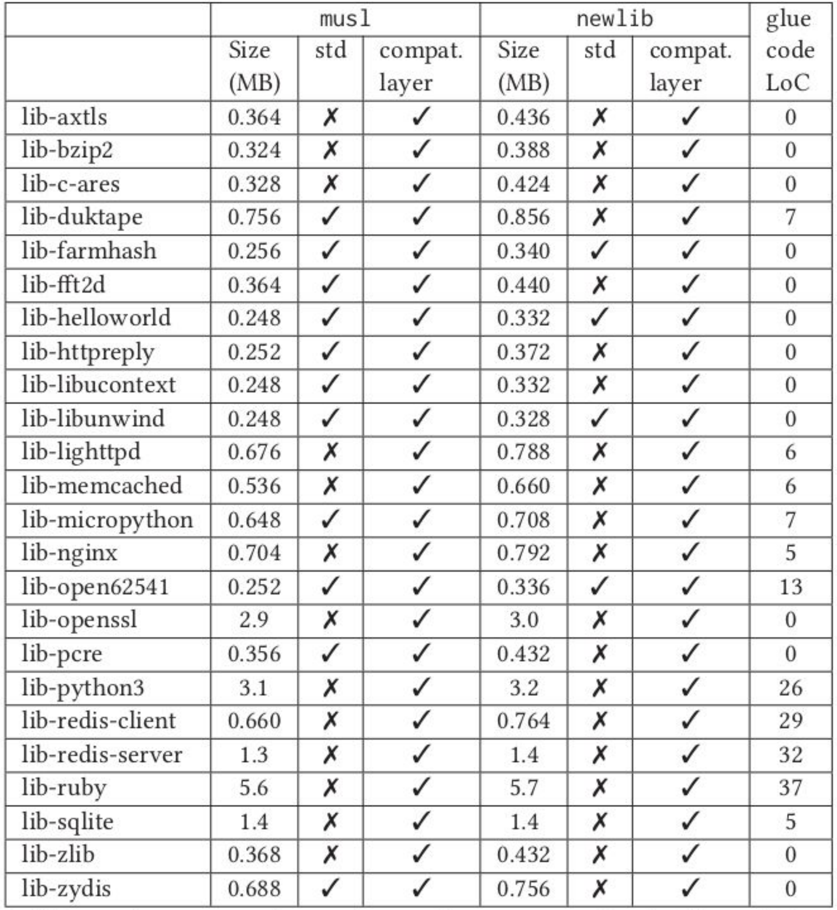
1. uknetdev
2. ukalloc
3. uksched and uklock



Application Porting

Application Porting

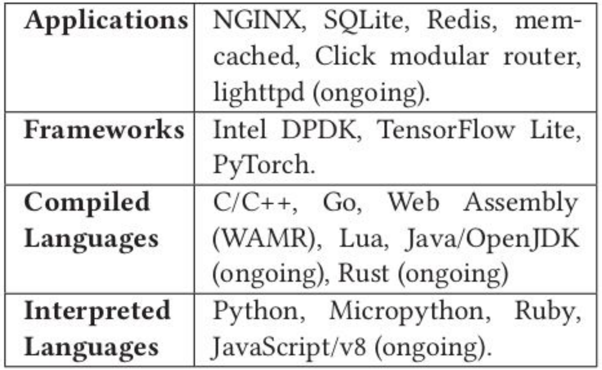
1. Application Compatibility
2. Manual Porting



	musl			newlib			glue
	Size (MB)	std	compat. layer	Size (MB)	std	compat. layer	code LoC
lib-axtls	0.364	✗	✓	0.436	✗	✓	0
lib-bzip2	0.324	✗	✓	0.388	✗	✓	0
lib-c-ares	0.328	✗	✓	0.424	✗	✓	0
lib-duktape	0.756	✓	✓	0.856	✗	✓	7
lib-farmhash	0.256	✓	✓	0.340	✓	✓	0
lib-fft2d	0.364	✓	✓	0.440	✗	✓	0
lib-helloworld	0.248	✓	✓	0.332	✓	✓	0
lib-httpreply	0.252	✓	✓	0.372	✗	✓	0
lib-libucontext	0.248	✓	✓	0.332	✗	✓	0
lib-libunwind	0.248	✓	✓	0.328	✓	✓	0
lib-lighttpd	0.676	✗	✓	0.788	✗	✓	6
lib-memcached	0.536	✗	✓	0.660	✗	✓	6
lib-micropython	0.648	✓	✓	0.708	✗	✓	7
lib-nginx	0.704	✗	✓	0.792	✗	✓	5
lib-open62541	0.252	✓	✓	0.336	✓	✓	13
lib-openssl	2.9	✗	✓	3.0	✗	✓	0
lib-pcre	0.356	✓	✓	0.432	✗	✓	0
lib-python3	3.1	✗	✓	3.2	✗	✓	26
lib-redis-client	0.660	✗	✓	0.764	✗	✓	29
lib-redis-server	1.3	✗	✓	1.4	✗	✓	32
lib-ruby	5.6	✗	✓	5.7	✗	✓	37
lib-sqlite	1.4	✗	✓	1.4	✗	✓	5
lib-zlib	0.368	✗	✓	0.432	✗	✓	0
lib-zydis	0.688	✓	✓	0.756	✗	✓	0

Table 2. Automated porting using externally-built archives linked against Unikraft using musl and newlib.

Application Porting



Applications	NGINX, SQLite, Redis, memcached, Click modular router, lighttpd (ongoing).
Frameworks	Intel DPDK, TensorFlow Lite, PyTorch.
Compiled Languages	C/C++, Go, Web Assembly (WAMR), Lua, Java/OpenJDK (ongoing), Rust (ongoing)
Interpreted Languages	Python, Micropython, Ruby, JavaScript/v8 (ongoing).

Table 3. Applications, frameworks and languages currently supported by Unikraft.

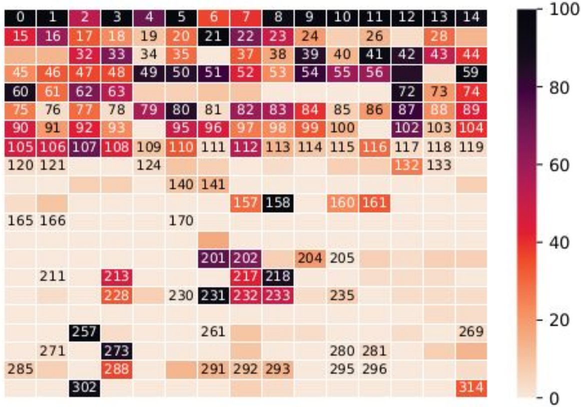


Figure 5. Syscalls required by 30 server apps vs syscalls supported by Unikraft.

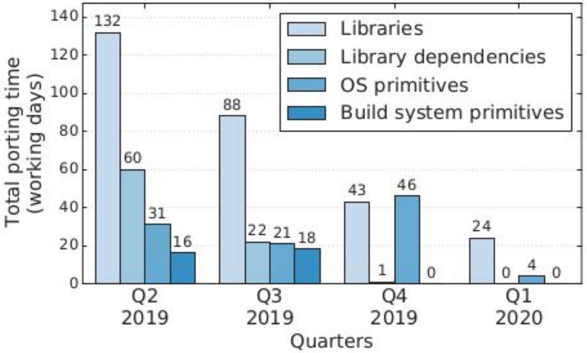


Figure 6. Devel survey of total effort to port a library, including dependencies, missing OS and build system primitives.

Application Porting

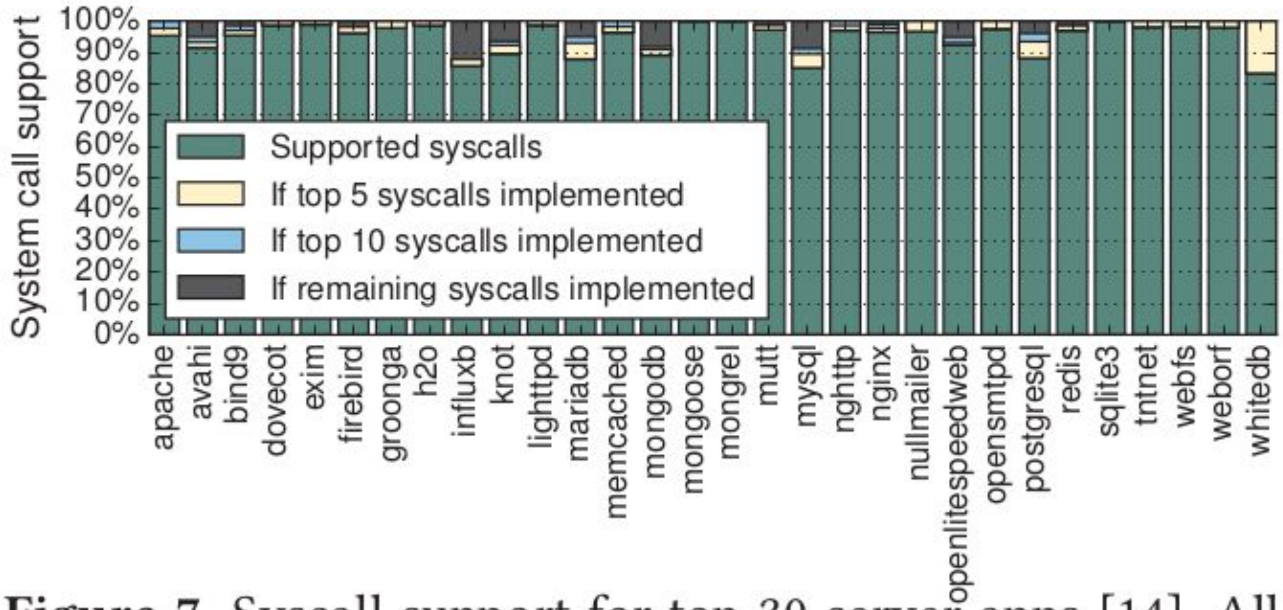


Figure 7. Syscall support for top 30 server apps [14]. All apps are close to being supported, and several already work even if some syscalls are stubbed (SQLite, nginx).

Evaluation and Results

Evaluation Environment

- Shuttle SH370R6 computer with an Intel i7 9700K 3.6 GHz (4.9 Ghz with Turbo Boost, 8 cores) and 32GB of RAM. For the DPDK experiment we use two of these connected via a direct cable and a pair of Intel X520-T2 cards with the 82599EB chipset.
- Disabled Hyper-Threading and isolated 4 CPU cores for the host using kernel boot parameters (isolcpus=4-7 noht)

Results: VM Image Sizes

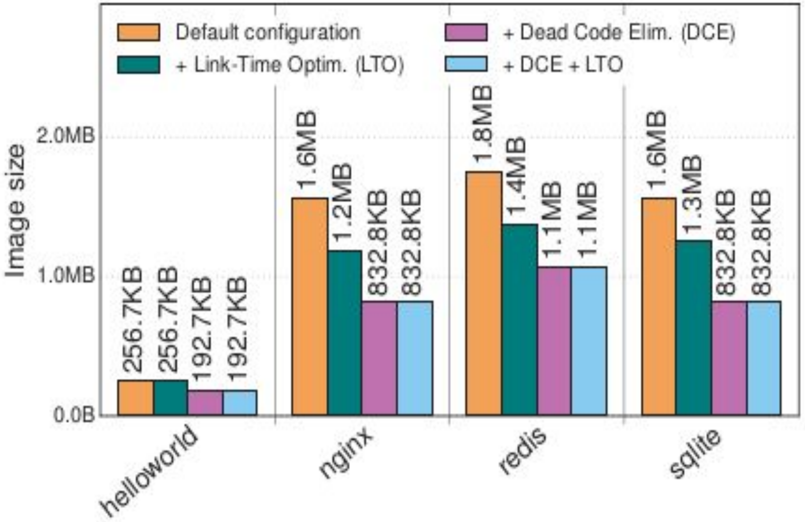


Figure 8. Image sizes of Unikraft applications with and without LTO and DCE.

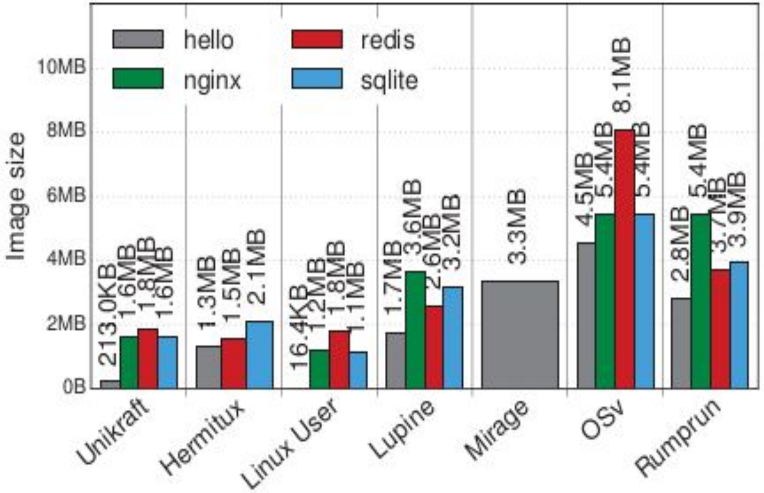


Figure 9. Image sizes for Unikraft and other OSes, stripped, w/o LTO and DCE.

Results: VM Boot times

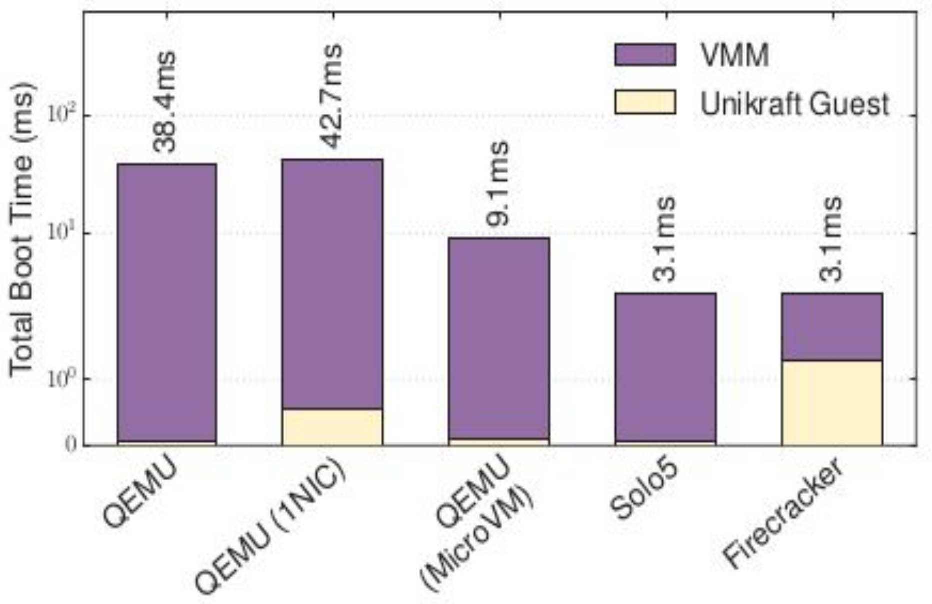


Figure 10. Boot time for Unikraft images with different virtual machine monitors.

Results: Memory requirements

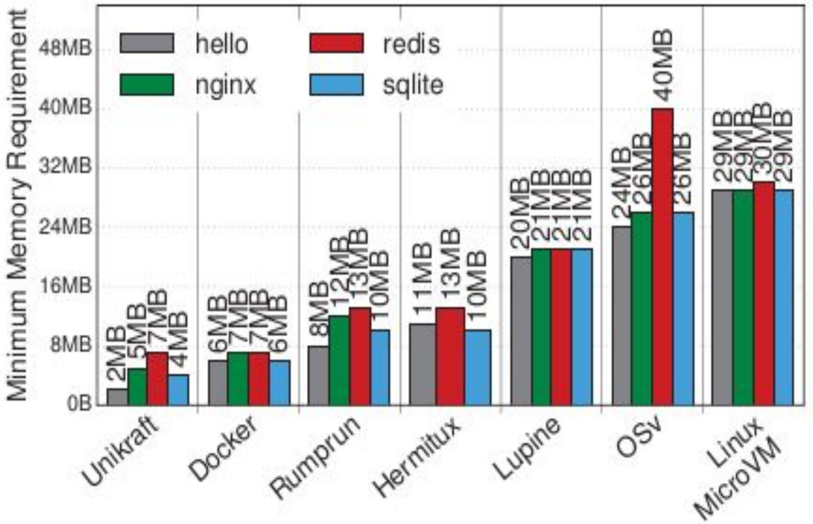


Figure 11. Minimum memory needed to run different applications using different OSes, including Unikraft.

Results: Filesystem performance

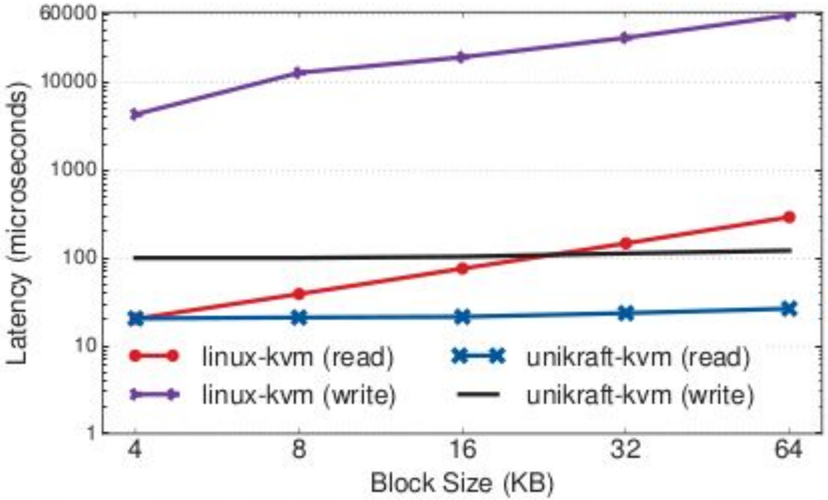


Figure 20. 9pfs latency for read and write operations, compared to Linux.

Results: Application throughput

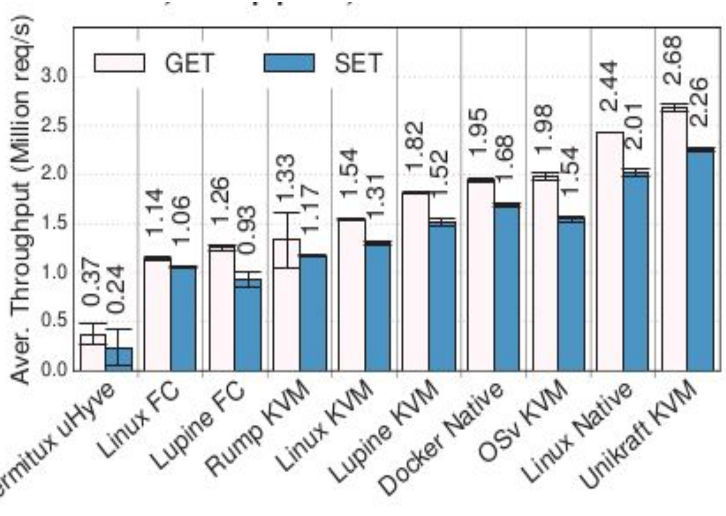


Figure 12. Redis perf (30 conns, 100k reqs, pipelining 16) with QEMU/KVM and Firecracker (FC).

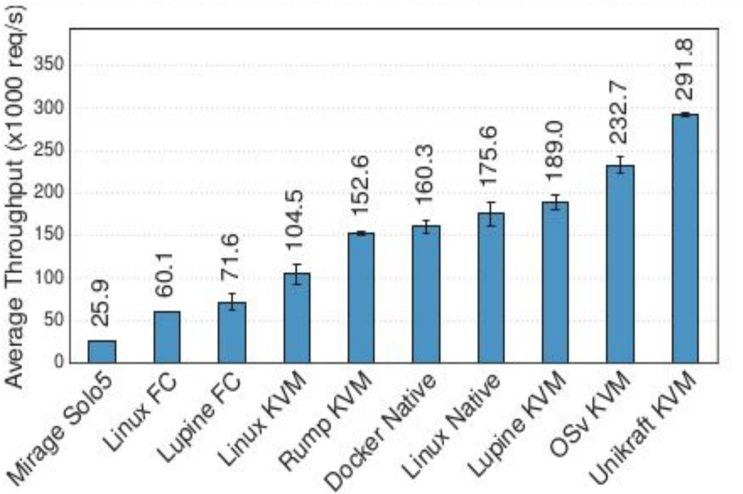


Figure 13. NGINX (and Mirage HTTP-reply) performance with wrk (1 minute, 14 threads, 30 conns, static 612B page).

Results: Performance of Automatically Ported Apps

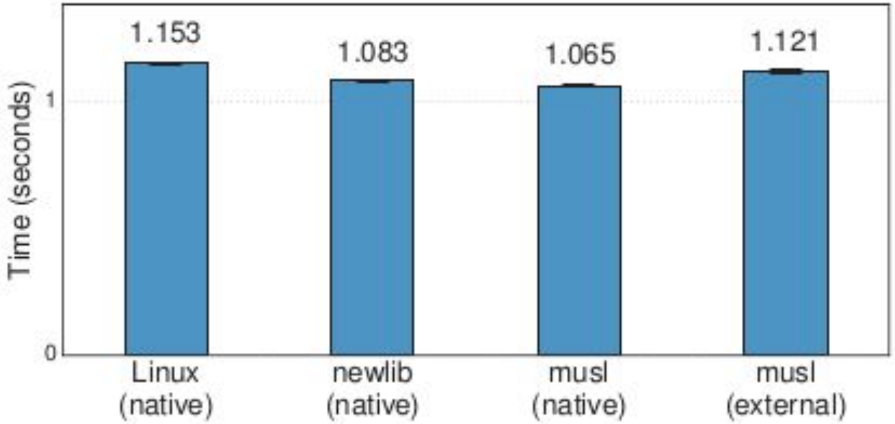


Figure 17. Time for 60k SQLite insertions for native Linux, newlib and musl on Unikraft and SQLite ported automatically to Unikraft (musl external).

Results: Performance of Memory Allocators

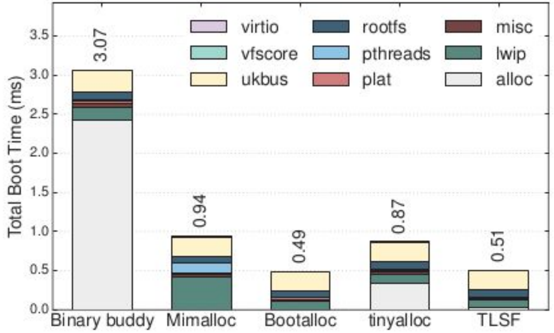


Figure 14. Unikraft Boot time for Nginx with different allocators.

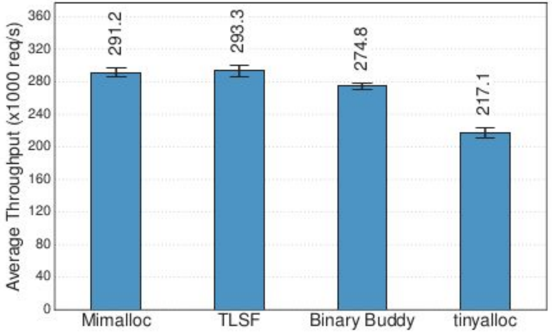


Figure 15. nginx throughput with different allocators.

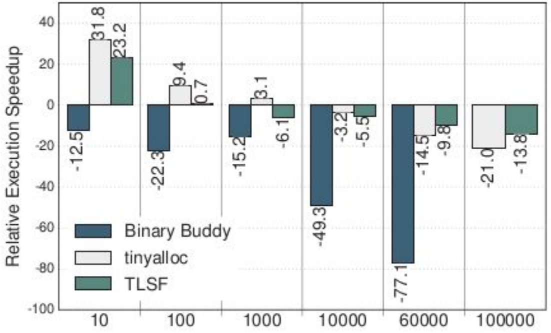


Figure 16. Execution speedup in SQLite Unikraft, relative to mimalloc [42].

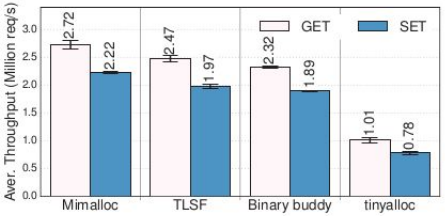


Figure 18. Redis throughput on Unikraft for different allocators (redis-benchmark, 30 conns, 100k requests, pipelining level of 16.)

Results: Performance of Memory Allocators

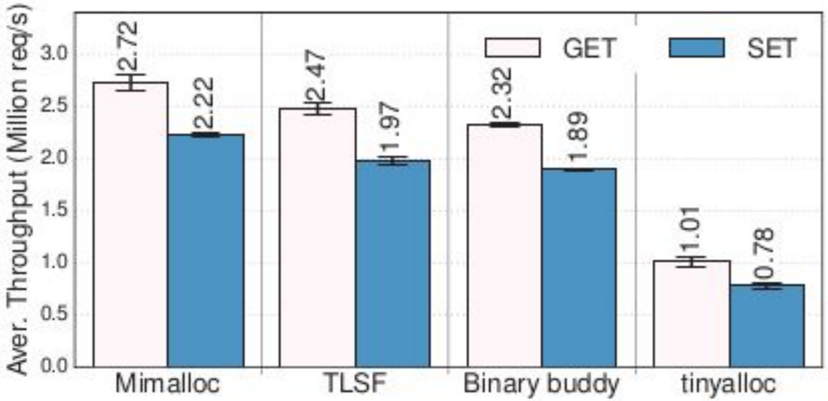


Figure 18. Redis throughput on Unikraft for different allocators (`redis-benchmark`, 30 conns, 100k requests, pipelining level of 16.)

Results: Specializing Applications

- Specialized Boot Code
- Networking Performance
- VFS and Filesystem Specialization
- Specializing a key-value store

Conclusion