Unikernels Made Easy

Simon Kuenzer <simon.kuenzer@neclab.eu>
Senior Researcher, NEC Laboratories Europe GmbH

FOSDEM’19

This work has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreements no. 675806 (“5G CITY”) and 761592 (“5G ESSENCE”). This work reflects only the author’s views and the European Commission is not responsible for any use that may be made of the information it contains.
NEC brings together and integrates technology and expertise to create the ICT-enabled society of tomorrow. We collaborate closely with partners and customers around the world, orchestrating each project to ensure all its parts are fine-tuned to local needs. Every day, our innovative solutions for society contribute to greater safety, security, efficiency and equality, and enable people to live brighter lives.
VMs vs. Containers

- VMs have been around for a long time
  - They allow consolidation, isolation, migration, ...

- Then containers came and many people LOVED them. Why?

Containers are much easier to create and deploy. I just write this Dockerfile and I’m done.

Containers are much faster to bring up than VMs.

Did you hear about Unikernels? VMs have their advantages, most importantly strong isolation.
Unikernels as VMs

**Traditional VMs**
- App A
  - Libs A
- App B
  - Libs B
- Kernel
- Hypervisor
- Hardware

**Unikernels**
- App A
  - Libs A
- App B
  - Libs B
- Kernel
- Hypervisor
- Hardware

- **Unikernels are purpose-built**
  - Thin kernel layer, *only what application needs*
  - Single monolithic binary *that contains OS and application*

- **No isolation within Unikernel, done with hypervisor**
  - One application → Flat and single address space

- **Further advantages from specialization**
Unikernel Gains

- Fast instantiation, destruction and migration time
  - 10s of milliseconds

- Low memory footprint
  - Few MB of RAM

- High density
  - 10k guests on a single server node

- High Performance
  - 10-40Gbit/s throughput with a single guest CPU

- Reduced attack surface
  - Less components exist in Unikernel
  - Strong isolation by hypervisor

LightVM [Manco SOSP 2017], Elastic CDNs [Kuenzer VEE 2017], Superfluid Cloud [Manco HotCloud 2015], ClickOS [Martins NSDI 2014]
In Numbers: Instantiation Times

Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8
MiniCache Unikernel: Purpose-built static HTTP Webserver

Legend: MiniCache (M) versus Debian (D) and Tinyx (T). L=lighttpd and N=nginx.

Experiments were conducted on Intel Xeon E5 1630v3 3.7GHz, 32GB DDR4 RAM, Mellanox ConnectX-3 40Git/s Ethernet, Xen 4.4.2, Debian Jessie with Linux 4.0.0 as Dom0 and booted from RAM.
**Application Domains**

**Minimal SW Stack**
- Fast boot, migration destroy
- Resource efficient

**Minimal SW Stack**
- Serverless, (Per-customer) vNFs, IoT, MEC, etc.

**Specialization**
- High performance
- Mission critical

**Specialization**
- NFV, MEC, etc.
- Small code base → Low attack surface → Cheaper verification
- Automotive, (Industrial) IoT, etc.
The Devil is in the Details

So, Unikernels:
- Give similar speed and size of containers
- But add **strong isolation** with *virtualization* and increase **security** due to *smaller code base*

The problem is *Unikernel development*: Optimized Unikernels are manually built
- Building takes several months or even longer
  - *We’ve done it before, multiple times*
- Potentially repeat the process for each target application
  - *We’ve done that too…*

That’s not an effective way of doing things!
Unikraft - A Unikernel Framework

Motivation

- Support wide range of use cases
- Simplify building and optimizing
- Simplify porting of existing applications
- Common and shared code base for Unikernel creators
- Support different hypervisors and CPU architectures

Concept: “Everything is a library”

- Decomposed OS functionality

Two components:

- Library Pool
- Build Tool
Unikraft

Overview
1) Library Pool

1. Select/create application

MyApplication

2. Select and configure libraries

- **main libs**
  - network stack
    - liblwip.o
    - libtcip.o
    - libhttp.o
  - filesystems
    - libvfs.o
    - libfat.o
    - libext3.o
  - drivers
    - libconsole.o
    - libixgbe.o
    - libvirtio.o
  - memory allocators
    - libbuddy.o
    - libheap.o
    - libmempool.o
  - runtimes
    - libcoap.o
    - libpreempt.o
    - librt.o
  - standard libs
    - musl.o
    - libnewlibc.o
    - libopenssl.o

3. Build

- **platform libs**
  - libxenplat.o
  - libkvmplat.o
  - liblinuxuplat.o
  - libxenplat.o

- **architecture libs**
  - libx86_64arch.o
  - libarm32arch.o
  - libmipsarch.o

4. Run

Unikernels
- myapp_xen_x86-64
- myapp_kvm_x86-64
- myapp_bare_arm64
- etc.
Example System

Python Unikernel for KVM on x86_64

<table>
<thead>
<tr>
<th>My Python App</th>
<th>libmicropython.o</th>
</tr>
</thead>
<tbody>
<tr>
<td>liblwip.o</td>
<td>libvfscore.o</td>
</tr>
<tr>
<td>libschedcoop.o</td>
<td>liballocbuddy.o</td>
</tr>
<tr>
<td>libkvmplat.o</td>
<td>libx86_64arch.o</td>
</tr>
</tbody>
</table>

Unikernel
2) Build Tool

KConfig based and Makefile “Magic”

Type "make menuconfig"

- Choose options in the menu that you want for your application
- Choose your target platform(s), e.g., Xen, KVM, bare-metal, Linux

Save your config and type "make"
An Baseline Example...

Xen PV x86_64 binary
Compiles to a 32.7kB image

unikraft_xen-x86_64.o (50.2kB)

Final linking

unikraft_xen-x86_64 (32.7kB)

Boots and prints messages to debug console (with min. 208kB RAM)
Unikraft 0.3 Iapetus

Upcoming Release
Supported Features

- **Target support**
  - Xen: x86_64, Arm32
  - KVM: x86_64, Arm64
  - Linux userspace: x86_64, Arm32
  - Bare-metal: x86_64 (with KVM target)

- **Core Functionality**
  - Cooperative scheduler
  - Binary buddy heap

- **Networking**
  - Low-level API for high-speed I/O
    - virtio-net
  - TCP/IP stack: Lightweight IP (lwIP)

- **Filesystems**
  - VFS

- **Libc’s**
  - nolibc (*Unikraft internal*)
  - Newlib
Roadmap

Concentrating effort on:

- Completing Arm64 support
  - (Virtual) Device drivers for Arm platforms
  - Other platforms

- More standard libraries
  - musl, libuv, zlib, openssl, libunwind, libaxtls (TLS), etc.

- Language environments
  - Javascript (v8), Python, Ruby, C++, etc.

- OCI container target support

- Filesystems
  - In-RAM and (Virtual) Disk filesystems

- Network drivers
  - Xen (netfront), Linux (tap)

- Frameworks:
  - Node.js, PyTorch, Intel DPDK, etc.
It is Open Source!

We need you!
Join Us!

Unikraft is OpenSource since Dec 2017 and under the umbrella of

Community is growing!
- Active contributors rose 91%, from 2 contributors to 23.

External contributors from
- Romania: networking, scheduling; from University Politehnica Bucharest
- Israel: bare-metal support, VGA driver
- China: Arm64 support from Arm

...but there is still a lot to do!
Get in touch with us!

Drop us a mail
minios-devel@lists.xen.org

Join our IRC channel
#unikraft on Freenode
Unikraft Resources

Wiki
- [https://wiki.xenproject.org/](https://wiki.xenproject.org/) (Search for Unikraft)

Documentation
- [http://www.unikraft.org](http://www.unikraft.org)

Sources (GIT)

Mailing list (shared with Mini-OS)
- [minios-devel@lists.xen.org](mailto:minios-devel@lists.xen.org)

IRC Channel on Freenode
- #unikraft

NEC-Team
- [http://sysml.neclab.eu](http://sysml.neclab.eu)
Orchestrating a brighter world

NEC