Unleashing the Power of Unikernels with Unikraft

Simon Kuenzer <simon.kuenzer@neclab.eu>
NEC Laboratories GmbH
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.
Unikernels

What are they? What are they good for?
“Google runs all services in containers”

Container-as-a-Service services:
- Amazon Lambda, EC2 Container Service
- Google Container Engine
- Azure Container service
- and so on...
The Container Advantage for Service Deployments

- Very fast instantiation times
  - 100s ms

- Small per-instance memory footprints
  - 10-100 MBs

- High density
  - 100-1000 instances on same host
The Unsafe Container

- Kernel API difficult to secure, lots of exploits
- An unsafe container affects all other containers on host
  - This includes DoS/exhaustion attacks (e.g., forkbombs, etc.)

![Graph showing the increase in number of syscalls over Linux release years from 2002 to 2018. The x-axis represents Linux release years, and the y-axis represents the number of syscalls. The graph shows a steady increase in syscalls over time.]
Picking a Poison

CONTAINERS

- ✔️ Lightweight
- ✗️ Iffy isolation

HYPERVERSORS

- ✔️ Strong isolation
- ✗️ Heavy weight
Picking a Poison

Can we break the myth of VMs being heavy weight?

Yes, with Unikernels!

- Iffy isolation
- Heavy weight
Unikernels are purpose-built
- Thin kernel layer (only what application needs)
- One application → Flat and single address space

Further advantages from specialization
In Numbers: Instantiation Times

Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

![Graph showing process creation times and number of running guests. The graph indicates process creation time ranges from 0.7ms to 10ms.](image-url)
In Numbers: Instantiation Times

Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Docker: 150ms-550ms
In Numbers: Instantiation Times

Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

Time [ms]

Number of running guests

Docker Boot
Debian Boot
Debian Create

Debian: 2.6-82 secs
Traditional VMs are in fact heavy-weight

Debian: 2.6-82 secs

In Numbers: Instantiation Times

Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8
In Numbers: Instantiation Times

Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

MiniOS Boot —— MiniOS Create ———

unikernel:, 163ms-1.4secs
In Numbers: Instantiation Times

Server: Intel Xeon E5-1630 v3 CPU@3.7GHz (4 cores), 128GB DDR4 RAM, Xen/Linux versions 4.8

MiniOS Boot ——— MiniOS Create ————

Unikernels similarly as fast as containers
In Numbers: Instantiation Times with LightVM

Server: 4 x AMD Opteron 6376 CPU@2.3GHz (64 cores total), 128GB DDR3 RAM, Xen 4.8/Linux 4.8

LightVM: http://sysml.neclab.eu/projects/lightvm

Guests: Noop/Unikernel and Noop Docker image / No devices
In Numbers: Instantiation Times with LightVM

Server: 4 x AMD Opteron 6376 CPU@2.3GHz (64 cores total), 128GB DDR3 RAM, Xen 4.8/Linux 4.8
LightVM: http://sysml.neclab.eu/projects/lightvm
Guests: Noop/Unikernel and Noop Docker image / No devices

Unikernels are even faster & lighter when using an optimized hypervisor toolstack
Unikernel Gains

- Fast instantiation, destruction and migration time
  - 10s of milliseconds
    - (LigthVM [Manco SOSP 2017], Jitsu [Madhvapeddy, NSDI 2015])

- Low memory footprint
  - Few MB of RAM (ClickOS [Martins NSDI 2014])

- High density
  - 10k guests on a single server node (LigthVM [Manco SOSP 2017])

- High Performance
  - 10-40Gbit/s throughput with a single guest CPU
    - (ClickOS [Martins NSDI 2014], Elastic CDNs [Kuenzer VEE 2017])

- Reduced attack surface
  - Less components exist in Unikernel
  - Strong isolation by hypervisor
When do I want a Unikernel?

Unikernels are applicable to many domains!
Question is about differentiators:

- Minimal SW stack
- Fast boot, migration destroy
- Resource efficient
- Small code base → cheap verification
- Specialization
- High performance
- Mission critical
- Strong isolation

Reactive VNFs, Lambda Functions, ...
Per-customer VNFs, IoT, MEC, ...
NFV, MEC, ...
Automotive, Industrial-grade, ...

© NEC Corporation 2018
Challenges with Unikernels

Cool, but why aren’t they more widespread?
The Devil is in the Details

- Each optimized Unikernel is manually built
- Building takes several months or longer
- Process needed to be repeated for each target application
Specialization in Practice

Standard OS/VM/container: 
**lots** of unnecessary code 
= **lots** of overhead!

Unikernel: only what’s needed is there **but lots** of development time!

General Purpose OS

Unikernel
Specialization in Practice

- Standard OS/VM/container: lots of unnecessary code = lots of overhead!

- Unikernel: only what’s needed is there but lots of development time!

Unikraft is about simplifying building & tuning of Unikernels

General Purpose OS

Unikernel
Unikraft

A Unikernel Build Framework
Unikraft - A Unikernel Framework

Motivation

- Core principle: Support everyone's use case!
- Simplify building and optimizing
- Concentrate efforts of each Unikernel project to a single base

Our Approach

- Concept: “Everything is a library”
- Decomposed OS functionality as libraries
- Unikraft’s 2 components:
  - Library Pool
  - Build Tool
Standard operating systems are *monolithic*: they are not modular so they are *not* designed for getting separated into their parts.

<table>
<thead>
<tr>
<th>Application(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>network stack</td>
</tr>
<tr>
<td>profiling</td>
</tr>
<tr>
<td>memory allocator</td>
</tr>
<tr>
<td>filesystem</td>
</tr>
<tr>
<td>timers</td>
</tr>
<tr>
<td>scheduler</td>
</tr>
<tr>
<td>drivers</td>
</tr>
</tbody>
</table>
Operating System Decomposition

Challenge: Have decomposed OS functionality as libraries, i.e., break apart an operating system.

Once decomposed, we can pick and choose which parts/libraries we actually need for our application!
1) Library Pool

1. SELECT APP
2. SELECT & CONFIG LIBS
3. BUILD
4. RUN

Main libs:
- Network stack: liblwip.o, libtcpipl.o, libhttp.o
- Drivers: libconsole.o, libixgbe.o, libnetfront.o
- Filesystems: libvfs.o, libfat.o, libext3.o
- Memory allocators: libbuddy.o, libheap.o, libmempool.o
- Runtimes: libocaml.o, libpython.o, liberlang.o
- Schedulers: libcoop.o, libpreempt.o, librt.o
- Standard libs: libc.o, libnewlibc.o, libopenssl.o
- Debug & profiling: libgdb.o, libucdebug.o, libperf.o

Platform libs:
- Libdockerplat.o, libbaremetalplat.o, Libkvmplat.o, libxenplat.o

Architecture libs:
- libx86_64arch.o, libarm32arch.o, libmipsarch.o

Unikernels:
- unicore_bare_x86_64
- unicore_bare_ARM32
- unicore_bare_MIPS
- unicore_xen_x86_64
- unicore_xen_ARM32
- unicore_xen_MIPS
- unicore_kvm_x86_64
- unicore_kvm_ARM32
- unicore_kvm_MIPS
- unicore_docker_x86_64

App: myapp
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)

- Xen, KVM, bare-metal, Linux

Save your config and type “make”
Example System

Python Unikernel for KVM on x86_64

<table>
<thead>
<tr>
<th>apppython.o</th>
<th>liblwip.o</th>
</tr>
</thead>
<tbody>
<tr>
<td>liballocbuddy.o</td>
<td>libconsole.o</td>
</tr>
<tr>
<td>libschedcoop.o</td>
<td>libfilesystem.o</td>
</tr>
<tr>
<td>libkvmplat.o</td>
<td>libx86_64arch.o</td>
</tr>
</tbody>
</table>
Example System

VNF Unikernel for KVM on x86_64

libdpdk.o
liballocbuddy.o  libconsole.o
libschedcoop.o  libfilesystem.o
libkvmlat.o  libx86_64arch.o

Click!

Modular ➔ Router

Click!

Modular ➔ Router

ClickOS
Unikraft 0.2 Titan

Current Status
1) Available Libraries

Main Libraries
- **libfdt**
  - Flat device tree parser
- **libnolibc**
  - A tiny libC replacement
- **libnewlib (external)**
  - Porting ongoing
- **libukalloc**
  - Memory allocator abstraction
- **libukallocbbuddy**
  - Binary buddy allocator for libukalloc
- **libukargparse**
  - Argument parser library
- **libukboot**
  - Unikraft bootstrapping
- **libukdebug**
  - Debug and kernel printing
  - Assertions, hexdump
- **libuksched**
  - Scheduler abstraction
- **libukschedcoop**
  - Cooperative scheduler for libsched

Architecture Libraries
- **libarmmath**
  - 64bit arithmetics on ARMv7

Platform Libraries
- **libxenplat**
  - Xen (PV)
  - X86_64, ARMv7
- **libkvmplat**
  - X86_64
- **liblinuxu**
  - Linux userspace
  - X86_64, ARMv7

More is coming!
2) Build System

- **Makefile**
  - **Makefile.uk**
  - **Config.uk**
  - **Sources**

- **Unikraft**
  - **Makefile System**
    - **KConfig**
      - **Internal Library**
        - **Config.uk**
          - **Sources**
        - **Makefile.uk**
          - **Sources**

- **Your Application**
  - ** Sources**
An Baseline Example...

- **Xen PV x86_64 binary**
  - Compiles to a 32.7kB image
  - `unikraft_xen-x86_64.o` (50.2kB)

- **Boots and prints messages to debug console (with min. 208kB RAM)**
We want you!

It is Open Source!
Recently: Unikraft got released as Xen incubator project
- OpenSource development under Xen and LinuxFoundation umbrella

We need you for our ambiguous goals!

Have look on the Xen Pages and Wiki
- https://www.xenproject.org/developers/teams/unikraft.html
- https://wiki.xenproject.org/

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

Join our IRC channel
- #unikraft on Freenode

We have plenty of open topics!
Get in touch with us!
Demo Time

It is real!
Unikraft Resources

Thanks!

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

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

- **Sources (GIT)**

- **Mailing list (shared with Mini-OS)**
  - minios-devel@lists.xen.org

- **IRC Channel on Freenode**
  - #unikraft

- **NEC-Team**
  - [http://sysml.neclab.eu](http://sysml.neclab.eu)
This work has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement no. 671566 (“Superfluidity”). 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.