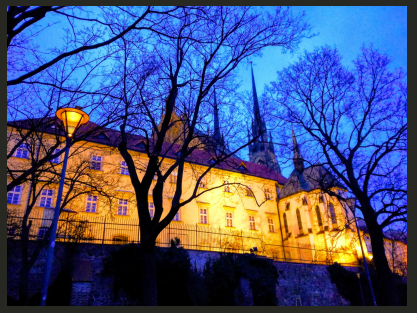
Unikernels in Action

28 January 2018, DevConf.cz, Brno

Michael Bright, Developer Evangelist @ Containous



Slides online @ https://mjbright.github.io/Talks/2018-Jan-28_Devconf.cz_Unikernels

@mjbright

Agenda

- What are Unikernels ?
    - What they are not.
    - Advantages / Characteristics
    - Application domains
    - Implementations & Tools
- IncludeOS demo
- What can we expect to see in 2018?

What are Unikernels?

"Unikernels are specialized, single-address-space machine images constructed by using library operating systems"

"What are Unikernels", unikernel.org

What are Unikernels?

"Unikernels are specialized, single-address-space machine images constructed by using library operating systems"

"What are Unikernels", unikernel.org

"VMs aren't heavy, OSes are"

Alfred Bratterud, #IncludeOS

@mjbright

What are Unikernels? - They are "Library OS"

Specialized applications built with only the "OS" components they need.

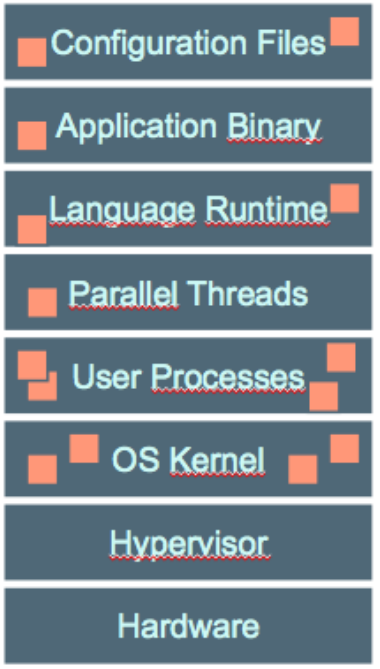
A Unikernel image runs directly as a VM

(or on bare metal?)

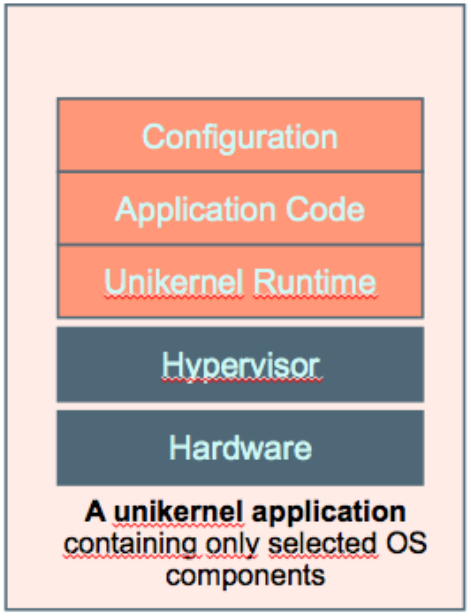
"OS" components such as Network stack, File-system, Device drivers are optional

Typically, there is no filesystem.

Configuration is stored in the unikernel application binary



Typical application running above an OS

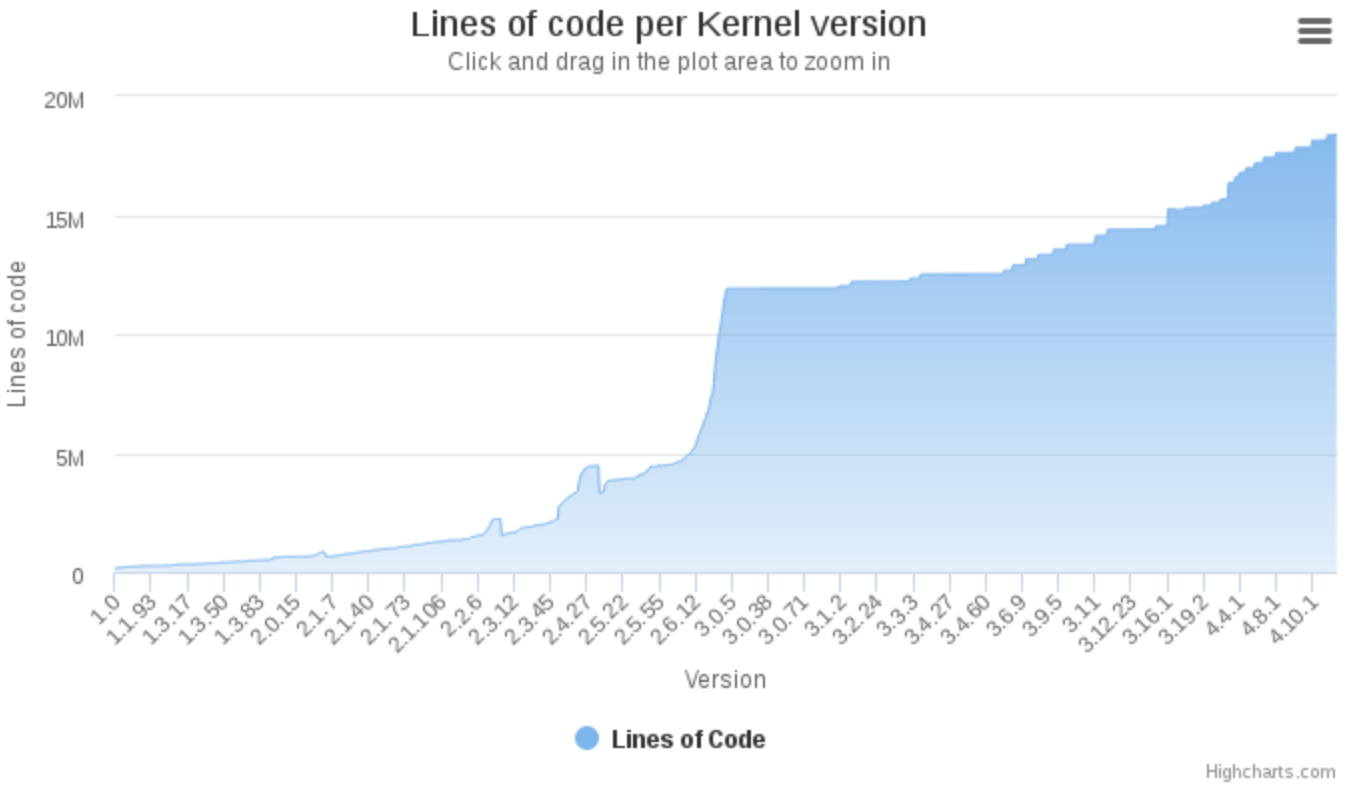


A unikernel application containing only selected OS components

@mjbright

Unikernels: What they are **not** ... General Purpose

OS kernels with unneeded features e.g. floppy drivers, designed to run any software on any hardware are huge - **lines of code**



@mjbright

Unikernels are **not** *"top-down"* minified versions of General Purpose OSes ...

Unikernels: **Are** ...

Very small compared to an application + OS

- use few resources - allows high density
- immutable, suitable for micro-services
- No legacy drivers
- No unneeded shell - **did I mention this?**

Have no separate **kernel space**

- No need to copy between kernel and **user space**

Unikernels: **Are** ...

Very small compared to an application + OS

- use few resources - allows high density
- immutable, suitable for micro-services
- No legacy drivers
- No unneeded shell - **did I mention this?**

Have no separate **kernel space**

- No need to copy between kernel and **user space**

More secure

- small attack surface
- If compromised, the attacker can't do much - **no** shell, users, processes ...

Unikernels: **Are** ...

Very small compared to an application + OS

- use few resources - allows high density
- immutable, suitable for micro-services
- No legacy drivers
- No unneeded shell - **did I mention this?**

Have no separate **kernel space**

- No need to copy between kernel and **user space**

More secure

- small attack surface
- If compromised, the attacker can't do much - **no** shell, users, processes ...

Fast to boot

- Possibility of on demand services

Unikernels: **Are** ...

Very small compared to an application + OS

- use few resources - allows high density
- immutable, suitable for micro-services
- No legacy drivers
- No unneeded shell - **did I mention this?**

Have no separate **kernel space**

- No need to copy between kernel and **user space**

More secure

- small attack surface
- If compromised, the attacker can't do much - **no** shell, users, processes ...

Fast to boot

- Possibility of on demand services

More difficult to develop

- libraries, languages, debugging limitations

Unikernels: Application Domains

Cloud Computing and *NFV*

- Fast to boot: On demand services
- Secure immutable images

Unikernels: Application Domains

Cloud Computing and *NFV*

- Fast to boot: On demand services
- Secure immutable images

IoT / Embedded

- Small images for OTA updates
- Secure immutable images

Unikernels: Application Domains

Cloud Computing and *NFV*

- Fast to boot: On demand services
- Secure immutable images

IoT / Embedded

- Small images for OTA updates
- Secure immutable images

HPC

- Secure in the cloud
- Very efficient (no context switches, just 1 process)

IETF draft on Containers for NFV expired Jan 2017

Taken from: draft-natarajan-nfvrg-containers-for-nfv-03.txt , Slides

4.2. Instantiation Times

Measurement of time to boot image, up to the 1st RST packet (to a SYN flood).

Technology Type	Time (msecs)
standardvm.xen	6500
standardvm.kvm	2988
Container	1711
tinyx.kvm	1081
tinyx.xen	431
unikernel.osv.kvm	330
unikernels.minios.xen	** 31 **

Note:

- These unikernels include just one application - iperf.
- Tinyx is "Tinyfied Linux" running 4.4.1 kernel - busybox+sshd+iperf
- Standard VM is Debian running 4.4.1 kernel + iperf
- Docker container including iperf

IETF draft on Containers for NFV expired Jan 2017

4.3. Throughput

TCP/IP throughput was measured using iperf from guest to host (to avoid physical medium limitations)

Technology Type	Throughput (Gb/s) Tx	Throughput (Gb/s) Rx
standardvm.xen	23.1	24.5
standardvm.kvm	20.1	38.9
Container	45.1	43.8
tinyx.kvm	21.5	37.9
tinyx.xen	28.6	24.9
unikernel.osv.kvm	** 47.9 **	** 47.7 **
unikernels.minios.xen	** 49.5 **	32.6

Note:

- Throughput depends not just on guest efficiency
- Xen is optimized for Tx but not Rx (similar to ClickOS experience)

IETF draft on Containers for NFV expired Jan 2017

4.4. RTT

Average round-trip time (RTT) measured from an external server using a ping flood.

Technology Type	Time (msecs)
standardvm.xen	34
standardvm.kvm	18
Container	** 4 **
tinyx.kvm	19
tinyx.xen	15
unikernel.osv.kvm	9
unikernels.minios.xen	** 5 **

IETF draft on Containers for NFV expired Jan 2017

4.5. Image Size

We measure image size using the standard "ls" tool.

Technology Type	Size (MBs)
standardvm.xen	913
standardvm.kvm	913
Container	61
tinyx.kvm	3.5
tinyx.xen	3.7
unikernel.osv.kvm	12
unikernels.minios.xen	** 2 **

IETF draft on Containers for NFV expired Jan 2017

4.6. Memory Usage

"top" and "xl" (on Xen) used to measure memory usage:

Technology Type	Usage (MBs)
standardvm.xen	112
standardvm.kvm	82
Container	** 3.8 **
tinyx.kvm	30
tinyx.xen	31
unikernel.osv.kvm	52
unikernels.minios.xen	8

Note:

- OSv pre-allocates memory, e.g for buffers
- Best result is Docker as it has no OS function

Unikernel implementations

Unikernel Implementations: 2 families

Clean-Slate	**Legacy**
- Minimalist approach	- POSIX compatibility
- Re-implement needed OS functions	- Re-use existing libraries
- Typically uses type safe language	- Possible binary compatibility
- Very small code size, resources	- Small to large code size/resources
- Harder to develop apps	- Easier to develop apps

Unikernel Implementations: 2 families

Clean-Slate	**Legacy**
- Minimalist approach	- POSIX compatibility
- Re-implement needed OS functions	- Re-use existing libraries
- Typically uses type safe language	- Possible binary compatibility
- Very small code size, resources	- Small to large code size/resources
- Harder to develop apps	- Easier to develop apps

Unikernel Implementations: 2 families

Clean-Slate	**Legacy**
- Minimalist approach	- POSIX compatibility
- Re-implement needed OS functions	- Re-use existing libraries
- Typically uses type safe language	- Possible binary compatibility
- Very small code size, resources	- Small to large code size/resources
- Harder to develop apps	- Easier to develop apps

We can see that *Legacy* Unikernels trade off some principles for ease of use ...

Unikernel Implementations:

Clean-Slate	**Legacy**
MirageOS (Ocaml)	OSv
HalVM (Haskell)	Rumprun (**+LKL**)
LING (Erlang)	.red[Runtime.js]
IncludeOS (C/C++)	HermitCore
	Graphene
	ClickOS
	Vorteil
Tools	Clive
Solo5/ukvm	Magnios
Unik	Ultibo
Unikraft	Drawbridge
Minios	… others ? …

Unikernel Implementations: IncludeOS

includeOS

includeos.org

Clean Slate

Open Source

Backing
(IncludeOS)

C/C++

includeos.
readthedoc.io

CppCon 2017

@mjbright

Open source Unikernels written in C++ - `#include <os>`

Runs on hypervisors (KVM, VMWare) *maybe* baremetal (E1000 support recently added) ...

Many features such as multi-threading, multi-cores can be compiled in (experimental today). Single-memory space.

Delegates to route messages between TCP/IP stack components.

No blocking POSIX calls implemented yet, only async i/o.

Recent developments:

- Currently integrating MUSL musl-libc.org
- Dashboard available as commercial product
- NaCl DSL to define network configurations
  - allows to build firewalls, routers, load-balancers
- Added Solo5 (ukvm) support
- Became 64-bit
- Added ARM support
- Worked with Mender (mender.io) for OTA updates

Demo

IncludeOS

- building IncludeOS unikernels
  - Native (could use Docker images)
- deploying IncludeOS on OpenStack (KVM)

Past demos include:

- deferpanic.net with rumpkernel/Python + remark.js slideset
- runtimejs under qemu
- MirageOS linux build/run, ukvm run, GCE run
- OSv/capstan tomcat

What can we expect to see in 2018?

More trials of **specialized** applications, e.g. networking components.

Unikernels becoming easier to use/deploy/debug

- Solo5: More backend support
- Unik as common unikernel compiler
- Unikraft as a tool for building Unikernels
- More Unikernel support from PaaS (kubernetes+virtlet)

IncludeOS

- Becomes production ready, trial deployments
- More capabilities around multi-thread, multi-core
- Limited bare-metal support
- More languages?

Docker / MirageOS ?

- MirageOS to support ReactML ?
- Progress on MirageSDK (part of LinuxKit)

Q&A

Resources

Resources - General

	URL
.	
Unikernel.org	site
Wikipedia	Wiki
.	
Scoop.It	Unikernels
Playlist	YouTube Unikernels

Resources - Unikernel Implementations

Technology	Backers	URL
.		
MirageOS	Xen	mirage.io
HalVM	Galois	galois.com/project/halvm
LING		erlangonxen.org
.		
IncludeOS	IncludeOS	includeos.org
Rumprun	NetBSD	rumpkernel.org
OSv	Cloudius	osv.io
HermitCore	Univ. Aachen	hermitcore.org
.		
Unik	CloudFoundry	github.com/cf-unik/unik
Solo5	IBM	github.com/Solo5/solo5
Ukvm	IBM	github.com/Solo5/solo5/tree/master/ukvm

Resources - Unikernel Implementations (2)

Technology	Backers	URL
.		
Ultibo (Raspi) Clive (Go) Magnios ClickOS	NEC	
.		
Drawbridge	Microsoft	project/drawbridge
.		
DeferPanic	DeferPanic	deferpanic.net

@mjbright