Unikernel
Only Run What You Need

by Aaron Grothe

Introduction

What is a Unikernel?

A Unikernel is a single address space program. It combines what we normally think of as a kernel and the program into one running executable. So it can usually be run on a hypervisor or bare metal.

There are several different approaches to accomplish this we'll talk about them a bit more later in this talk.

Introduction (Continued)

If you have questions/comments please feel free to ask them anytime. You don't have to hold them until the end of the talk.

If there are other resources similar to these that you think might be useful to people please let the group know.

Hopefully this will be an interactive and productive session.

OPS.city

Let's build a unikernel to show how it goes

We'll use ops.city's product for this since it is pretty easy to use.

% curl https://ps.city/get.sh -sSfL | sh

This will install the basic components for the nanovms software

OPS.city - (cont'd)

Now we need an application

We'll create an echo server in the go programming language:

We'll use the one from Rosetta Code.org

https://rosettacode.org/wiki/Echo_server#Go

OPS.city - (cont'd)

Now we need to compile the program

```
GOOS=linux go build echo.go
```

Now we should have an executable, lets go ahead and run it locally.

```
% ./echo
```

OPS.city - (cont'd)

Connect to the server and try it out

% telnet localhost 8080

Kill it and verify it isn't working anymore

% killall echo; telnet localhost 8080 # should fail

OPS.city - (cont'd)

Now lets go ahead and create and fire up a unikernel

% ops run -p 8080 -c config.js echo

What does this do. It creates an image in the .ops/image folder named echo
It fires up qemu and starts that image with config.js and echo programs

Time to try it out

% telnet localhost 8080

OPS.city - (cont'd)

So we can kill the program now

Let's take a look at the image. The image by default is in the ~/.ops/images folder - should be named echo

% file ~/.ops/images/echo.img

It is a raw image.

OPS.city - (cont'd)

Now we'll run it outside of the ops environment. We have a raw image so we should be able to boot it up.

```
% qemu-system-x86_64 -drive
file=/home/aarong/.ops/images/echo.img,format=raw,if=non
e,id=hd0 -device virtio-blk,drive=hd0 -device
virtio-net,netdev=n0 -netdev
user,id=n0,hostfwd=tcp::8080-:8080 -nodefaults -no-reboot
-device isa-debug-exit -m 2G -display none -serial stdio
```

Yes it is fugly. Could simplify a bit but it gives you an idea.

OPS.city - (cont'd)

Give it a minute to boot and let's try it out

% telnet localhost 8080

Let's kill the qemu session

% killall qemu

% telnet localhost 8080

OPS.city - (cont'd)

So what have we done here.

- We've compiled a go program into an executable.
- We've created a config.json file and created an ops image
- We've run the ops image both inside and outside of the ops environment

OPS.city - (cont'd)

OPS.city has a lot of additional capabilities

We took an executable and built it into an image. We'll take a look at some of the packages included with it.

% ops pkg list

Includes things like ruby, php, nodejs, java, python and so on

OPS.city - (cont'd)

Also OPS.city can do things like automatically deploy to the google cloud platform.

OPS.city also has instructions for deploying to other clouds like Digital Ocean and so on.

OPS.city - (cont'd)

Language support/Stability

- Right now OPS.city works pretty well with golang, php, nodejs, and ruby
- Java experiments did not go well :-(Started with Echo Server, and worked my way down to Hello World without success
- Tried various options like nightly builds and so on without success
- Still early days with this, but cool stuff :-)

Types of Unikernels

- Generic Unikernels - these are able to run general programs. Can be in many different locations, other examples of this include RumpRUN
- Language Specific Unikernels - these are designed to support one specific language/runtime. E.g. Clive for Go programming language. Kind of a glorified Read-Evaluation-Print-Loop (REPL)
- Reduced O/S. An example of this is Hermitux, that is able to run Linux executables with a reduced O/S size
- Other - there are a lot of other types included as well

Some Other Unikernels

- Clive - Operating System written in Go programming language
- ClickOS - Supposed to be very fast, boot 20 milliseconds, minimal Xen OS
- HalVM - Haskell Compiler Tool Suite
- IncludeOS - library used to generate OS images, designed to run C++ code
- RumpRun - NetBSD based system
- UniK - system that is similar to docker to build Unikernels
- Hermitux - takes posix executables from Linux and runs them under a reduced O/S

Unikernel Linux

- Recent paper was published proposing a set of extensions for the Linux kernel to generate Unikernels
- Interesting part of this is the approach they plan on taking as it would be a simple build of a Linux kernel with the UKL option to generate a self-contained bootable executable
- Is still very early in design, but looks very interesting. Compares to Hermitux
- Leverages off of previous designs such as User Mode Linux (UML) to help with design

Benefits of Unikernels

- Less code
- Smaller environment
- Works well in a devops type of environment
- Reduced attack surface
- Better Security?

Drawbacks of Unikernels

- "Unikernels are unfit for production" - article by Bryan Cantrill that is pretty tough
  - Debugging can be tough, hello printf
- Limitations of program (no memory swapping, processes)
- Can be tough to do complicated programs
- Early days, e.g. Rumprun has been unsupported for quite some time
- Once a unikernel is compromised the attacker has full privs to the environment
- No concept of UserIDs, User permissions, memory checks and so on

Security of Unikernels?

- This is complicated
- There is a really good paper on this by the NCC group titled "Assessing Unikernel Security"
  - Some security features aren't addressed in some Unikernels
    - ASLR - Address Space Layout Randomization
    - W^X, NX protection not enabled
  - The CEO of OPS.City had a blog post rebutting most of the findings of the paper
- Some of the issues are less dangerous when running on a VM since it is running in Ring 3 instead of Ring 0

History of Unikernels

- Unikernels go back to the 1990s and have been around in various forms since then
- With rise of containers and DevOPs, unikernels are being reevaulated
- Some people say that Forth is the original Unikernel
  - Still being used by Nasa :-)
  - Forth goes back to the 60s, btw

Future of Unikernels

- There will be some consolidation in this area and more startups and companies closing
- Internet of Things (IoT) offers some potential new places for the deployment of Unikernels
- When a large company announces a project using a Unikernel will be interesting
- Kubernetes plus Unikernels might be very interesting :-)

Q & A

Any Questions?

Thanks for listening.

Links

OPS.city

https://ops.city

Rossetta Code - Echo Server samples

https://rosettacode.org/wiki/Echo_server#Go

Links

- Clive - http://lsub.org/ls/clive.html
- ClickOS - http://cnp.neclab.eu/projects/clickos/
- HalVM - http://galois.com/project/halvm/
- IncludeOS - http://www.includeos.org/
- RumpRun - http://rumpkernel.org/
- UniK - https://github.com/emc-advanced-dev/unik
- Hermitux - https://ssrg-vt.github.io/hermitux/

Links.

Assessing Unikernel Security

https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2019/april/assessing-unikernel-security/

Unikernels are unfit for production

https://www.joyent.com/blog/unikernels-are-unfit-for-production

Forth Programing Language

https://en.wikipedia.org/wiki/Forth_(programming_language)

Links.

Unikernels: The Next Stage of Linux Dominance

https://www.cs.bu.edu/~jappavoo/Resources/Papers/unikernel-hotos19.pdf