Rumprun for Rump Kernels:

Instant Unikernels for POSIX applications

Martin Lucina, @matolucina

So many Kernels, what are they?

- Monolithic kernel, Microkernel: standard stuff.
- Rump kernel: an existing monolithic kernel componentized into an *anykernel*. Mix and match kernel components any way you want.
- Unikernel: turns the OS into a “library” for a single application. Normally requires writing your application from the ground up for the Unikernel. Mirage OS, for example.

Rump kernels

- Today's talk by @anttikantee has all the details, won't repeat them here.
- Many different use cases:
  - Use Rump Kernels to bootstrap a new OS quickly.
  - Use Rump Kernel components directly in a userspace application.
  - Use Rump Kernels as Unikernels for *unmodified* POSIX applications.

The last point is what will be demonstrated in this talk.

Rump kernels as Unikernels for POSIX applications

Rump kernels already provide most of the components we need:

- Core system calls.
- File systems, network stack, device drivers, ...

What else do we need?

1. Threads, scheduling, memory management, locking, event handling (interrupts): Provided by the hypervisor and “firmware”.
2. A C library: We re-use the NetBSD libc with minimal modifications.
3. A magic* build system: *app-tools*.
4. A magic* deployment system: *rumprun*.

* One that just works. No fiddling.

Rumprun

We need an easy way to provision and deploy the application on the various different stacks:

- Configure network devices.
- Configure block devices and mount filesystems.
- Platform-specific launching (Xen, KVM, ...).

The `rumprun` tool and `rumpconfig` module which I have been working on is the beginning of this:

```
rumprun xen -di -n inet,static,10.10.10.10/16 \
    -b images/stubetc.iso,/etc \
    -b images/data.iso,data \
    .../mathopd -n -f /data/mathopd.conf
```

The Xen version uses Xenstore to communicate configuration to the application. KVM, bare metal, etc. will need to use a mechanism yet TBD.

The fun part, the demo!

This demo is done using rumprun-xen, since that stack is complete right now. This stack runs on the Xen hypervisor, repurposing Mini-OS as the “firmware”. Thus the application runs as a PV guest on Xen.

Other stacks exist. rumpuser-baremetal is in progress and should work with KVM, as a Xen HVM guest or on bare metal. Deployment will be different on each of these, but rumprun can be extended to support all the stacks.

I will demonstrate:

1. Building the entire stack and running “Hello World”.
2. Building an *unmodified* POSIX HTTP server, mathopd.
3. Deploying and running it on a Xen host with *rumprun*.

This slide intentionally left blank for the demo.

What is it good for?

Security:

- An alternative to containers, with much stonger isolation guarantees.
- Making the standard OS go away reduces the attack surface.
  - If there is no shell, there is nothing to break in to!

Performance:

- Application and Unikernel run at the same privilege level.
- Greatly reduced cost of context switches.
- Should help latency-sensitive workloads.

Next steps

- Stabilise & fix bugs.
- Upstream Mini-OS work to Xen.
- More POSIXy interfaces. "Processes" and *fork()* emulation?
- Improve rumprun and merge with rumpuser-baremetal stack.

Questions?

Resources

- Rump Kernels: http://rumpkernel.org/
- rumprun-xen: http://repo.rumpkernel.org/rumprun-xen
- This demo: https://github.com/mato/rump-mathopd

Thank you for listening.

Martin Lucina, November 2014
@matolucina, https://lucina.net/