Deploying real-world software today as unikernels on Xen with Rumprun

Martin Lucina
@matolucina
mato@rumpkernel.org

Rumprun: Introduction

- Rump kernels provide production quality kernel drivers such as file systems, POSIX system call handlers and a TCP/IP network stack.
    - Development started circa 2007 as part of the NetBSD project.
    - Use the NetBSD *anykernel* architecture to provide *unmodified* drivers.
- The Rumprun unikernel is a software stack for building and running *unmodified* POSIX applications as unikernels.
    - Initial support for the Xen platform was committed almost exactly two years before this talk as rumpuser-xen, and later morphed into the Rumprun unikernel.
    - Today we support Xen, KVM and bare metal as platforms, on the x86, x86_64 and ARM architectures.
- More detailed history in @anttikantee's article, On rump kernels and the Rumprun unikernel.

Rumprun: Hello, World

The first non-trivial program which ran as a rump kernel-based unikernel on Xen was netcat, so we'll use it again as our "Hello, World" example today:

```
$ x86_64-rumprun-netbsd-gcc -o netcat netcat.c
$ rumpbake xen_pv netcat.bin netcat
```

As root, we `rumprun`* our freshly-baked unikernel:

```
# rumprun xen -di -I xen0,xenif -W xen0,inet,static,10.0.120.120/24 \
>    -- ./netcat.bin -l -p 3333
(...)
xenif0: Ethernet address 00:16:3e:5f:d2:3c

=== calling "./netcat.bin" main() ===

rumprun: call to ``sigaction'' ignored
rumprun: call to ``_sys___sigprocmask14'' ignored
```

* `rumprun` command syntax is not final and subject to change. See august 2015 status report for details.

Rumprun: Hello, World

Elsewhere, we connect to it and send our message:

```
$ echo "Hello, World" | nc -q0 10.0.120.120 3333
```

Our artisanal netcat unikernel processes the message and dutifully shuts down:

```
Hello, World

=== _exit(0) called ===
rump kernel halting...
syncing disks... done
(...)
#
```

Finished with engines!

Rumprun: Hello, World

What just happened?

- We took an existing, *unmodified* POSIX application, vintage 2011.
- Cross-compiled it into a unikernel using the rumprun cross toolchain.
- Baked the final unikernel image using `rumpbake` for a Xen PV platform.
- Booted the unikernel as a Xen domU, processed a request and shut it down.

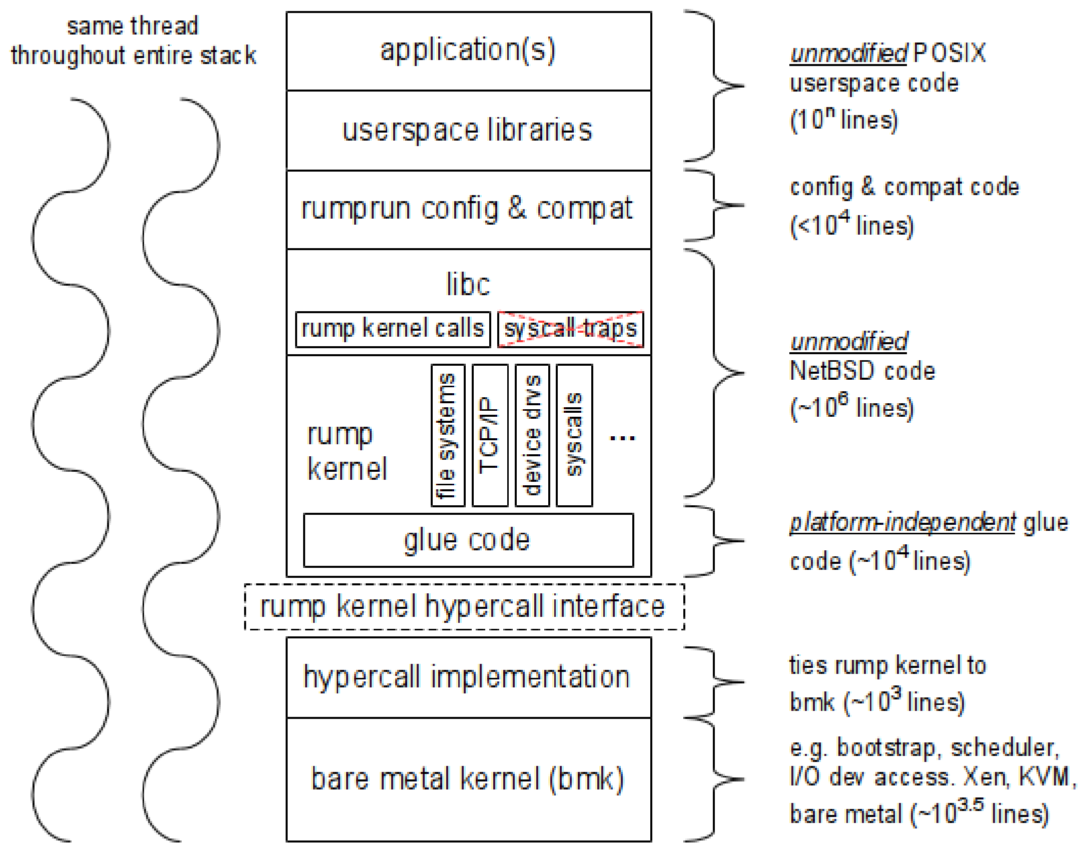
Ok, but what's inside this thing?

```
-rwxr-xr-x 1 mato mato 36320 Aug 13 22:54 netcat
-rwxr-xr-x 1 mato mato 19234176 Aug 13 22:56 netcat.bin

$ size netcat.bin
   text    data     bss     dec     hex filename
2261696  358500 2135152 4755348  488f94 netcat.bin
```

Seems a bit big, let's take a closer look.

Rumprun: Architecture


same thread throughout entire stack
application(s)
userspace libraries
rumprun config & compat
libc
rump kernel calls
~~syscall traps~~
rump kernel
file systems
TCP/IP
device drvs
syscalls
...
glue code
rump kernel hypercall interface
hypercall implementation
bare metal kernel (bmk)
unmodified POSIX userspace code (10^n lines)
config & compat code ($<10^4$ lines)
unmodified NetBSD code ($\sim10^6$ lines)
platform-independent glue code ($\sim10^4$ lines)
ties rump kernel to bmk ($\sim10^3$ lines)
e.g. bootstrap, scheduler, I/O dev access. Xen, KVM, bare metal ($\sim10^{3.5}$ lines)

Rumprun: Components

- Rump kernel components:
  - POSIX system calls, TCP/IP stack, file systems, device drivers.
- Full C and C++ standard library (direct from NetBSD upstream), including pthreads.
- Rumprun config & compat code:
  - Integration with the `rumprun` script and toolchain.
- Bare metal kernel:
  - *Co-operative* thread scheduler.
  - For the Xen platform, this also contains the remains of Mini-OS, mostly as early boot code and the Xen PV frontend drivers (`netfront`, `blkfront`, `pcifront`).

It turns out this is enough* to run a lot of existing applications out of the box.

* Notable omissions: No `fork()` or `exec()`, no virtual memory.

Rumprun: Toolchain

Rumprun provides a standard compiler toolchain, indistinguishable from a normal cross compiler, so that you can build existing software out of the box.

Support for common build systems is provided:

- GNU autoconf: `./configure --host=x86_64-rumprun-netbsd`
- CMake: `cmake -DCMAKE_TOOLCHAIN_FILE=.../x86_64-rumprun-netbsd-toolchain.cmake`
- Plain make: `make CC=x86-64-rumprun-netbsd-gcc`

No shared libraries or dynamic linking, everything is linked together statically into a single address space.

The `rumpbake` tool is used for the final link ("baking") of the unikernel, and can be configured to leave out components which are not required by the application or target platform.

Rumprun: Rumpbake

The definition of the `xen_pv` configuration is in `app-tools/rumpbake.conf`:

```
# Paravirtualized Xen
addconfig xen_pv rumprun-xen \
    RND FS KERNFS DISK NETINET NETUNIX NETBPF NETCONFIG XEN XEN_NET SYSPROXY
```

The definition of the `FS` components:

```
LIBS_FS="rumpfs_ffs rumpfs_cd9660 rumpfs_ext2fs rumpfs_tmpfs"
```

For netcat, we can strip this configuration down:

```
addconfig xen_pv_minimal rumprun-xen KERNFS NETINET NETCONFIG XEN_NET
```

If we now re-bake netcat as `rumpbake xen_pv_minimal netcat.min netcat`, we can see that the resulting unikernel is smaller:

```
   text    data     bss     dec     hex filename
2261696  358500 2135152 4755348  488f94 netcat.bin
1831377  305724 2023280 4160381  3f7b7d netcat.min
```

(4MB is still embarassingly large, reducing the size further is low-hanging fruit to fix.)

Rumprun: Packages

We provide a nascent packaging system at rumprun-packages, similar to BSD pkgsrc. The goal is to encourage community participation and to provide pre-packaged software builds for rumprun.

Packaged so far: hiawatha, leveldb, libxml2, mathopd, mpg123, *mysql*, *nginx*, ngircd, pcre, *php*, python, redis, roundcube.

Most packages build with just `make`. Where patches are required, 99% of the time these are related to the build system and cross compilation.

A lot of upstream software does not support cross compilation out of the box (Nginx, I'm looking at you)!

Thankfully, quality patches for cross compiling existing software have been developed, for example by the buildroot project. These can often be re-used for rumprun.

Serving a static website with Nginx

With `rumprun-packages/nginx`:

1. Put your files in `images/data/www`.
2. Edit `images/data/conf/nginx.conf` to taste.
3. Run `make` and `rumpbake xen_pv bin/nginx.bin bin/nginx`.
4. Run the unikernel:

   ```
   rumprun xen -M 128 -di \
   -I xen0,xenif -W xen0,inet,static,10.0.120.120/24 \
   -b images/stubetc.iso,/etc \
   -b images/data.iso,/data \
   -- bin/nginx.bin -c /data/conf/nginx.conf
   ```

The above invocation runs the `nginx.bin` unikernel with 128 MB of memory, a single network interface using the Xen toolstack defaults, and two block devices using ISO image files.

Rumprun: Network configuration

Creating a guest network interface:

```
-I tag,xenif[,backendopts]
```

Creates `xenif0` inside the guest. Passes `backendopts` to the Xen toolstack, so can contain any standard Xen network configuration options, such as `bridge=`, `mac=`, `script=`.

Configuring the created network interface inside the guest:

```
-W tag,method,...
```

- Static IPv4 address: `-W tag,inet,static,addr/mask[,gateway]`
- IPv4 with DHCP: `-W tag,inet,dhcp`
- IPv6 stateless autoconfiguration: `-W tag,inet6,auto`

`tag` has no functional significance, is used only to match up `-I` with corresponding `-W` option(s).

Rumprun: Storage configuration

Creating a block device in the guest:

```
-b hostpath,[mountpoint]
```

Creates a block device in the guest, mapped to the host path `hostpath`.

- `hostpath` may be either a regular file or a block device.
- If `mountpoint` is specified and `hostpath` contains a supported file system (FFS, CD9660, ext2), it will be mounted at `/mountpoint` in the guest.

Why file system image files?

- Easy to work with, re-create as needed.
- Using ISO images for read-only static data ensures the data cannot be modified by the guest.

Could use LVM volumes, but that needs more infrastructure (snapshots?) for the common "update data and reboot" life cycle.

Deployment: Startup and shutdown

`rumprun xen -di` can be fed to a pipe, and the process will terminate when the domU shuts down. This works quite well with, for example, `systemd`:

```
[Unit]
Description=Rumprun (Nginx) (u-67)
Requires=xen.service
After=xen.service

[Service]
Environment="U_MAC=00:16:3e:04:04:04" "U_IP=62.176.XXX.XXX" "U_GW=62.176.XXX.XXX"
Environment="U_DATADIR=/srv/rumprun/nginx-u-67" "U_VIFNAME=vif.nginx-u-67"
Restart=always
ExecStart=/srv/rumprun/bin/rumprun xen -N %p -M 128 -di \
    -I xen0,xenif,mac=${U_MAC},ip=${U_IP},vifname=${U_VIFNAME},script=vif-route \
    -W xen0,inet,static,${U_IP}/24,${U_GW} \
    -I xen1,xenif,bridge=xenbr1 \
    -W xen1,inet,static,10.1.1.67/24 \
    -b /srv/rumprun/common/stubetc.iso,/etc \
    -b ${U_DATADIR}/data.iso,/data \
    -- \
    /srv/rumprun/bin/nginx.bin -c /data/conf/nginx.conf
ExecStop=/usr/sbin/xl destroy %p

[Install]
WantedBy=local.target
```

Deployment: Logging

Logging options:

- Pipe the Xen console (`rumprun -i`) to `logger` or `systemd`.
    - Relies on small patches to `xenconsole`, coming in Xen 4.6.
- Use a remote syslog server.
    - `xenbr1` in the previous slide is used as a management network and provides access to a syslog server.
    - Nginx has support for logging to a remote server via syslog:
      `access_log syslog:server=10.1.0.1,facility=local7 combined;`

Rsyslog can log into separate files per hostname (`-N` option to `rumprun`):

```
ruleset(name="rumprun") {
    $template RumprunDynamic,"/var/log/rumprun-%hostname%.log"
    local7.* ?RumprunDynamic
}

$ModLoad imudp
$UDPServerRun 514
input(type="imudp" port="514" ruleset="rumprun")
```

Deployment: Monitoring and vital statistics

Currently there is no easy way to extract vital statistics (memory usage, for example) from a Rumprun unikernel.

The easiest approach is to periodically print statistics to the console or syslog. I have some proof of concept code for this, but it's not ready yet.

An alternative is to use the *sysproxy* mechanism provided by rump kernels, which allows remote clients to execute system calls *inside* the unikernel. This would allow us to use *unmodified* NetBSD tools such as `df`, `netstat` and `vmstat` to inspect a running unikernel.

So, more work is needed in this area. In the mean time, make sure you run your unikernels with enough resources (especially memory).

The RAMP stack

MySQL, Nginx and PHP make up the *RAMP* stack, the Rumprun Answer to MySQL and PHP:

- An Nginx unikernel, serving HTTP(S) on the frontend and using FastCGI on the backend.
- A PHP unikernel, running your legacy PHP application.
    - To serve multiple requests concurrently, just run multiple PHP unikernels.
- A MySQL unikernel, providing the database for PHP.

Millions of lines of existing production code, running *unmodified* as unikernels.

I have packaged Roundcube webmail as an example application which you can use today.

(There's nothing special about Roundcube. I don't trust any PHP application, and as I already run Rouncube, I've wanted it as a unikernel since I started working on Rumprun.)

RAMP: Roundcube webmail

Get up and running:

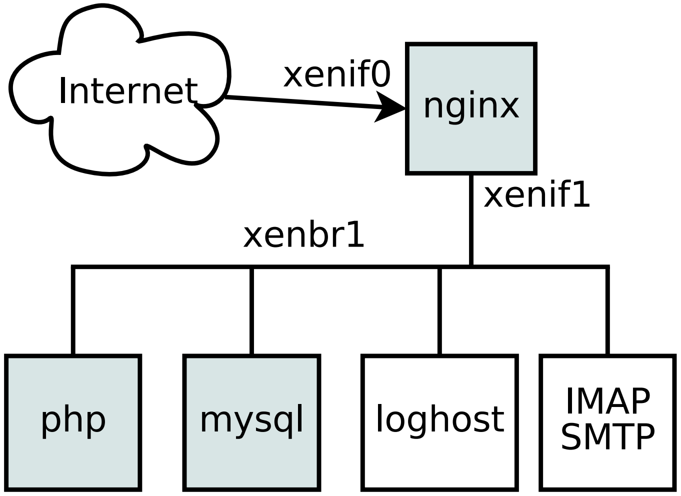
- Build and `rumpbake` the `nginx`, `php` and `mysql` packages.
- With the `roundcube` package:
  - Edit `config/etc-hosts` and `run/*.sh` to suit your network configuration (see next slide).
  - Edit `config/config.inc.php` to set your IMAP and SMTP configuration.
  - Review `config/nginx/nginx.conf` and optionally provide `config/nginx/cert.{key,pem}`.
  - Run `make`.

Run the built unikernels using the scripts in `run/`:

```
# run/u-nginx-1.sh
# run/u-php-1.sh
# run/u-mysql-1.sh
```

RAMP: Roundcube webmail network

The Xen network configuration used by the example script in the `roundcube` package:



`loghost` and IMAP/SMTP are provided by other (non-unikernel) Xen domUs.

This is "okay" except that, without complicated bridge firewalling, all domUs on `xenbr1` can talk to each other.

RAMP: Securing the network

Ideally, we want an arrangement where:

- The `nginx` unikernel(s) can only connect *to* `php` unikernel(s).
- The `php` unikernel(s) can only connect *to* the `mysql` unikernel.

There is no such thing as a "point to point link" in Xen networking:

- Could be done with routed networking, but then all traffic goes via dom0, and the firewalling is just as complicated.
- Could be done with separate bridges (`br-nginx-to-php`, `br-php-to-mysql`), but that rapidly gets *hard* to manage.

Xen provides the vchan mechanism for inter-domain communication via shared memory, which is exactly what we need.

A major future goal for Rumprun is to develop support for using vchan with existing *unmodified* applications. This will allow linking individual unikernels together easily and securely.

Rumprun unikernels and security

Rumprun unikernels can be an alternative to containers, with much stronger isolation guarantees.

A Rumprun unikernel does not contain a traditional OS, with a shell, and its program data is generally read-only and static.

If there is no shell, there is very little an exploit can do to gain a permanent foothold in the system. We can look at ways to make this even harder, e.g. ask Xen to "drop our privileges" and force all future page mappings to be NX.

With vchan links between domUs, a PHP unikernel need not even include a TCP stack.

The TCB of the Xen hypervisor is orders of magnitude smaller than that of container technologies.

Minimalism is a virtue! Rumprun unikernels are about starting with nothing and providing services with the *minimal* amount of code. Containers tend to be developed the other way round.

Future directions and challenges

- A `PF_UNIX` to `vchan` shim for Rumprun.
  - This will allow us to use a secure `vchan` channel for `sysproxy`.
  - And, combine different unikernel technologies:
  - Such as, replacing nginx with a type-safe MirageOS unikernel serving frontend HTTP and TLS.
- A better solution for host (and other domU) file store access.
  - NFS would probably just work, feel free to try. But, it's a pain to set up.
  - The KVM folks are using 9P over virtio. We could do the same with vchan.
- Improve debgging, logging and monitoring support.
- `rumprun ec2` support for running on AWS out of the box.
- Run more software and more language runtimes. This is a great way for people to get involved with the project!

Conclusion and getting involved

- Real-world, *unmodified* POSIX applications can be deployed as Rumprun unikernels, today.
- There is nothing special about the set of applications I have presented, except that I needed them.
- The project is a a level of maturity where people have started using Rumprun successfully to deploy applications they are interested in.
- You don't need to be a systems girl/guy to get involved!

Resources

- Rump Kernels: http://rumpkernel.org/
- The Rumprun unikernel stack: http://repo.rumpkernel.org/rumprun
- Mailing list: rumpkernel-users@freelists.org
- `#rumpkernel` on IRC (freenode)

Questions?

Thank you for listening.

Martin Lucina, Xen Project Developer Summit, August 2015
@matolucina, http://lucina.net/, mato@rumpkernel.org