Unikernels

Yiying Zhang
Announcements

• Project proposal graded
• AWS credits and lab machines assigned
• Project progress report due on Feb 20th
• Second quiz next Mon
  • Covers everything from container (1/24) to Firecracker (Wed)
Different OS Structures

- **User-Mode**
  - Monolithic Kernel
  - MicroKernel
  - ExoKernel (Library OS)

- **Kernel-Mode**
ExoKernel

- Only does protection and multiplexing
- Present hardware resources directly to users
Traditional Library OS

• Most OS functionalities implemented in the user space as libraries

• The kernel-space OS part only ensures protection and multiplexing

• Applications get to access hardware resources directly (faster)

• But isolation is hard

• and a lot of software (esp. device drivers) need to be rewritten
Traditional Library OS

- Most OS functionalities implemented in the user space as libraries
- The kernel-space OS part only ensures protection and multiplexing
- Applications get to access hardware resources directly (faster)
- But isolation is hard
- and a lot of software (esp. device drivers) need to be rewritten

What if instead we run libOS on hypervisor (as VM)?
Unikernel

- “Unikernels are specialised, single-address-space machine images constructed by using library operating systems.” — unikernel.org
Unikernel

• “Unikernels are specialised, single-address-space machine images constructed by using library operating systems.” — unikernel.org
Unikernel

“Unikernels are specialised, single-address-space machine images constructed by using library operating systems.” — unikernel.org

What if all the software layers in a virtual appliance were compiled within the same safe, high-level language framework?

By Anil Madhavapeddy and David J. Scott

Unikernels: The Rise of the Virtual Library Operating System
VM, Container, and Unikernel

Virtual Machines

Containers

Unikernels
VM, Container, and Unikernel

Virtual Machines

VM
- App
- App
- Guest OS
- Hypervisor
- Hardware

Host OS

Containers

Container
- App
- App
- LibOS

LibOS

Unikernels

VM
- App
- App
- LibOS
- Hypervisor
- Hardware
VM, Container, and Unikernel

VM

- + Strong isolation/security
- - Heavy-weight

Containers

- - Weak isolation/security
  + Light-weight

Unikernels

- + Strong isolation/security
  + Light-weight

+ Strong isolation/security

- Heavy-weight

- Weak isolation/security

+ Light-weight

Virtual Machines

- Container

- Host OS

- Hypervisor

Hardware

Containers

- Unikernels

- LibOS

Hardware

Hypervisor
Unikernel Designs

• Integrating configurations into the compilation process
  • All related services, applications packed into a single application
  • Features not used are not compiled => extensive dead-code elimination

• Single-purpose libOS VMs perform only what the application needs and rely on hypervisor for isolation and resource multiplexing

• Within a unikernel VM, there’s no privilege difference between application and libOS (single address space)
  • Which mode/ring should unikernel run in?
  • What does single address space imply?

• Single (type-safe) language for everything

• Unikernel is sealed at run time and cannot dynamically add code (better security)
  • No writable and executable, no heap expansion
Unikernel Benefits

- Lightweight
  - Only what the application uses is compiled and deployed
- Faster startup time (compared to VMs)
- Better security
  - Isolates libOS’s by hypervisor
  - Small attack surface
  - Single type-safe language, page table sealing, compile-time address space randomization
- Fits many new cloud environments well
  - Serverless, microservices, NFV
MirageOS

- Implemented in OCaml, runs on Xen
  - OCaml: a high-level language that is type safe (statically checked) and uses managed memory
- Single address space (process), with one virtual CPU
- Modified language runtime’s memory management and concurrency
MirageOS Memory Layout

External I/O pages for communicating with other VMs

fast minor heap for short-lived values

larger major heap for long-lived values

Figure 2: Specialised virtual memory layout of a 64-bit Mirage unikernel running on Xen.
Figure 4: Example zero-copy write for an HTTP GET. The application writes its request into an I/O page, and the network stack segments it into fragments to write to the device ring. When a response arrives the pages are collected and the write thread notified.
Results - Boot Time

Figure 6: Boot time using an asynchronous Xen toolstack.
Results - Running DNS Server

Figure 10: DNS performance with increasing zone size.
## Results - Image Size

<table>
<thead>
<tr>
<th>Appliance</th>
<th>Binary size (MB)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard build</td>
<td>Dead code elimination</td>
<td></td>
</tr>
<tr>
<td>DNS</td>
<td>0.449</td>
<td>0.184</td>
<td></td>
</tr>
<tr>
<td>Web Server</td>
<td>0.673</td>
<td>0.172</td>
<td></td>
</tr>
<tr>
<td>OpenFlow switch</td>
<td>0.393</td>
<td>0.164</td>
<td></td>
</tr>
<tr>
<td>OpenFlow controller</td>
<td>0.392</td>
<td>0.168</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Sizes of Mirage unikernels, before and after dead-code elimination. Configuration and data are compiled directly into the unikernel.

- Typical Linux image size is ~400MB
Other Unikernels

- OSv: new OS designed to run single application
- Rumprun: running unmodified POSIX software as a unikernel
- runtime.js: libOS that runs JavaScript
- IncludeOS: libOS for running C++ code on virtual hardware
- ClickOS: A high-performance, virtualized software middlebox platform (e.g., for NFV)
- Clive: an OS designed to work in distributed and cloud environments, written in Go
- Erlang on Xen

Find more info at [http://unikernel.org/projects/](http://unikernel.org/projects/)
Discussion

• Pros and cons of unikernel? (vs. container, traditional VM, etc.)

• Do you think unikernel will be widely adopted?
Unikernels as Processes

Dan Williams, Ricardo Koller (IBM Research)
Martin Lucina (robur.io/Center for the Cultivation of Technology)
Nikhil Prakash (BITS Pilani)
Virtualization is a mixed bag

- Good for *isolation*, but...

- **Tooling** for VMs not designed for lightweight (e.g., lightVM)
- How do you debug **black-box** VMs?
- Poor VM **performance** due to vmexits
- **Deployment** issues on already-virtualized infrastructure
Why not run unikernels as processes?

• Unikernels are a **single process** anyway!

• Many benefits as a process
  • Better *performance*
  • Common tooling (gdb, perf, etc.)
  • ASLR
  • Memory sharing
  • Architecture independence

• **Isolation** by limiting process interface to host
  • 98% reduction in accessible kernel functions
Unikernel architecture

- ukm unikernel monitor
  - Userspace process
  - Uses Linux/KVM

- Setup and loading
- Exit handling
Unikernel as process architecture

- **Tender**: modified ukvm unikernel monitor
  - Userspace process
  - Uses seccomp to restrict interface

- Setup and loading
- “Exit” handling
Unikernel isolation comes from the interface

• Direct mapping between 10 hypercalls and system call/resource pairs

• 6 for I/O
  • Network: packet level
  • Storage: block level

• vs. >350 syscalls

<table>
<thead>
<tr>
<th>Hypercall</th>
<th>System Call</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>walltime</td>
<td>clock_gettime</td>
<td></td>
</tr>
<tr>
<td>puts</td>
<td>write</td>
<td>stdout</td>
</tr>
<tr>
<td>poll</td>
<td>ppoll</td>
<td>net_fd</td>
</tr>
<tr>
<td>blkinfo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blkwrite</td>
<td>pwrite64</td>
<td>blk_fd</td>
</tr>
<tr>
<td>blkread</td>
<td>pread64</td>
<td>blk_fd</td>
</tr>
<tr>
<td>netinfo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>netwrite</td>
<td>write</td>
<td>net_fd</td>
</tr>
<tr>
<td>netread</td>
<td>read</td>
<td>net_fd</td>
</tr>
<tr>
<td>halt</td>
<td>exit_group</td>
<td></td>
</tr>
</tbody>
</table>
Implementation: nabla

• Extended Solo5 unikernel ecosystem and ukvm
• Prototype supports:
  • MirageOS
  • IncludeOS
  • Rumprun
• [https://github.com/solo5/solo5](https://github.com/solo5/solo5)
Results

• Unique kernel functions accessed: normal processes and VMs have 5-6x and 2-3x more than Nabla

• Application throughput: Nabla 101% - 245% higher than ukvm

• CPU utilization: Nabla has 12% reduction over ukvm

• Startup time: ukvm takes 30-370% longer
Conclusion

• Library OS (Exokernel) is an old idea aiming to expose more hardware interface directly to applications running in user space

• Unikernels: run app+libOS as VMs on hypervisor
  • Better isolation
  • Much more lightweight
  • But need a lot of reimplementation and can’t use existing tooling

• Unikernels as processes: run app+libOS as processes on host OS, limit interface to OS for security
  • Can use existing tooling, more flexible
  • Even more lightweight

*Better fit for modern cloud environments*