A Binary Compatible Unikernel

Pierre Olivier*, Daniel Chiba*, Stefan Lankes†, Changwoo Min*, Binoy Ravidnran*

*Virginia Tech, †RWTH Aachen University

VEE’19 - 04/14/2019
Unikernels

Presentation

Full-fledged Virtual Machine

- Application
- Libraries
- OS interface used
- Linux Kernel
- Linux distribution

Hypervisor

Hardware
Unikernels
Presentation

Full-fledged Virtual Machine

Legend:
- Useful software
- Software bloat

Application
Libraries
OS interface used
Linux Kernel

Hypervisor
Hardware
Unikernels

Presentation

**Legend:**
- Useful software
- Software bloat

**Full-fledged Virtual Machine**
- Linux distribution
  - Application
  - Libraries
- OS interface used
  - Linux Kernel

**Unikernel**
- Application
- Libraries
- OS Layer

Hypervisor

Hardware
Unikernels
Presentation (2)

Unikernel: application + dependencies + thin OS compiled as a static binary running on top of a hypervisor
Unikernels
Presentation (2)

Unikernel: application + dependencies + thin OS compiled as a static binary running on top of a hypervisor

▶ single-*
  ▶ Single purpose: run 1 application
  ▶ Single process
  ▶ Single binary and single address space for application + kernel
    ▶ No user/kernel protection needed
Unikernels
Presentation (2)

Unikernel: application + dependencies + thin OS compiled as a static binary running on top of a hypervisor

- single-*
  - Single purpose: run 1 application
  - Single process
  - Single binary and single address space for application + kernel
    - No user/kernel protection needed

- Lightweight virtualization, alternative to containers
  - Security advantage: small attack surface and high isolation
Unikernels
Presentation (2)

Unikernel: application + dependencies + thin OS compiled as a static binary running on top of a hypervisor

- single-*
  - Single purpose: run 1 application
  - Single process
  - Single binary and single address space for application + kernel
    - No user/kernel protection needed

- Lightweight virtualization, alternative to containers
  - Security advantage: small attack surface and high isolation

- Per-application tailored kernel
  - LibOS/Exokernel model
Unikernels
Presentation (2)

Unikernel: application + dependencies + thin OS compiled as a static binary running on top of a hypervisor

▶ single-*
  ▶ Single purpose: run 1 application
  ▶ Single process
  ▶ Single binary and single address space for application + kernel
    ▶ No user/kernel protection needed

▶ Lightweight virtualization, alternative to containers
  ▶ Security advantage: small attack surface and high isolation

▶ Per-application tailored kernel
  ▶ LibOS/Exokernel model

▶ Reduced OS noise, increased performance
  ▶ Low system call latency
    ▶ App + kernel in ring 0, system calls are function calls
Unikernels
The Issue

- Unikernels have plenty of benefits to bring
- Unikernels have plenty of application domains
- They are very popular in academia . . .
- . . . but why (nearly) nobody uses them in the industry?
Unikernels
The Issue

- Unikernels have plenty of benefits to bring
- Unikernels have plenty of application domains
- They are very popular in academia . . .
- . . . but why (nearly) nobody uses them in the industry?

Because it is hard to port existing applications!
Unikernels
The Issue: Porting to Unikernels

Application src ➔ Unikernel & libraries src ➔ Build (compile, link, etc.) ➔ Unikernel binary ➔ Execution

- Proprietary software → source code not available
- Incompatible language
- Unsupported features
- Porting is hard, needs knowledge about both application and unikernel
- Complex build toolchains
Unikernels
The Issue: Porting to Unikernels

- Proprietary software → source code not available
Unikernels
The Issue: Porting to Unikernels

- Proprietary software → source code not available
- Incompatible language
Unikernels
The Issue: Porting to Unikernels

- Proprietary software → source code not available
- Incompatible language
- Unsupported features
Unikernels
The Issue: Porting to Unikernels

- Proprietary software → source code not available
- Incompatible language
- Unsupported features
- Porting is hard, needs knowledge about both application and unikernel
Unikernels
The Issue: Porting to Unikernels

- Proprietary software → source code not available
- Incompatible language
- Unsupported features
- Porting is hard, needs knowledge about both application and unikernel
- Complex build toolchains
Unikernels
The Issue: Porting to Unikernels

- Proprietary software → source code not available
- Incompatible language
- Unsupported features
- Porting is hard, needs knowledge about both application and unikernel
- Complex build toolchains

HermiTux Solution
- A unikernel binary-compatible with Linux
  - For x86-64 for now
Unikernels
Overview

- Linux ABI convention:
  - ELF loader convention
  - Load-time Stack layout
  - Syscalls
  - Kernel adapted from HermitCore
  - Complete/partial support for 80+
syscalls

- How to maintain unikernel benefits without access to the application sources?
  - Fast system calls and modularity
Unikernels Overview

- Linux ABI convention:
  - ELF loader convention
  - Load-time Stack layout

- Complete/partial support for 80+ syscalls

- How to maintain unikernel benefits without access to the application sources?

- Fast system calls and modularity
Unikernels
Overview

- Linux ABI convention:
  - ELF loader convention
  - Load-time Stack layout
  - Syscalls

How to maintain unikernel benefits without access to the application sources?
- Fast system calls
- Modularity
Unikernels
Overview

- Hypervisor: uHyve
- Host: Linux kernel

- Linux ABI convention:
  - ELF loader convention
  - Load-time Stack layout
  - Syscalls

- Kernel adapted from HermitCore

How to maintain unikernel benefits without access to the application sources?

Fast system calls and modularity
Unikernels
Overview

- Host: Linux kernel
- Hypervisor: uHyve
- Single-address space VM

- Linux ABI convention:
  - ELF loader convention
  - Load-time Stack layout
  - Syscalls

- Kernel adapted from HermitCore

Host: Linux kernel  \( KVM \)
Hypervisor: uHyve

Single-address space VM
Unikernels
Overview

- **Linux ABI convention:**
  - ELF loader convention
  - Load-time Stack layout
  - Syscalls

- Kernel adapted from HermitCore

---

Host: Linux kernel
Hypervisor: uHyve
Native Linux App.
Unikernels Overview

- Host: Linux kernel
- Hypervisor: uHyve
- Native Linux App.
- Load
- Guest

- Linux ABI convention:
  - ELF loader convention
  - Load-time Stack layout
  - Syscalls

- Kernel adapted from HermitCore
Unikernels
Overview

- Host: Linux kernel
- Hypervisor: uHyve
- Native Linux App.
- Load Init. stack and jump to entry point
- ▶ Linux ABI convention:
  - ELF loader convention
  - Load-time Stack layout
  - Syscalls
- ▶ Kernel adapted from HermitCore

Diagram:
- Hermitux kernel
- Native Linux App.
- Hypervisor: uHyve
- Host: Linux kernel

Note: Load Time Stack Layout and Kernel Adapted from HermitCore are features specific to Unikernels, providing benefits such as fast system calls and modularity.
Unikernels Overview

- **Linux ABI convention:**
  - ELF loader convention
  - Load-time Stack layout
  - Syscalls

- Kernel adapted from HermitCore
- Complete/partial support for 80+ syscalls
Unikernels
Overview

- Linux ABI convention:
  - ELF loader convention
  - Load-time Stack layout
  - Syscalls
- Kernel adapted from HermitCore
- Complete/partial support for 80+ syscalls

Init. stack and jump to entry point

Host: Linux kernel
Hypervisor: uHyve
Native Linux App.
Load
Syscall
Hermitux
kernel
handler
Unikernels
Overview

Host: Linux kernel
Hypervisor: uHyve
Native
Linux
App.
Load
Init. stack and jump to entry point

▶ Linux ABI convention:
  ▶ ELF loader convention
  ▶ Load-time Stack layout
  ▶ Syscalls

▶ Kernel adapted from HermitCore
▶ Complete/partial support for 80+ syscalls

▶ How to maintain unikernel benefits without access to the application sources?
  ▶ Fast system calls and modularity
Unikernels
Fast Syscalls with Libc Substitution

- HermiTux’s syscall handler is invoked by the syscall instruction
  - Reintroduce high latency for system calls due to the world switch
Unikernels
Fast Syscalls with Libc Substitution

- HermiTux’s syscall handler is invoked by the syscall instruction
  - Reintroduce high latency for system calls due to the world switch
- For dynamically compiled programs:
  - At runtime load a unikernel-aware Libc
    - Making for system calls (fast) function calls directly into the kernel
  - Automatically transformed version of Musl Libc with Coccinelle
Unikernels
Fast Syscalls with Binary Rewriting

- What about static binaries?
- (Statically) binary-rewrite syscall instructions to direct jumps to the syscall implementation
  - Problem: syscall is 2 bytes long and any call/jmp instruction will be larger

```plaintext
Original code
mov $0, %rax (read)
syscall (2 bytes)
mov $2, %esi (5 bytes)
mov $3, %rdi
...

Rewritten code
mov %r10, %rcx
callq 0x200457 (read)
mov $2, %esi
jmp 0x400aac
```

```
Snippet
... 
jmp 0x200042 (5 bytes) 
nop (1 byte) 
nop (1 byte) 
mov $3, %rdi (5 bytes) 
...
```

```
Syscall binary rewriting
... 
mov $0, %rax (read) 
syscall (2 bytes) 
mov $2, %esi (5 bytes) 
mov $3, %rdi 
...
```
Unikernels
Fast Syscalls with Binary Rewriting

- What about static binaries?
- (Statically) binary-rewrite syscall instructions to direct jumps to the syscall implementation
  - Problem: syscall is 2 bytes long and any call/jmp instruction will be larger

![Execution time comparison chart]

- Left: Linux (native)
- Middle: Hermitux Handler
- Right: Hermitux Rewrite
- Bottom: Hermitux Lib. Substitution

Legend:
- null (getppid)
- read
- write

- [8/11]
Unikernels
System-call-based Modularity

- System-call based modularity
  - Compile a kernel with support for only the necessary system calls
  - How to identify syscall needed without access to the sources?
    - **Use binary analysis** to find out what is the value in %rax for each syscall invocation

<table>
<thead>
<tr>
<th>Program</th>
<th>Number of system calls</th>
<th>Kernel .text size reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimal</td>
<td>5</td>
<td>21.87 %</td>
</tr>
<tr>
<td>Hello world</td>
<td>10</td>
<td>19.84 %</td>
</tr>
<tr>
<td>PARSEC Blackscholes</td>
<td>15</td>
<td>17.05 %</td>
</tr>
<tr>
<td>Postmark</td>
<td>26</td>
<td>14.36 %</td>
</tr>
<tr>
<td>Sqlite</td>
<td>31</td>
<td>11.34 %</td>
</tr>
<tr>
<td>Full syscalls support</td>
<td>64</td>
<td>00.00 %</td>
</tr>
</tbody>
</table>
Unikernels Evaluation

- Image 650x smaller, boot time 780x faster, RAM usage 9x lower than a Linux VM!
Conclusion

- Porting to unikernels is hard
  - Hinders their adoption in the industry

- HermiTux provides binary-compatibility with Linux applications

- HermiTux maintains unikernel benefits:
  - Fast boot times, small footprints
  - Various techniques to get fast system calls and modularity

It’s open source, try it out!
https://ssrg-vt.github.io/hermitux/