ToroV, a kernel in user-space, or sort of

www.torokernel.io

Matias Vara Larsen
matiasevara@torokernel.io
Who am I?

- I enjoy working on operating systems and playing with virtualization
- I worked at Citrix, Tttech, Huawei ...

- https://github.com/MatiasVa ra
Monolithic Application

<table>
<thead>
<tr>
<th>Logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
</tr>
<tr>
<td>Catalog</td>
</tr>
</tbody>
</table>

Decomposed Application into Services

<table>
<thead>
<tr>
<th>Logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
</tr>
<tr>
<td>Catalog</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Microservice #0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microservice #1</td>
</tr>
<tr>
<td>Microservice #2</td>
</tr>
</tbody>
</table>

e.g., Amazon website
Monolithic Application

- Logging
- Order
- Catalog

e.g., Amazon website

Decomposed Application into Services

- Logging
- Order
- Catalog

Deployed as server-less

The Cloud provider deals with deployment, i.e., performance and isolation

- Microservice #0
- Microservice #1
- Microservice #2
How server-less applications can be deployed?

- By using containers, i.e., software-based virtualization
- By using VMs, i.e., hardware-based virtualization
  - General Purpose OS
  - unikernel

- These mechanisms are chosen based on a trade-off between performance and security
Guests consume a lot of resources, e.g., memory, CPU, on-disk image.

A different set of drivers may be needed depending on the Cloud provider device model.

VMs take long time to be up and running.

These issues end up by limiting the number of instances that a server can host and the cost of maintenance.

, e.g., QEMU, Firecraker

Device Model

These issues end up by limiting the number of instances that a server can host and the cost of maintenance.
Virtual Machine #0

- Hardware
- Memory
- CPUs

Operating System

- Kernel
- Ring 0
- Ring 3

Microservice #0

syscalls

Device Model

VMExit

Linux Host

Kernel

KVM

Device Model

,e.g., QEMU, Firecraker
Virtual Machine #0

- Microservices #0
  - e.g., Toro, Osv, MirageOS, Unikraft, NanoVMs

Kernel

Device Model

Linux/KVM

Hardware

Memory

CPUs

Ring 3
Ring 0
Ring -1

Virtual Machine #1

- Microservices #1

Unikernel

Device Model

Linux/KVM

[2] “Unikernels: library operating systems for the cloud”, Madhavapeddy et al., 2013
Could we simply offload the guest’s kernel in the host?

[2] “Unikernels: library operating systems for the cloud”, Madhavapeddy et al., 2013
What is a kernel in user’s space?

- It is a way to offload guest’s kernel in the host
- It allows a guest application to run without a device model thus reducing the attack surface of the host
- It prevents the host to be exposed by emulating kernel services.
- Approaches:
  - User-Mode Linux
  - Gvisor
  - ToroV
Virtual Machine #0

- Hardware
- Memory
- CPUs

Operating System
- Lightweight Kernel
- Ring 0
- Ring 3

Microservice #0

VMExit

KVM

Kernel at host’s user-space
- Filesystem
- Networking

Reduced syscalls

Linux Host

syscalls
Virtual Machine #0

Operating System

Microservice #0

Hardware

Memory

CPUs

Lightweight Kernel

Ring 0

Ring 3

Ring -1

VMExit

Reduced syscalls

Kernel at host’s user-space

Filesystem

Networking

Linux Host

KVM

Kernel

Hardware

Memory

CPUs
Virtual Machine #0

- Hardware
- Memory
- CPUs

Operating System
- Lightweight Kernel
- Ring 0
- Ring 3

Microservice #0

syscalls

VMExit

Kernel at host’s user-space
- Filesystem
- Networking

Reduced syscalls

Linux Host

KVM

Hardware

Memory

CPUs
Virtual Machine #0

Operating System

Lightweight Kernel (Sentry)

Microservice #0

VMExit (halt)

syscalls

Ring 3

Ring 0

Ring -1

Linux Host

KVM

Gvisor

Sentry

Gopher

Reduced syscalls

IO syscalls

https://gvisor.dev/docs/
What is ToroV?

- It is a minimalist kernel in user space in which syscalls from the guest are forwarded to the host.
- It allows the user to configure what syscalls are allowed per application.
- It provides a modified stdlib that the user’s application must be compiled within.
- It exposes a POSIX API based on hypercalls to the guest.
- It runs as a containerized process to reduce host attack surface.
- It allows the user to debug guest’s applications by simply using a GDB.
Reduced PID Namespace

Hardware

Memory

CPUs

Ring 0

Virtual Machine #0

Microservice #0

Function call

Glibc / fpcrtl

Hypercall (Linux 64 bits ABI)

Ring -1

KVM

ToroV

Reduced PID Namespace

Chroot

VMM

HyperCalls

Configurable syscalls

KVM

Hardware

Memory

CPUs

{  
  "Binary": "HelloWorld.bin",
  "MountPoint": "/./mountpoint",
  "Newpid": "False",
  "Debug":
  
  
  
  
  
  "Allowed": "False",
  "Port": "1234"
  
  
  

  "Hypercalls": [
  
  
  
  

  
  "Name": "Ioctl",
  "Allowed": "True"
  
  
  

  
  "Name": "Write",
  "Allowed": "True"
  
  
  

}
Compilation Process

Glibc / fpcrnl

Includes

Microservice

MyMicroservice.elf

64 bits long mode in Ring0

VM

Launches

vmm

MyMicroservice.json

MyMicroservice.bin

MyMicroservice.dbg
Memory Layout

Virtual Machine

```
<table>
<thead>
<tr>
<th>PDT</th>
<th>App's binary</th>
<th>Heap</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0x2000</td>
<td>0x400200</td>
<td>0x600000</td>
</tr>
<tr>
<td>0x400200</td>
<td>0x600000</td>
<td>0x800000</td>
<td></td>
</tr>
</tbody>
</table>
```

Copy to/from user

VMM (host’s containerized process memory)

MMAP
Memory Layout

Virtual Machine

PDT

App’s binary

Heap

Stack

0x0

0x2000

0x400200

0x600000

0x800000

Copy to/from user

VMM (host’s containerized process memory)

MMAI
Memory Layout

- **Virtual Machine**
  - PDT
  - App's binary
  - Heap
  - Stack

- **VMM (host's containerized process memory)**

- **Copy to/from user**

- **MMAP**
HelloWorld example

```
strace -f ../../src/vmm/vmm helloworld.json

ioctl(5, KVM_CREATE_VCPU, 0) = 6
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_SHARED, 6, 0) = 0x7f8908125000
ioctl(6, KVM_GET_SREGS, {cs={base=0xffff0000, limit=65535, selector=61440, type=11, present=1, dpl=0, db=0, s=1, l=0, g=0, avl=0}, ...}) = 0
ioctl(6, KVM_SET_SREGS, {cs={base=0, limit=4294967295, selector=8, type=11, present=1, dpl=0, db=0, s=1, l=1, g=1, avl=0}, ...}) = 0
ioctl(6, KVM_SET_REGS, {rax=0, ..., rsp=0x7ffe28, rbp=0x7ffe38, ..., rip=0x4005c5, rflags=0x2}) = 0
clone(child_stack=0x513a30, flags=CLONE_VM|CLONE_FILES|CLONE_NEWNS|SIGCHLD) = 16528
wait4(16528, strace: Process 16528 attached
<unfinished ...>
[pid 16528] chroot("./mountpoint") = -1 ENOENT (No such file or directory)
[pid 16528] chdir("/") = 0
[pid 16528] ioctl(6, KVM_RUN, 0) = 0
[pid 16528] ioctl(6, KVM_GET_REGS, {rax=0x1, ..., rsp=0x7ffe28, rbp=0x7ffe38, ..., rip=0x4005c5, rflags=0x2}) = 0
[pid 16528] write(2, "Hello World, I am ToroV\n", 25Hello World, I am ToroV!
) = 25
[pid 16528] ioctl(6, KVM_SET_REGS, {rax=0x19, ..., rsp=0x7ffe28, rbp=0x7ffe38, ..., rip=0x4005c5, rflags=0x2}) = 0
[pid 16528] ioctl(6, KVM_RUN, 0) = 0
```
HelloWorld example

- ~ 1.5 MB of memory (top)
- ~ 7 ms (median)
- Write() syscall ~ 0.10 ms ~ x10 slower (0.012 ms)
Future Work

- Work on Glibc and other languages like Go or Rust
- Research about how ToroV compares with seccomp
- Enable the use of binaries without recompilation by replacing “syscall” opcode by “out” opcode thus starting the program as-is
- Port the whole project to Rust
- Replace the current syscall mechanism for an asynchronous mechanism, e.g., virtio device for syscalls.
- Enable that different components handles different syscalls, e.g., SOA
Resources

• GitHub repository at https://github.com/torokernel/torov
• “Debugging applications that run as VM by using GDB”, https://youtu.be/QC8pYtMOWe4
• “Using ToroV to isolate an app by using virtualization and containerization technologies”, https://youtu.be/YDpE8jlwVPA
• “Simple Echo server by relying on POSIX hypercalls”, https://youtu.be/aJpcmZhDqMw
Thanks!

www.torokernel.io