# URVirt: A U-mode trap-and-emulate hypervisor

dram

# Virtualization

# Classically virtualizable

The RISC-V instruction set is *classically virtualizable.*

*[The] technical decision to orthogonalize the RISC-V user ISA and privileged architecture [...] simplifies the implementation of full virtualization. Exposing privileged features to unprivileged software adds complexity to hardware-assisted virtualization, and can make classical virtualization impossible.* (Waterman, 2016)

# Classically virtualizable





How does classical virtualization work?

# Classically virtualizable

How does classical virtualization work?

*As long as all privileged instructions generate traps when executed in user mode, it also suffices to support classical virtualization, in which guest OSes systems run in unprivileged mode.* (Waterman, 2016)

# Trap-and-emulate hypervisors

```
csrw satp, a3
```

# Trap-and-emulate hypervisors

```
csrw satp, a3
```

- In privileged mode: Instruction is executed on hardware
- In unprivileged mode: Generates an *illegal instruction* exception

# Trap-and-emulate hypervisors

```
csrw satp, a3
```

- A trap-and-emulate hypervisor, running in privileged mode:
  - *Traps* illegal instruction exception
  - *Emulates* the `csrrw`
    - E.g. sets up *shadow page tables*
  - Advances `sepc` and resumes guest OS execution
- Guest OS running in unprivileged mode:
  - `csrrw` handled transparently

# Trap-and-emulate hypervisors

```
csrw satp, a3
```

- A trap-and-emulate hypervisor, running in privileged mode:
  - *Traps* illegal instruction exception
  - *Emulates* the `csrrw`
    - E.g. sets up *shadow page tables*
  - Advances `sepc` and resumes guest OS execution
- Guest OS running in unprivileged mode:
  - `csrrw` handled transparently
- What if: The trap-and-emulate hypervisor runs in unprivileged mode too?

# URVirt

# U-mode trap-and-emulate hypervisor

– Running as a normal user program under Linux

# U-mode trap-and-emulate hypervisor

- Running as a normal user program under Linux
- Privileged instructions still trap in Linux
  - Generates `SIGILL`
- Handle and emulate instruction without privileges?

# What can we trap?



- All privileged instructions (`SIGILL`)
- `ecall` (`SIGSYS`)
- Page faults (`SIGSEGV`)

# What can we emulate?

- Enough privileged instructions to run something
- Privilege modes
- SBI calls
- Timers
- A subset of virtual memory facilities

# Demo

# Running rCore-Tutorial

```
[dram@sayori:~/urvirt]$ sudo ./urvirt-loader urvirt-stub.bin os.bin fs.img 2>log.txt
[kernel] Hello, world!
[ INFO] last 1020 Physical Frames.
[ INFO] .text [0x80200000, 0x80210000)
[ INFO] .rodata [0x80210000, 0x80215000)
[ INFO] .data [0x80215000, 0x803f3000)
[ INFO] .bss [0x803f3000, 0x80604000)
[ INFO] mapping .text section
[ INFO] mapping .rodata section
[ INFO] mapping .data section
[ INFO] mapping .bss section
[ INFO] mapping physical memory
[ INFO] remap_test passed!
after initproc!
/**** APPS ****
exit
fantastic_text
forktest
forktest2
forktest_simple
forktree
hello_world
initproc
matrix
```

Figure 1: rCore-Tutorial[1] running under URVirt

[1] https://github.com/LearningOS/rCore-Tutorial-2021Autumn

# Features

- Miscellaneous SBI calls: `shutdown`, `console_putchar`, `console_getchar` ✓
- S and U Privilege modes ✓
- Timer interrupts (SBI call `set_timers`) ✓
- Virtual memory (Sv39, partial support)
- A very simple block device ✓

# Performance

- Run performance test program in URVirt/rCore-Tutorial and qemu-system-riscv64/rCore-Tutorial
- Four benchmarks:
  - `pure`: Loop through some in-register computation
  - `memory`: Write things around in an 1 MiB memory region
  - `badsyscall`: Write to invalid fd, context switch only
  - `goodsyscall`: Write to stdout, context switch and SBI calls
- URVirt prints some counters
  - `tlb`: Map in pages on SIGSEGV
  - `priv`: Emulated privileged instruction
  - The rest should be self-descriptive
- Timers *disabled*

# Performance

| | Benchmark time (secs) | | | URVirt stats | | | | |
|---|---|---|---|---|---|---|---|---|
| | urvirt | qemu | q/u | tlb | priv | sret | uecall | secall |
| `pure` | 0.922 | 1.928 | 2.09 | 20 | 16 | 1 | 1 | 0 |
| `memory` | 1.433 | 4.365 | 3.05 | 564 | 32 | 2 | 2 | 0 |
| `badsyscall` | 3.045 | 1.296 | 0.43 | 19021 | 16016 | 1001 | 1001 | 0 |
| `goodsyscall` | 4.900 | 1.175 | 0.24 | 32021 | 16016 | 1001 | 1001 | 9000 |

# Performance

| | Benchmark time (secs) | | | URVirt stats | | | | |
|---|---|---|---|---|---|---|---|---|
| | urvirt | qemu | q/u | tlb | priv | sret | uecall | secall |
| `pure` | 0.922 | 1.928 | 2.09 | 20 | 16 | 1 | 1 | 0 |
| `memory` | 1.433 | 4.365 | 3.05 | 564 | 32 | 2 | 2 | 0 |
| `badsyscall` | 3.045 | 1.296 | 0.43 | 19021 | 16016 | 1001 | 1001 | 0 |
| `goodsyscall` | 4.900 | 1.175 | 0.24 | 32021 | 16016 | 1001 | 1001 | 9000 |

Speed:

Trap-and-emulate $<$ QEMU TCG $<$ Native

# Internals

# Components

- Loader: Normal Linux program
- Stub: Linux program, *without* heap or libc
  - Runs as the *signal handler* for the running kernel
- The guest kernel
- The filesystem image

# Initialization



In loader:

- Opens relevant files
- Maps stub into memory
- Jump to stub

# Initialization

In loader:

- Opens relevant files
- Maps stub into memory
- Jump to stub

In stub:

- Signal handlers
- Memory map
- Jump to kernel

# Privileged instructions / ecall





- Privileged instructions generate SIGILL
- Use Seccomp BPF filter to generate SIGSYS on ecall

```
void handler(int sig, siginfo_t *info, ucontext_t* ucontext);
```

- ucontext_t: All registers and PC at time of trapping instruction
- Emulate instruction / SBI call, modify ucontext, return

# Virtual memory

- (Emulated) `satp.MODE` is set to ‘bare’ at startup
- RAM is a memfd[2], initially mapped at `0x8000_0000`

[2] https://man7.org/linux/man-pages/man2/memfd_create.2.html

# Virtual memory

- (Emulated) `satp.MODE` is set to 'bare' at startup
- RAM is a memfd[2], initially mapped at `0x8000_0000`

- On every write to `satp`, `sret`, `sfence.vma`, unmap everything
- Next access generates a `SIGSEGV`, with virtual address
- Read from the RAM memfd, decode PTE to find PPN

[2] https://man7.org/linux/man-pages/man2/memfd_create.2.html

# Virtual memory (RAM case)



– If PPN is in RAM, map the corresponding ‘physical’ page at virtual page

```
mmap(
    virtual_page_start /* vpn << 12 */, PAGE_SIZE /* 4096 */,
    flags /* translated from PTE */,
    MAP_SHARED | MAP_FIXED_NOREPLACE,
    RAM_FD, physical_page_start /* ppn << 12 */ - RAM_START
);
```

# Virtual memory (MMIO case)

- If PPN is an MMIO register, decode the faulting instruction and emulate the load/store
- Loads: `lb`, `lbu`, `lh`, `lhu`, `lw`, `lwu`, `ld`, `c.lw`, `c.ld`, `c.lwsp`, `c.ldsp`
- Stores: `sb`, `sh`, `sw`, `sd`, `c.sw`, `c.sd`, `c.swsp`, `c.sdsp`

# Various exception cases

Trap to guest S-mode:

- U-mode ecall
- `SIGSEGV`, but PTE is not valid / has wrong permissions
- Illegal instruction in U-mode

# Comparisons

# Comparisons



| | Runs in | Emulates | Implementation |
|---|---|---|---|
| URVirt | U-mode | S-mode | Trap-and-emulate |
| RVirt[3] | S-mode | S-mode | Trap-and-emulate |
| QEMU (system)[4] | U-mode | M-mode | Binary translation + Soft MMU |
| QEMU (user) | U-mode | U-mode | Binary translation + Host MMU |
| KVM[5] | (H)S-mode | S-mode | RISC-V Hypervisor extension |
| UML[6] | U-mode | U-mode | Trap-and-emulate |

[3] `https://github.com/mit-pdos/RVirt`

[4] `https://www.qemu.org`

[5] `https://github.com/kvm-riscv`, hardware implementation of H-ext not available at time of writing

[6] User-mode Linux, not yet available on RISC-V at time of writing

# Limitations and alternatives

- Full Sv39 memory space impossible (due to Linux restrictions)
- Trap-and-emulate speed in userspace is abysmal
- Goal of URVirt: An experimentation in virtualization, not a practical hypervisor
- Possible alternatives
  - Using other Linux facilities: `ptrace`/`userfaultfd`/`seccomp_unotify`...

# References

# References



Waterman, A. (2016). *Design of the RISC-V Instruction Set Architecture.* PhD thesis, EECS Department, University of California, Berkeley.

# Thank you

Check it out on GitHub: `https://github.com/dramforever/urvirt`