CERTIFIED INTERRUPTIBLE OS KERNALS AND DEVICE DRIVERS

Zhong Shao
Joint work with Hao Chen, Newman Wu, Joshua Lockerman, and Ronghui Gu
Yale University

Formal Verification of OS Kernel

- seL4
- CertiKOS
- Verve

Application
- Trap
- Virtualization
- Process Mgmt.
  - Kernel
  - IPC
- Thread
- Memory Mgmt.
- CPU
- Memory
Formal Verification of OS Kernel

Applications

Trap

Virtualization

Process Mgmt.

IPC

Thread

Memory Mgmt.

CPU

Memory

LAPIC

IOAPIC

AHCI / SATA (disk)

USB

NIC

Serial

Keyboard

VGA (video)

Formal Verification of OS Kernel

Applications

Trap

Virtualization

Process Mgmt.

IPC

Thread

Memory Mgmt.

VM

Monitor

Virtual Dev. 1

... Virtual Dev. N

Applications

CPU

Memory

LAPIC

IOAPIC

AHCI / SATA (disk)

USB

NIC

Serial

Keyboard

VGA (video)
Device Drivers in Mainstream OS

- 70% of Linux 2.4.1 kernel are device drivers.
- 70% of Windows crash are caused by third-party driver code.

mCertikOS Overview [POPL’15]

- Single-core version of CertiKOS.
- 3k LOC, can boot Linux as guest.
- Aggressive use of abstraction over deep specification (37 layers).
Main Challenge

Every fine-grained processor step could be interrupted.

Other Challenges

- Interrupt hardware can be dynamically configured.
- Devices and CPU run in parallel.
- Device drivers are written in both C and assembly.
- The correctness results of different components should be linked formally.
Our Contributions [PLDI’16]

The *first* formally verified *interruptible* OS kernel with device *drivers*.

- New techniques for certifying abstraction layers with multiple *logical CPUs* and devices.
- New techniques for building formal *certified device hierarchies*.
- An abstraction-layer-based approach for reasoning about *interrupts*.
- **Case study**: interruptible mCertiKOS with device drivers.
Linux Kernel Map

Kernel components are sorted into different stacks of abstraction layers based on their underlying hardware device.

New Machine Model

...
Our Contributions

- New techniques for certifying abstraction layers with multiple logical CPUs and devices.
- New techniques for building formal certified device hierarchies.
- An abstraction-layer-based approach for reasoning about interrupts.
- Case study: interruptible mCertiKOS with device drivers.

Hardware Device Model

- Devices are modeled as transition systems parameterized by all possible lists of external events.
- Example external events:
  - `Recv` (s: list char)
  - `KeyPressed` (c: Z)
- State: observable registers.
- Transition:
  - environmental transition: $\delta_{\text{ENV}}$
  - I/O transition: $\delta_{\text{CPU}}$
Raw Device Object

- Local log for the list of observed external events.
- Multiple local logs to handle disjoint set of external events asynchronously.
- Read/Write instructions: IN/OUT, memory mapped I/O, etc.

Extended Device Object

Driver as a logical device.
Recap: Machine Model

... 

Our Contributions

- New techniques for certifying abstraction layers with multiple logical CPUs and devices.
- New techniques for building formal certified device hierarchies.
- An abstraction-layer-based approach for reasoning about interrupts.
- Case study: interruptible mCertiKOS with device drivers.
Interrupt Models

New HW Interrupt Model
Semantics of `intr_disable`

- Scans external events.
- Recursively performs the environmental transition.
- Synchronizes unhandled interrupts.

\[
\begin{align*}
\text{DISABLENOINTR: Disable with no unhandled interrupt} \\
(s, l_i) & = \text{next}(e^{in}, l_i) \quad s_{\text{irq}} = \text{assert}(s, e) \\
& \quad \text{env}.s_{\text{irq}} = \text{false} \\
& \quad s' = s[\text{Flag} \leftarrow 0] \\
& \quad \text{intr\_disable}(s, l_i, e^{in}) = (s', l_i)
\end{align*}
\]

\[
\begin{align*}
\text{DISABLEINTR: Disable with unhandled interrupts} \\
(s, l_i) & = \text{next}(e^{in}, l_i) \quad s' = \text{assert}(s, e) \\
& \quad s'.s_{\text{irq}} = \text{true} \quad (s', l_i) = \text{intr\_handler}(s', l_i, e^{in}) \\
& \quad (s^{in}, l^{in}) = \text{intr\_disable}(s', l_i, e^{in}) \\
& \quad \text{intr\_disable}(s, l_i, e^{in}) = (s^{in}, l^{in})
\end{align*}
\]

Semantics of `intr_enable`

- Recursively discharges pending interrupts.
- Delayed interrupts that occur while the interrupt is disabled.

\[
\begin{align*}
\text{ENABLENOINTR: Enable with no pending interrupt} \\
& \quad s_{\text{irq}} = \text{false} \quad s' = s[\text{Flag} \leftarrow 1] \quad \text{intr\_enable}(s, l_i, e^{in}) = (s', l_i)
\end{align*}
\]

\[
\begin{align*}
\text{ENABLEINTR: Enable with pending interrupts} \\
& \quad s_{\text{irq}} = \text{true} \quad (s', l_i) = \text{intr\_handler}(s, l_i, e^{in}) \\
& \quad (s^{in}, l^{in}) = \text{intr\_enable}(s', l_i, e^{in}) \\
& \quad \text{intr\_enable}(s, l_i, e^{in}) = (s^{in}, l^{in})
\end{align*}
\]
Refinement btw. The HW & Abstract Interrupt Model

Kernel/User IC Dev
- Event arrival
- intr_handler
- Primitive called

Interrupt Model

Keep/ICDevEvent arrivalintr_handlerPrimitive called

Our Approach
- The driver code of each device runs on its own "logical CPU", operates its own internal states.
- Interruptible code can be naturally reasoned on top of the abstract interrupt model.
Our Contributions

- New techniques for certifying abstraction layers with multiple logical CPUs and devices.
- New techniques for building formal certified device hierarchies.
- An abstraction-layer-based approach for reasoning about interrupts.
- Case study: interruptible mCertiKOS with device drivers.

Interruptible mCertiKOS with Drivers
Case Study: Modeling HW Devices

- Serial Port, I/O APIC, Local APIC, CPU interrupt handling.

Case Study: Serial Device

- States: see figure
- Transitions: serial_trans_env + serial_trans_IO
- Read/Write primitives: serial_read / serial_write
Serial Interrupt Handler

```c
void serial_intr () {
  unsigned int hasMore;
  int t = 0;
  hasMore = serial_getc ();
  while (hasMore && t < CONSOLE_BUFFER_SIZE) {
    hasMore = serial_getc ();
  }
  if (hasMore) {
    t++;
    unsigned int serial_getc () {
      unsigned int rv = 0;
      unsigned int rx;
      if (serial_exists()) {
        if (serial_read(COM1 + COM_LSR, BIT1) % 2 == 1) {
          rx = serial_read(COM1 + COM_RX, M_ALL);
          cons_buf_write(rx);
          rv = 1;
        }
      }
      return rv;
    }
    return rv;
  }
```

Serial Driver

```c
void serial_puts(char * s, int len) {
  int i = 0;
  while (i < len && s[i] != 0) {
    serial_intr_disable ();
    serial_putchar (s[i]);
    serial_intr_enable ();
    i++;
  }
}
```

```c
void serial_putchar (unsigned int c) {
  unsigned int lsr = 0, i;
  if (serial_exists()) {
    for (i = 0; !lsr && i < 12800; i++) {
      lsr = serial_read(0x3FD) & 0x20;
      delay();
    }
    serial_write (0x3F8, c);
  }
  ...
What We Have Proved

- Total functional correctness.
- Safety.
- Contextual refinement between the lowest and the top level abstract machine:
  \[ \forall P, \left[ [K \triangleright P]_{\text{x86}} \subseteq [P]_{\text{mCertiKOS}} \right] \]
- Data invariants:
  - Console’s circular buffer is always well-formed.
  - Interrupt controller states are always consistent.
- The framework also ensures that:
  - No code injection attacks, buffer overflow, integer overflow, null pointer access, etc.

Size of TCB and Spec/Proof

- In the TCB
  - X86 hardware model
  - Hardware device/interrupt model (510 LOC)
  - System call specification (126 LOC)
  - Bootloader
  - Coq proof checker
  - Pretty-printing phase of the CompCert compiler

- Rest of the spec/proof (about 20k LOC)
  - Intermediate and auxiliary specifications and definitions
  - Coq proof scripts
Conclusion

- Compositional framework for building certified interruptible kernel with device drivers.
  - Certified abstraction layers with multiple logical CPUs.
  - An abstraction-layer-based approach for expressing interrupts.
- The first formally verified interruptible OS kernel with device drivers.
- Extensions:
  - Other drivers
  - Concurrency
  - Larger kernel