TOWARD COMPOSITIONAL VERIFICATION OF INTERRUPTIBLE OS KERNELS AND DEVICE DRIVERS

Xiongnan (Newman) Wu Joint work with Hao Chen, Zhong Shao, Joshua Lockerman, and Ronghui Gu

Yale University



June 16, 2016









Formal Verification of OS Kernel



Formal Verification of OS Kernel



Formal Verification of OS Kernel



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

The first formally verified interruptible OS kernel with device drivers.



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.

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: δ^{env}
 - **I**/O transition: δ^{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.

DISABLENOINTR: Disable with no unhandled interrupt

$$\frac{(e, \ell'_i) = \mathsf{next}(\ell^{env}, \ell_i) \qquad s_{\mathsf{tmp}} = \delta^{\mathsf{env}}(s, e)}{s_{\mathsf{tmp}}.irq = \mathsf{false} \qquad s' = s[\mathsf{iFlag} \leftarrow 0]}$$
$$\frac{\mathsf{intr_disable}(s, \ell_i, \ell^{env}) = (s', \ell_i)}{\mathsf{intr_disable}(s, \ell_i, \ell^{env}) = (s', \ell_i)}$$

DISABLEINTR: Disable with unhandled interrupts

$$\begin{array}{c} (e,\ell_i') = \mathsf{next}(\ell^{env},\ell_i) & s' = \delta^{\mathsf{env}}(s,e) \\ s'.irq = \mathsf{true} & (s'',\ell_i'') = \mathsf{intr_handler}(s',\ell_i',\ell^{env}) \\ & (s''',\ell_i''') = \mathsf{intr_disable}(s'',\ell_i'',\ell^{env}) \\ \hline & \mathsf{intr_disable}(s,\ell_i,\ell^{env}) = (s''',\ell_i''') \end{array}$$

Semantics of intr_enable

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

ENABLENOINTR: Enable with no pending interrupt

$$\frac{s.irq = \text{false} \quad s' = s[\text{iFlag} \leftarrow 1]}{\text{intr_enable}(s, \ell_i, \ell^{env}) = (s', \ell_i)}$$

ENABLEINTR: Enable with pending interrupts

$$s.irq = true (s', \ell'_i) = intr_handler(s, \ell_i, \ell^{env}) (s'', \ell''_i) = intr_enable(s', \ell'_i, \ell^{env}) intr_enable(s, \ell_i, \ell^{env}) = (s'', \ell''_i)$$

Refinement btw. The HW & Abstract Interrupt Model

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

```
1 void serial intr() {
    unsigned int hasMore;
2
    int t = 0:
3
    hasMore = serial getc ();
4
    while (hasMore && t < CONSOLE_BUFFER_SIZE) {</pre>
5
      hasMore = serial getc ();
6
      t++;
7
                             1 unsigned int serial_getc () {
    }
8
                                 unsigned int rv = 0;
                             2
9 }
                                 unsigned int rx;
                             3
                                 if (serial exists()) {
                             4
                                   if (serial read(COM1 + COM LSR, BIT1) % 2 == 1)
                             5
                                   {
                                     rx = serial read(COM1 + COM RX, M ALL);
                             6
                                     cons buf write(rx);
                             7
                                     rv = 1;
                             8
                                   }
                             9
                                 }
                            10
                                 return rv;
                            11
                            12 }
```

Serial Driver

2

3

4

5

6

7

8

9

}

. . .

```
void serial_puts(char * s, int len) {
                                   1
                                       int i = 0:
                                   2
                                       while (i < len && s[i] != 0) {</pre>
                                   3
                                         serial_intr_disable ();
                                   4
                                         serial_putc (s[i]);
                                   5
                                   6
                                         serial_intr_enable ();
                                         i++;
                                   7
                                     }
                                   8
                                   9 }
1 void serial_putc (unsigned int c) {
   unsigned int lsr = 0, i;
   if ( serial exists() ){
      for (i = 0; !lsr && i < 12800; i++) {</pre>
       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:
 ∀P, [[K ⋈ P]]_{x86} ⊑ [[P]]_{mCertiKOS}
- 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

Linux kernel map						
functionalities	human interface	system	processing	memory	storage	networking
User space interfaces reder calls and opsern files	HI char devices	Interfaces core Spare Cal Interface Incurrently Spare Cal Interface Incurrently Spare Spare News Incurrently Spare Spare Spare Call Spare	Interest Processes National Control Non-Interest Sparad Control Sp5, Sp5, Sp5, Sp5, Sp5, Sp5, Sp5, Sp5,	memory access sys_bh	files & directories	sockets access vg.,soletzil vg.,soletzil sol,soletzil vg.,soletzil
virtual	sourie soli pot my pot todi pot my pot todi partistan socirity pot socirity pot within you	Device Model deversions heatiger heatiger ball year ball tool year deversions	threads scholar with an and with your guild you with for the scholar scholar thread scholar thre	virtual memory virtual memory ind viso prese viso memory viso memo	Virtual File System	protocol families we we we way to be we one protocol we one protocol we one protocol we one protocol protocol we one protocol protocol set
bridges crass-functional mobiles	debugging texutrice log but region-jonte print handeugen epoleumit tyduteeepoin epoleumit	Start, register arver, film pote toat, roode module module armodule barrer Janen korrer Janen	synchronization body tarrel tarriting all sear torriting tarriting all sear torriting tarritin	www.mag.et memory of revea mapping formers.gath term.data.at maj.bath maj.bath maj.bath	Bill State Bill S	socket Se.ops rking age socket splice socket splice socket splice socket splice socket splice socket splice socket splice socket splice socket splice
logical	HI subsystems	Norman System run berret Dood, Jarkaban Dood, Jarkaban Markaban	Scheduler aereningest task_struct schedule_treas_ and_treas_ arrangement arra	Popular memory physical magnet memory en, ist memory https:// pagi.s	Iogical file systems ent.gentus ent.gentus	Description of the second seco
device control	Abstract devices and HID class drivers ensystem ket mounceev and and and and and and and and and and	generic HW access report, report request, response ub, price ub, price ub, price ub, price ub, brot ub, brot	interrupts core	Page Allocator	block devices	network interface reading of the series and the series of the series and the series of the series and the series of the series of the series of the series of the series of the series of the the series of the series of the series of the the series of the series of the series of the the series of the series of the series of the the series of th
hardware interfaces chives. registers and interrupts	HI peripherals device drive sector and an accurate perbas	device access and bus drivers webw metw and website and website an	CPU specific which and service and press and press problemation whom, mp. PUTOP access, g. 40	physical memory operations accessing accessing body accessing body	disk controller drivers Sociation Sociation another and another and	ettop_er
electronics	user peripherals	LO Inen LO PCI 10 2013 ACP1 USB carlooder controller	CPU mpsies AP2 controler	memory	disk controllers	network controllers

Thank You

