Low-Overhead Virtualization of Mobile Systems

Gernot Heiser
NICTA and University of New South Wales
Sydney, Australia
Types of Virtualization

<table>
<thead>
<tr>
<th>Type</th>
<th>Hypervisor</th>
<th>Operating System</th>
<th>Processor</th>
<th>VM Process</th>
<th>OS</th>
<th>Java Program</th>
<th>Java VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type-1</td>
<td>Hosted</td>
<td>Native</td>
<td>Native</td>
<td>Hosted</td>
<td>Native</td>
<td>Native</td>
<td>Native</td>
</tr>
<tr>
<td>Type-2</td>
<td>Native</td>
<td>Native</td>
<td>Native</td>
<td>Native</td>
<td>Native</td>
<td>Native</td>
<td>Native</td>
</tr>
</tbody>
</table>
Why Virtual Machines?

Traditional (enterprise) uses:

• Server consolidation
  – Hardware & energy savings with QoS isolation
  – Migrating, checkpointing, debugging
  – Concurrent use of multiple OSes
    • … or OS versions

• Security
  – Partitioning to limit reach of intrusions
  – Sandboxing untrusted apps

Virtualizing mobile systems – crazy idea?
Mobile Phones

Dumb phone
- Apps
- RTOS
- Processor

Baseband Software

Smartphone
- Apps
- Rich OS (Linux)
- Hypervisor
- Processor

Consolidated phone
- Core
- Baseband Software

Heterogenous Operating Systems!
Consolidated Phone: Motorola Evoke

- Linux+BREW OS
- Linux+BREW apps
- Seamless UI integration
- Released April 2009

RTOS+BB

Apps

Linux

BREW OS

RTOS+BB

OKL4 Hypervisor

200 MHz ARM926
Dual-Persona Smartphone

- Phones increasingly used to access business data
  - Companies lock down phones, no arbitrary apps
  - Employees end up carrying two phones

- Integrate two virtual phones into one physical
  - Locked-down business phone
  - Open personal phone

- Only one used at a time
  - Perfect use of virtualization

Will reach market soon
Secure Communication on COTS Phone

- Secure phones are expensive (small product runs)
  - Strong push for COTS devices in defence etc
- Use virtualization to provide secure communication on standard smartphone
  - Encrypt voice, data and tunnel through open OS
- Hypervisor guarantees isolation
  - With controlled communication
- Small *trusted computing base*

Presently under evaluation by various agencies
Energy Management in Future Devices

- Load-based dynamic re-mapping of activities to cores
Virtualization Mechanics: Instruction Emulation

• “Pure” virtualization: *Trap and emulate* approach:
  – Guest attempts to access physical resource
  – Hardware raises exception (trap), invoking hypervisor’s handler
  – Hypervisor emulates result, based on access to virtual resources

• Most instructions do not trap
  – Makes efficient virtualization possible
  – Requires that VM ISA is (almost) same as physical processor ISA

• Works as long as architecture is “virtualizable”:
  – All instructions exposing or modifying physical resources must trap
  – Not the case e.g. for ARM

Guest

```assembly
ld  r0, curr_thrd
ld  r1, (r0, ASID)
mv  CPU_ASID, r1
ld  sp, (r1, kern_stk)
```

Hypervisor

```assembly
lda r1, vm_reg_ctxt
ld  r2, (r1,ofs_r0)
sto r2, (r1,ofs_ASID)
```
Para-Virtualization

- Manual modification of guest OS source
  - Port from hardware ISA to hypervisor API
    - Replace ISA instructions by trapping code ("hypercalls")
    - Expensive in terms of engineering time (& error prone)
  - Mandatory for non-virtualizable architecture (eg. ARM)
  - Optionally for performance improvements
    - Minimise costly hypervisor entries
    - Amortize hypercall cost over many instructions
Minimising Overheads

• Hypervisor design and implementation is important
  – Para-virtualization requires well-designed API
    • Minimise hypervisor entries
  – Tight implementation as hypervisor is on critical path
    • Small cache footprint
    • “Fast paths” for optimising common case
    • Many processor-specific optimisations
  – Keeping it small helps:
    • 10 kLOC is much easier to optimise than 100 kLOC!
# Overheads: Imbench Microbenchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Native</th>
<th>Virtualized</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>null syscall</td>
<td>0.6 µs</td>
<td>0.96 µs</td>
<td>0.36 µs</td>
</tr>
<tr>
<td>read</td>
<td>1.14 µs</td>
<td>1.31 µs</td>
<td>0.17 µs</td>
</tr>
<tr>
<td>stat</td>
<td>4.73 µs</td>
<td>5.05 µs</td>
<td>0.32 µs</td>
</tr>
<tr>
<td>fstat</td>
<td>1.58 µs</td>
<td>2.24 µs</td>
<td>0.66 µs</td>
</tr>
<tr>
<td>open/close</td>
<td>9.12 µs</td>
<td>8.23 µs</td>
<td>-0.89 µs</td>
</tr>
<tr>
<td>select(10)</td>
<td>2.62 µs</td>
<td>2.98 µs</td>
<td>0.36 µs</td>
</tr>
<tr>
<td>sig handler</td>
<td>1.77 µs</td>
<td>2.05 µs</td>
<td>0.28 µs</td>
</tr>
<tr>
<td>pipe latency</td>
<td>41.56 µs</td>
<td>54.45 µs</td>
<td>12.89 µs</td>
</tr>
<tr>
<td>UNIX socket</td>
<td>52.76 µs</td>
<td>80.90 µs</td>
<td>28.14 µs</td>
</tr>
<tr>
<td>fork</td>
<td>1,106 µs</td>
<td>1,190 µs</td>
<td>84 µs</td>
</tr>
<tr>
<td>fork+execve</td>
<td>4,710 µs</td>
<td>4,933 µs</td>
<td>223 µs</td>
</tr>
<tr>
<td>system</td>
<td>7,583 µs</td>
<td>7,796 µs</td>
<td>213 µs</td>
</tr>
</tbody>
</table>

OKL4 Microvisor on Beagle Board (500 MHz Cortex A8 ARMv7)
Overheads: Networking

Netperf networking benchmark on Linux

<table>
<thead>
<tr>
<th>Type</th>
<th>Measure</th>
<th>Native</th>
<th>Virtualized</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>Throughput [Mib/s]</td>
<td>651</td>
<td>630</td>
<td>3 %</td>
</tr>
<tr>
<td></td>
<td>CPU load [%]</td>
<td>99</td>
<td>99</td>
<td>0 %</td>
</tr>
<tr>
<td></td>
<td>Cost [µs/KiB]</td>
<td>12.5</td>
<td>12.9</td>
<td>3 %</td>
</tr>
<tr>
<td>UDP</td>
<td>Throughput [Mib/s]</td>
<td>537</td>
<td>516</td>
<td>4 %</td>
</tr>
<tr>
<td></td>
<td>CPU load [%]</td>
<td>99</td>
<td>99</td>
<td>0 %</td>
</tr>
<tr>
<td></td>
<td>Cost [µs/KiB]</td>
<td>15.2</td>
<td>15.8</td>
<td>4 %</td>
</tr>
</tbody>
</table>

OKL4 Microvisor on Beagle Board (500 MHz Cortex A8 ARMv7)
Virtualizing Devices: Two Possibilities

VM-Owned

- Device regs exposed to VM
  - unmodified native guest driver accesses device directly

Shared

- Virtual device exposed to VM
  - Virtual driver communicates with real driver in hypervisor
Shared Devices: Pure vs Para-Virtualized

**Pure: Unmodified guest driver**
- Each device register access by guest driver traps to hypervisor
  - real driver emulates
- Many traps – expensive!

**Para: Modified device API**
- Virtual device is simplified
  - possibly explicit driver communication API
  - virtual driver is very simple
- Can dramatically reduce traps
- **But: need new driver**
  - real driver ported to hypervisor
- Real driver can be
  - inside hypervisor
  - separate driver VM
  - one for all drivers
  - separate for each driver
Coming Up: Hardware Support

- ARMv7 virtualization extensions announced Q3/2010
- Anticipate Si samples in 2011, products in 2012
- Presently only simulator (**not** cycle accurate!)

### New privilege level: hyp
- Strictly higher than kernel
- Virtualizes or traps *all* sensitive instructions
- Only available in ARM TrustZone “non-secure” mode

### Note: different from x86
- VT-x “root” mode is orthogonal to x86 protection rings
ARM Virtualization Extensions (1)

Configurable Traps

- **Kernel mode**
  - Hyp mode
  - User mode
  - Can configure traps to go directly to guest OS

- **User mode**
  - Native syscall
  - Virtual syscall

- **Virtual syscall**

- **Hyp mode**

**CASES’11**
ARM Virtualization Extensions (2)

Emulation

1) Load faulting instruction
   • Compulsory L1-D miss!

2) Decode instruction
   • Complex logic

3) Emulate instruction
   • Usually straightforward

IR

R2

L1 I-Cache

L1 D-Cache

L2 Cache

mv CPU_ASID,r1

ld r1,(r0,ASID)

mv CPU_ASID,r1

ld sp,(r1,kern_stk)

mv CPU_ASID,r1

ld r1,(r0,ASID)

mv CPU_ASID,r1

ld sp,(r1,kern_stk)

...
ARM Virtualization Extensions (2)

Emulation Support

- HW decodes instruction
  - No L1 miss
  - No software decode
- SW emulates instruction
  - Usually straightforward

```c
mv CPU_ASID, r1
ir

ld r1, (r0, ASID)

mv CPU_ASID, r1

ld sp, (r1, kern_stk)

L1 I-Cache

L1 D-Cache

... ... ...

L2 Cache

ld r1, (r0, ASID)

mv CPU_ASID, r1

ld sp, (r1, kern_stk)
```
ARM Virtualization Extensions (3)

2-stage translation

- Hardware PT walker traverses both PTs
- Loads combined (guest-virtual to physical) mapping into TLB

Diagram:

1. User
   - load r0, adr
   - Guest virtual address

2. 1st PT ptr (Hardware)
   - Guest OS
   - (Virtual) guest page table

3. 2nd PT ptr (Hardware)
   - Hypervisor (Hardware)
   - Hypervisor's guest memory map

4. Memory
   - Physical address
   - data
ARM Virtualization Extensions (4)

Virtual Interrupts

- ARM has 2-part IRQ controller
  - Global “distributor”
  - Per-CPU “interface”
- New H/W “virt. CPU interface”
  - Mapped to guest
  - Used by HV to forward IRQ
  - Used by guest to acknowledge
- Reduces hypervisor entries for interrupt virtualization
Experience: Hypervisor Size

- Resonably complete prototype hypervisor utilising extensions
  - Runs Linux
  - Simulator only (no hardware)

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>ISA</th>
<th>Type</th>
<th>Kernel</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>OKL4</td>
<td>ARMv7</td>
<td>para-virtualization</td>
<td>9.8 kLOC</td>
<td>0</td>
</tr>
<tr>
<td>Prototype</td>
<td>ARMv7</td>
<td>pure virtualization</td>
<td>6 kLOC</td>
<td>0</td>
</tr>
<tr>
<td>Nova</td>
<td>x86</td>
<td>pure virtualization</td>
<td>9 kLOC</td>
<td>27 kLOC</td>
</tr>
</tbody>
</table>

- Much smaller than x86 pure-virtualization hypervisor
  - Mostly due to greatly reduced need for instruction emulation
- Size (& complexity) reduced about 40% wrt to para-virtualization
### Overheads (Estimated)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Pure virtualization</th>
<th>Para-virtualiz.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Instruct</td>
<td>Cycles (est)</td>
</tr>
<tr>
<td>Guest system call</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hypervisor entry + exit</td>
<td>120</td>
<td>650</td>
</tr>
<tr>
<td>IRQ entry + exit</td>
<td>270</td>
<td>900</td>
</tr>
<tr>
<td>Page fault</td>
<td>356</td>
<td>1500</td>
</tr>
<tr>
<td>Device emul.</td>
<td>249</td>
<td>1040</td>
</tr>
<tr>
<td>Device emul. (accel.)</td>
<td>176</td>
<td>740</td>
</tr>
<tr>
<td>World switch</td>
<td>2824</td>
<td>7555</td>
</tr>
</tbody>
</table>

- Note: *Rough* estimates due to lack of cycle-accurate simulation
- Interesting tradeoffs:
  - Fast syscalls (no emulation)
  - Slower hypervisor invocation, world switch
- Pure virtualization almost certainly unsuitable for device drivers
Future of Hypervisors: seL4 Microkernel

- Q: Can you trust separation by the hypervisor?
- A: Yes: we have proof!

By implication
⇒
implementation enforces integrity

Refinement proof
⇒
implementation correct

Refinement proof
⇒
spec enforces integrity

Abstract kernel spec
4,800 lines

Kernel implementation
8,700 lines C

Integrity requirements
1,000 lines
seL4 WCET Analysis

Open system - untrusted code

Closed system

Clearly early days, aiming for 10 \(\mu s\) WCET
Conclusions

• Virtualization is coming to mobile devices!
  – Hardware utilization
  – Security
  – Energy management

• Manufacturers are providing extensions to accelerate

• The art of para-virtualization is far from dying

• Isolation can have the strength of mathematical proof