Quest-V: A Secure and Predictable System for IoT and Beyond

Richard West
richwest@cs.bu.edu
Goals

• Develop system for high-confidence (embedded) systems
  – Mixed criticalities (timeliness and safety)

• Predictable – real-time support
• Secure – resistant to component failures & malicious attacks
• Self-healing
• Online recovery of software component failures
Target Applications

- Healthcare
- Avionics
- Automotive
- Factory automation
- Robotics
- Space exploration
- Secure/safety-critical domains
- Internet-of-Things (IoT)
Internet of Things

- Number of Internet-connected devices
  > 12.5 billion in 2010
- World population > 7 billion (2015)
- Cisco predicts 50 billion Internet devices by 2020

Challenges:
  - **Secure** management of vast quantities of data
  - **Reliable + predictable** data exchange b/w “smart” devices
In the Beginning...Quest

- Initially a “small” RTOS
- ~30KB ROM image for uniprocessor version
- Page-based address spaces
- Threads
- Dual-mode kernel-user separation
- Real-time Virtual CPU (VCPU) Scheduling
- Later SMP support
- LAPIC timing

FreeRTOS, uC/OS-II etc  →  Quest  ←  Linux, Windows, Mac OS X etc
From Quest to Quest-V

- Quest-V for multi-/many-core processors
  - Distributed system on a chip
  - Time as a first-class resource
    - Cycle-accurate time accountability
  - Separate sandbox kernels for system components
  - Memory isolation using h/w-assisted memory virtualization
  - Also CPU, I/O, cache partitioning
- Focus on safety, efficiency, predictability + security
Related Work

- Existing virtualized solutions for resource partitioning
  - Wind River Hypervisor, XtratuM, PikeOS, Mentor Graphics Hypervisor
  - Xen, Oracle PDOMs, IBM LPARs
  - Muen, (Siemens) Jailhouse
Problem

Traditional Virtual Machine approaches too expensive

- Require traps to VMM (a.k.a. hypervisor) to mux & manage machine resources for multiple guests
- e.g., ~1500 clock cycles VM-Enter/Exit on Xeon E5506

Traditional Virtual Machine approaches too memory intensive for embedded systems in areas such as IoT!
Traditional Approach (Type 1 VMM)

Type 1 VMM / Hypervisor

Hardware (CPUs, memory, devices)
Quest-V Approach

Eliminates hypervisor intervention during normal virtual machine operations

Hardware (CPUs, memory, devices)
Quest-V Architecture Overview

Sandbox 1

- VCPU
- VCPU
- Monitor
- PCPU(s)
- IO Devices

Sandbox 2

- VCPU
- Monitor
- PCPU(s)
- IO Devices

Sandbox M

- VCPU
- VCPU
- Monitor
- PCPU(s)
- IO Devices

Communication + Migration

Sandbox

Address Space

Thread
Memory Partitioning

- Guest kernel page tables for GVA-to-GPA translation
- EPTs (a.k.a. shadow page tables) for GPA-to-HPA translation
  - EPTs modifiable only by monitors
  - Intel VT-x: 1GB address spaces require 12KB EPTs w/ 2MB superpaging
Quest-V Memory Partitioning
I/O Partitioning

- Device interrupts directed to each sandbox
  - Use I/O APIC redirection tables
  - Eliminates monitor from control path
- EPTs prevent unauthorized updates to I/O APIC memory area by guest kernels
- Port-addressed devices use in/out instructions
- VMCS configured to cause monitor trap for specific port addresses
- Monitor maintains device "blacklist" for each sandbox
  - DeviceID + VendorID of restricted PCI devices
CPU Partitioning

• Scheduling local to each sandbox
  – partitioned rather than global
  – avoids monitor intervention

• Uses real-time VCPU approach for Quest native kernels [RTAS'11]
Predictability

- VCPUs for budgeted real-time execution of threads and system events (e.g., interrupts)
  - Threads mapped to VCPUs
  - VCPUs mapped to physical cores
- Sandbox kernels perform local scheduling on assigned cores
  - Avoid VM-Exits to Monitor – eliminate cache/TLB flushes
VCPUs in Quest(-V)

Address Space

Threads
Main VCPUs
I/O VCPUs

PCPUs (Cores)
Example VCPU Schedule
Utilization Bound Test

- Sandbox with 1 PCPU, n Main VCPUs, and m I/O VCPUs
  - $C_i = \text{Budget Capacity of } V_i$
  - $T_i = \text{Replenishment Period of } V_i$
  - Main VCPU, $V_i$
  - $U_j = \text{Utilization factor for I/O VCPU, } V_j$

\[
\sum_{i=0}^{n-1} \frac{C_i}{T_i} + \sum_{j=0}^{m-1} (2 - U_j) \cdot U_j \leq n \left( \sqrt[n]{2} - 1 \right)
\]
Cache Partitioning

- Shared caches controlled using color-aware memory allocator [COLORIS – PACT'14]
- Cache occupancy prediction based on h/w performance counters
  - $E' = E + (1 - E/C) \cdot m_h - E/C \cdot m_o$
  - Enhanced with hits + misses

[Book Chapter, OSR'11, PACT'10]
Linux Front End

• For low criticality legacy services
• Based on Puppy Linux 3.8.0
• Runs entirely out of RAM including root filesystem
• Low-cost paravirtualization
  – less than 100 lines
  – Restrict observable memory
  – Adjust DMA offsets
• Grant access to VGA framebuffer + GPU
• Quest native SBs tunnel terminal I/O to Linux via shared memory using special drivers
Quest-V Linux Screenshot

1 CPU + 512 MB

No VMX or EPT flags
Quest-V Performance

100 Million Page Faults

1 Million *fork-exec-exit* Calls
Quest-V Summary

• Separation kernel built from scratch
  – Distributed system on a chip
  – Uses (optional) h/w virtualization to partition resources into sandboxes
  – Protected comms channels b/w sandboxes

• Sandboxes can have different criticalities
  – Linux front-end for less critical legacy services

• Sandboxes responsible for local resource management
  – avoids monitor involvement
Proposed Work

- Port of Quest to Intel Galileo [Done]
- Qduino API [Ongoing]
- Port of Quest(-V) to Intel Edison and Minnowboard Max [Started]
- IoT Applications: 3D printing / manufacturing, robotics, secure home automation, etc [To Do]
- (Secure) Information Flow Analysis [To Do]
- Real-time Communication [Ongoing]
Quest on Galileo

- Porting Quest to the Galileo board:
  - Added multiboot support back to 32-bit GRUB EFI (GRUB Legacy)
  - Developed I2C, SPI controller drivers
  - Developed Cypress GPIO Expander and AD7298 ADC drivers

- Original Arduino API Support
- New real-time multithreaded Qduino API
Qduino

- Qduino – Enhanced Arduino API for Quest
  - Parallel and predictable loop execution
  - Real-time communication b/w loops
  - Predictable and efficient interrupt management
  - Real-time event delivery
  - Simplifies multithreaded real-time programming
Qduino Multi-loop Example

• Multiple loop sketch example:

```cpp
loop (1, 40, 100) { /* VCPU: C = 40, T = 100 */
    digitalWrite(LED1, HIGH);
    ... /* Blink LED1 */
}
loop (2, 20, 100) { /* VCPU: C = 20, T = 100 */
    analogWrite(LED2, brightness);
    ... /* Change brightness of LED2 */
}
setup () {
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
}
```
# Qduino New APIs

<table>
<thead>
<tr>
<th>Function Signatures</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>• loop(loop_id, C, T)</td>
<td>Structure</td>
</tr>
<tr>
<td>• interruptsVcpu(C,T)</td>
<td>Interrupt</td>
</tr>
<tr>
<td>• attachInterruptVcpu(pin,ISR,mode,C,T)</td>
<td>Interrupt</td>
</tr>
<tr>
<td>• spinlockInit(lock)</td>
<td>Spinlock</td>
</tr>
<tr>
<td>• spinlockLock(lock)</td>
<td>Spinlock</td>
</tr>
<tr>
<td>• spinlockUnlock(lock)</td>
<td>Spinlock</td>
</tr>
<tr>
<td>• channelWrite(channel,item)</td>
<td>Four-slot</td>
</tr>
<tr>
<td>• item channelRead(channel)</td>
<td>Four-slot</td>
</tr>
<tr>
<td>• ringbufInit(buffer,size)</td>
<td>Ring buffer</td>
</tr>
<tr>
<td>• ringbufWrite(buffer,item)</td>
<td>Ring buffer</td>
</tr>
<tr>
<td>• ringbufRead(buffer,item)</td>
<td>Ring buffer</td>
</tr>
</tbody>
</table>
Qduino Event Handling

Sketch

Handler

pthread_create
attachInterrupt

Real Time Event

User

Kernel

Main VCPU

Main VCPU

IO VCPU

Scheduler

GPIO Driver

CPU Core(s)

GPIO Expander

Pin State Monitoring
Qduino Temporal Isolation

- Foreground loop increments counter during loop period
- 2-4 background loops act as potential interference, consuming remaining CPU capacity
- No temporal isolation or timing guarantees w/ Linux
Possible Use Cases

- Mixed-criticality automotive system
- Secure home automation
- 3D printer controller
- IoT interoperability sandboxing
  - Secure virtual networks of untrusted devices
- Many others...
Mixed-Criticality Automotive System

More Critical

Less Critical

User

Real-time Command & Control

Real-time Sensor Data Processing

Display & External Comms

V2V, V2I Infotainment

Kernel

VCPU(s)

Monitor

MONITOR

MONITOR

MONITOR

Core(s)

Core(s)

Core(s)

I/O Devices e.g. Motors, Servos

I/O Devices e.g. Cameras, LIDAR

I/O Devices e.g. GPU, NIC

Hardware

Monitor

Monitor

Monitor

Sandbox 1

Sandbox 2

Sandbox M

INTERNET

Sandboxes on multicore platform replace CAN bus nodes
Secure Home Automation

More Secure

Real-time Sensor Data Processing

Comms

Web Server / App “Plugins”

Less Secure

Kernel

VCPU(s)

Monitor

More Secure

User

Core(s)

Memory

I/O Devices e.g. Cameras, CO+Fire Alarm

Sandbox 1

3rd Party untrusted services

INTERNET

Sandbox M

Linux

Core(s)

Memory

I/O Devices e.g. NIC

Hardware
Secure Home Automation

- Home equipped w/ cameras, alarms, window/door actuators, HVAC + appliance controls
- “Home owner” sandbox(es) for localized control of data, sensors + actuators
  - e.g., smartphone ↔ appliance control
- 3rd party sandbox(es) for plugin app services
  - e.g., Emergency (police/fire/ambulance) callouts
Secure Home Automation

- Challenges:
  - Prevent homeowner generating false alarms
    - Apply penalties from service provider
  - Prevent 3\textsuperscript{rd} parties accessing sensitive homeowner data (e.g., raw camera feeds)
    - Enforce secure inter-sandbox comms
    - Require services across sandboxes to be digitally signed by separate entities (non-collusion)
Secure Home Automation

- External system interface via public Internet only accesses 3rd party (untrusted) sandboxes
- Internal system interface via home network accesses trusted sandboxes

- Replicated monitors observe suspicious activity
  - e.g., high frequency access to “root” mode (monitor) via VM-exits
- Monitors akin to security guards
  - An attacker would have to compromise all such guards to prevent system recovery
Edison 3D Printer Controller

Real-time Sensing & Control
Real-time Job Scheduling
Web Server / Verification
Linux

Quark MCU
Sandbox 1
Core(s)
Memory
I/O Devices e.g. Motors, Extruder, Temp Sensors

Dual Core Atom Silvermont
Sandbox 2
Core(s)
Memory
I/O Devices e.g. Flash Storage

Sandbox 3
Core(s)
Memory
I/O Devices e.g. NIC

Hardware
Kernel
User

Untrusted
Trusted
INTERNET
Distributed Virtual Manufacturing

- Extend 3D print service to distributed “customizable” one-off manufacturing
  - A “Kinkos” 3D printing/manufacturing service
- Submit requests via web interface
  - Need to verify correctness
- Verified requests spooled for processing
- Use real-time comms + Qduino for real-time machine control
  - Possible to form “job shop” style assembly lines
IoT Interoperability Sandboxing

- Collaborative open-source frameworks
  - IoTivity (Open Interconnect Consortium: Intel, Samsung, Cisco, GE + many others)
  - Alljoyn (Allseen Alliance), 160+ partners
- Communication across different transport media, OSes, and protocols
- Microsoft Device System Bridges (DSBs) for Z-wave and BACnet
- Google's Brillo Weave, Apple Home Kit
IoT Interoperability Sandboxing

- Use Quest-V sandboxes to isolate IoTivity / Alljoyn software stacks
  - Promote secure isolation of networks of devices
- Use replicated / distributed monitor network to identify “unusual” (potentially malicious) network activity
What Next?

- Continue port of Quest(-V) to Edison and Minnowboard Max
- Develop 3D printer controller
  - Investigate techniques to quarantine and verify 3rd party service requests before processing
- Develop autonomous vehicle system
  - Look at real-time control in presence of injected faults
- Home automation prototype
  - Provide secure services for 3rd party plugins
Conclusions

- Quest-V uses one monitor per sandbox
  - Heightens security & safety
  - Monitors are small
    - Not needed for resource multiplexing
  - Can potentially exploit this to build new security models
    - Monitors like multiple system guards
- Chip-level distributed system
  - Real-time inter-sandbox communication
  - Isolation of 3rd party services
The Quest Team

- Richard West
- Ye Li
- Eric Missimer
- Matt Danish
- Gary Wong
- Ying Ye
- Zhuoqun Cheng