Quest – A Journey in Space and Time

Richard West
richwest@cs.bu.edu
Goals

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

• Predictable – real-time support
• Resistant to component failures & malicious manipulation (Secure)
• 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)
Case Studies

• $327 million Mars Climate Orbiter
  – Loss of spacecraft due to Imperial / Metric conversion error (September 23, 1999)

• 10 yrs & $7 billion to develop Ariane 5 rocket
  – June 4, 1996 rocket destroyed during flight
  – Conversion error from 64-bit double to 16-bit value

• 50+ million people in 8 states & Canada in 2003 without electricity due to software race condition
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
• 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 Approach (Type 1 VMM)

VM (Novell)  VM  VM  VM  VM  ...

Type 1 VMM / Hypervisor

Hardware (CPUs, memory, devices)
Contributions

- Quest-V Separation Kernel [WMC'13, VEE'14]
  - Uses H/W virtualization to partition resources amongst services of different criticalities
  - Each partition, or sandbox, manages its own CPU cores, memory area, and I/O devices w/o hypervisor intervention
  - Hypervisor typically only needed for bootstrapping system & managing comms channels b/w sandboxes
Contributions

• Quest-V Separation Kernel

Eliminates hypervisor intervention during normal virtual machine operations
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 Linux Memory Layout

- BIOS: 0x00000000 to 0x00100000
- Quest Sandbox 0: SANDBOX_OFFSET+0x100000
- Quest Sandbox 1: 2*SANDBOX_OFFSET+0x100000
- Quest Sandbox 2: 3*SANDBOX_OFFSET+0x100000
- Quest Sandbox 3: 4*SANDBOX_OFFSET+0x100000
- Shared Memory Region: PHYS_SHARED_MEM_HIGH
- Reserved for Module: 0x80000000
- Reserved: 0xFFFFFFFF
- Linux Kernel: 2G starting address
Quest-V Memory Partitioning

SB Kernel
- Guest Virtual Address
- Kernel Paging Data Structures
- Guest Physical Address

Monitor
- EPT Data Structures
- Host Physical Address

EPT Data Structure
- PML4
- Directory Ptr
- Directory
- Table
- Offset
- PDPT
- PDE
- Phy Addr
- PML4E
- PTE

Legend:
- Guest Domain
- Host Domain
Memory Virtualization Costs

- Example Data TLB overheads
- Xeon E5506 4-core @ 2.13GHz, 4GB RAM
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
Quest-V I/O Partitioning

Data Port: 0xCFC

Address Port: 0xCF8
Monitor Intervention

During normal operation only one monitor trap every 3-5 mins by CPUID

<table>
<thead>
<tr>
<th></th>
<th>No I/O Partitioning</th>
<th>I/O Partitioning (Block COM and NIC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception (TF)</td>
<td>0</td>
<td>9785</td>
</tr>
<tr>
<td>CPUID</td>
<td>502</td>
<td>497</td>
</tr>
<tr>
<td>VMCALL</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>I/O Instruction</td>
<td>0</td>
<td>11412</td>
</tr>
<tr>
<td>EPT Violation</td>
<td>0</td>
<td>388</td>
</tr>
<tr>
<td>XSETEBV</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table: Monitor Trap Count During Linux Sandbox Initialization
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)
VCPUs in Quest(-V)

- **Two classes**
  - **Main** → for conventional tasks
  - **I/O** → for I/O event threads (e.g., ISRs)

- **Scheduling policies**
  - **Main** → sporadic server (SS)
  - **I/O** → priority inheritance bandwidth-preserving server (PIBS)
SS Scheduling

• Model periodic tasks
  – Each SS has a pair (C,T) s.t. a server is guaranteed C CPU cycles every period of T cycles when runnable
    • Guarantee applied at foreground priority
    • background priority when budget depleted
  – Rate-Monotonic Scheduling theory applies
PIBS Scheduling

- IO VCPUs have utilization factor, $U_{V,\text{IO}}$

- IO VCPUs inherit priorities of tasks (or Main VCPUs) associated with IO events
  - Currently, priorities are $f(T)$ for corresponding Main VCPU
  - IO VCPU budget is limited to:
    - $T_{V,\text{main}} \times U_{V,\text{IO}}$ for period $T_{V,\text{main}}$
• IO VCPUs have *eligibility* times, when they can execute

• $t_e = t + \frac{C_{\text{actual}}}{U_{V,\text{IO}}}$
  - $t =$ start of latest execution
  - $t \geq$ previous eligibility time
Example VCPU Schedule
Example Replenishments

Replenishment Queue Element

- VCPU 0 (C=10, T=40, Start=1)
- VCPU 1 (C=20, T=50, Start=0)
- IOVCPU (Utilization=4%)

Interval [t=0,100] (A) VCPU 1 = 40%, (B) VCPU 1 = 46%
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, $V_i$
  - $U_j = \text{Utilization factor for I/O VCPU, } V_j$

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

• Shared caches controlled using color-aware memory allocator [COLORIS – PACT'14]

• Cache occupancy prediction based on h/w performance counters
  \[ E' = E + (1-E/C) \times m_l - E/C \times m_o \]
  
  - 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
Quest-V Linux Screenshot

1 CPU + 512 MB

No VMX or EPT flags
Quest-V Performance

- Measured time to play back 1080P MPEG2 video from the x264 HD video benchmark
- Mini-ITX Intel Core i5-2500K 4-core, HD3000 graphics, 4GB RAM
Quest-V Network Performance

- netperf UDP send
- netperf UDP receive (netserver)

- Realtek gigabit NIC to remote host
- Virtio enabled for Xen
- IOP = I/O partitioning w/o blacklist
Quest-V Performance

100 Million Page Faults

1 Million fork-exec-exit Calls
Conclusions

- Quest-V 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
Quest-V Status

• About 11,000 lines of kernel code
• 200,000+ lines including lwIP, drivers, regression tests
• SMP, IA32, paging, VCPU scheduling, USB, PCI, networking, etc
• Quest-V requires BSP to send INIT-SIPI-SIPI to APs, as in SMP system
  – BSP launches 1st (guest) sandbox
  – APs “VM fork” their sandboxes from BSP copy
Current & Future Work

- Online fault detection and recovery
- Technologies for secure monitors
  - e.g., Intel TXT + VT-d
- SLIPKNOT for IoT
  - SecureLy Isolated Predictable Kernels for Networks of Things
- Inter-sandbox real-time communication & migration (4-slot async comms etc)

See www.questos.org for more details
Internet of Things

• Number of Internet-connected devices
  > 12.5 billion in 2010
• World population > 7 billion (2014)
• Cisco predicts 50 billion Internet devices by 2020

Challenges:
  • Secure management of vast quantities of data
  • Reliable + predictable data exchange b/w “smart” devices
SLIPKNOT Example

Galileo running Quest

Galileo

QBOX

Fire Alarm

Internet

USB

Ethernet

Wireless

802.11p

4G Network

Wireless

Ethernet

SLIPKNOT Services

Quest

Monitor

VCPU

CPU m

SLIPKNOT Services

Quest

Monitor

VCPU

CPU m

SLIPKNOT Services

Linux Kernel

Monitor

CPU m

Comms channel (e.g. shared memory)
Other (Current) Developments

- Port of Quest to Intel Galileo Arduino
- Applications: RacerX, manufacturing, etc
- Quest RT-USB host controller stack [RTAS'13]
Quest-V Demo

• Bootstrapping Quest native kernel (cores 0-2)
  + Linux (core 3)
    - Linux kernel + filesystem in RAM
    - Secure comms channel b/w Quest SB & Linux SB using a pseudo-char device
      - /dev/qSBx device for each sandbox x
• Triple modular redundancy (TMR) fault recovery for unmanned aerial vehicle (UAV)

http://quest.bu.edu/demo.html
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
Quest on Galileo

• Arduino+ API Support
  – Parallel and predictable loop execution
  – Real-time communication b/w loops
  – Predictable and efficient interrupt management
  – Real-time event delivery
• 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);
}
```
The Quest Team

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