Quest-V – a Virtualized Multikernel

Richard West
richwest@cs.bu.edu

Ye Li, Eric Missimer
{liye, missimer}@cs.bu.edu
Goals

- Develop system for high-confidence (embedded) systems
- Predictable – real-time support
- Resistant to component failures & malicious manipulation
- Self-healing
- Online recovery of software component failures
Target Applications

- Healthcare
- Avionics
- Automotive
- Factory automation
- Robotics
- Space exploration
- Other safety-critical domains
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
Approach

- Quest-V for multicore processors
  - Distributed system on a chip
  - Time as a first-class resource
    - Cycle-accurate time accountability
  - Separate sandbox kernels for system sub-components
  - Isolation using h/w-assisted memory virtualization
    - Extended page tables (EPTs – Intel)
    - Nested page tables (NPTs – AMD)
  - Security enforcible using VT-d + interrupt remapping (IR)
    - Device interrupts scoped to specific sandboxes
    - DMA xfers to specific host memory
Isolation

• Memory virtualization using EPTs isolates sandboxes and their components
• Dedicated physical cores assigned to sandboxes
• Temporal isolation using Virtual CPUs (VCPUs)
Extended Page Tables

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

Monitor
- EPT Data Structures
  - Host Physical Address

PML4
Directory Ptr
Directory
Table
Offset

PML4E
PDPTE
PDE
PTE
Phy Addr

EPT Data Structure

Guest Domain
Host Domain
Quest-V Memory Layout

Virtual Memory Layout

Physical Memory Layout

Shared Memory Region

User Space

EPT Data Structure 1

Shared Driver

Monitor 1

Sandbox Kernel 1

BIOS

0xFFFF FFFFFFFF

0x00000000

Sandbox 1

User Space

Shared Memory Region

EPT Data Structure 1

Shared Driver

Monitor 1

Sandbox Kernel 1

BIOS

0xFFFF FFFFFFFF

0x00000000

Sandbox M

User Space

Shared Memory Region

EPT Data Structure M

Shared Driver

Monitor M

Sandbox Kernel M

BIOS

0xFFFF FFFFFFFF

0x00000000
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)

Threads
Main VCPUs
I/O VCPUs
PCPs (Cores, HTs)
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, $U_{V,IO}$

- 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:
    - $T_{V,main} \times U_{V,IO} \times U_{V,main}$ for period $T$
PIBS Scheduling

• IO VCPUs have *eligibility* times, when they can execute

\[ t_e = t + \frac{C_{\text{actual}}}{U_{\text{v,io}}} \]

  - \( t = \) start of latest execution
  - \( t \geq \) previous eligibility time
Example VCPU Schedule

Timeline

Budgets

- VCPU0 (C=1, T=3)
- VCPU1 (C=1, T=4)
- VCPU2 (C=2, T=10)
Sporadic Constraint

- Worst-case preemption by a sporadic task for all other tasks is not greater than that caused by an equivalent periodic task

(1) Replenishment, R must be deferred at least $t + T_v$

(2) Can be deferred longer

(3) Can merge two overlapping replenishments
   - $R1.time + R1.amount \geq R2.time$ then MERGE
   - Allow replenishment of $R1.amount + R2.amount$ at $R1.time$
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
  - $C_i =$ Budget Capacity of $V_i$
  - $T_i =$ Replenishment Period of $V_i$
  - Main VCPU, $V_i$
  - $U_j =$ Utilization factor for I/O VCPU, $V_j$

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

• Lightweight I/O virtualization & interrupt passthrough capabilities
  • e.g., VNICS provide separate interfaces to single NIC device
• Avoid VM-Exits into monitor for scheduling & I/O mgmt
I/O Passthrough

Sandbox 1
- Apps
- Kernel
  - Main VCPU
  - IO VCPU
- Monitor
- CPU 1

Sandbox 2
- Apps
- Kernel
  - Main VCPU
  - IO VCPU
- Monitor
- CPU 2

Sandbox M
- Apps
- Kernel
  - Main VCPU
  - IO VCPU
- Monitor
- CPU M

Shared Mem / Msg Channel

Migration

Shared Drivers

I/O Device (e.g., NIC)
Virtualization Costs

- Example Data TLB overheads
- Xeon E5506 4-core @ 2.13GHz, 4GB RAM
Device (Driver) Sharing

- Example NIC RX Ring Buffer

RX Ring Buffer at Time t

RX Ring Buffer at Time t + 1

Index Moving

- Sandbox x read index
- Driver DMA index

- Ready for DMA
- Not Ready for DMA
- DMA Data Available
Shared Driver Costs

Netperf UDP Throughput Test

UDP Throughput (Mbps)

- Quest-V
- Linux
- Xen (PVM)
- Xen (HVM)

1xNetperf
2xNetperf
4xNetperf
Quest
Example Fault Recovery

SB Kernel (Guest) Monitor (Host)

Component Failure Detection
Component Recovery in Local Sandbox
Component Recovery in Remote Sandbox
Remote Event Notification via IPI

Fault Identification And Handling

1. Component Failure Detection
2. Fault Identification And Handling
3. Component Recovery in Remote Sandbox
4. Remote Event Notification via IPI

Kernel
Main VCPU
IO VCPU

Monitor

Msg Channel

NIC Driver

NIC

(1)

(2)

(3)

(4)
Faulting Driver for Web Server

- httpperf with web server in presence of Realtek NIC driver fault

- Requests / replies set at 120/s under normal operation
  - Single-threaded server
  - Focus on one process
  - Recovery time rather than throughput
Performance Costs

- Core i5-2500K with 8GB RAM

<table>
<thead>
<tr>
<th>Recovery Phases</th>
<th>CPU Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Local Recovery</td>
</tr>
<tr>
<td>VM-Exit</td>
<td>885</td>
</tr>
<tr>
<td>Driver Switch</td>
<td>10503</td>
</tr>
<tr>
<td>IPI Round Trip</td>
<td>N/A</td>
</tr>
<tr>
<td>VM-Enter</td>
<td>663</td>
</tr>
<tr>
<td>Driver Re-initialization</td>
<td>1.45E+07</td>
</tr>
<tr>
<td>Network Re-initialization</td>
<td>78351</td>
</tr>
</tbody>
</table>
Inter-Sandbox Communication

- Via Communication VCPUs
  - High rate VCPUs: 50/100ms
  - Low rate VCPUs: 40/100ms
The Quest Team

• Rich West
• Ye Li
• Eric Missimer
• Matt Danish
• Gary Wong
Further Information

• Quest website
  • http://www.cs.bu.edu/fac/richwest/quest.html

• Github public repo
  • http://questos.github.com
Quest(-V) Summary

- About 11,000 lines of kernel code
- 175,000+ 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
Final Remarks

• Quest-V multikernel
  − Leverages H/W virtualization for safety/isolation
  − Avoids VM-Exits for VCPU/thread scheduling
  − Online fault recovery
  − Shared memory communication channels
  − Lightweight I/O virtualization
  − Predictable VCPU scheduling framework
Isolation

- 4 sandboxes: SB0,..., SB3
  - SB1 sends msgs to SB0, SB2 & SB3 at 50ms intervals
    - SB0, SB2 & SB3 rx at 100, 800, 1000ms intervals, respectively
  - SB0 handles ICMP requests
    - sent remotely at 500ms intervals
  - Observe failure + recovery in SB0

- Messaging threads on Main VCPUs: 20ms/100ms
- NIC driver I/O VCPU: 1ms/10ms
Next Steps

- VCPU/thread migration
- API extensions
- Application development
- Hardware performance monitoring
- RT-USB sub-system
- Fault detection
Real-Time Migration

- At $t$, guarantee VCPU, $V_{src}$, moves from $SB_{src} \rightarrow SB_{dest}$ without violating:
  
  (a) Remote VCPU requirements, $\forall V_{dest} \in SB_{dest}$

  (b) Requirements of $V_{src}$

- Use migration VCPUs, $V_{migrate} \ [C_{mig}, T_{mig}]$

- Ensure: $U_{dest} + \frac{C_{src}}{T_{src}} \leq (n+1)(\frac{n+1}{\sqrt{2}} - 1), |V_{dest}| = n @ t' < t$

- Ensure: $C[\text{memcpy of } V_{src} + \text{thread(s)}] \leq C_{mig}$
  
  - while $V_{src}$ is ineligible for execution
Real-Time Migration

Make migration decision (Find destination)

Resume local scheduling

Migration thread event received

Resume local scheduling

SB Kernel (Guest)

Monitor (Host)

Push quest_tss address(es) to destination

Copy quest_tss structure(s)

Move addr space and VCPU from source

Main VCPU

IO VCPU

Kernel

Scheduler

Monitor

Main VCPU

Migration Thread

Main VCPU

Scheduler

Monitor

Main VCPU

IO VCPU

Kernel

Scheduler

Monitor

Main VCPU

IO VCPU

Kernel

Scheduler

Monitor
VCPU API

- Full thread support
  - NB: Limit a VCPU to one address space
  - Reduces migration costs

```c
int VCPU_create(struct vcpu_param *param)
struct vcpu_param {
    int vcpuid;
    policy; // SCHED_SPORADIC, SCHED_PIBS
    int mask; // affinity mask
    int C; // budget
    int T; // period
}
```
VCPU API

- `int VCPU_destroy(int vcpuid, int force);`
- `int VCPU_setparam(int vcpuid, struct vcpu_param *param);`
- `int VCPU_getparam(struct vcpu_param *param);`
- `Int VCPU_bind_task(int vcpuid);`

- Policy:
  - Which sandboxes assigned which VCPUs?
    - Utilization considerations
    - Cache usage (perfmon)
    - Have SBs announce their utilization (bidding)
Real-Time Fault Recovery

- Real-time fault recovery
  - Local & remote
  - Requires SB with working scheduler for predictable recovery
  - Remote recovery can avoid re-initialization of faulting service
Real-Time Fault Recovery

- Real-Time Fault Recovery
- Fault Recovery Thread
- Exit Code
- Entry Code
- Sandbox Kernel
- Monitor
- Schedule
- De-schedule
- Restore Machine State for Recovery Code
- Save Machine State for Recovery Code
- Start / Continue Recovery Procedure
- Monitor LAPIC Timer Handler
- LAPIC Timer Interrupt
Applications

- RacerX
- TORCS
- Benchmarks
  - Web server
  - Netperf
  - Canny
  - Others?
Performance Monitoring

- (LLC) Cache hits, misses, instrs retired, TSC,...
- Can predict s/w thread LLC occupancy in real-time
  \[ E' = E + (1-E/C)M_1 - E/CM_0 \]
  - See West, Zaroo, Waldspurger & Zhang
    - OSR, December 2010
Experiments

- Intel Core2 Extreme QX6700 @ 2.66GHz
- 4GB RAM
- Gigabit Ethernet (Intel 8254x “e1000”)
- UHCI USB Host Controller
  - 1GB USB memory stick
- Parallel ATA CDROM in PIO mode

- Measurements over 5sec windows using bandwidth-preserving logging thread
Experiments

- CPU-bound threads: increment a counter
- CD ROM/USB threads: read 64KB data from filesystem on corresponding device
## I/O Effects on VCPUs

<table>
<thead>
<tr>
<th>VCPU</th>
<th>V&lt;sub&gt;c&lt;/sub&gt;</th>
<th>V&lt;sub&gt;T&lt;/sub&gt;</th>
<th>threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCPU0</td>
<td>2</td>
<td>5</td>
<td>CPU-bound</td>
</tr>
<tr>
<td>VCPU1</td>
<td>2</td>
<td>8</td>
<td>Reading CD, CPU-bound</td>
</tr>
<tr>
<td>VCPU2</td>
<td>1</td>
<td>4</td>
<td>CPU-bound</td>
</tr>
<tr>
<td>VCPU3</td>
<td>1</td>
<td>10</td>
<td>Logging, CPU-bound</td>
</tr>
<tr>
<td>IOVCPU</td>
<td>10%</td>
<td>ATA</td>
<td></td>
</tr>
</tbody>
</table>
I/O Effects on VCPUs
# PIBS vs SS IO VCPU Scheduling

<table>
<thead>
<tr>
<th>VCPU</th>
<th>$V_c$</th>
<th>$V_T$</th>
<th>threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCPU0</td>
<td>1</td>
<td>20</td>
<td>CPU-bound</td>
</tr>
<tr>
<td>VCPU1</td>
<td>1</td>
<td>30</td>
<td>CPU-bound</td>
</tr>
<tr>
<td>VCPU2</td>
<td>10</td>
<td>100</td>
<td>Network, CPU-bound</td>
</tr>
<tr>
<td>VCPU3</td>
<td>20</td>
<td>100</td>
<td>Logging, CPU-bound</td>
</tr>
<tr>
<td>IOVCPU</td>
<td>1%</td>
<td></td>
<td>Network</td>
</tr>
</tbody>
</table>
t=50 start ICMP ping flood. Here, we see comparison overheads of two scheduling policies.
Network bandwidth of two scheduling policies
## IO VCPU Sharing

<table>
<thead>
<tr>
<th>VCPU</th>
<th>$V_c$</th>
<th>$V_T$</th>
<th>threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCPU0</td>
<td>30</td>
<td>100</td>
<td>USB, CPU-bound</td>
</tr>
<tr>
<td>VCPU1</td>
<td>10</td>
<td>110</td>
<td>CPU-bound</td>
</tr>
<tr>
<td>VCPU2</td>
<td>10</td>
<td>90</td>
<td>Network, CPU-bound</td>
</tr>
<tr>
<td>VCPU3</td>
<td>100</td>
<td>200</td>
<td>Logging, CPU-bound</td>
</tr>
<tr>
<td>IO VCPU</td>
<td>1%</td>
<td></td>
<td>USB, Network</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VCPU</th>
<th>$V_c$</th>
<th>$V_T$</th>
<th>threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCPU0</td>
<td>30</td>
<td>100</td>
<td>USB, CPU-bound</td>
</tr>
<tr>
<td>VCPU1</td>
<td>10</td>
<td>110</td>
<td>CPU-bound</td>
</tr>
<tr>
<td>VCPU2</td>
<td>10</td>
<td>90</td>
<td>Network, CPU-bound</td>
</tr>
<tr>
<td>VCPU3</td>
<td>100</td>
<td>200</td>
<td>Logging, CPU-bound</td>
</tr>
<tr>
<td>IO VCPU1</td>
<td>1%</td>
<td></td>
<td>USB</td>
</tr>
<tr>
<td>IO VCPU2</td>
<td>1%</td>
<td></td>
<td>Network</td>
</tr>
</tbody>
</table>
IO VCPU Sharing

![Bar chart showing kB/s for shared and separate scenarios with different VCPU configurations.]

- USB
- Network
- USB (pingflood)

Shared vs Separate VCPU configurations.
Conclusions

• Temporal isolation on IO events and tasks
• PIBS + SS Main & IO VCPUs can guarantee utilization bounds
• Future investigation of higher-level policies
• Future investigation of h/w performance counters for VCPU-to-PCPU scheduling
Example Fault Recovery

Component Failure Detection → Fault Identification And Handling → Remote Event Notification (IPI)

SB Kernel (Guest) → Monitor (Host)

VM-Exit → VM-Entry