The Quest OS for Real-Time Computing

Richard West
richwest@cs.bu.edu
Emerging “Smart” Devices
Goals

• High-confidence (embedded) systems
• **Mixed criticalities** – timeliness and safety
• **Predictable** – real-time support
• **Secure** – resistant to component failures & malicious attacks
• **Fault tolerant** – online recovery from soft errors and timing violations
Target Applications

• Healthcare
• Avionics
• Automotive
• Factory automation
• Robotics
• Space exploration
• Internet-of-Things (IoT)
• Industry 4.0 “smart factories”
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 data
• Reliable + predictable data exchange
• Device interoperability
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, RTEMS 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
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 many real-time embedded systems
Traditional Approach (Type 1 VMM)

VM
VM
VM
VM
...

Type 1 VMM / Hypervisor

Hardware (CPUs, memory, devices)
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

Address Space

Thread
Mixed-Criticality Automotive System

More Critical

Less Critical

User
Real-time Command & Control

Real-time Sensor Data Processing

Display & External Comms

Kernel
VCPUs

Monitor

Monitor

Monitor

Hardware
Core(s)

Core(s)

Core(s)

Memory

Memory

Memory

I/O Devices e.g. Motors, Servos

I/O Devices e.g. Cameras, LIDAR

I/O Devices e.g. GPU, NIC

Sandbox 1

Sandbox 2

Sandbox M

Sandboxes on multicore platform replace CAN bus nodes

INTERNET

V2V, V2I Infotainment

...
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
Quest-V Linux Memory Layout

- BIOS: 0x00000000 - 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
- Linux Kernel
  - 4*SANDBOX_OFFSET+0x100000 + LINUX_MEM_SIZE
- Reserved for Module
  - 0x80000000
- Shared Memory Region
  - PHYS_SHARED_MEM_HIGH
- Reserved for Hardware
  - 0xFFFFFFFF
Memory Virtualization Costs

- Example Data TLB overheads
- Xeon E5506 4-core @ 2.13GHz, 4GB RAM

![Graph showing CPU cycles vs. number of pages for different VM exit and TLB flush operations.]
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)
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 $T_{V,\text{main}}$
PIBS Scheduling

- IO VCPUs have *eligibility* times, when they can execute
- \( t_e = t + \frac{C_{\text{actual}}}{U_{V,IO}} \)
  - \( t = \) 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
   - \( R_1.\text{time} + R_1.\text{amount} \geq R_2.\text{time} \) then MERGE
   - Allow replenishment of \( R_1.\text{amount} + R_2.\text{amount} \) at \( R_1.\text{time} \)
Example SS-Only Schedule

\[ \tau_1 \text{ Main Application} \\
\text{Sporadic Server} \\
C = 8 \quad T = 16 \]

\[ \tau_2 \text{ I/O Interrupt BH} \\
\text{Sporadic Server} \\
C = 4 \quad T = 16 \]
Example SS+PIBS Schedule

- \( \tau_1 \) Main Application
  - Sporadic Server
  - \( C=8 \)  \( T=16 \)

- \( \tau_2 \) I/O Interrupt BH
  - PIBS
  - \( U=0.25 \)

I/O Event Initiated
Interrupts Occur
No Missed Deadline
Utilization Bound Test

- Sandbox with 1 PCPU, n Main VCPUs, and m I/O VCPUs
  - \( Ci = \) Budget Capacity of \( Vi \)
  - \( Ti = \) Replenishment Period of \( Vi \)
  - Main VCPU, \( Vi \)
  - \( Uj = \) Utilization factor for I/O VCPU, \( Vj \)

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

• Shared caches controlled using color-aware memory allocator [COLORIS – PACT'14]
• Cache occupancy prediction based on h/w performance counters
  – \( E' = E + (1-E/C) \times m_1 - E/C \times m_0 \)
  – 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**

- **CPU and Memory**: 1 CPU + 512 MB
- **Kernel Flags**: No VMX or EPT flags
Quest-V Performance

100 Million Page Faults

1 Million fork-exec-exit Calls
• 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
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
  – **Backward compatible** with Arduino API
  – Simplifies multithreaded real-time programming
// Sketch 1: toggle GPIO pin 9
// every 2s
int val9 = 0;

void setup() {
    pinMode(9, OUTPUT);
}

void loop() {
    val9 = !val9; //flip the output value
digitalWrite(9, val9);
    delay(2000); //delay 2s
}

// Sketch 2: toggle pin 10 every 3s
int val10 = 0;

void setup() {
    pinMode(10, OUTPUT);
}

void loop() {
    val10 = !val10; //flip the output value
digitalWrite(10, val10);
    delay(3000); //delay 3s
}

How do you merge the sketches and keep the correct delays?
Interleaved Sketches

- Do scheduling by hand
- Inefficient
- Hard to scale

```c
int val9, val10 = 0;
int next_flip9, next_flip10 = 0;

void setup() {
    pinMode(9, OUTPUT);
    pinMode(10, OUTPUT);
}

void loop() {
    if (millis() >= next_flip9) {
        val9 = !val9; //flip the output value
        digitalWrite(9, val9);
        next_flip9 += 2000;
    }
    if (millis() >= next_flip10) {
        val10 = !val10; //flip the output value
        digitalWrite(10, val10);
        next_flip10 += 3000;
    }
}
```
```cpp
int val9, val10 = 0;
int C = 500, T = 1000;

void setup() {
    pinMode(9, OUTPUT);
    pinMode(10, OUTPUT);
}

void loop(1, C, T) {
    val9 = !val9; // flip the output value
    digitalWrite(9, val9);
    delay(2000);
}

void loop(2, C, T) {
    val10 = !val10; // flip the output value
    digitalWrite(10, val10);
    delay(3000);
}
```
Qduino Organization

Diagram with layers:
- Quest Native App
- Quest Native App
- Sketch
- loop1 to loopN
- Qduino Libs
- VCPU Scheduler
- GPIO Driver
- SPI Driver
- I2C Driver
- User Kernel
- x86 SoC
- Galileo
- Edison
- Minnowboard
# 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) → <strong>I/O VCPU</strong></td>
<td>Interrupt</td>
</tr>
<tr>
<td>• attachInterruptVcpu(pin,ISR,mode,C,T) → <strong>Main VCPU</strong></td>
<td></td>
</tr>
<tr>
<td>• spinlockInit(lock)</td>
<td>Spinlock</td>
</tr>
<tr>
<td>• spinlockLock(lock)</td>
<td></td>
</tr>
<tr>
<td>• spinlockUnlock(lock)</td>
<td></td>
</tr>
<tr>
<td>• channelWrite(channel,item)</td>
<td>Four-slot</td>
</tr>
<tr>
<td>• item channelRead(channel)</td>
<td></td>
</tr>
<tr>
<td>• ringbufInit(buffer,size)</td>
<td>Ring buffer</td>
</tr>
<tr>
<td>• ringbufWrite(buffer,item)</td>
<td></td>
</tr>
<tr>
<td>• ringbufRead(buffer,item)</td>
<td></td>
</tr>
</tbody>
</table>
Qduino Event Handling

```
Scheduler

<table>
<thead>
<tr>
<th>Main VCPU</th>
<th>Main VCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I/O VCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

CPU Core(s)

GPIO Expander

Sketch Thread

User Interrupt Handler

attachInterruptVcpu

interrupt return

User

Kernel

Wakeup

Interrupt Bottom Half

Hardware Interrupt

GPIO Driver

```
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
Qduino Rover

- Autonomous Vehicle
  - Collision avoidance using ultrasonic sensor

- Two tasks:
  - A **sensing task** detects distance to an obstacle – `delay(200)`
  - An **actuation task** controls the motors - `delay(100)`
Rover Performance

• Measure the time interval between two consecutive calls to the motor actuation code
  
  • **Clanton Linux single loop**
    • delay from both sensing and actuation task
  
  • **Qduino multi-loop**
    • No delay from sensing loop
    • No delay from sensor timeout

• The shorter the worst case time interval, the faster the vehicle can drive
RacerX Autonomous Vehicle
Secure Home Automation

Real-time Sensor Data Processing

Web Server / App “Plugins”

VCPU(s)

Linux

Monitor

Monitor

Core(s)

Core(s)

Memory

Memory

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

I/O Devices e.g. NIC

User

Kernel

More Secure

Less Secure

Sandbox 1

Sandbox M

3rd Party untrusted services

INTERNET

3rd Party untrusted services

INTERNET
Edison 3D Printer Controller

Real-time Sensing & Control

Real-time Job Scheduling

Web Server / Verification

I/O Devices e.g. Motors, Extruder, Temp Sensors

I/O Devices e.g. Flash Storage

I/O Devices e.g. NIC

Linux

VCPU(s)

Core(s)

Monitor

Memory

Sandbox 1

Sandbox 2

Sandbox 3

DUAL CORE ATOM SILVERTOND

INTEGRATED MEMORY

Trusted

Untrusted

INTERNET

Hardware

Kernel

User
Minnowboard 3D Printer Controller
Demos

- Qduino Rover
- **Quest-V Triple modular redundancy (TMR)** fault recovery for unmanned aerial vehicle (UAV)

http://quest.bu.edu/demo.html
Conclusions

- Quest-V uses one monitor per sandbox
  - Heightens security & safety
  - Monitors are small
    - Not needed for resource multiplexing

- Possible to refactor legacy apps into separate components mapped to different sandboxes
  - Eases transition to a new OS

- Qduino real-time multi-loop programming
The Quest Team

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