Quest(-V): A Secure and Predictable System for Smart IoT Devices

Richard West
richwest@cs.bu.edu
Emerging “Smart” Devices Need an OS

- Multiple cores
- GPIOs
- PWM
- Virtualization support
- Integrated Graphics
- Various bus interfaces
- Timing + data security requirements
Recap: Quest-V Separation Kernel

Exploit VT-x/EPT capabilities on Intel multicore processors for efficient sandboxing
VCPUs in Quest(-V)

- Temporal isolation between VCPUs
- Guarantee budget C every T cycles (or time units)
- I/O VCPUs use simpler bandwidth preservation scheme
  - Reduces timer reprogramming overheads for short-lived interrupts
Proposed Work

- Implement and study Quest(-V) on Intel SBCs
- Port of Quest to Intel Galileo [Done]
- Port of Quest(-V) to Intel Edison and Minnowboard Max [Quest is working]
- Qduino API [Version 1 complete]
  - Now working on QduinoMC [In progress]
- IoT “smart” devices/apps: 3D printing / manufacturing, robotics, secure home automation, UAVs, etc [In progress]
Smart Devices

• **Dumb device?**
  – Requires remote inputs to function
  – No autonomy

• **Smart device?**
  – Ability to make own decisions, at least partly, based on sensory inputs that determine the state of the environment and the device itself

• e.g., **Smart 3D printer**
  – Spool requests via webserver
  – High level (STL file) requests rather than g-codes
  – Local slicer engine & g-code parser
  – Local verifier for “correctness” of requests
  – Possible communication/coordination with other smart devices
Developments 1/2

● Built 3D printer controller circuit using:
  - MinnowMax/Turbot
  - RAMPS 1.4
  - ADS7828 I2C Analog-to-Digital Converter
  - 4 x 4988 Pololu Stepper Motor drivers
  - PNP/NPN transistors, resistors etc for level shifting

● Tested on a Printrbot Simple Metal
  - See: www.cs.bu.edu/fac/richwest/smartprint3d.php
Developments 2/2

- Ported Marlin 3D printer firmware to Yocto Linux
  - Used Intel IoT devkit libmraa library to interface with I2C ADC and GPIOs via sysfs
- Ported Quest to MinnowMax and Turbot
  - Developed test scenarios for 3D print objects
  - Details to follow
- Papers
  - Qduino – RTSS'15
  - Quest-V – ACM TOCS
Marlin on Arduino

• One loop and two timer interrupt handlers
  - Loop: read G-code commands, translate them to motor movements and fan/heater operations
  - A high frequency, sporadic timer interrupt to drive motors (up to 10 Khz)
    • Trapezoidal speed control
  - A low frequency, periodic timer interrupt to read extruder temperature (1 KHz)
Real-Time Challenges

- Nanosleep timing for stepper motor control
- Matching extrusion rate with bed motion
- Let:
  - $B =$ gear pitch (e.g., 2mm for GT2 pulley)
  - $C =$ gear tooth count (e.g., 20)
  - $S =$ stepper motor steps per revolution (e.g., 200)
  - $\alpha =$ microstepping (e.g., 16 for 4988 driver)
  - $V =$ feedrate in given axis (e.g., 125mm/s)
- GPIO stepper pulse frequency, $F$:
  - $F = \frac{(V \times S \times \alpha)}{(B \times C)} = 10$kHz using above params
  - Requires 100 microsecond pulse timing
  - Won't work with Linux scheduling accuracy!
Marlin on Linux/MinnowBoard Max

• Ported Marlin to a Linux program
  - Replaced hardware timer interrupts with high resolution software timers
    • Linux hrtimer-based nanosleep
  - Replaced architecture-dependent I/O operations with mraa library functions

- **Cons:** approach fails to utilize underlying hardware parallelism
Marlin on Linux

File system

Read Gcode

Translate coordinates to steps

Use temperature to do PID control

Buffer: each block contains steps for one command

Extract steps from the block and pulse the steppers

Motors

Temperature

PID output

Read temperature and adjust Fan & heater

Fan & Heater
Quest on MinnowBoard Max

• Ported Quest to MinnowBoard Max
  - Added I2C Driver
  - Added GPIO Driver
  - Updated ACPI firmware to latest version

• Implemented partial mraa library on Quest
  - I2C Module (read/write bytes on I2C bus)
  - GPIO Module (get/set value+direction of GPIOs)

• Qduino Framework
• Three Qduino loops
  - Loop 1: command reading and path planning
    - Calculate & buffer steps+direction along each axis
  - Loop 2: motor driving
    - Smallest period and largest CPU utilization
  - Loop 3: temperature reading & adjustment
    - Largest period and smallest utilization
Marlin on Quest/MinnowMax

Loop 2
- Extract steps from the block and pulse the steppers

Loop 1
- G-code translation
- Temperature PID control

Loop 3
- Read temperature and adjust fan/heater

Qduino Library

VCPU

Quest Kernel

Core 1

MinnowBoard
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
Interleaved Sketches

// 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

```cpp
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;
  }
}
```
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

Quest Native App

Quest Native App

Sketch

loop1

loopN

Qduino Libs

User

Kernel

VCPU Scheduler

GPIO Driver

SPI Driver

I2C Driver

Galileo

Edison

Minnowboard

x86 SoC
## 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) ← I/O VCPU</td>
<td>Interrupt</td>
</tr>
<tr>
<td>• attachInterruptVcpu(pin,ISR,mode,C,T) ← Main VCPU</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

- **Sketch Thread**
- **User Interrupt Handler**
- **attachInterruptVcpu**
- **interrupt return**
- **Main VCPU**
- **I/O VCPU**
- **GPIO Driver**
- **CPU Core(s)**
- **GPIO Expander**

Diagram shows the flow of events between different components such as Event Handling, Scheduler, User Interrupt Handler, and 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
Edison 3D Printer Controller

User
- Real-time Sensing & Control
  - VCPU(s)
  - Monitor
- VCPU(s)
- Core(s)
- Memory
- I/O Devices e.g. Motors, Extruder, Temp Sensors

Kernel
- Real-time Job Scheduling
  - Monitor
- Web Server / Verification
  - Linux
  - Monitor

Hardware
- 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

Untrusted

Trusted
MinnowMax 3D Printer Controller
Future Directions

• **QduinoMC**
  - API support to map loops to cores
  - Load balancing via MARACAS [RTSS'16] framework
  - Pub/sub communications between Quest-V sandboxes
    - e.g., Linux ↔ Quest

• **QROS**
  - Legacy Linux ROS nodes communicate w/ time-critical Quest services
Future Directions

• Smart Devices / Apps
  – Use Intel SBCs/SoCs (Up board, Edison, MinnowMax, Celeron Braswell, Skylake U, Kaby/Apollo Lake NUCs)
  – Energy + CPU + GPU + latency-sensitive I/O requirements
  – RacerX autonomous rover
  – Smart drones
    • Configurable mission objectives (indoor / outdoor)
    • Search & rescue, surveillance, package delivery, SLAM, target tracking
    • Real-time adaptive control (e.g. in windy conditions)
  – Biokinematic / body sensor network (Edison/Curie)
Future Directions

• Quest-V on Edison, MinnowMax, Up board, other Intel SBCs/SoCs
  – Mixed-criticality: Linux + Quest
  – Mixed-criticality scheduling (ECRTS'16)
  – TMR fault tolerance using replicated sandboxes
MARACAS Framework

- Quest memory+cache-aware scheduling framework
  - Supports VCPU load balancing to share background cycles across cores
    - Background cycles: 1-C/T
    - Uses h/w perf counters to identify bus congestion
    - Congestion? Throttle select cores with available background cycles
    - Reduces memory/bus congestion while guaranteeing VCPU foreground timing requirements
MARACAS Framework

• Avg memory request latency = Occupancy / Requests
• Occupancy = UNC_ARB_TRK_OCCUPANCY.ALL
  – Cycles weighted by queued memory requests
• Requests = UNC_ARB_TRK_REQUEST.ALL
  – # of requests to memory controller request queue
• If latency exceeds threshold apply weighted throttling to cores
• Use COLORIS [PACT'14] dynamic page coloring for cache isolation
Lessons Learned

• Intel SBCs for “smart” devices
  – Multiple cores (good for multi-tasking)
  – VT-x capabilities for security/isolation/fault tolerance
  – GPIOs for interfacing sensors + actuators
  – PWMs for motor & servo control
  – Serial interfaces for device communication
  – Shared caches + memory bus affects temporal isolation (not good for real-time!)
  • ARINC 653 requires space-time isolation b/w cores
Wish List 1/2

• Intel SBCs for “smart” devices
  – Temporal isolation b/w cores
    • TI ARM PRU-like features for dedicated core(s)
    • Quark offers something close on the Edison
    • Support for cache + bus isolation (way partitioning, page coloring, TDMA bus management?)
  – Better GPU support
    • Needed for vision+AI+deep learning tasks
    • Georgia Tech AutoRally vehicle uses Mini-ITX + Nvidia GTX 750 Ti PCIe card, which is too power-hungry and heavy
  – Low-wattage “PC” with GPIOs, serial buses, GPU ala Nvidia Jetson (but better!)
    • e.g., “smart” drone has energy and weight restrictions
• Intel SBCs for “smart” devices
  – Simplified VT-x support
    • Basic memory partitioning b/w sandboxes (no EPT walking)
    • Like segmentation with simplified VMCS
    • Simplified IOMMU w/ DMA to sandbox physical offset address
  – Tagged memory for confidentiality + integrity on secure information flows between sandboxes
  – H/W-assisted port-based I/O interposition
    • To prevent sandbox discovery/access to unauthorized devices
The End

? || /* */
Extra Slides
Goals

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

• Predictable
• Secure
• Safe / Fault tolerant
• Efficient
Target Applications

- Healthcare
- Avionics
- Automotive
- Factory automation
- Robotics
- Space exploration
- Internet-of-Things (IoT)
- Industry 4.0 smart factories
- Smart drones, other devices
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 processing & exchange
  - Device interoperability
Background: Quest Real-Time OS

- 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  \(\leftrightarrow\)  Quest  \(\leftrightarrow\)  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
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
• 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:
    • $T_{V,\text{main}} * U_{V,\text{IO}}$ for period $T_{V,\text{main}}$
PIBS Scheduling

- IO VCPUs have *eligibility* times, when they can execute
- \( t_e = t + \frac{C_{\text{actual}}}{U_{v,\text{IO}}} \)
  - \( t \) = start of latest execution
  - \( t \geq \) previous eligibility time
**Example SS-Only Schedule**

- **τ₁ Main Application**
  - Sporadic Server
  - C = 8, T = 16

- **τ₂ I/O Interrupt BH**
  - Sporadic Server
  - C = 4, T = 16

Execution timeline:

- I/O Event Initiated
- Interrupts Occur

**Missed Deadline**

Timeline points:

- [8,0] [8,16]
- [4,9] [2,11] [1,25] [1,27] [2,27] [1,29] [1,41]
- [4,0] [3,11] [1,25] [2,25] [1,27] [1,29]
- [3,9] [1,25]
Example SS+PIBS Schedule

τ₁ Main Application
Sporadic Server
C=8  T=16

τ₂ I/O Interrupt BH
PIBS
U=0.25

Execution

I/O Event
Initiated

Interrupts Occur

No Missed
Deadline
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, $V_j$

$$\sum_{i=0}^{n-1} \frac{C_i}{T_i} + \sum_{j=0}^{m-1} (2-U_j) \cdot U_j \leq n \cdot (\sqrt{2} - 1)$$
• Shared caches controlled using color-aware memory allocator \[\text{COLORIS} – \text{PACT'14}\]
• Cache occupancy prediction based on h/w performance counters
  – $E' = E + (1-E/C) * m_i - E/C * m_o$
  – Enhanced with hits + misses
    \[\text{Book Chapter, OSR'11, PACT'10}\]
• 5 patents (3 awarded so far) w/ VMware