Orlando
2005

Freescale
Technology
Forum

Network. Connect. Explore.

Real-Time Technology in Linux

Sven-Thorsten Dietrich

Real-Time Architect

freescale
semiconductor

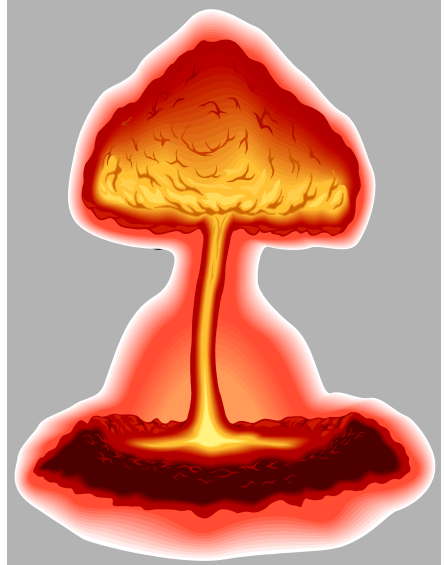
Introductions

- **MontaVista Software is a leading global supplier of systems software and development tools for intelligent connected devices and the associated infrastructure.**
  - We provide a commercial-grade Linux-based operating system and universal development platform. Our products address software developer needs encompassing applications ranging from communications infrastructure to consumer electronics.
  - We are engaged with the Linux community in developing code that communicates the needs of MontaVista customers **directly** to the developers maintaining the Software.

- **Sven-Thorsten Dietrich**
  - Real-Time Systems Architect, Technology R & D

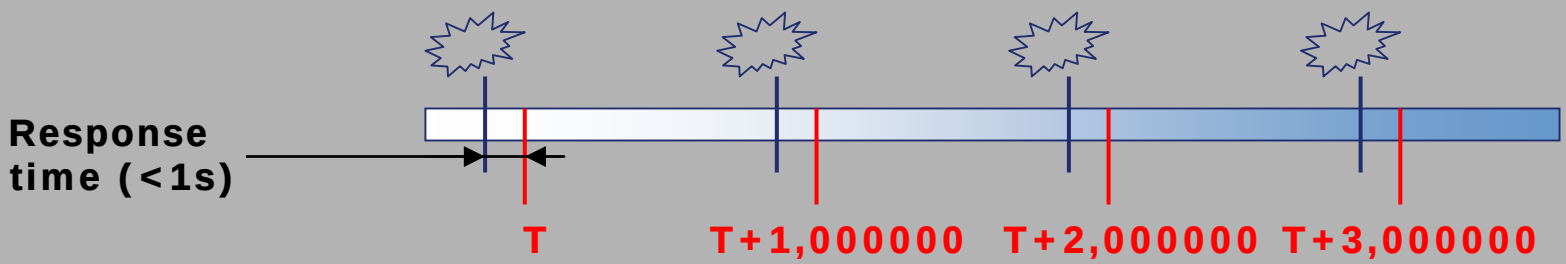
Real-Time Processing

- Deterministic Program Execution
  - Program Execution Time Known
  - Program Execution Time Bounded by Maximum
- Soft Real-Time
  - Value of Computation Diminishes after Deadline Passes
    - > Mouse pointer slow-to-change to hour-glass after mouse click
- Hard Real-Time
  - Value of Computation at Most 0 after Deadline Passes
    - > Nuclear Plant Fuel-Rod Retraction



Real-Time Processing ≠ FAST Code

- Real-Time Programming is Predictable Programming
  - Atomic (Cesium) Clock Example: Pulse Once Per Second (PPS)
    - > Accuracy Quantified by Spacing of Pulses
    - > Execution time must be less than 1s, constant offset calibrated
    - > PPS Code Path Must NEVER Vary
      - – Synchronous Operation
      - – No CPU cache + No Memory Paging



Why Real-Time in Handheld and Embedded Systems

- Cost / Performance / Power / Weight Compromise
  - Competitive, high-volume, low-margin Markets
  - Maximum Feature-set, Add-ons, Responsive UI feel
  - Devices have minimal CPU & Memory & Power
  - Minimal CPU = High CPU utilization
  - High CPU load + Time-Critical functionality = RT specs
  - Real-time Requirements will **never** be alleviated by Improvements in Hardware Performance / Efficiency
    - > Software utilizing latest hardware technologies easily keep up with and outpace advances in hardware technology

Why Linux in Embedded Systems

- NOT because of Linux Kernel's Real-Time performance
- Linux with its UNIX legacy, are inherently NOT Real-Time Operating Environments.
- Au contraire, Fairness, Progress and Resource sharing stand in stark contrast with the requirements of Embedded or Time-Critical Applications
- Linux IS Evolving to become a Formidable RTOS

Linux and Real-Time: History

- Early Linux Not Designed for Real-Time Processing
  - Early Linux (1.x Kernel) installations on retired Windows PCs
    - > Older hardware still useful under Linux due to efficiency of the fledgling OS
    - > Linux outperformed Windows in reliability and uptime (and still does)
  - Linux Design: Fairness, Throughput and Resource-Sharing
    - > Basic Unix development design principles applied in Kernel
    - > Heavily (over)-loaded systems continue to make progress
    - > Does not drop network connections or starve users / applications
  - Fairness- and resource-sharing design objectives
    - > contributed to make Linux competitive and popular in the enterprise-server and development-application environments
    - > Gave rise to RedHat and others.
    - > essential to the evolution of Linux, endemic of UNIX legacy

Linux and Real-Time: Evolution over Time

- Gradual SMP-Oriented Linux Kernel Optimizations

Early Kernel 1.x	No Kernel preemption
SMP Kernel 2.x	No Kernel preemption, “BKL” SMP Lock
SMP Kernel 2.2 - 2.4	No preemption, Spin-locked Critical Sections
“Preempt” Kernel 2.4	Kernel Preemption outside Critical Sections Spin-locked Critical Sections
Current Kernel 2.6	Kernel Preemption outside Critical Sections, Preemptible “BKL”
“RT-Preempt” Kernel	Kernel Critical sections Preemptible IRQ Subsystem Preemptible Mutex Locks with Priority Inheritance

MontaVista Realtime Preemption R & D

- Fall 2000 – MontaVista Unveils Linux 2.4 Kernel Preemption
  - Preemption concept expands on SMP locking
  - Audio community endorses new Preemptible Kernel technology
- December 2003 – Linux 2.6.0 Kernel Released
  - Incorporates into Main-stream MontaVista's Preemptible Kernel Concept
- Spring 2004 – MontaVista Software focuses on 2.6 Kernel RT
  - Linux 2.6 Real-Time performance baseline vs. Linux 2.4 kernel
    - > MontaVista Linux 2.4 kernel outperforms the Community's 2.6 kernel
  - Proof of Concept: Soft IRQs in task context
- Summer 2004
  - Real-Time Kernel Prototype Development
    - > Interrupts executing in thread context
    - > Kernel Mutex integration, debugging
  - MontaVista Engineers recognize Overlapping Community Development

MontaVista Realtime Preemption R & D

- October 2004
  - MontaVista announces RT prototype to Linux Community
  - Audio Community Tests and Strongly Endorses RT Preemption
  - Ingo Molnar incorporates RT code into Voluntary Preemption, renames to Real-Time Preemption
  - Other Companies Open-Source their Real-Time Code
- November - December 2004
  - Community (led by Ingo Molnar) Integrates and refines RT patch
- January 2005 – June 2005
  - RT patch for Linux 2.6 i386 Kernel available in Community
  - MontaVista releases RT ports to Arm, MIPS, PPC, PPC64, X64
- June 8, 2005
  - **MontaVista Software, Inc. announces Hard-Real-Time Linux**

Linux Real-Time Technology Overview

- Real-time Technology Linux 2.6 Kernel Enhancements
  - Preemptible Interrupt Handlers in Thread Context
  - Priority-Inheritance Mutex substituted for Spin-Locks
    - > PI Mutex-protects Kernel's critical sections
  - Preemptible Big Kernel Lock (BKL)
  - Preemptible Read-Write Locks
  - Preemptible RCU Locks
  - Improved Concurrency / SMP correctness / SMP scalability
    - > additional per-CPU variables, cache coherence
  - Debugging, Diagnostic and Performance-tuning tools:
    - > Mutex Deadlock-Detect
    - > IRQs-off Tracing and Latency Instrumentation
    - > Preemption Latency Tracing

Thread-Context Interrupt Handlers

- Legacy Linux IRQ Subsystem Shortcomings
  - IRQ Subsystem Preempts Tasks Unconditionally
  - Preemption Latency Correlated with Interrupt Load
  - Bottom-Half (SoftIRQ) Processing Unbounded
    - > SoftIRQs may re-activate
    - > SoftIRQ daemon defers re-activating-SoftIRQ activity to task space
  - No Interrupt Priorities

- Real-Time Solution: Interrupts in Thread Context
  - Promote SoftIRQ daemon to handle ALL Bottom-half processing
  - Demote IRQ Top-half execution into Thread-space IRQ Handlers
  - IRQ activates corresponding Handler-thread, returns immediately
  - IRQ Handlers scheduled in Bounded-time by O(1) Scheduler

Thread-Context Interrupt Handlers

- Threaded IRQs Pros
  - IRQ Processing does not Interfere with Task Execution
  - Priority Assignment Flexibility
    - > Real-Time Task-Priority can be higher than IRQ Priority
  - Real-Time Tasks can Preempt IRQ handlers
  - Inter-leaving of RT and IRQ tasks
  - Uncontended IRQ Execution-space for RT IRQ functions
    - > RT IRQs do not contend with common IRQs
    - > RT IRQs latency predictable & subject to minimal variation
- Threaded IRQs Cons
  - IRQ-Thread Scheduling Overhead
  - Throughput / Efficiency Reduced
    - > IRQs no longer running at the highest priority
    - > Response-Time / Throughput tradeoff

Priority-Inheriting Kernel Mutex

- High-Performance Priority-Inheriting Kernel Mutex
  - fundamental RT Technology
- Generic Implementation
  - easy to port to Architectures
- Priority Inheritance eliminates Priority Inversion Delays
  - MontaVista currently working on Generic PI implementation
    - > Applies to Robust Mutex / Futex
    - > Applies to RT Mutex
- Priority-based O(1) Wait Queues
  - Constant-time waiter-list Processing
- Deadlock Detect
  - identifies Lock-ordering errors, Cycles

Critical-region Management: PI Mutexes

- PI Mutex Technology Fundamental for RT
  - Preemptible alternative to Spin-lock / Non-Preemptible Regions
    - > Logical progression of MontaVista's 2.4 Preemptible technology
    - > Multiple Concurrent Tasks in independent Critical Sections
  - Replaces SMP Spin-locks in the Kernel
    - > Spinlock typing Preserved
    - > Spin locks, Read / Write Locks and RCU locks
  - Enabler for user-space PI mutexes, condition variables, Posix
- Dramatic reduction in 2.6 preemption delays
  - Almost all kernel code becomes Preemptible
    - > Non-Preemptible: Interrupt-off Paths and IRQ Thread Dispatch
    - > Non-Preemptible: Scheduling and context switching code
- RT Tasks designed to use Kernel-resources in managed ways can eliminate or reduce Priority-Inheritance delays
  - Rate-Monotonic / EDF / SJF tasks, with limited inter-dependence can achieve near-Hard-RT performance (Liu / Layland, ~~1970)

Real-time Linux 2.6 Kernel Preemption Enablers

- SMP Scalability improves with Fine-granular Preemption
  - Mainstreaming of Dual-Processor Technology
    - > Dual-Core Pentium 4
    - > Dual-Core Athlon
    - > Dual-Core PPC
  - Pending availability of Quad, Octal Entertainment platforms
- Pro-Audio Performance Requirements
  - Audio community very involved in Kernel preemption since 2.2
  - Audio community strongly endorsing RT technology
- Embedded Application Domain
  - Single-Chip, Mobile Applications (Mobilinux TM)
  - Predictable OS performance eliminates design uncertainty
    - > Reliable Prototyping
    - > Improved Product Scheduling

Linux Real-Time Technology Status

- Real-Time Performance
  - All IRQs running in threads
  - Critical sections in kernel are fully Preemptible
  - Kernel task preempt latency 10s of μs on late x86 CPU (worst case)
- Current R & D on RT Kernel
  - Driver bug reports (RT preemption reveals sub-standard coding)
  - RT “awareness” extensions to Power-management subsystem
  - Queuing for Integration with Community Kernel in progress
  - High Resolution Timers
  - Robust Mutexes (with Priority Inheritance)
    - > Generic PI, Generic DD

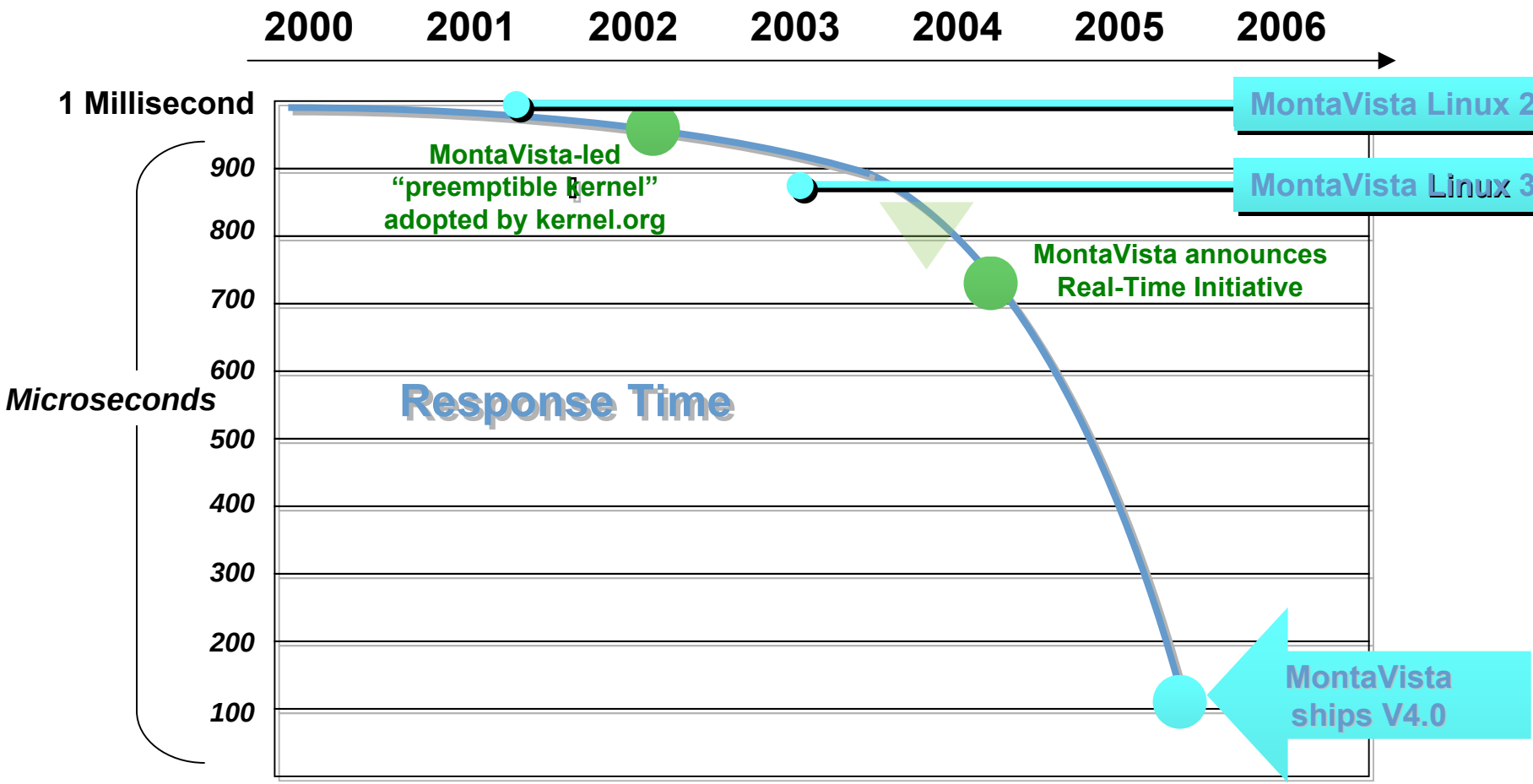
Real-time Linux 2.6 Kernel Acceptance and Integration

- Community Status
  - Stable RT kernel is Development-stable in Community
  - Generic implementation facilitates portability, stability
- Real-Time Linux 2.6 Kernel Performance
  - Far exceeds most stringent Audio performance requirements
  - Enables Hard Real Time for RT-aware Applications
- Real-Time Linux 2.6 Technology Confidence
  - Rapid Adoption into Community Kernel in Progress
  - Growing Community RT-awareness, acceptance
  - First Class MontaVista Engineering Support
- Real-Time Linux 2.6 Kernel Future Enhancements
  - Smart IRQ disable, for quantifiable Hard-Real-Time Performance

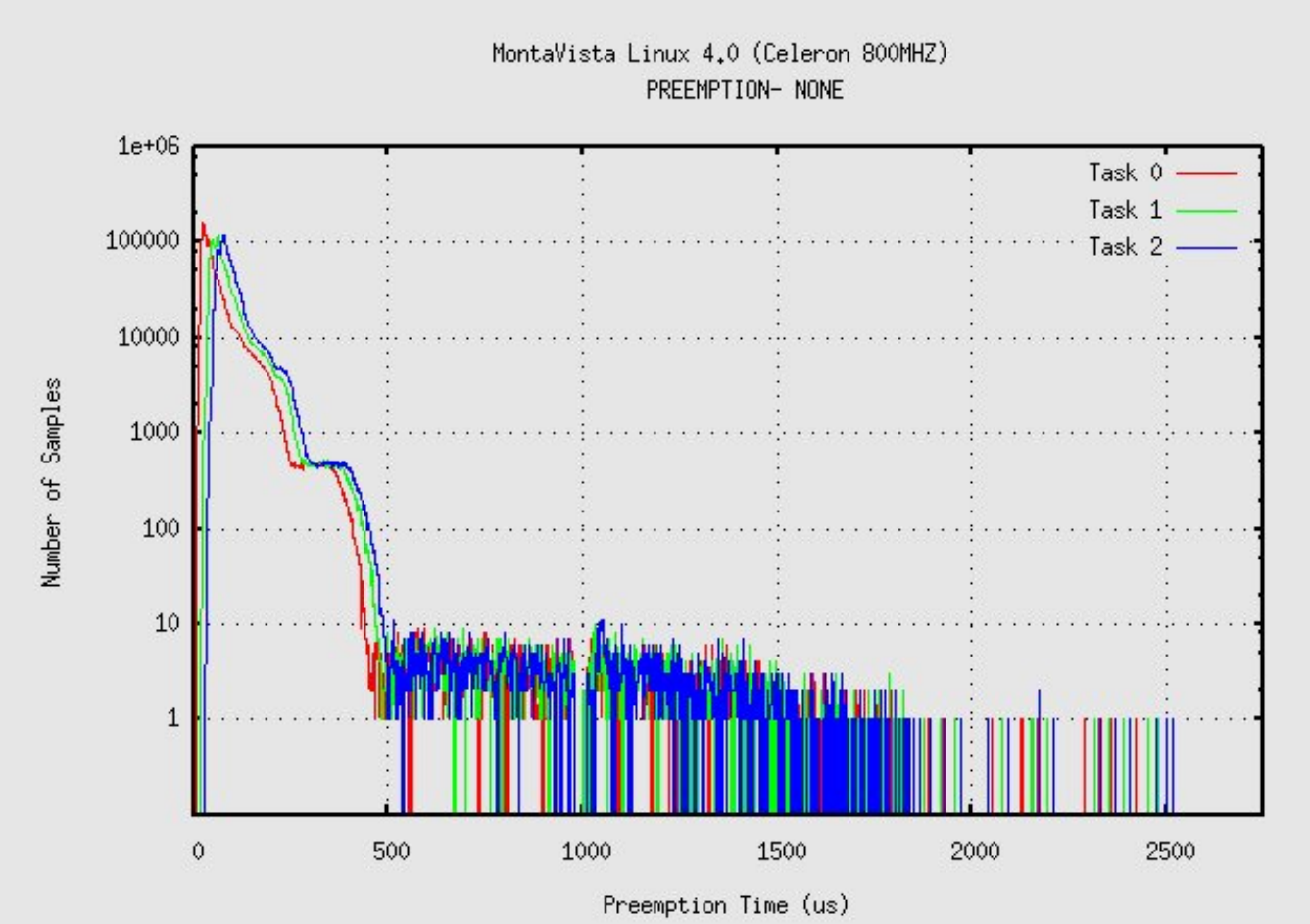
Summary

- MontaVista:
  - Is Leading Real-Time Linux Kernel Innovation
    - > First to Open-Source RT Kernel Prototype
    - > First to Open-Source RT Kernel Arch Ports
    - > Spearheading Hard-Real-Time Efforts
  - Is Leading User-Space RT-Mutex development:
    - > Mutex Special Interest Group:
    - > http://developer.osdl.org/dev.mutexes
  - MontaVista Staff Expertise in Linux Real-Time Technology is Unprecedented

Native Real-Time in Linux
Powerful Recent Technology Advancements

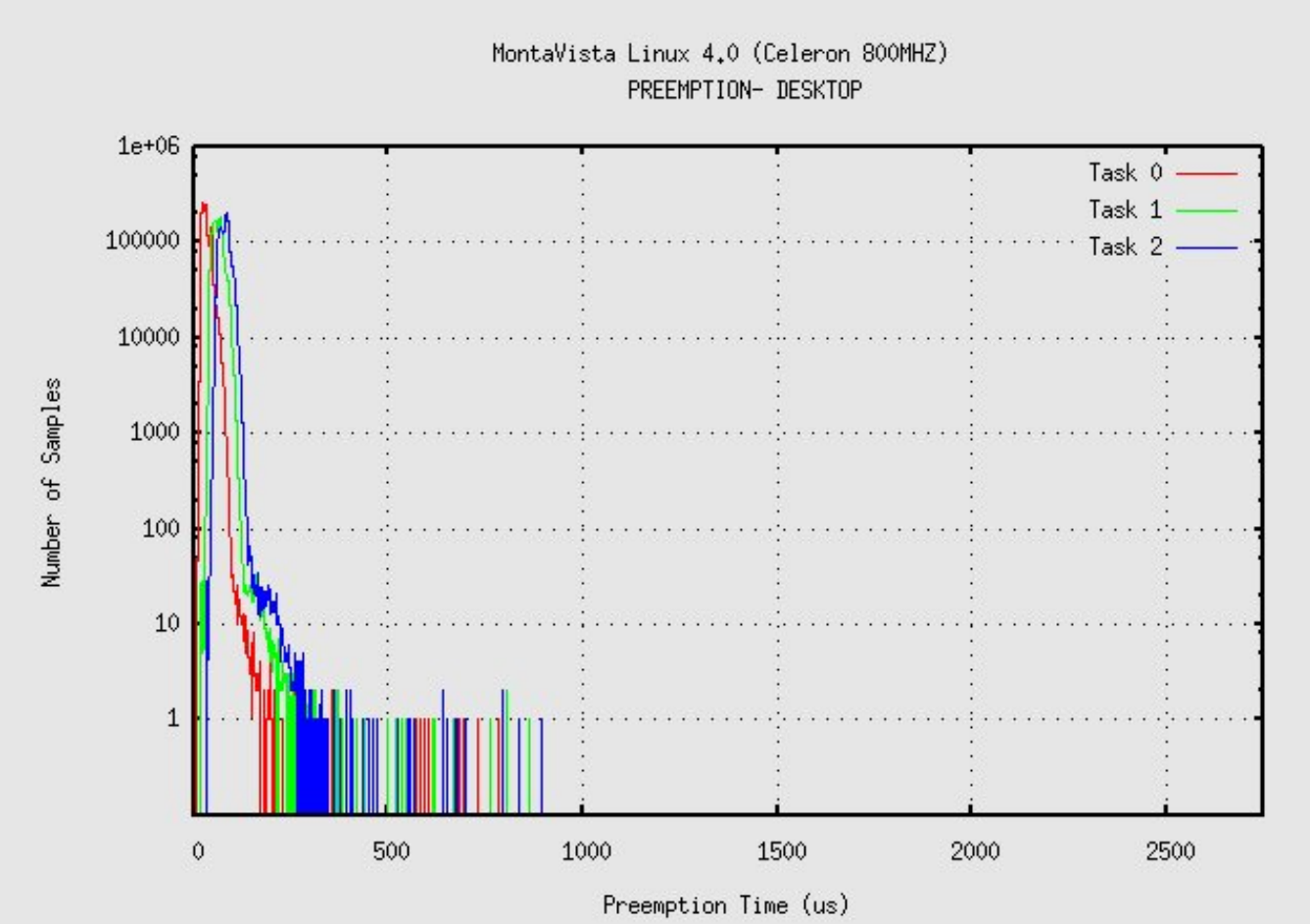


Linux 2.6 Kernel Preemption Latency – No Preemption



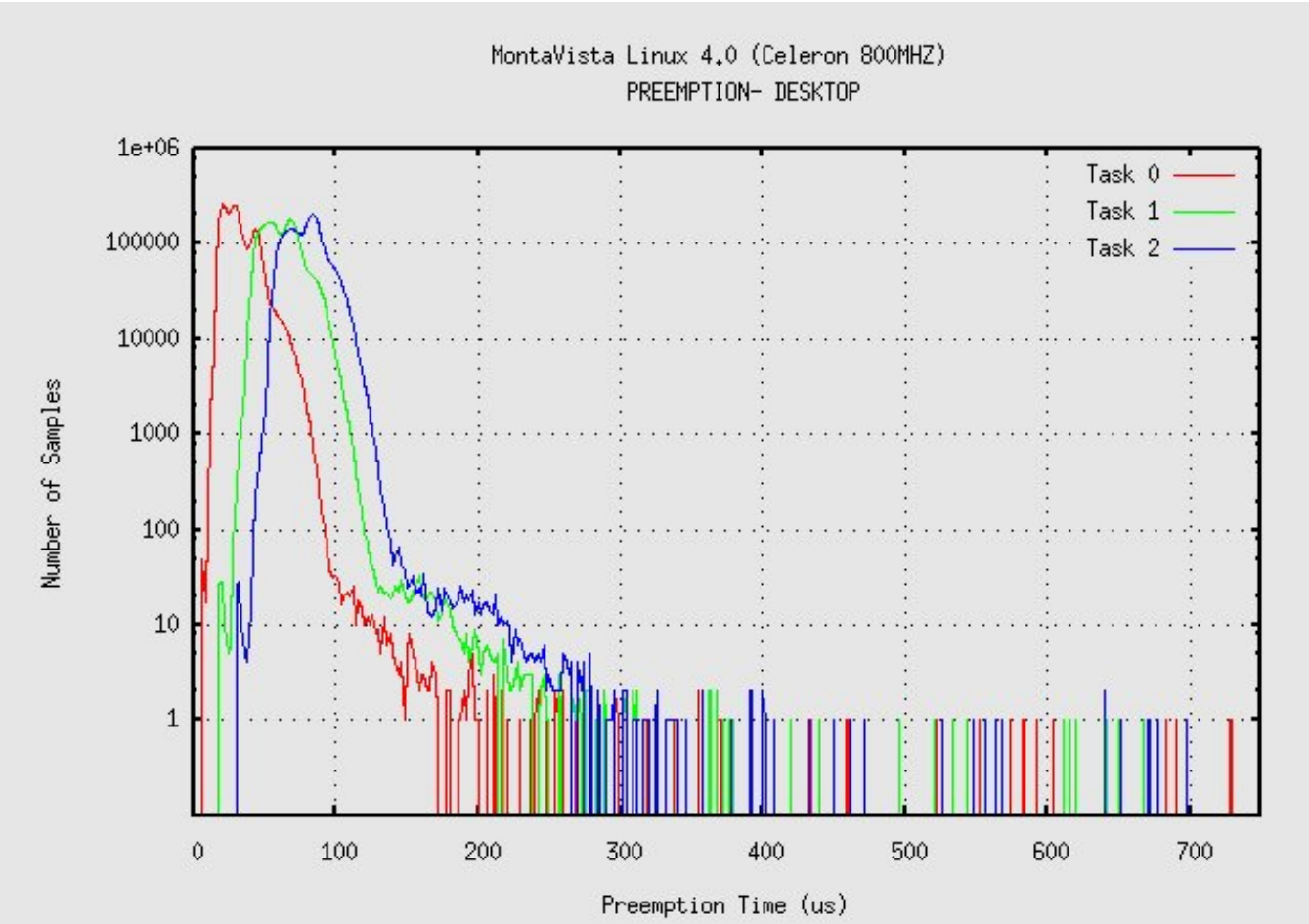
Source:

Linux 2.6 Kernel Preemption Latency – Legacy Preemption



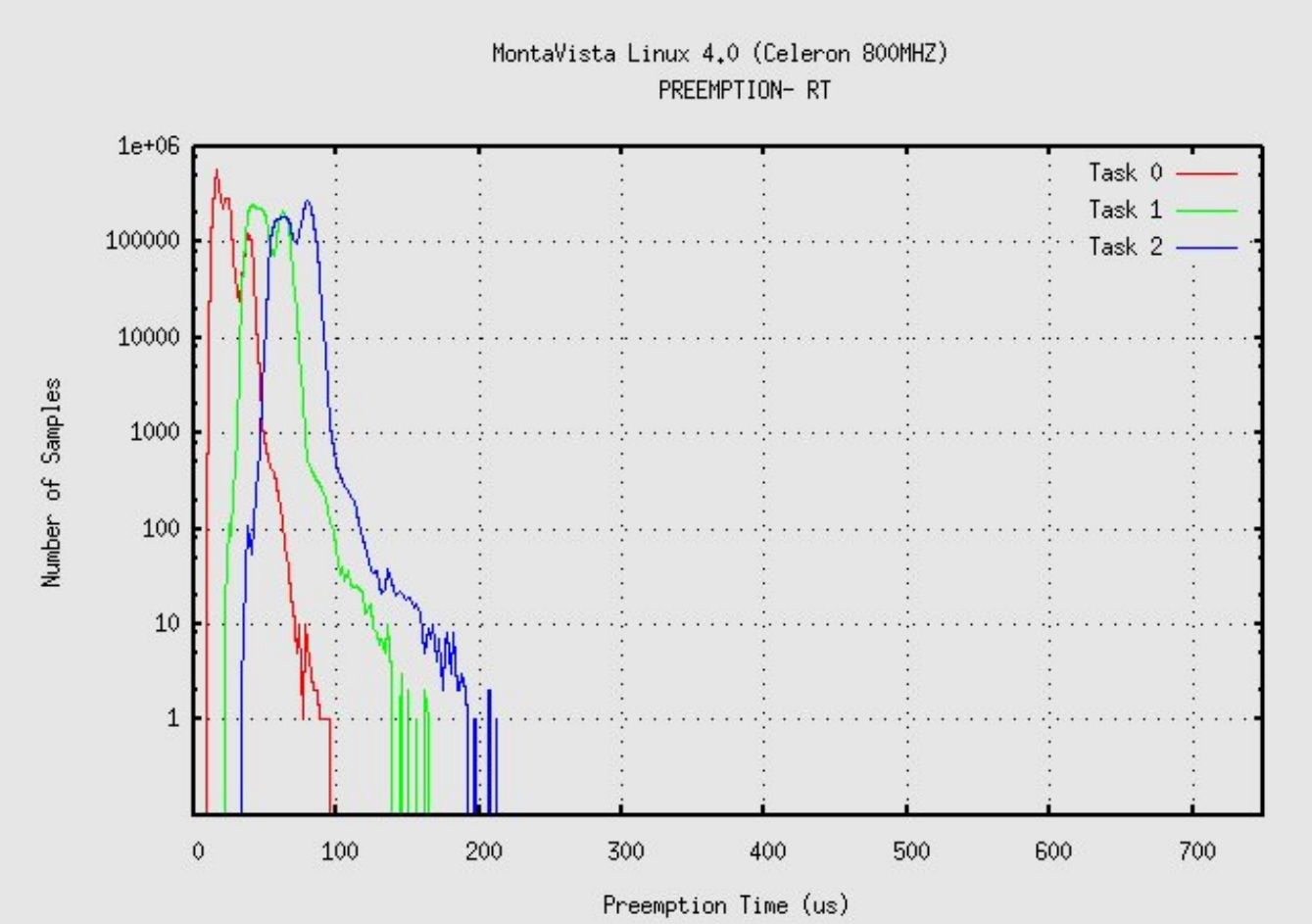
Source:

Linux 2.6 Kernel Preemption Latency – Legacy Preemption



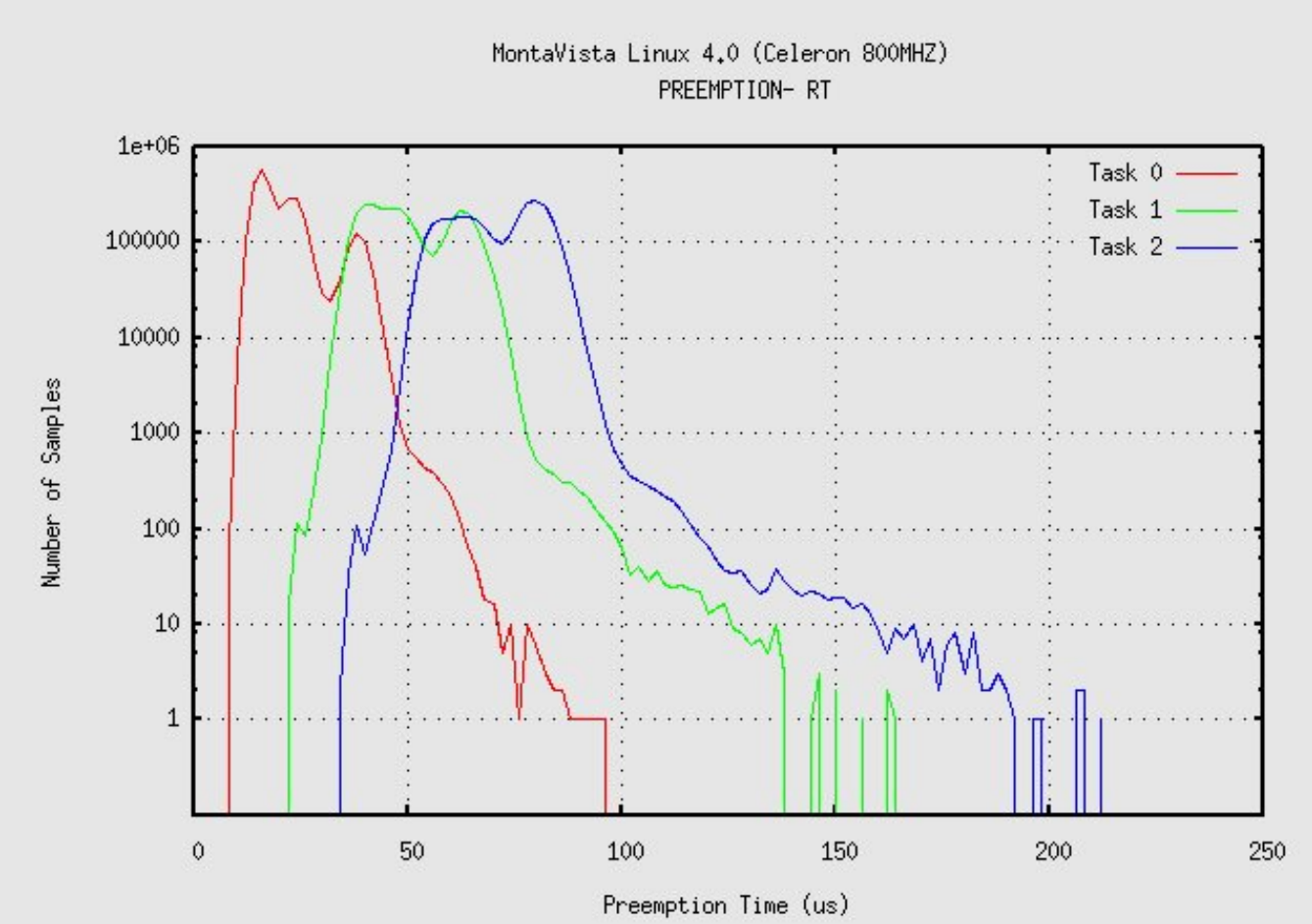
Source:

Linux 2.6 Kernel Preemption Latency - RT Preemption



Source:

Linux 2.6 Kernel Preemption Latency - RT Preemption



Source:

What it takes to Win

- For more information
  - Visit www.mvista.com
  - Contact me at sdietrich@mvista.com
  - Contact MontaVista Software at info@mvista.com

Thank you !


freescale™
semiconductor