What are Interrupt Threads and How Do They Work?

Interrupt Threads in Linux

Mike Anderson
Chief Scientist
The PTR Group, Inc.
mailto: mike@theptrgroup.com
http://www.theptrgroup.com

What We Will Talk About

- What is latency?
- Sources of latency in Linux
- The Molnar RT Patch for the Linux kernel
- Executing interrupt code in a thread context
- Interrupt threads in Linux
- Some notional performance comparisons
- Summary

A Definition of Latency

- Latency can best be described as the difference in time between when an event is signaled and when code starts to run
- Operating systems have:
  - Scheduling latency
  - Interrupt latency
  - And more…
- Because we deal with the real world, we must deal with latency
  - The real world is not a very deterministic place

Scheduling Latency

- Scheduling latency is the amount of time between when a high-priority thread becomes ready to run and when it gets the CPU
- Affected by:
  - Disabling the scheduler (BKL)
  - Non-preemptible system calls

Interrupt Latency

- The amount of time between when an interrupt is signaled and when the ISR begins to execute
- Affected by:
  - Long-duration ISRs
  - Disabling interrupts
  - Prioritization of interrupts

Taxonomy

- Deterministic execution
  - This means that code takes the same amount of time to run every time
    - The holy grail of real-time systems
- Real-time computing
  - Computing with a deadline
- Soft real time
  - Deadlines are squishy
    - Executing after the deadline has diminishing value
- Hard real time
  - If you miss the deadline, people get hurt or data is lost permanently

Source: graphpaper.com

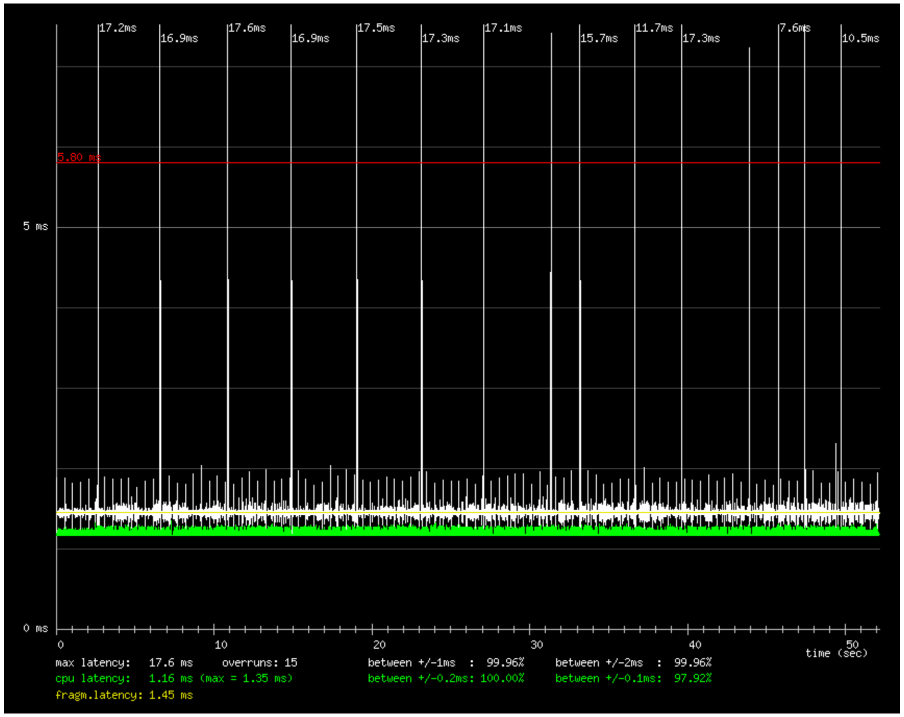
Real-time isn't Fair

- Embedded RTOS developers know that real-time applications are decidedly unfair
- In fact, many RTOSes don't support round-robin scheduling very well
  - Preemptive, priority-based is the scheduler of choice
    - That's SCHED_FIFO to us Linux folks
- This unfairness requires a different mindset from traditional Linux
  - Can take some getting used to

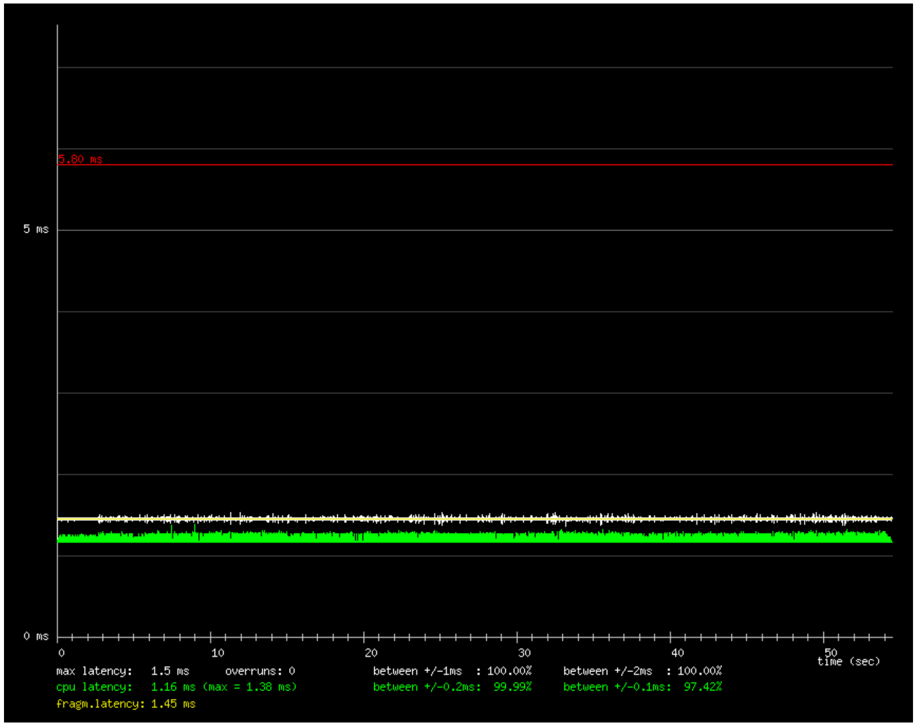
Preemption in the Linux Kernel

- Early Linux kernels were almost totally non-preemptible
- Preemption has been gradually phased into the Linux kernel over several years
- The "preemptible kernel" patch came in late in the 2.4 kernel series
  - Addressed many performance issues

Preemption in 2.4.17



```
max latency:   17.6 ms     overruns: 15          between +/-1ms   :  99.96%     between +/-2ms   :  99.96%
cpu latency:   1.16 ms (max = 1.35 ms)           between +/-0.2ms: 100.00%     between +/-0.1ms:  97.92%
fragm.latency: 1.45 ms
```



```
max latency:   1.5 ms      overruns: 0           between +/-1ms   : 100.00%     between +/-2ms   : 100.00%
cpu latency:   1.16 ms (max = 1.38 ms)           between +/-0.2ms:  99.99%     between +/-0.1ms:  97.42%
fragm.latency: 1.45 ms
```

Source: linuxjournal.com

MP3 without Preemption

MP3 with Preemption

2.6 Kernel R-T Regression

- When the 2.6 kernel was first released, performance dropped to below 2.4 levels
  - Critical regions of code were not preemptible
    - Spinlocks were being held too long
- This caused a lot of developers to stick with the 2.4 kernel for longer than everyone would have liked

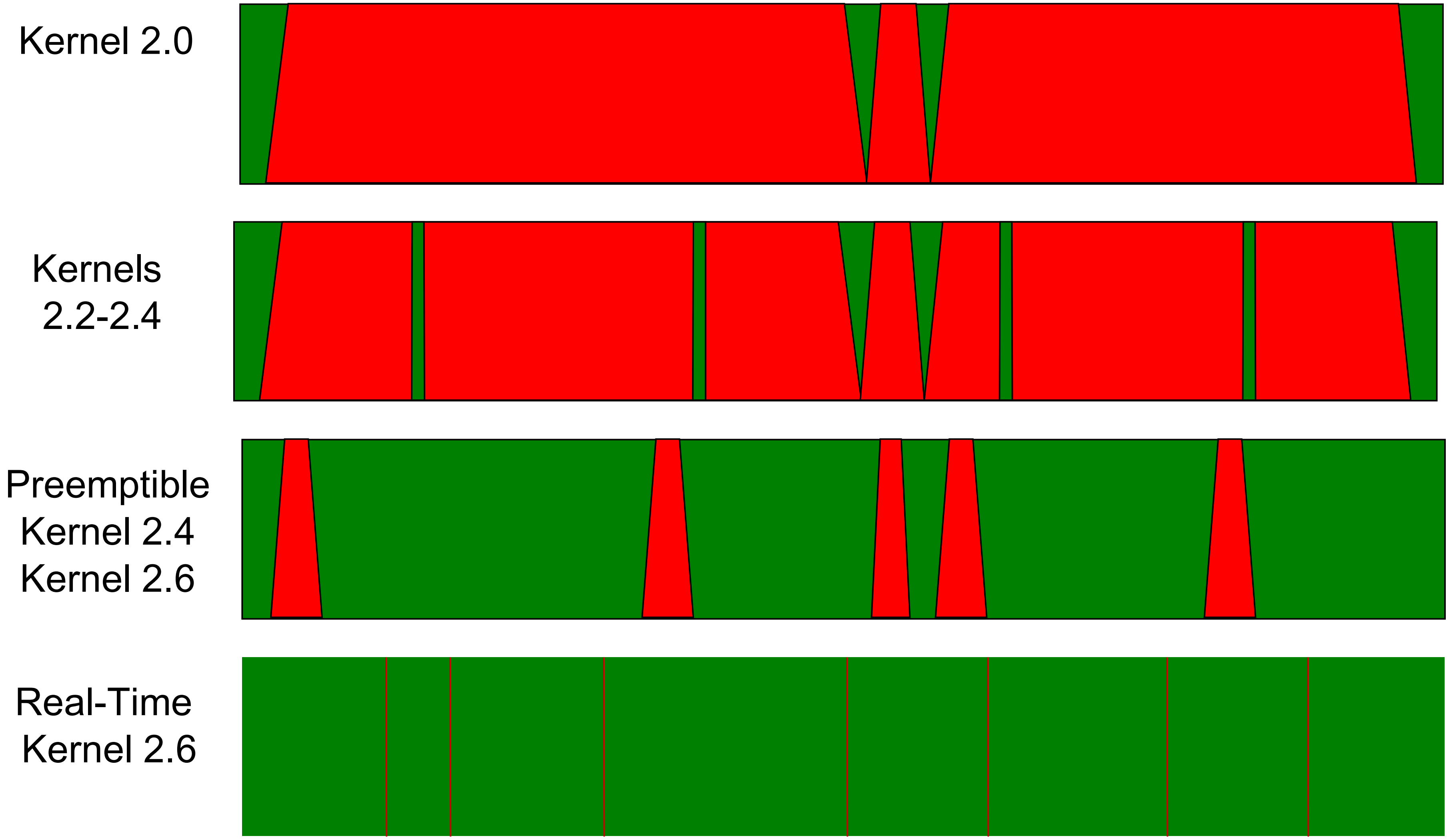
Fully Preemptible Kernel

- The low-latency desktop (PREEMPT_DESKTOP) work fixed most of the regressions in 2.6 responsiveness
  - However, full preemption is still not the default in the mainline kernel
    - Voluntary preemption is the default
- However, making the kernel more responsive means we're likely sacrificing total throughput
  - Preemption leads to more context switches

Audio Community Wanted More

- Even though the PREEMPT_DESKTOP option enabled soft real-time performance, the audio community wanted determinism
  - Needed to maintain sampling rates
- This lead to the development of the RT_preempt patch set
  - A.k.a. Molnar real-time patches

Linux Preemption Evolution



Kernel 2.0

Kernels 2.2-2.4

Preemptible Kernel 2.4 Kernel 2.6

Real-Time Kernel 2.6

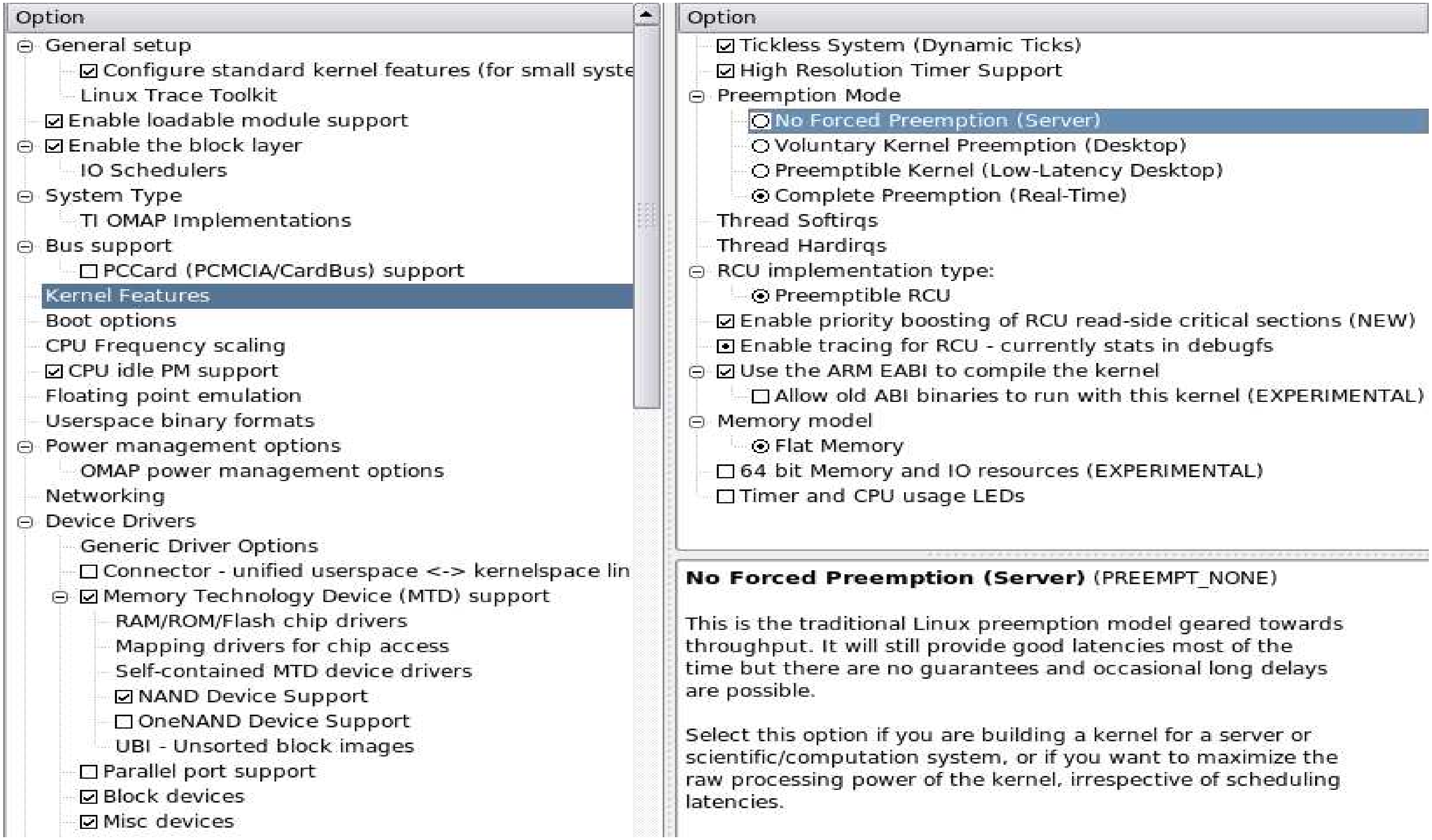
🟩 Preemptible 🟥 Non-Preemptible

Source: mvista.com

RT-Patches not Mainline Yet

- As of 2.6.28.7, the R-T patches are still not mainline
  - You can download them from kernel.org but not all kernels are supported
  - Or, use a distribution that integrates the patch set for you
- Beware: not all distros are created equal
  - Ubuntu 8.10 had an R-T kernel for 2.6.27 that only worked in uniprocessor mode
- Technically, 2.6.27 & 2.6.28 were skipped by R-T patch community
  - Focusing on 2.6.29 currently

Selecting Preemption Models



```
Option
⊖ General setup
      ☑ Configure standard kernel features (for small syste
      Linux Trace Toolkit
  ☑ Enable loadable module support
⊖ ☑ Enable the block layer
      IO Schedulers
⊖ System Type
      TI OMAP Implementations
⊖ Bus support
      □ PCCard (PCMCIA/CardBus) support
  Kernel Features
  Boot options
  CPU Frequency scaling
  ☑ CPU idle PM support
  Floating point emulation
  Userspace binary formats
⊖ Power management options
      OMAP power management options
  Networking
⊖ Device Drivers
      Generic Driver Options
      □ Connector - unified userspace <-> kernelspace lin
   ⊖ ☑ Memory Technology Device (MTD) support
          RAM/ROM/Flash chip drivers
          Mapping drivers for chip access
          Self-contained MTD device drivers
          ☑ NAND Device Support
          □ OneNAND Device Support
          UBI - Unsorted block images
      □ Parallel port support
      ☑ Block devices
      ☑ Misc devices
```

```
Option
  ☑ Tickless System (Dynamic Ticks)
  ☑ High Resolution Timer Support
⊖ Preemption Mode
      ○ No Forced Preemption (Server)
      ○ Voluntary Kernel Preemption (Desktop)
      ○ Preemptible Kernel (Low-Latency Desktop)
      ◉ Complete Preemption (Real-Time)
  Thread Softirqs
  Thread Hardirqs
⊖ RCU implementation type:
      ◉ Preemptible RCU
  ☑ Enable priority boosting of RCU read-side critical sections (NEW)
  ▣ Enable tracing for RCU - currently stats in debugfs
⊖ ☑ Use the ARM EABI to compile the kernel
      □ Allow old ABI binaries to run with this kernel (EXPERIMENTAL)
⊖ Memory model
      ◉ Flat Memory
  □ 64 bit Memory and IO resources (EXPERIMENTAL)
  □ Timer and CPU usage LEDs
```

No Forced Preemption (Server) (PREEMPT_NONE)

This is the traditional Linux preemption model geared towards throughput. It will still provide good latencies most of the time but there are no guarantees and occasional long delays are possible.

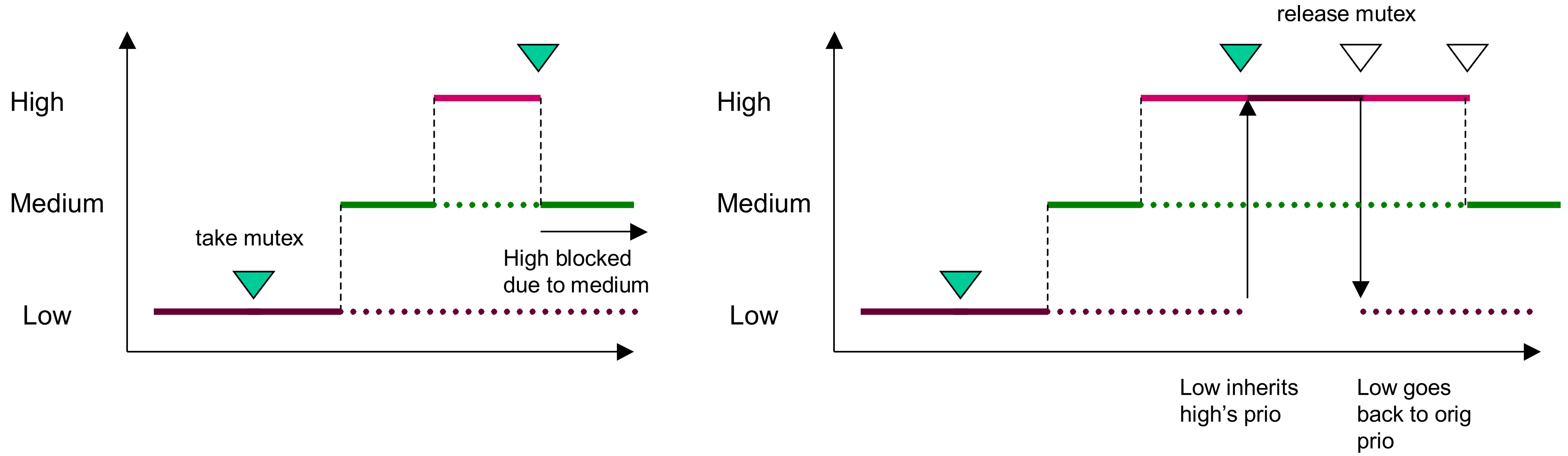
Select this option if you are building a kernel for a server or scientific/computation system, or if you want to maximize the raw processing power of the kernel, irrespective of scheduling latencies.

What R-T Patches Bring

- The major features of the R-T patch set are:
  - Spinlocks are replaced by PI-Mutexes
    - Support for priority inheritance
    - raw_spinlock() implements old spinlock behavior
  - Critical sections protected by spinlock_t and rwlock_t are now preemptible
  - Converted old Linux timer API into separate mechanisms for high-resolution and normal Linux kernel timers
    - Enables high-resolution POSIX timers in user space as well
  - Runs interrupt handlers in preemptible thread context
    - Both hard and soft IRQs can run in thread context

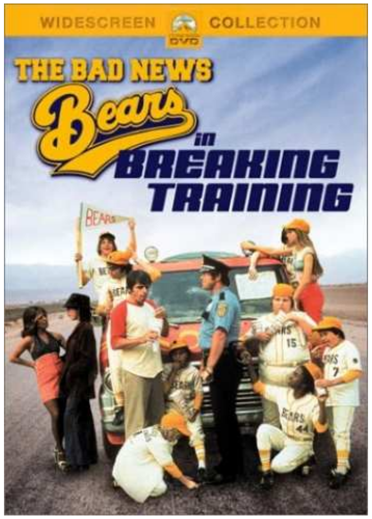
Priority Inversion

- A major problem for Linux and real-time work was something called priority inversion
  - Fixed with PI-Mutex



Breaking Training

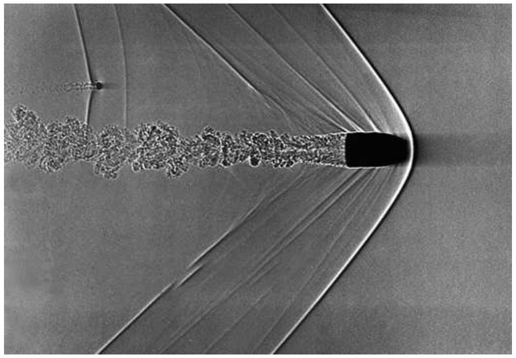
- We've been trained to think that interrupt code must be:
  - Fast
  - Atomic
  - Run in a special context
- But, what processor instructions *must* be run in interrupt context?
  - Return from interrupt
    - E.g., PPC RFI or x86 IRET
  - That's about it
- OK, what about fast and atomic?



Source: sportonvideo.com

How Fast is Fast Enough?

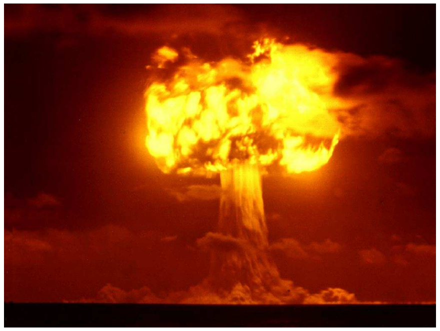
- Well, it depends…
  - Do we have a buffer that will be overrun?
  - When does the hardware interrupt get re-enabled?
- The kernel NAPI interface shows us that we can reduce the number of interrupts and still have excellent service
  - Buffering may be automatic and in hardware
- If we have to re-arm the interrupt in our ISR, then it's likely that the re-arm can wait until we get to it
  - Will data be lost? Is it important?



Source: nasa.gov

OK, How about Atomic?

- In Linux, if interrupts are marked as “slow” we can have interrupts interrupting interrupts
  - Our interrupt stack must handle worst case nesting
- It might be important to prioritize interrupts
  - We may want highest priority interrupt to run to completion
  - Unfortunately, many buses don’t support this



Source: deskpicture.com

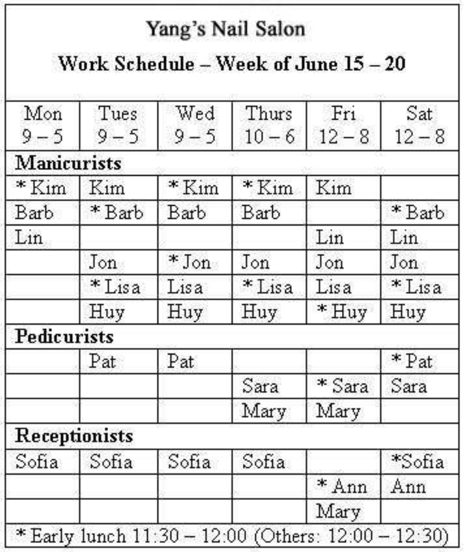
Interrupt Latency Reduction

- We’ve learned to use bottom halves to reduce interrupt latency
  - Lengthy copy operations can be moved to SoftIRQ/tasklet/work queue to re-enable interrupts while the copy proceeds
- Work queues are kernel threads
  - They’re scheduled, have priorities and can sleep
- The ISR top half can be a single schedule_work() call
  - This makes the top half deterministic

PTR

Scheduling Work

- The Linux scheduler is O(1)
  - Deterministic dispatch time
- This means that the work queue will be scheduled in constant time
- Since the work queue is a thread, it can run as long as needed (SCHED_FIFO)
  - Highest priority wins with the scheduler
- This means we can use R-T priorities to prioritize execution of bottom half
  - This is something we didn't have with tasklets/softIRQs



Yang's Nail Salon

Work Schedule – Week of June 15 – 20

Mon 9 – 5	Tues 9 – 5	Wed 9 – 5	Thurs 10 – 6	Fri 12 – 8	Sat 12 – 8
Manicurists					
* Kim	Kim	* Kim	* Kim	Kim	
Barb	* Barb	Barb	Barb		* Barb
Lin				Lin	Lin
	Jon	* Jon	Jon	Jon	Jon
	* Lisa	Lisa	* Lisa	Lisa	* Lisa
	Huy	Huy	Huy	* Huy	Huy
Pedicurists					
	Pat	Pat			* Pat
			Sara	* Sara	Sara
			Mary	Mary	
Receptionists					
Sofia	Sofia	Sofia	Sofia		*Sofia
				* Ann	Ann
				Mary	
* Early lunch 11:30 – 12:00 (Others: 12:00 – 12:30)					

Source: johnmh.com

R-T Patch to the Rescue

- What the R-T patch does is to institutionalize the work queue idea
  - All hardIRQs and softIRQs execute in high-priority kernel threads
- Highest priority wins
- Threaded hard and soft IRQs can be disabled via kernel command line or in /proc
  - hardirq-preempt=0/1
  - /proc/sys/kernel/hardirq_preemption
  - Similar options for softIRQs

Source: incredimazing.com

Threads are Created Automatically

- You don’t have to do anything special to run your code in a thread
  - request_irq() call creates the thread and includes your function

```
if (!(new->flags & IRQF_NODELAY))
 if (start_irq_thread(irq, desc))
       return -ENOMEM;
```

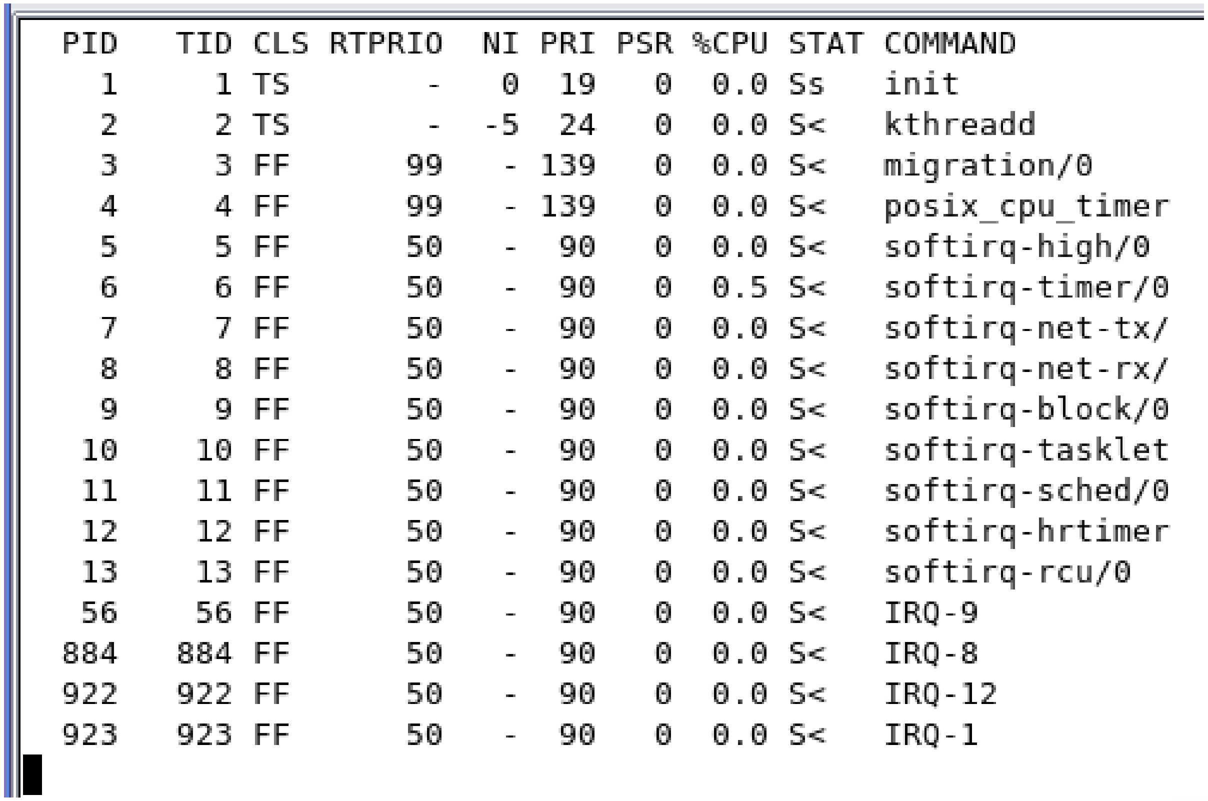
- This code will pass your ISR to the start_irq_thread function
  - Creates a kernel thread that calls your ISR code

The start_irq_thread Call

```
static int start_irq_thread(int irq, struct irq_desc *desc)
{
   if (desc->thread || !ok_to_create_irq_threads)
        return 0;

   desc->thread = kthread_create(do_irqd, desc, "IRQ-%d", irq);
   if (!desc->thread) {
        printk(KERN_ERR "irqd: could not create IRQ thread %d!\n", irq);
        return -ENOMEM;
   }

   /*
    * An interrupt may have come in before the thread pointer was
    * stored in desc->thread; make sure the thread gets woken up in
    * such a case:
    */
   smp_mb();
   wake_up_process(desc->thread);

   return 0;
}
```

View of Threaded IRQs

- With the RT patch set enabled, the hard/softIRQs are automatically run in kernel threads
  - Kernel threads use the kernel’s API and share the address space with drivers, the kernel etc.



```
  PID   TID CLS RTPRIO  NI PRI PSR %CPU STAT COMMAND
    1     1 TS       -   0  19   0  0.0 Ss   init
    2     2 TS       -  -5  24   0  0.0 S<   kthreadd
    3     3 FF      99   - 139   0  0.0 S<   migration/0
    4     4 FF      99   - 139   0  0.0 S<   posix_cpu_timer
    5     5 FF      50   -  90   0  0.0 S<   softirq-high/0
    6     6 FF      50   -  90   0  0.5 S<   softirq-timer/0
    7     7 FF      50   -  90   0  0.0 S<   softirq-net-tx/
    8     8 FF      50   -  90   0  0.0 S<   softirq-net-rx/
    9     9 FF      50   -  90   0  0.0 S<   softirq-block/0
   10    10 FF      50   -  90   0  0.0 S<   softirq-tasklet
   11    11 FF      50   -  90   0  0.0 S<   softirq-sched/0
   12    12 FF      50   -  90   0  0.0 S<   softirq-hrtimer
   13    13 FF      50   -  90   0  0.0 S<   softirq-rcu/0
   56    56 FF      50   -  90   0  0.0 S<   IRQ-9
  884   884 FF      50   -  90   0  0.0 S<   IRQ-8
  922   922 FF      50   -  90   0  0.0 S<   IRQ-12
  923   923 FF      50   -  90   0  0.0 S<   IRQ-1
```

Once it’s a Thread

- Now that your ISR is in the context of a thread:
  - You can change the priority using sys_sched_setscheduler()
  - Allows you to create an interrupt priority scheme
- You can also set CPU affinity masks to limit thread migration and optimize the use of processor caches
  - taskset() command from command line or via sys_sched_setaffinity() calls

Source: germes-online.com

Writing ISRs for Interrupt Threads

- Use the CONFIG_PREEMPT_RT #define to determine if you’re compiling for a kernel with the RT patch
- Do bottom halves still work?
  - Yes, but you don’t need to use them in this case
- You can use the in_irq() call to determine if you’re running in a normal IRQ
  - It returns false if you’re in an interrupt thread
- Use this to know if you need to schedule a tasklet or not

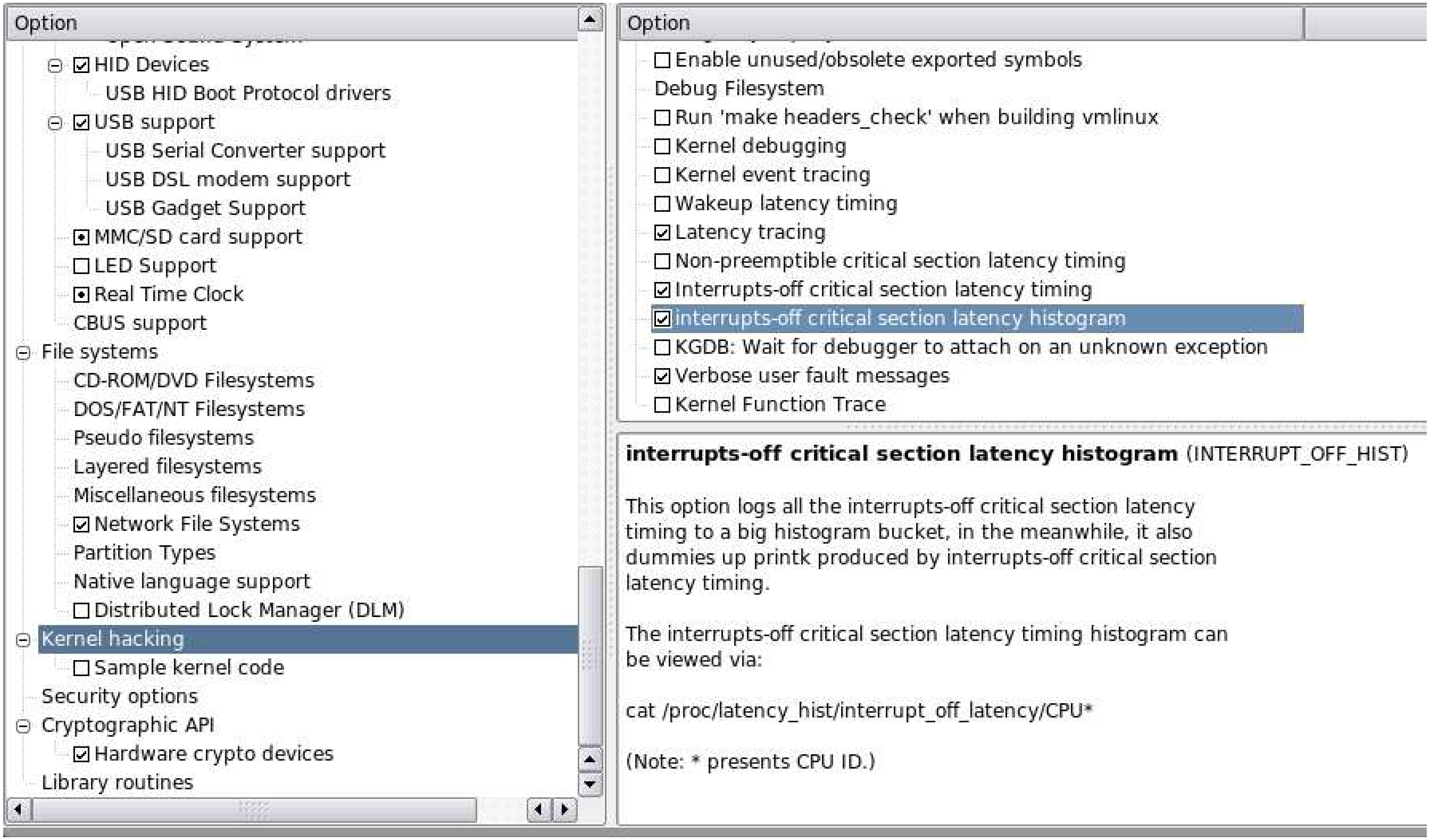
Threading isn't Always Best

- Just because you can thread your ISRs doesn't mean that you should
- The overhead of scheduling a thread doesn't make sense for simple devices
  - Timers, serial ports, etc.
    - Their behavior was already deterministic
- The request_irq() call has a solution to this
  - IRQF_NODELAY or IRQF_TIMER flags
  - If either of these flags are present, the ISR runs the old-fashioned way

Quantifying Performance

- Along with the R-T patch set are a number of performance measurement tools
- Instrumentation for interrupt latency, wakeup latency and histograms for worst offenders
  - Some latency measurements use the same entries in /proc
    - Only one of these measurements can be active at a time
  - Read the kernel configurator help to learn how to control them
- Beware: collecting data will change your timing
  - Don’t leave these measurements enabled in a shipping product!

Enabling Data Collection



Option

- ⊟ ☑ HID Devices
  - USB HID Boot Protocol drivers
- ⊟ ☑ USB support
  - USB Serial Converter support
  - USB DSL modem support
  - USB Gadget Support
- ⊡ MMC/SD card support
- □ LED Support
- ⊡ Real Time Clock
- CBUS support

⊟ File systems

- CD-ROM/DVD Filesystems
- DOS/FAT/NT Filesystems
- Pseudo filesystems
- Layered filesystems
- Miscellaneous filesystems
- ☑ Network File Systems
- Partition Types
- Native language support
- □ Distributed Lock Manager (DLM)

⊟ Kernel hacking

- □ Sample kernel code

Security options

⊟ Cryptographic API

- ☑ Hardware crypto devices

Library routines

Option

- □ Enable unused/obsolete exported symbols
- Debug Filesystem
- □ Run 'make headers_check' when building vmlinux
- □ Kernel debugging
- □ Kernel event tracing
- □ Wakeup latency timing
- ☑ Latency tracing
- □ Non-preemptible critical section latency timing
- ☑ Interrupts-off critical section latency timing
- ☑ interrupts-off critical section latency histogram
- □ KGDB: Wait for debugger to attach on an unknown exception
- ☑ Verbose user fault messages
- □ Kernel Function Trace

interrupts-off critical section latency histogram (INTERRUPT_OFF_HIST)

This option logs all the interrupts-off critical section latency timing to a big histogram bucket, in the meanwhile, it also dummies up printk produced by interrupts-off critical section latency timing.

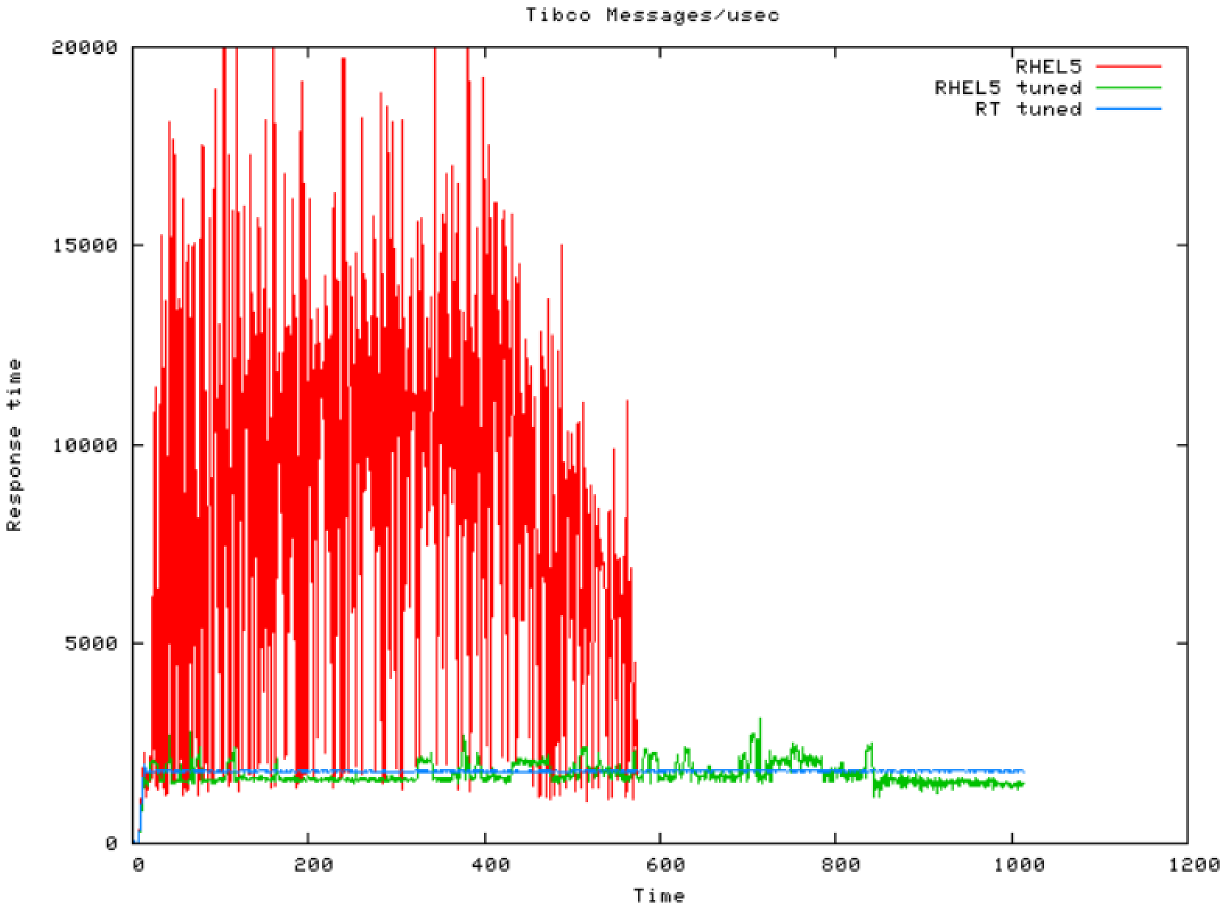
The interrupts-off critical section latency timing histogram can be viewed via:

```
cat /proc/latency_hist/interrupt_off_latency/CPU*
```

(Note: * presents CPU ID.)

Comparative Benchmarks #1

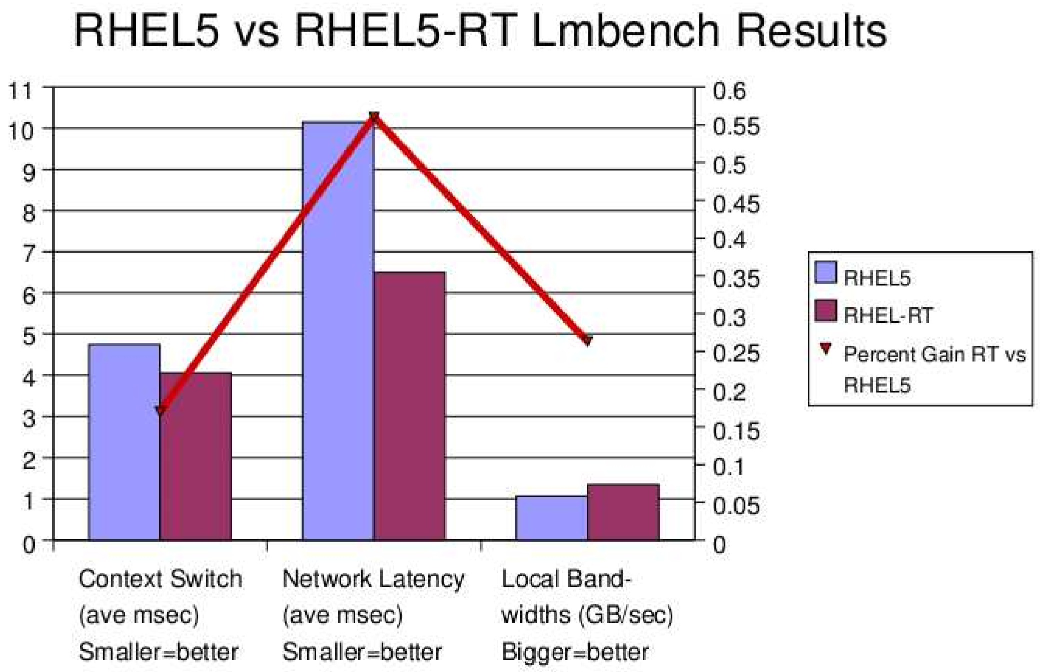
- RH did some benchmarking for their collateral material
- Focused on message passing on x86
- Shows relative stability of R-T kernel compared to stock kernel



Source: redhat.com

Comparative Benchmarks #2

- Lmbench shows better context switch times, smaller average network latency and an increase in local bandwidth with the R-T patch enabled
- Some applications may see an increase in throughput due to preemption
  - Instead of waiting on hardware to respond



Source: redhat.com

Interesting Links

- Real-time patches
  - http://www.kernel.org/pub/linux/kernel/projects/rt/
- R-T kernel How-To on the R-T Wiki
  - http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO
- Thomas Gleixner's R-T tests
  - http://www.kernel.org/pub/linux/kernel/people/tglx/rt-tests/
- linux-rt-users mailing list
  - http://vger.kernel.org/vger-lists.html#linux-rt-users
- Tibco messaging benchmark
  - http://www.tibco.com/software/messaging/enterprise_messaging_service
- Red Hat R-T Performance Whitepaper
  - http://www.redhat.com/f/pdf/mrg/mrg_realtime_whitepaper.pdf

Summary

- Real-time means being fast enough
  - Determinism is nice to have when you can get it
    - Some applications, like audio, require it
- The R-T patch set includes many key enhancements including interrupt threads that make the kernel more responsive
  - However, some throughput may be sacrificed
- The use of interrupt threads enables developers to prioritize interrupts and make interrupt servicing more deterministic
  - Jitter goes way down
  - May require some system redesign to take full advantage of threading
- The R-T patch set is making its way into enterprise and desktop applications via SUSE, RH and Ubuntu
  - Hopefully, it will be mainstreamed soon