Realtime In the Enterprise

(definitely not your grandfather's realtime)

Clark Williams
Team Lead, MRG Realtime
Red Hat Inc.

Realtime
Back in the Day...

- Usually about meeting strict timing deadlines
- Special purpose hardware
  - Uniprocessor
    - Many times a relatively weak 8 or 16-bit processor
  - Limited RAM
  - Limited mass storage (usually EPROM)
  - Serial communications
  - Single purpose system
  - I/O devices were generally purpose-built for a task

Realtime
Today

- Less about meeting deadlines and more about reducing the time between an event and the servicing of that event (latency)
- High-end, general-purpose systems
  - Multi-core processors
  - Gigabytes of RAM
  - Large mass storage array
  - High-speed network connections
  - More use of standardized I/O devices

General Purpose Systems Make RT Hard to Do

- SMP complicates the scheduler
- Large numbers of interrupts + large numbers of cores complicate interrupt handling
- Migration costs cause us to lose performance as number of processor cores rise
- Work is being done on different schedulers and new ways of mutual exclusion (EDF and RCU)
- RT is a *magnifier* of existing kernel problems
  - “If RT sees it now, mainline will see it in 3-5 years”

Enterprise Realtime Applications

- Financial
  - Market data (operating on a multicast stream)
  - Trading (transaction oriented)
- Military
  - C^3 (Command, Control, Communications)
  - Navigation (image recognition)
- Multimedia
  - Audio recording/playback (millisecond deadlines)
- Seismic Analysis

Realtime Applications differ from Most Regular Applications

- Fairness goes out the window
  - Some threads are more important than others
  - Realtime priorities used to indicate relative importance of threads (SCHED_FIFO)
  - Resources are locked down (e.g. mlock(2))
- Systems are *partitioned*
  - Core (or groups of cores) dedicated to applications
  - Interrupt affinity used to move important interrupts to lightly used core(s)
    - Tuna or taskset is useful for this

Issues with RT Linux on General Purpose Hardware

- Throughput Loss
  - Determinism has its cost
  - 10-20% loss over vanilla kernel in network throughput loads (e.g. netperf)
  - Work is ongoing to identify and remove performance hotspots
- Unexplained Latencies
  - System Management Interrupts (SMIs)

RT Tools

- Many tools being developed for finding/fixing problems and tuning RT Linux
  - perf
  - ftrace
  - rteval
  - hwlat_detector
  - tuna
- All of these tools run on mainstream Linux

perf

- Interface to kernel Performance Events system
- Command line tool with pluggable sub-commands, similar to 'git':
  - perf top – provides updating view of top kernel routines
  - perf record – profiles a command
  - perf report – analyzes data from a 'perf record' run
  - perf annotate – another way to view 'perf record' data
  - perf stat – gather perf counter statistics on a command
- Main use is to analyze kernel performance
- Kept in kernel source tree: tools/perf

ftrace

- Originally grew from Ingo Molnar's latency-tracer
- Steven Rostedt generalized it and added functions
- Many, many tracer's available to gather different views of kernel execution
- More info in kernel tree in Documents/tracing
- See Steven's talk this afternoon
  - git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/linux-2.6-trace.git

rteval

- Used to measure latency on a system under moderate load
- Two loads:
  - Kernel compile:
    - make -j<ncores*2> bzImage modules
  - hackbench
- Uses cyclictest to measure latency
- Produces XML output with statistical analysis of run data
- git://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/rteval.git

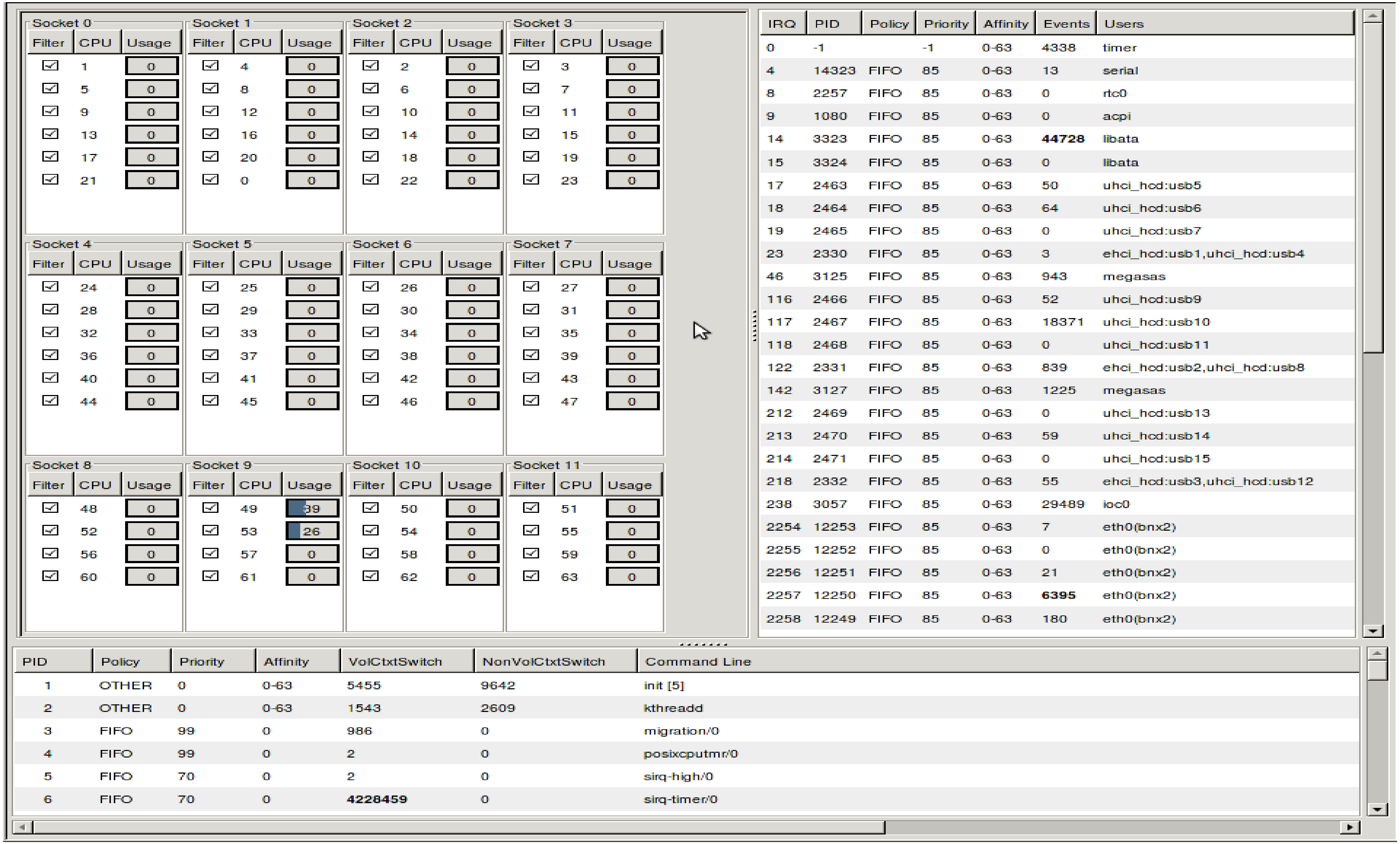
hwlat_detector

- Kernel module written by Jon Masters used to detect hardware related latencies (e.g. SMIs)
  - debugfs interface
- Uses stop_machine() to poll system clock, looking for gaps where control transferred away from the OS
- Python wrapper in rt-tests package, named hwlatdetect.py, drives module
- Tentative plan is to rewrite hwlatdetect.py functionality as perf sub-command

tuna

- Graphical tool for interactively tuning a running RT system
- Has command line backend, so things done in GUI can be scripted later
- Still under development
  - Python + GTK
  - Should be out in Fedora 12 (already in MRG)
  - git://git.kernel.org/pub/scm/linux/kernel/git/acme/tuna.git

tuna Screenshot



RT Things We Want to Get Upstream

- Finish up threaded IRQ support
  - Lots of drivers need to be evaluated/converted
- Full preemption (sleeping spinlocks)
  - Still working on how to indicate that locks will change behavior based on config options
- More tools
  - tuna
  - rteval
  - perf framework needs extending

Questions?