Real-time Linux

FreedomHEC 2012 Taipei
Insop Song
Ericsson

Contents

- Linux & Real-time
- Linux scheduler
- Latency
- Real-time Linux (PREEMPT_RT)
- Benchmark test results
- Tools
- Tips
- EDF scheduler
- Summary

Linux

- Linux is a general purpose OS
  - It’s about fair scheduling
  - Main goal is to optimize the average throughput
  - Not deterministic
  - Large latencies

Real time systems

- Real Time system is ..
  - not “real fast”
  - determinism
- Types of real-time systems are …
  - Hard real-time
    - Missing a deadline is a total system failure [wiki]
    - Mission critical systems
  - Soft real-time
    - The usefulness of a result degrades after its deadline, thereby degrading the system's quality of service [wiki]
    - Non-mission critical systems

Linux Scheduler

Linux scheduler 1

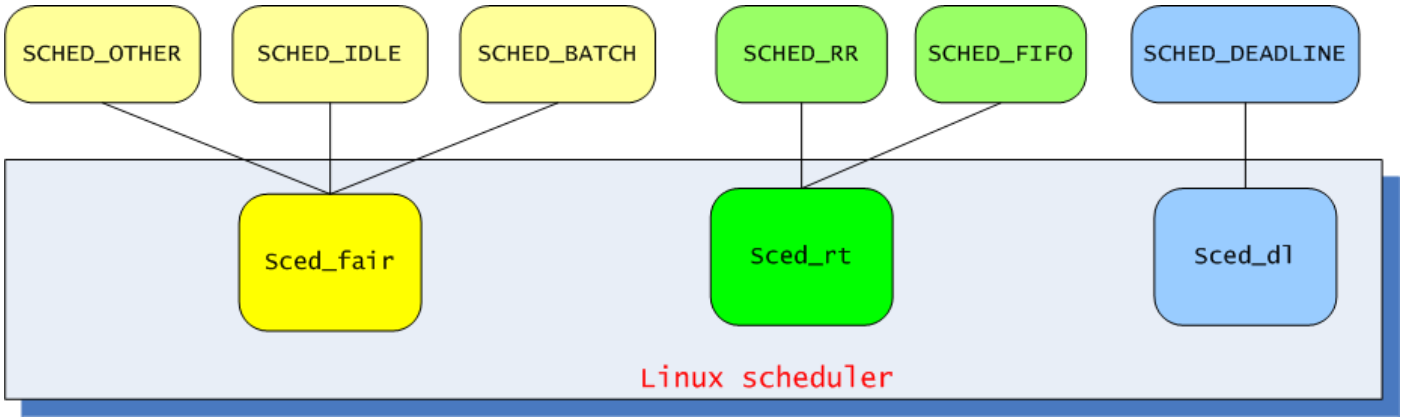
- Linux kernel scheduler
  - Decides which process to run next
  - Allocates processor(s)' time among run-able processes
  - Realizes multi-tasking in a single processor machine
  - Schedules SMP as well
  - Affects how the system behaves, responsiveness

Linux scheduler 2

- Scheduler framework
  - Kernel supports various scheduling policies by plug-in
  - Scheduling classes contains details of the scheduling policy
    - Decides which task runs next
    - Each operation can be requested by the global scheduler;
    - Allows for creating of the generic scheduler without any knowledge about different scheduler classes

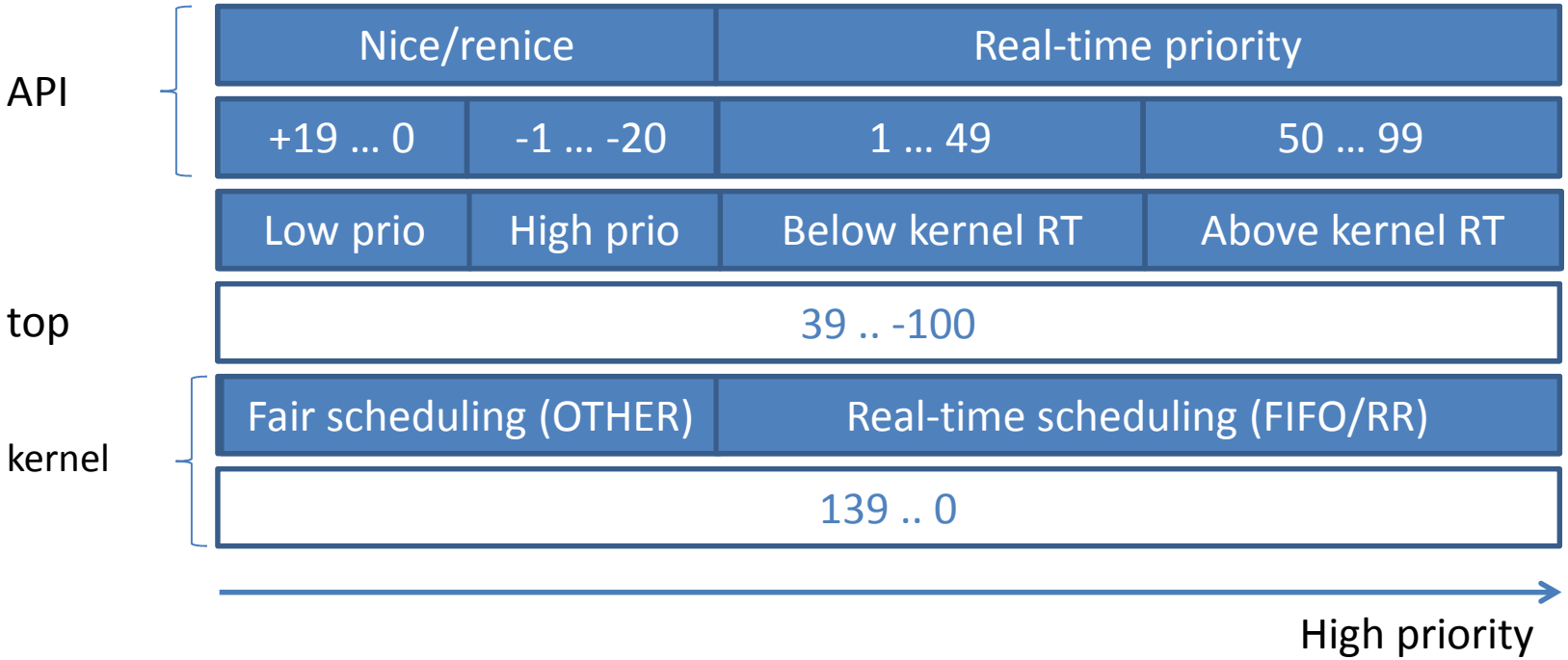
Linux schedule classes

- Supports multiple schedule classes
  - SCHED_OTHER
    - Default scheduler, non-real-time scheduling
  - SCHED_FIFO
    - Use FIFO real-time scheduling
  - SCHED_RR
    - Use round-robin real-time scheduling
  - SCHED_DEADLINE (not in mainline yet)
    - EDF scheduling with reservation-based scheduler



Linux scheduler priority

- Two separate priority ranges
  - Nice value: -20 … +19 (19 being the lowest)
  - Real-time priority: 0 … 99 (higher value is higher prio)
  - *ps* priority: 0 … 139 (0: lowest and 139: highest)



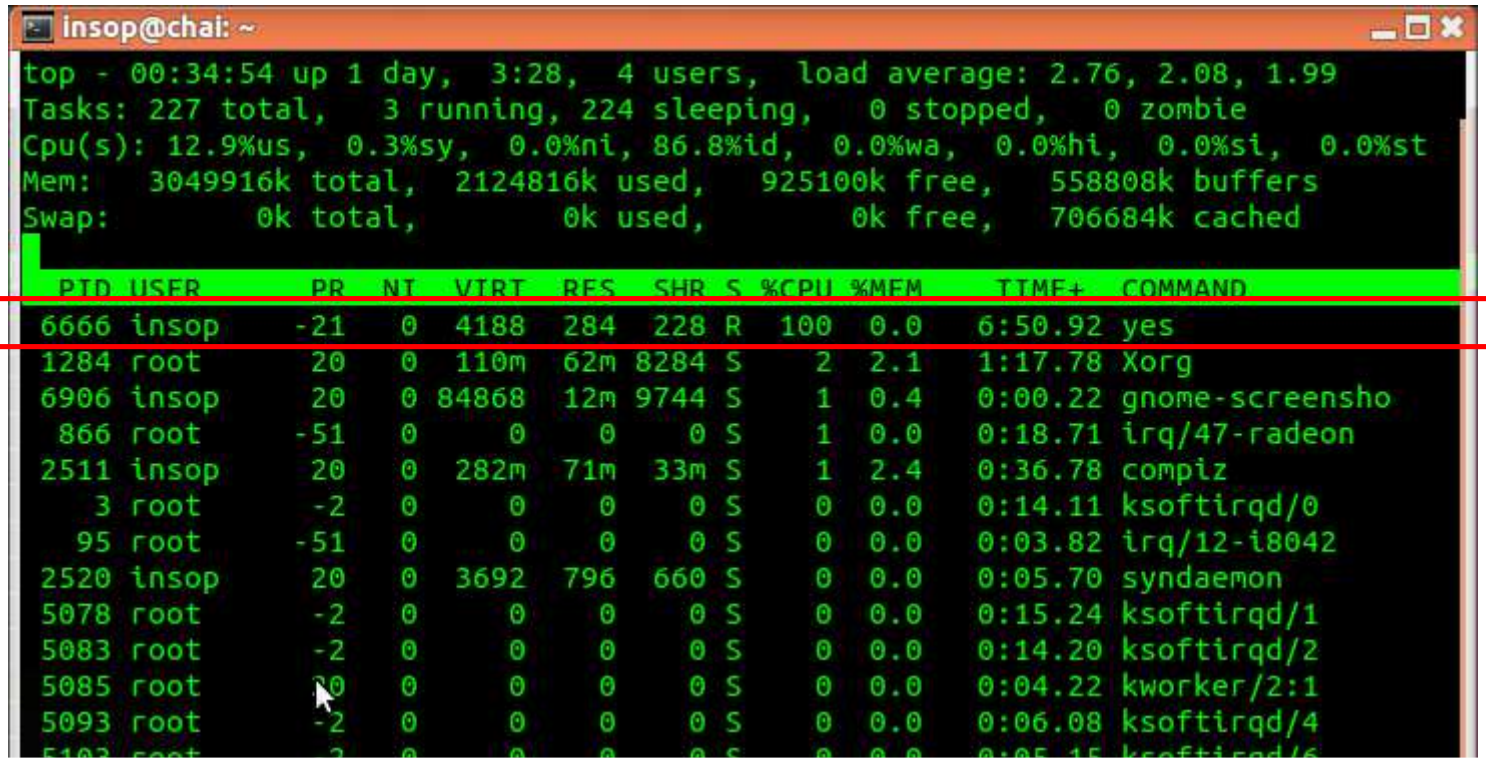
Linux schedule system calls

- Scheduler related sysetm calls

System call	description
nice()	Sets a process' nice values
sched_setscheduler()	Sets a process' scheduling policy
sched_getscheduler()	Gets a process' scheduling policy
sched_setparam()	Sets a process' real-time policy
sched_getparam()	Gets a process' real-time policy
sched_get_priority_max()	Gets the max real-time priority
sched_get_priority_min()	Gets the min real-time priority
sched_rr_get_interval()	Gets a process' timeslice value
sched_setaffinity()	Sets a process' processor affinity
sched_getaffinity()	Gets a process' processor affinity
sched_yield()	Temporarily yields the processor

Linux scheduler

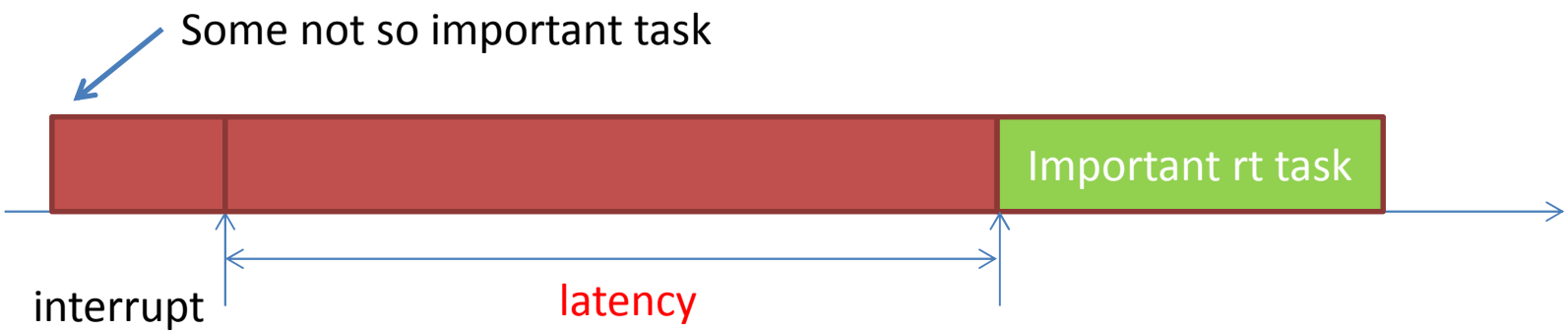
- How to config priority
  - *> chrt [option] -p [prio] pid*
    - -r: SCHED_RR
    - -f: SCHED_FIFO
  - sched_setscheduler()
  - Checking priority, nice, rt-priority
    - *> ps -eo pid,class, pri,nice,rtprio,comm*



Latency

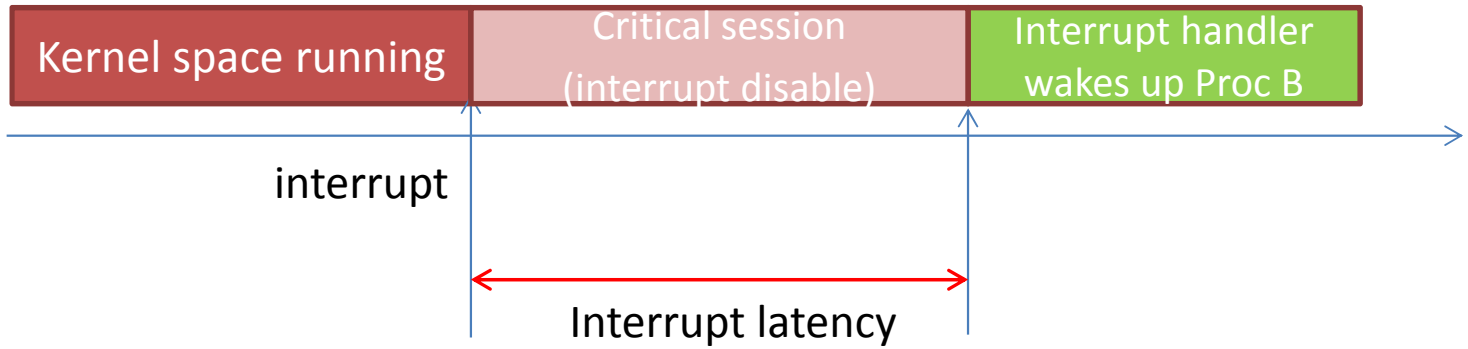
Latency in Kernel

- Real time means external event should be handled within the bounded time
- Interrupt handler responds to the event and inform user-space process
- Latency
  - Time taken from external interrupt till a user-space process to react to the interrupt



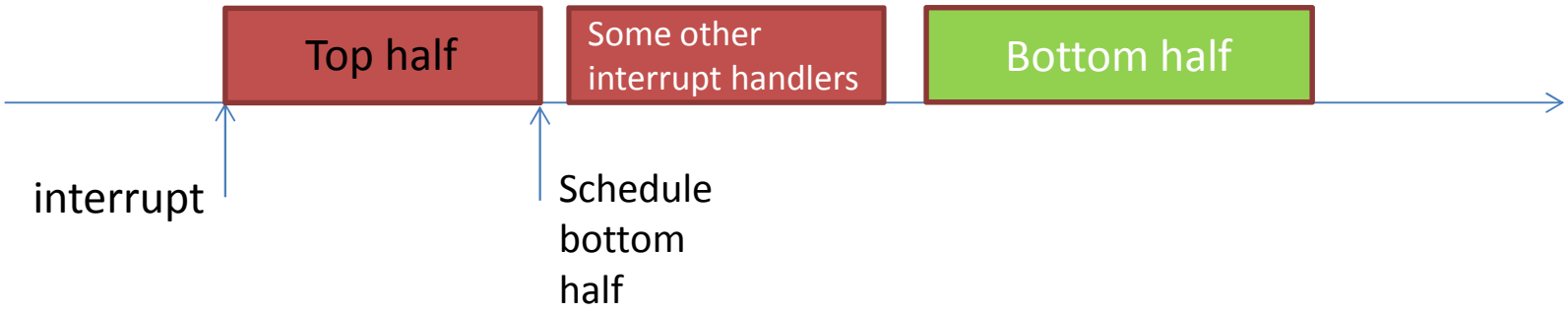
Source of interrupt latency

- Interrupt latency
  - Time elapsed before the interrupt handle starts
- One of the main cause of the latencies
  - Unbounded critical secsion (interrupt disabled section)
    - By spinlock and explicit interrupt disable



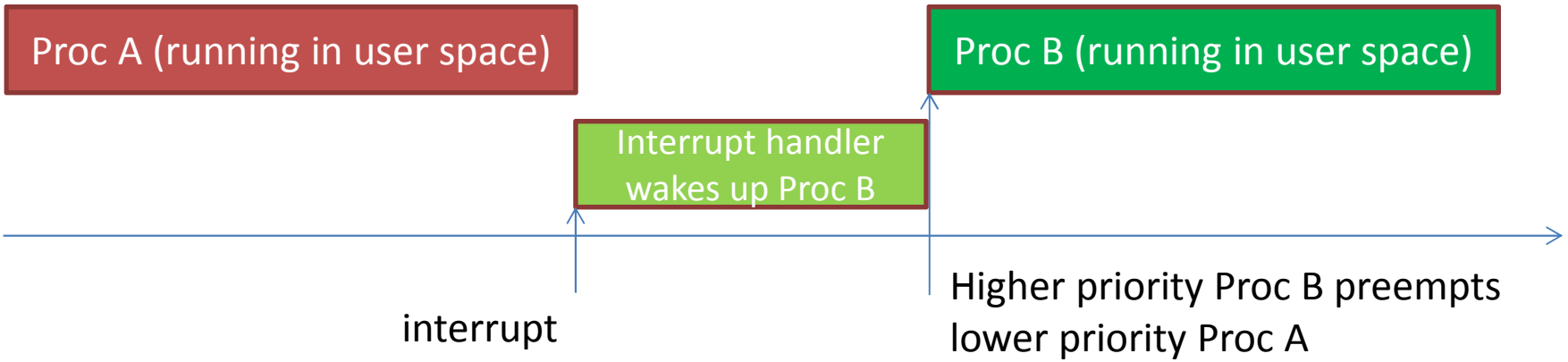
Interrupt handler

- Interrpt handlers are splitted into two parts
  - A top-half:
    - Process, as quickly as possible, the work during interrupt disabled, such as queue the information for the bottom-half
    - Schedule the bottom-half
  - A bottom-half
    - Process the required tasks from the triggered intrrupt



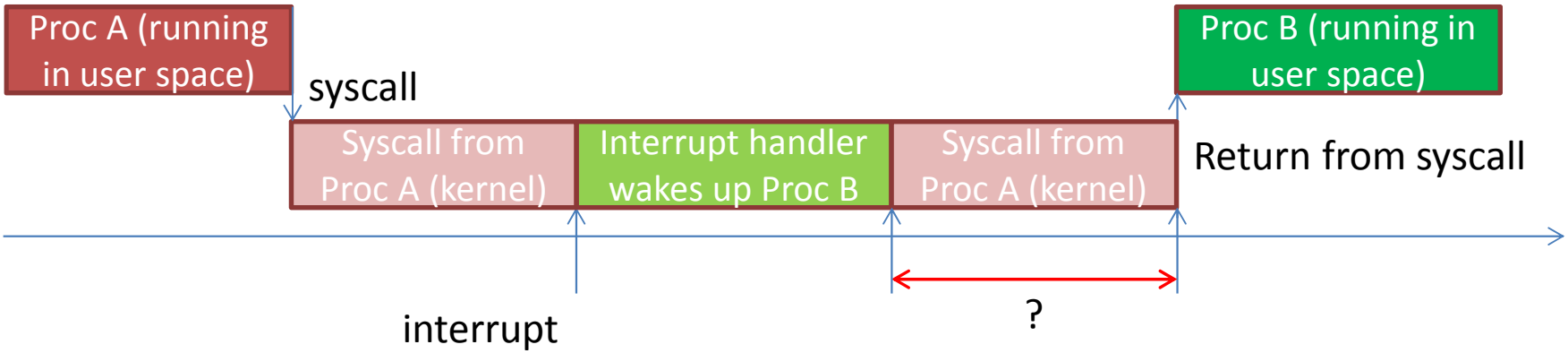
Kernel preemption 1

- Kernel can be configured preemptible
- If an intrrupt is arrived during Proc A (lower priority) runs
  - Proc A is intrrupted
  - Interrupt handler triggers another user space proc (Proc B, higher priority then Proc A)
- When Interrupt handler finished
  - Proc B preempted previously running Proc A



Kernel preemption 2

- Interrupt triggered during Proc A was invoking syscall, which runs in the kernel space
  - Interrupt handler wakes up Proc B
- However, Proc A running in the kernel space will not be preempted
  - *By default*, the Linux kernel does not do *kernel preemption*
- The time before Proc B runs (scheduler to be called) is unbounded



Scheduler latency/duration

- Scheduler latency
  - Time elapsed before the scheuler runs
- Scheduler duration
  - Time taken to decide what is the next task to run
  - SCHED_FIFO & SCHED_RR use bit map to find out the next task
  - CFS (Complete Fair Scheduler) uses Red-black tree as a sorted queue

Other issues

Other souces of latencies

- Linux is highly based on virtual memory, as provided by an MMU,
  - so that memory is allocated on demand. Whenever an application accesses code or data for the first time, it is loaded on demand, which can creates huge delays.
- Many C library services or kernel services are not designed with realtime constraints in mind.

Real-time Linux

Real time approaches

- Two major approaches real time Linux
- Approaches
  - Real-time Linux (PREEMPT_RT patch)
    - Allows preemption, so minimize latencies
    - Execute all activities (including IRQ) in “schedulable/thread” context
    - Many of the RT patch have been merged
  - Linux extention
    - Add extra layer between hardware and the Linux kernel to manage real-time tasks separately
    - (This will not be covered in this presentation)

Approaches in Real-time Linux

- PREEMPT_RT
  - A project led by kernel developers including Ingo Molnar, Thomas Gleixner, and Steven Rostedt
    - Large testing efforts at RedHat, IBM, OSADL, Linutronix
  - Goal is to improve real time performance
  - Separate branch, not yet in main-line kernel, but it is getting close
    - Many of small features from PREEMPT_RT are in main-line kernel over the years

PREEMPT_RT

- Background
  - Replace non-preemptible constructs with preemptible ones
- Make OS preemptible as much as possible
  - except preempt_disable and interrupt disable
- Make Threaded (schedulable) IRQs
  - so that it can be scheduled
- spinlocks converted to mutexes (aka sleeping spinlocks)
  - Not disabling interrupt and allows preemption
  - Works well with thread interrupts

Approaches in Real-time Linux

- Mainline kernel
  - CONFIG_PREEMPT option in mainline
  - Allows preemption
  - Mainline 2.6 & 3.0 kernel
    - CONFIG_PREEMPT_NONE
      - No forced kernel preemption
    - CONFIG_PREEMPT_VOLUNTARY
      - Explicit preemption points in kernel
    - CONFIG_PREEMPT
      - All kernel code (except critical section) preemptible

High resolution timers

- Resolution of the system timer (HZ)
  - 100 HZ to 1000 HZ depends on arch and configuration
  - Resolution of 10 ms to 1 ms
  - Adds overheads
- High resolution timers introduced since 2.6.21
  - Allows to use availble h/w timers to program interrupts
  - Granularity nano second

Threaded interrupts

- Handle interrupt by interrupt handler thread
- Interrupt handlers run in normal kernel threads
  - Priorities can be configured
- Main interrup handler
  - Do minimal work and wake-up the corresponding thread
- Thread interrupts allows to use spleeping spinlocks
- Merged since 2.6.30
  - Conversion is manual in mainline
  - *cf*) in PREEMPT_RT, all interrupt handlers are switched to threaded interrupt

Threaded interrupt handler

- Priority must be set
  - Interrupt handler threads
  - Softirq threads
  - Other kernel threads
  - Real time application processes/threads

PREEMPT_RT

- Almost all kernel space is now preemptible
  - An interrupt can occur at any time
  - The woken up processes by interrupt can start immediately
- Treaded IRQs
  - Kernel thread per ISR

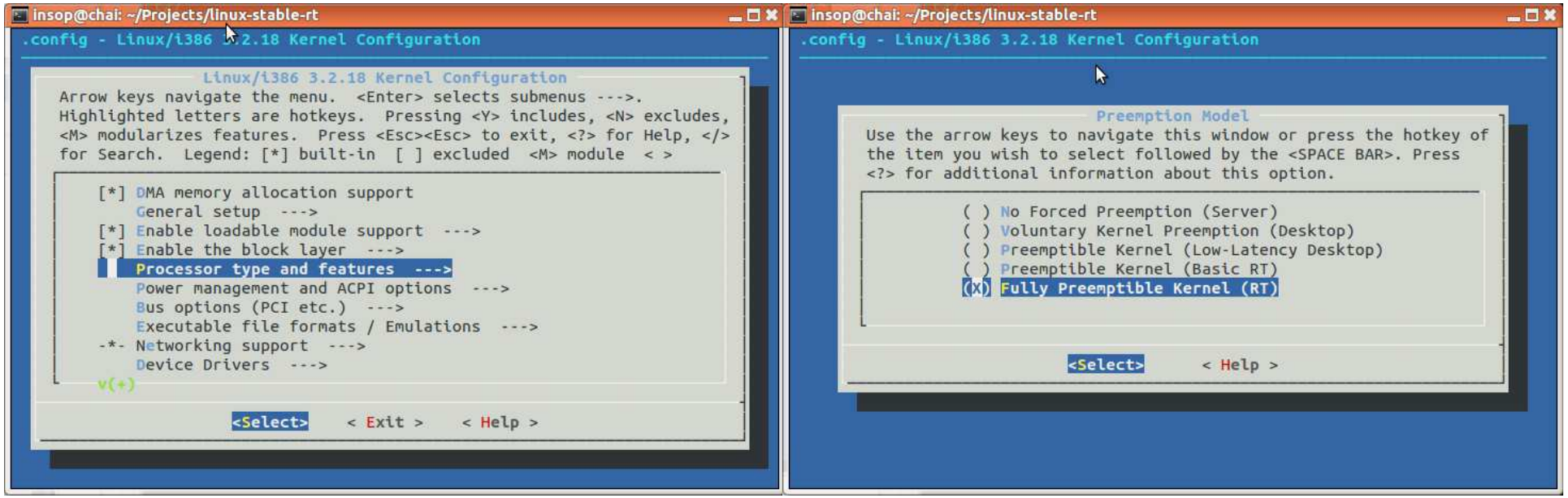
Configuring PREEMPT_RT

PREEMPT_RT config

- Where to get:
  - http://www.kernel.org/pub/linux/kernel/projects/rt/
- Pre 3.0
  - Use -rt patch, i.e. download mainline & apply patch
- Post 3.0
  - Patch as well as git availble
    - git clone git://git.kernel.org/pub/scm/linux/kernel/git/rt/linux-stable-rt.git
- Yocto project supports PREEMPT_RT as well
  - Reference images
    - core-image-rt
    - core-image-rt-sdk
  - Detail information:
    - http://www.yoctoproject.org/docs/current/poky-ref-manual/poky-ref-manual.html

PREEMPT_RT config

- Processor type and features
  - Preemption model
    - Fully Preemtible Kernel (RT)
  - High-resolution timer support
  - disabled all Power Management Options like ACPI or APM



RT hello world

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <sched.h>
#include <sys/mman.h>
#include <string.h>

#define MY_PRIORITY (49) /* we use 49 as the PRREMPT_RT use 50
                        as the priority of kernel tasklets
                        and interrupt handler by default */

#define MAX_SAFE_STACK (8*1024) /* The maximum stack size
which is
                                guranteed safe to access without
                                faulting */
#define NSEC_PER_SEC    (1000000000) /* The noof nsecs per sec. */
void stack_prefault(void) {
        unsigned char dummy[MAX_SAFE_STACK];
        memset(dummy, 0, MAX_SAFE_STACK);
        return;
}
…
```

https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

```
int main(int argc, char* argv[])
{
        struct timespec t;
        struct sched_param param;
        int interval = 50000; /* 50us*/
        param.sched_priority = MY_PRIORITY; /* Declare ourself as a real time task */
        if(sched_setscheduler(0, SCHED_FIFO, &param) == -1) {
                perror("sched_setscheduler failed");                exit(-1);
        }
        /* Lock memory */
        if(mlockall(MCL_CURRENT|MCL_FUTURE) == -1) {
                perror("mlockall failed");                exit(-2);
        }
        stack_prefault(); /* Pre-fault our stack */
        clock_gettime(CLOCK_MONOTONIC ,&t);
        /* start after one second */
        t.tv_sec++;
        while(1) {
                /* wait until next shot */
                clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &t, NULL);
                /* do the RT stuff */
                t.tv_nsec += interval; /* calculate next shot */
                while (t.tv_nsec >= NSEC_PER_SEC) {
                        t.tv_nsec -= NSEC_PER_SEC;      t.tv_sec++;
                }
  }
}
```

PREEMPT_RT benchmark results

Benchmark result of vanilla kernel

- Cyclictest
  - measuring accuracy of sleep and wake operations of highly prioritized realtime threads
  - https://rt.wiki.kernel.org/index.php/Cyclictest
  - git clone git://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/rt-tests.git

  - *sudo ./cyclictest -a -t -n -p99*

Vanilla kernel: 3.2.0-24 on 8 core PC, CONFIG_PREEMPT_VOLUNTARY

```
insop@chai:~/Projects/rt-tests$ uname -a
Linux chai 3.2.0-24-generic-pae #39-Ubuntu SMP Mon May 21 18:54:21 UTC 2012 i686 i686 i386 GNU/Linux
insop@chai:~/Projects/rt-tests$ sudo ./cyclictest -a -t -n -p99
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.54 0.69 0.67 6/417 3256

T: 0 ( 2772) P:99 I:1000 C:1008249 Min:      4 Act:   18 Avg:   11 Max:     701
T: 1 ( 2773) P:99 I:1500 C: 672166 Min:      4 Act:   35 Avg:   11 Max:     491
T: 2 ( 2774) P:99 I:2000 C: 504124 Min:      4 Act:    9 Avg:   11 Max:     363
T: 3 ( 2775) P:99 I:2500 C: 403299 Min:      4 Act:   14 Avg:   11 Max:    2013
T: 4 ( 2776) P:99 I:3000 C: 336082 Min:      4 Act:   14 Avg:   14 Max:     804
T: 5 ( 2777) P:99 I:3500 C: 288071 Min:      3 Act:   13 Avg:    9 Max:     190
T: 6 ( 2778) P:99 I:4000 C: 252062 Min:      3 Act:    9 Avg:    9 Max:     343
T: 7 ( 2779) P:99 I:4500 C: 224055 Min:      3 Act:   13 Avg:   10 Max:     224
```

Worst case latency: hundreds of usec

Benchmark result of PREEMPT_RT

- Cyclictest
  - measuring accuracy of sleep and wake operations of highly prioritized realtime threads
  - https://rt.wiki.kernel.org/index.php/Cyclictest
  - git clone git://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/rt-tests.git

  - *sudo ./cyclictest -a -t -n -p99*

RT kernel: 3.2.18-rt on 8 core PC

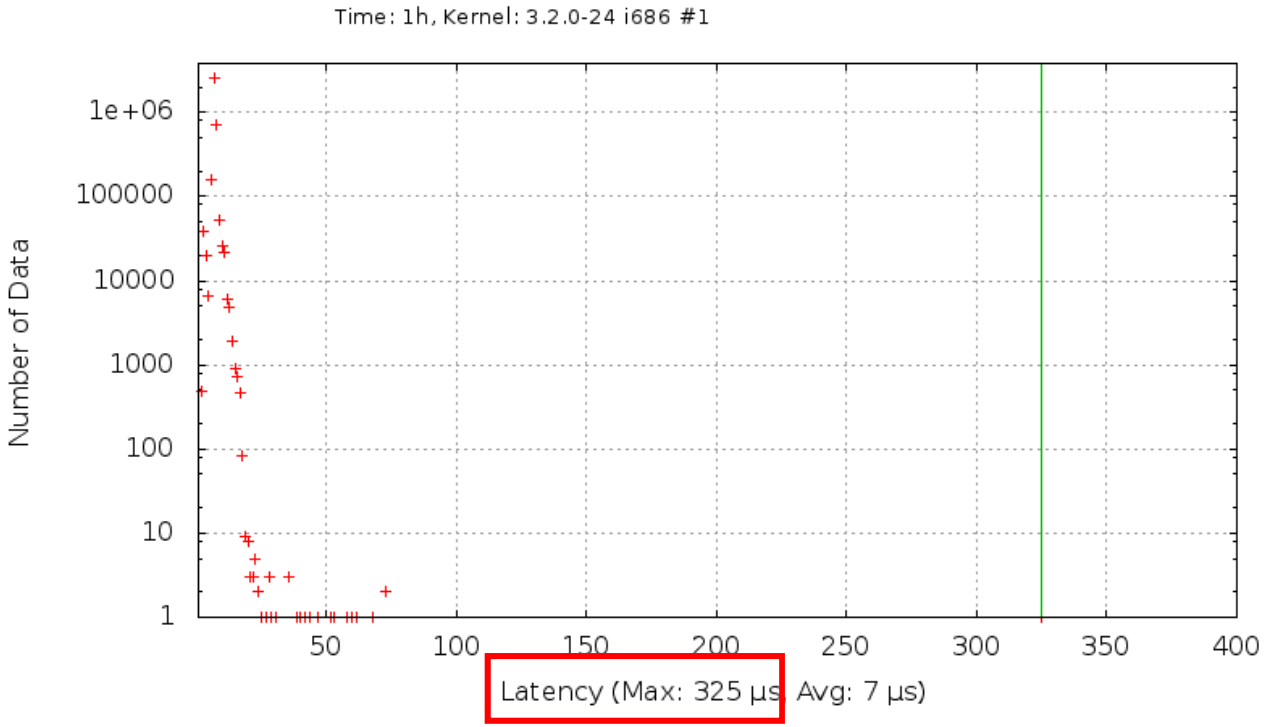
```
insop@chai:~/Projects/rt-tests$ uname -a
Linux chai 3.2.18-iss-rt #1 SMP PREEMPT RT Fri May 25 23:04:58 PDT 2012 i686 i686 i386 GNU/Linux
insop@chai:~/Projects/rt-tests$ sudo ./cyclictest -a -t -n -p99
[sudo] password for insop:
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.69 0.79 0.77 1/434 3037

T: 0 ( 2995) P:99 I:1000 C:1030921 Min:      5 Act:    7 Avg:   12 Max:      32
T: 1 ( 2996) P:99 I:1500 C: 687280 Min:      5 Act:    7 Avg:   13 Max:      53
T: 2 ( 2997) P:99 I:2000 C: 515455 Min:      4 Act:    6 Avg:   13 Max:      34
T: 3 ( 2998) P:99 I:2500 C: 412364 Min:      5 Act:    7 Avg:   13 Max:      34
T: 4 ( 2999) P:99 I:3000 C: 343637 Min:      6 Act:   11 Avg:   16 Max:      31
T: 5 ( 3000) P:99 I:3500 C: 294546 Min:      3 Act:    4 Avg:   11 Max:     338
T: 6 ( 3001) P:99 I:4000 C: 257727 Min:      4 Act:    5 Avg:   11 Max:      24
T: 7 ( 3002) P:99 I:4500 C: 229091 Min:      3 Act:    5 Avg:   11 Max:      29
```

Worst case latency: tens of usec

Benchmark result of vanilla kernel 2

- Cyclictest running 1 hour
  - *./cyclictest -p 80 -n -D 1h -v | gzip > /tmp/cyclic-rt.gz*

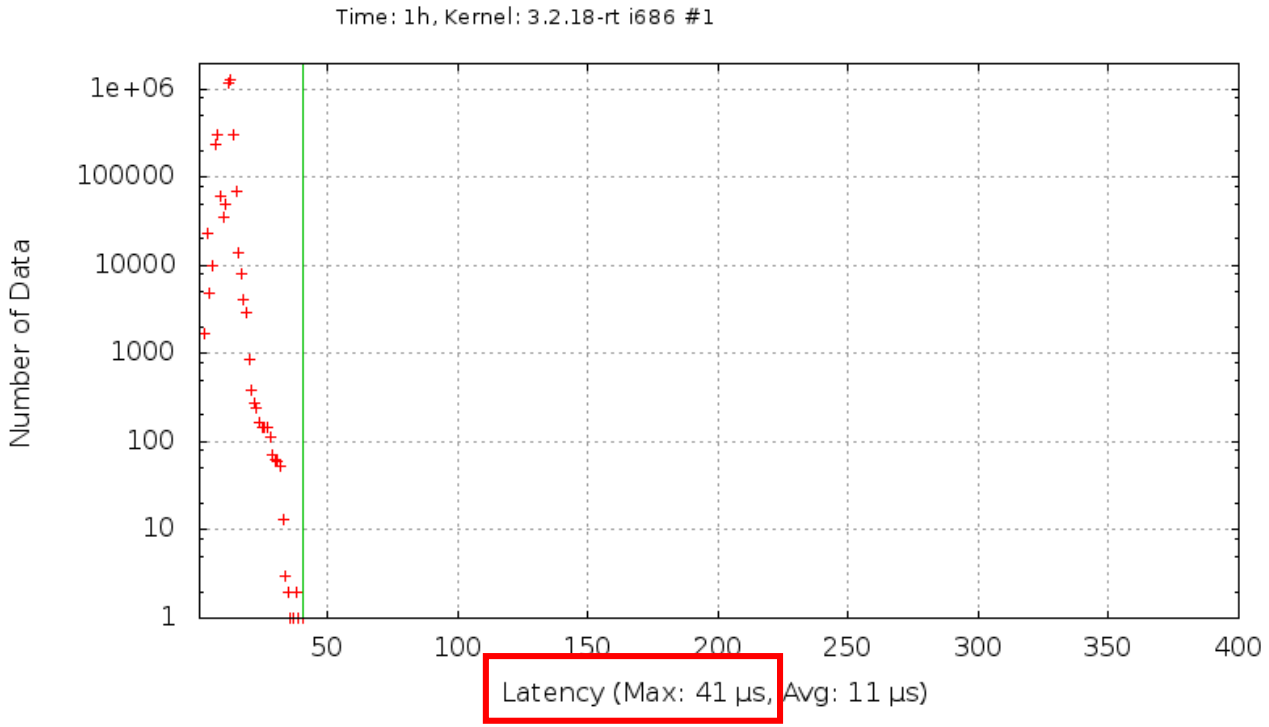
Vanilla kernel: 3.2.0-24, CONFIG_PREEMPT_VOLUNTARY



Benchmark result of PREEMPT_RT 2

- Cyclictest running 1 hour
  - *./cyclictest -p 80 -n -D 1h -v | gzip > /tmp/cyclic-rt.gz*

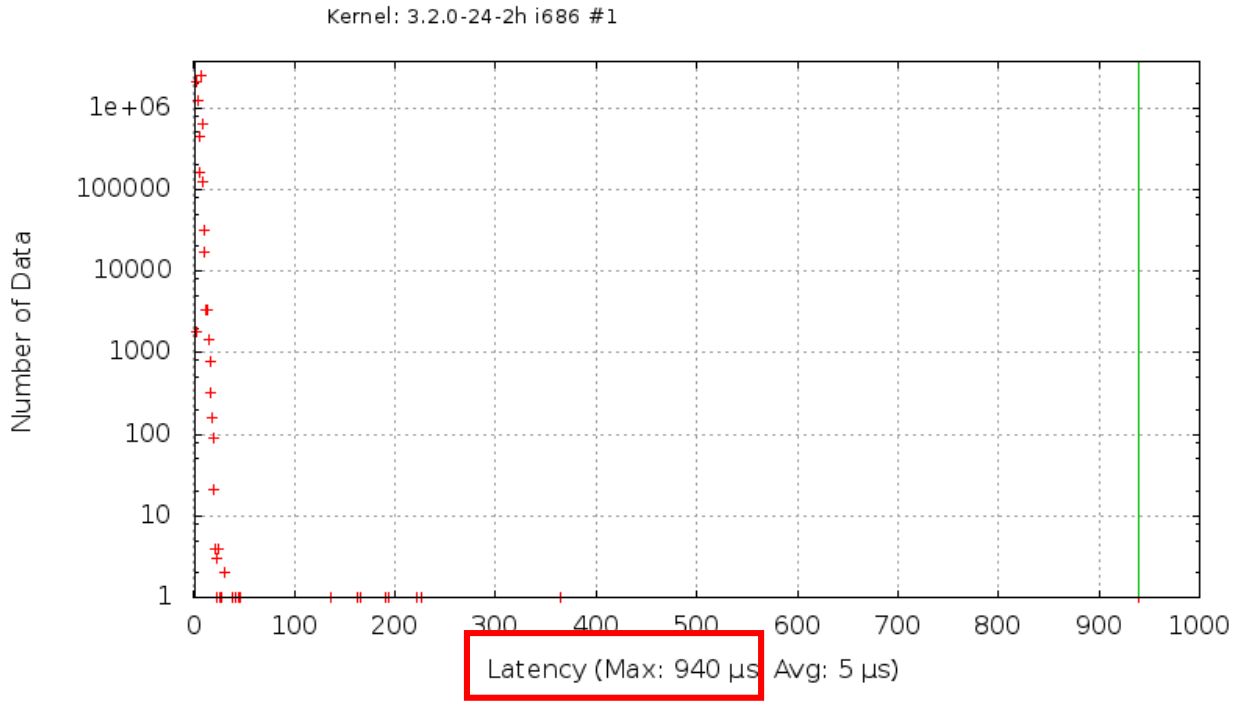
RT kernel: 3.2.18-rt



Benchmark result of vanilla kernel 3

- Cyclictest running 2 hours
  - *./cyclictest -p 80 -n -D 2h -v | gzip > /tmp/cyclic-rt.gz*

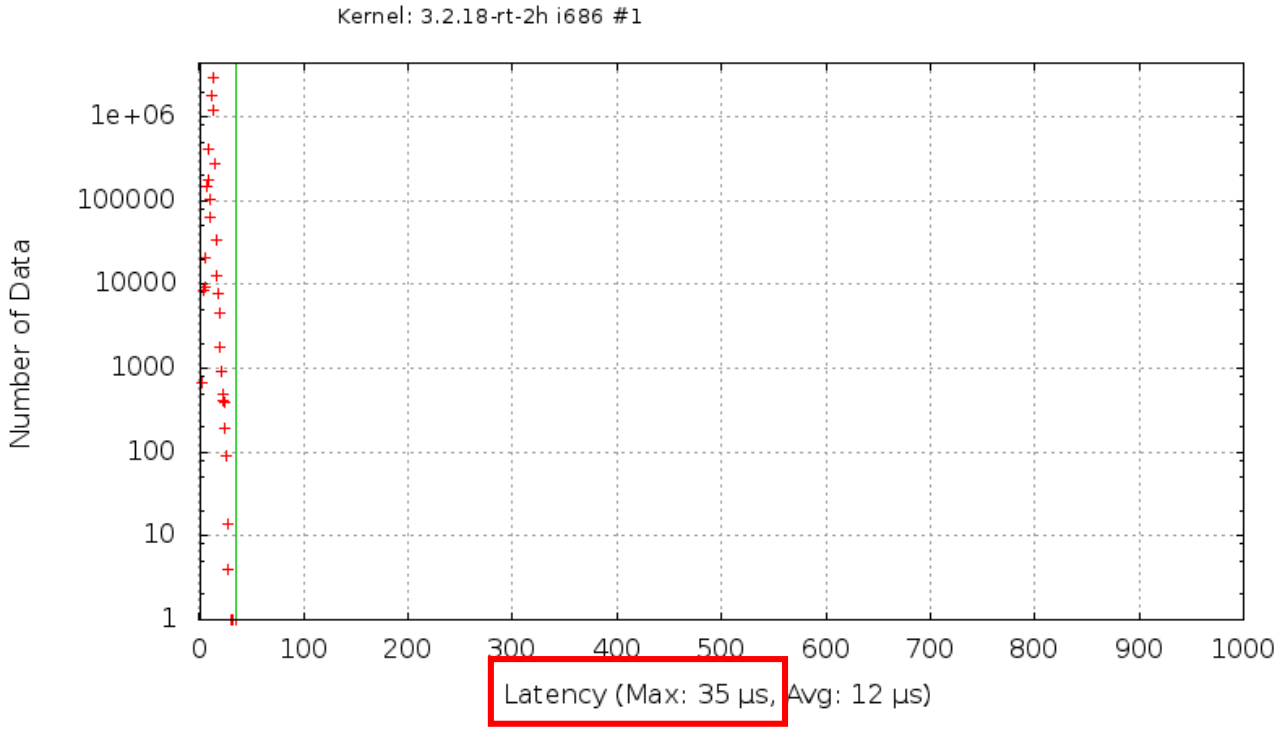
Vanilla kernel: 3.2.0-24, CONFIG_PREEMPT_VOLUNTARY



Benchmark result of PREEMPT_RT 3

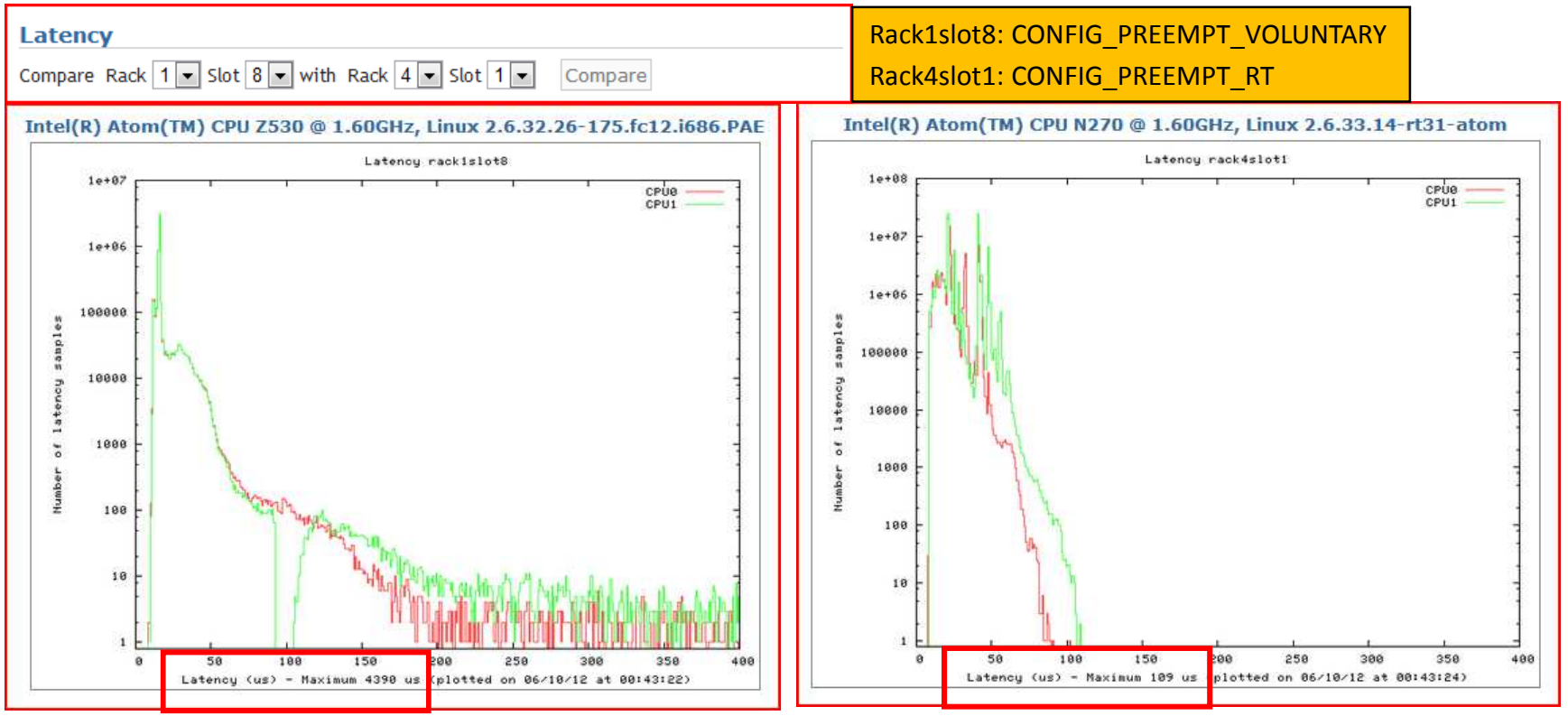
- Cyclictest running 2 hours
  - *./cyclictest -p 80 -n -D 2h -v | gzip > /tmp/cyclic-rt.gz*

RT kernel: 3.2.18-rt



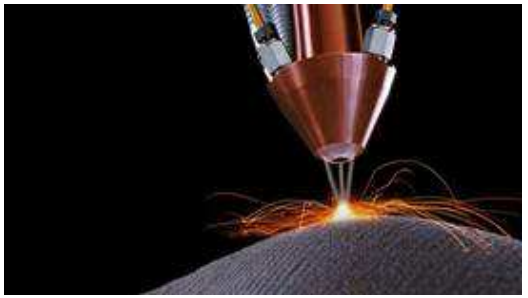
Benchmark result from OSADL

- QA farm running at OSADL.ORG
  - QA FARM: https://www.osadl.org/QA-Farm-Realtime.qa-farm-about.0.html
  - Comparison between 2.6.32 and 2.6.33-rt
  - https://www.osadl.org/Compare-systems.qa-farm-compare-latency.0.html
  - Worst case latency in non-rt is over 4,000 usec, in rt around 100 usec
  - The following is worst case latency distribution comparisions



Few selected deployment examples

- US Navy Zumwalt
  - Total Ship Computing Environment (TSCE)
- US Navy VTOL drone (MQ8B Fire Scout)
  - Ground control by uncle Linus who's replacing uncle Gates suffering from various security issues
- Laser for cutting welding
  - TRUMPF Laser Division
- Robots
  - Lot of them [here]



PREEMPT_RT Roadmap

- Complete full PREEMPT_RT patch merge to mainline
  - "*And yeah, I still think the hard-RT people are mostly crazy*" by Linus 2010
    - (read) getting close, but might take little more time …
- However,..
  - "*The RT people have actually been pretty good at slipping their stuff in, in small increments, and always with good reasons for why they aren't crazy*" by Linus 2010
    - (read) small and well defined piecewise stuff can be merged …

From http://www.redhat.com/archives/utrace-devel/2010-January/msg00319.html

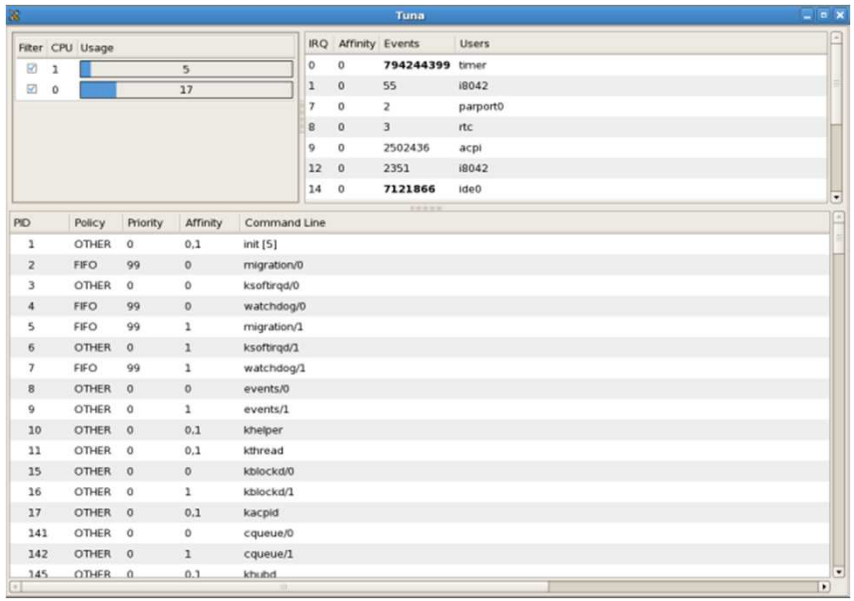
Performance tuning tools

- Ftrace
  - Developed by Steven Rostedt at Redhat
  - Well documented in Documentation/ftrace.txt
  - Run time trace capture of longest latency paths, kernel and user space
  - Peak detector
  - Detailed kernel profiles
    - Can be used to trace any kernel function

[8]

Performance tuning tools

- TUNA
  - Redhat MRG Realtime
  - Dynamic parameter tunings including process affinity, scheduling policy, device IRQ priority



- Linux performance tools
  - Oprofile: system level statistical profileing

Practical tips

- Tune the system
  - Tweak prioirity
    - chrt, sched_setscheduler()
    - SCHED_FIFO and SCHED_RR for real-time scheduling
    - Apply RM (Rate monotonic) scheduling scheme
      - Assign higher priorities to more frequently runing process
    - Avoid SCHED_FIFO priority 99
  - Find hotspot
    - oprofile, gprofile
  - Find the long latency
    - Measure and estimate worst case response time
    - Ftrace
- Processor affinity
  - Bind threads to specific processors
  - Use cpu affinity field in /proc/irqs/<n>/smp_affinity to bind interrupts to specific processors

Response time and throughput

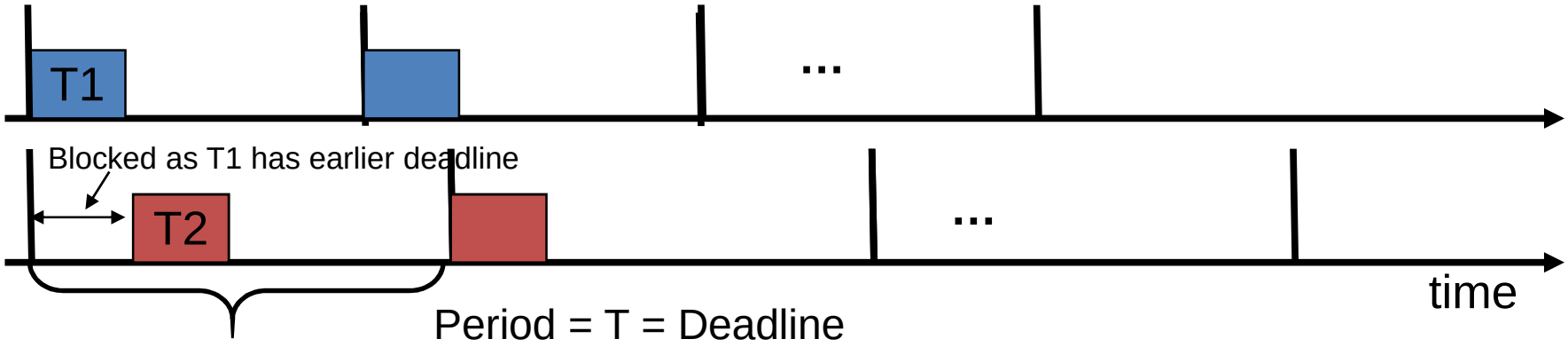
- Overhead for real-time preemption
  - Mutex instead of spin lock
  - Priority inheritance
  - Threaded interrupt handler
- Due to overhead of real-time preemption
  - Throughput is reduced
- Due to the flexibility of preemption
  - Much better worst case latency

Deadline scheduler

Deadline scheduler

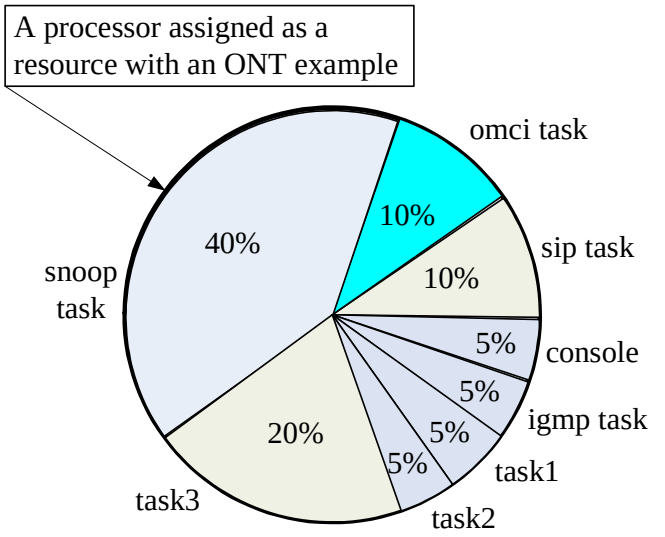
- Optimal dynamic-priority scheduling algorithm
  - A task with a shorter deadline has a higher priority
  - Executes a job with the earliest absolute deadline
- Be able to schedule periodic and aperiodic tasks
- Real-time system is schedulable under EDF if and only if
  - $\Sigma C_i/T_i \leq 1$
    - where Ci: worst-case execution time
    - where Ti: period of task i
  - Full processor utilization

[5]



Deadline scheduler

- Enables a task to be reserved for fractions of the full CPU resource
  - Assigns percentage of CPU to a task instead of assigning priority
  - Ensures each task can only use up to the assigned percentage of the processor even if it demands more
- Guarantees temporal isolation
  - Tasks are temporally isolated, i.e. misbehaving tasks will not impact normal task

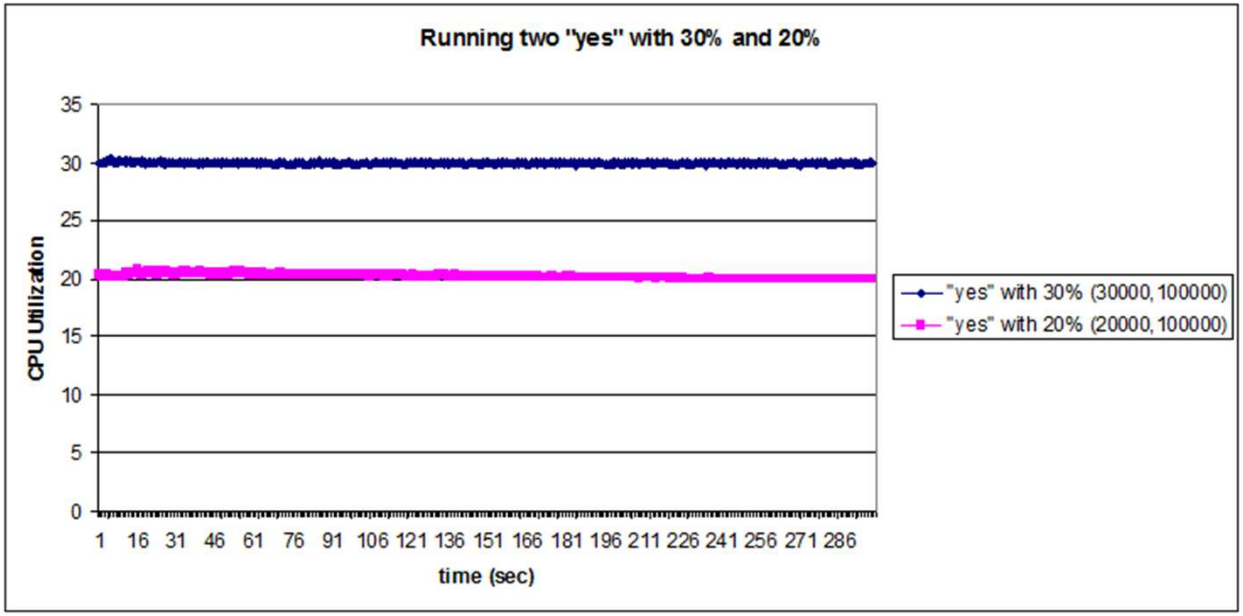


Temporal Isolation

Each task can use assigned percentages of the CPU resource, so tasks are temporally isolated

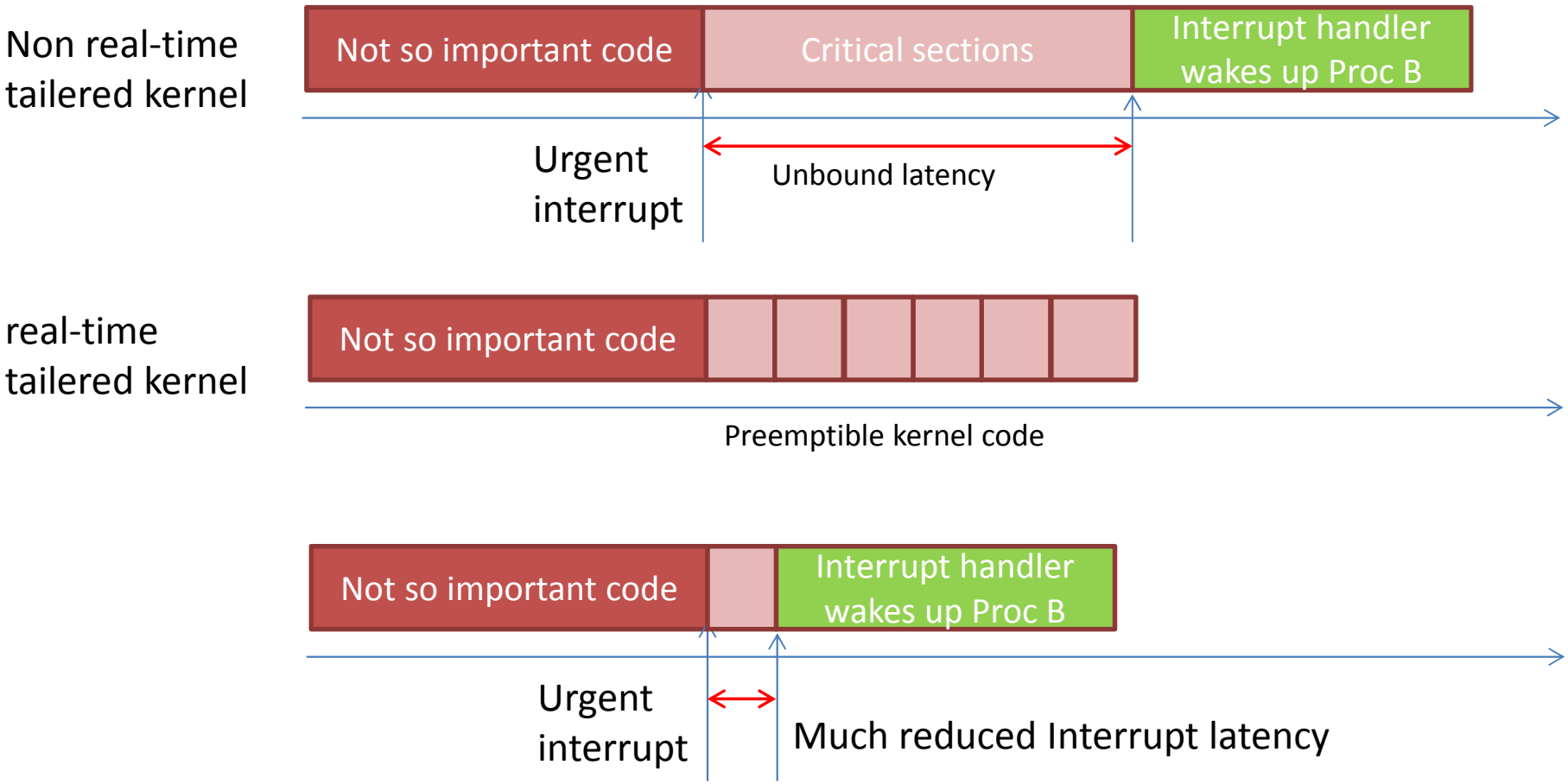
Deadline/reservation test result

- Running two “yes” with 30% and 20%
  - Yes1: 30%, Q:30000, P:100000
  - Yes2: 20%, Q:20000, P:100000



Summary

- Most important aspects of Real-time
  - Controlling latency by allowing kernel to be preemptible everywhere..



Summary

- Mainline kernel
  - Preemtible
    - CONFIG_PREEMPT
  - Real-time scheduler class
    - SCHED_FIFO, SCHED_RR with proper priority
- PREEMPT_RT patch
  - Virtually most of the kernel code is preemptible
  - Many of features are already merged
    - Many kernel developers from various companies have long been working
    - It is anticipated that more code will be merged to mainline
  - Better worst time latency
  - *“So I can work with crazy people, that's not the problem. They just need to _sell_ their crazy stuff to me using non-crazy arguments, and in small and well-defined pieces.”* Linus

References

1. Linux-rt: Turning a General Purpose OS into a Real-Time OS, Peter Zijlstra
2. Inside The RT Patch, Steven Rostedt
3. Real-Time Preemption Patch-Set, Manas Saksena
4. Realtime in embedded Linux systems, Michael Opdenacker et al. free-electrons.com, largely based on this material)
5. Reservation-Based Scheduler for Network Equipments, Insop Song
6. Linux Kernel Development 3rd Edition, Robert Love
7. Professional Linux Kernel Architecture, Wolfgang Mauerer
8. Introduction to realtime linux, Bryan Che, SCaLE, Feb 2009
9. Using Real-time Linux, Klaas van Gend, ELC 2008
10. Linux Realtime-Fähigkeiten, Stefan Agner, 2009
11. Real-time Track, Linux Plumbers Conference, Boston, 2010

Web links

1. Real-Time Linux Wiki,

https://rt.wiki.kernel.org/

2. Mainline RT-preempt patchset,

http://elinux.org/Mainline_RT-preempt_patchset

Disclaimer

- This work represents the view of the author and does not necessarily represent the view of Ericsson

- Thank you for your attention.
- Question?

Backup

Real-time Linux

- How to make linux real-time
  - Minimize interrupt disable time
  - Interrupt handling with schedulable threads
  - Almost fully preemptible kernel
    - except short critical sections
  - Synchronization using mutexs instead of spin locks
    - Allows preemtible
  - Prioirity inheritance support for mutex
  - Priority queue supported mutex
  - High resolution timers

[9]

Kernel config options

- CONFIG_PREEMPT
- Kernel preemption
- Better Posix real-time API
- Priority inheritance support for Mutexes
- High-resolution timers
- Threaded interrupts
- Spinlock annotations

Priority inversion

- Priority inversion

Priority inheritance

- Prioirity inheritance