Kernel Preemption

Linux Internals Seminar WS 2003/2004

Max-Gerd Retzlaff <m.retzlaff@gmx.net>

Overview

I. Introduction

II. The kernel preemption patch

III. Comparison to other efforts and appraisal

IV. References

Overview

I. Introduction

II. The kernel preemption patch

III. Comparison to other efforts and appraisal

IV. References

The goal

- increase system response
- reduce latency, resp.
- in a nutshell:
  A system that is responsive, even under high load caused by:
  - CPU utilization and/or
  - high I/O throughput.

What for?

- musicians
  - audio hard disc recording and MIDI
- (pseudo) real-time applications
  - embedded systems for industrial automation
- the *usual* user
  - a fast and responsive desktop
  - “neither jerky video nor choppy audio”

hard real-time

- *real-time* or *hard real-time* means:
  - *guaranteed* time frames / deadlines
  - Disaster happens if deadline is missed, so the *maximum* response time *must be* within the time frame.
    example: an airplane's computer system
  - very time-consuming design (but possible!)

“pseudo” real-time

- Take a fast processor, break up long-held locks, make the kernel preemptible, etc.
  - You have got a “real-time” capable system!
- Of course, this is *wrong*...
  - *reduced average latency* but
    no guaranteed *maximum response time*
- Nevertheless enough for video streaming and maybe even for some industrial automation.

History I: low latency patches

- low latency patches for 2.2 and later 2.4 by Ingo Molnar and Andrew Morton, resp.
- use scheduling points / preemption points to break up long-held locks (traversals of long lists)
    - `if (current->need_resched) schedule();`
    - experimental approach: Measure latencies of particular kernel regions and place scheduling points.
    - better referenced as: *lock-breaking patches*
- remarkable lobby: “a joint letter on low latency and linux” on June 28th, 2000

History II: kernel preemption patches

- at least two independent efforts:
- MontaVista press release on Sep. 7th, 2000
  - Originally written by Nigel Gamble (MontaVista).
  - Presumedly since October, 2001 maintained by Robert Love (employee of MontaVista since January, 2002).
  - Merged into the main linux kernel-tree
    as of v2.5.4-pre6 on Feb. 10, 2002.
- TimeSys's implementation seems to be a tad superior.

Overview

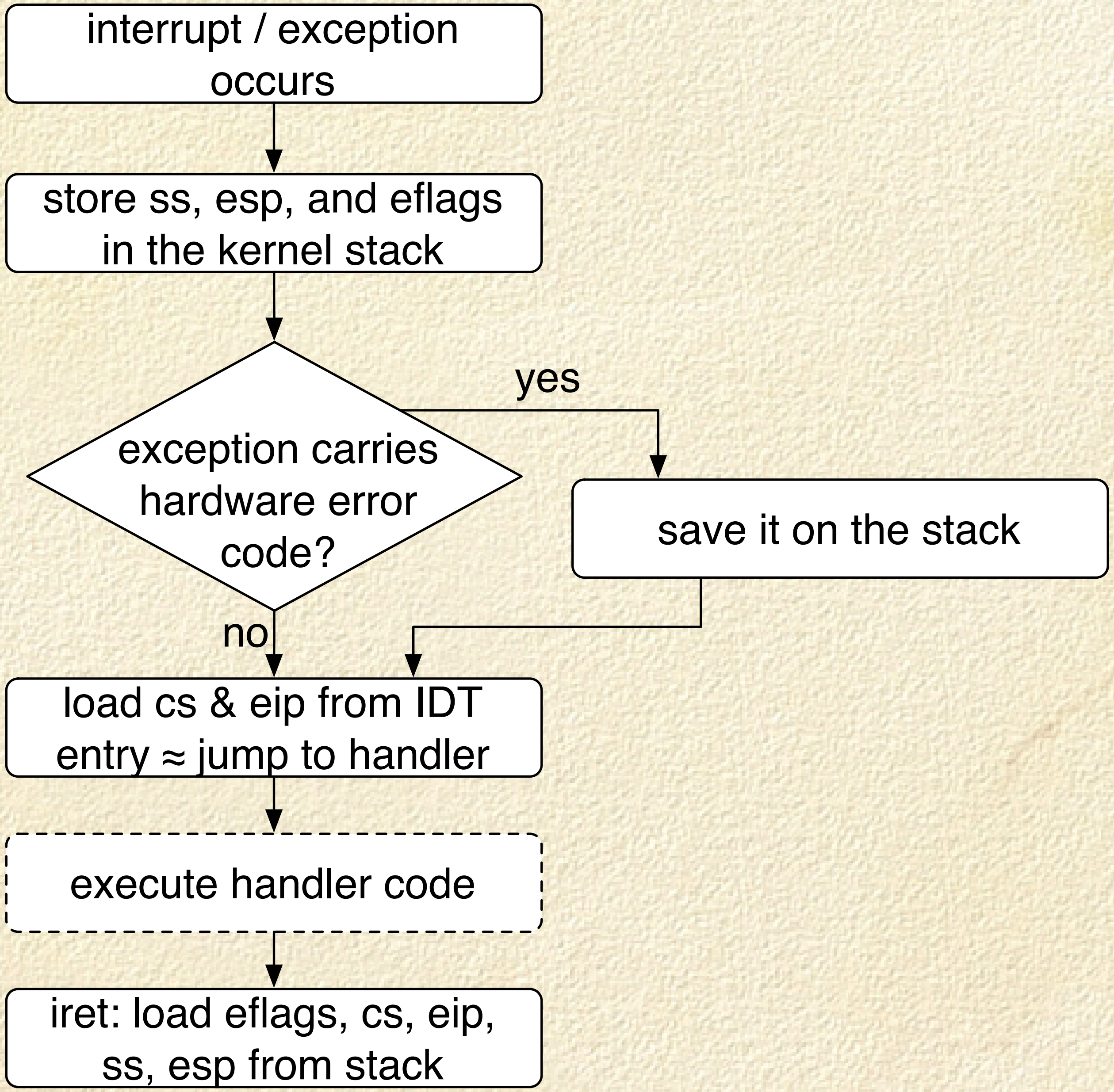
I. Introduction

II. The kernel preemption patch

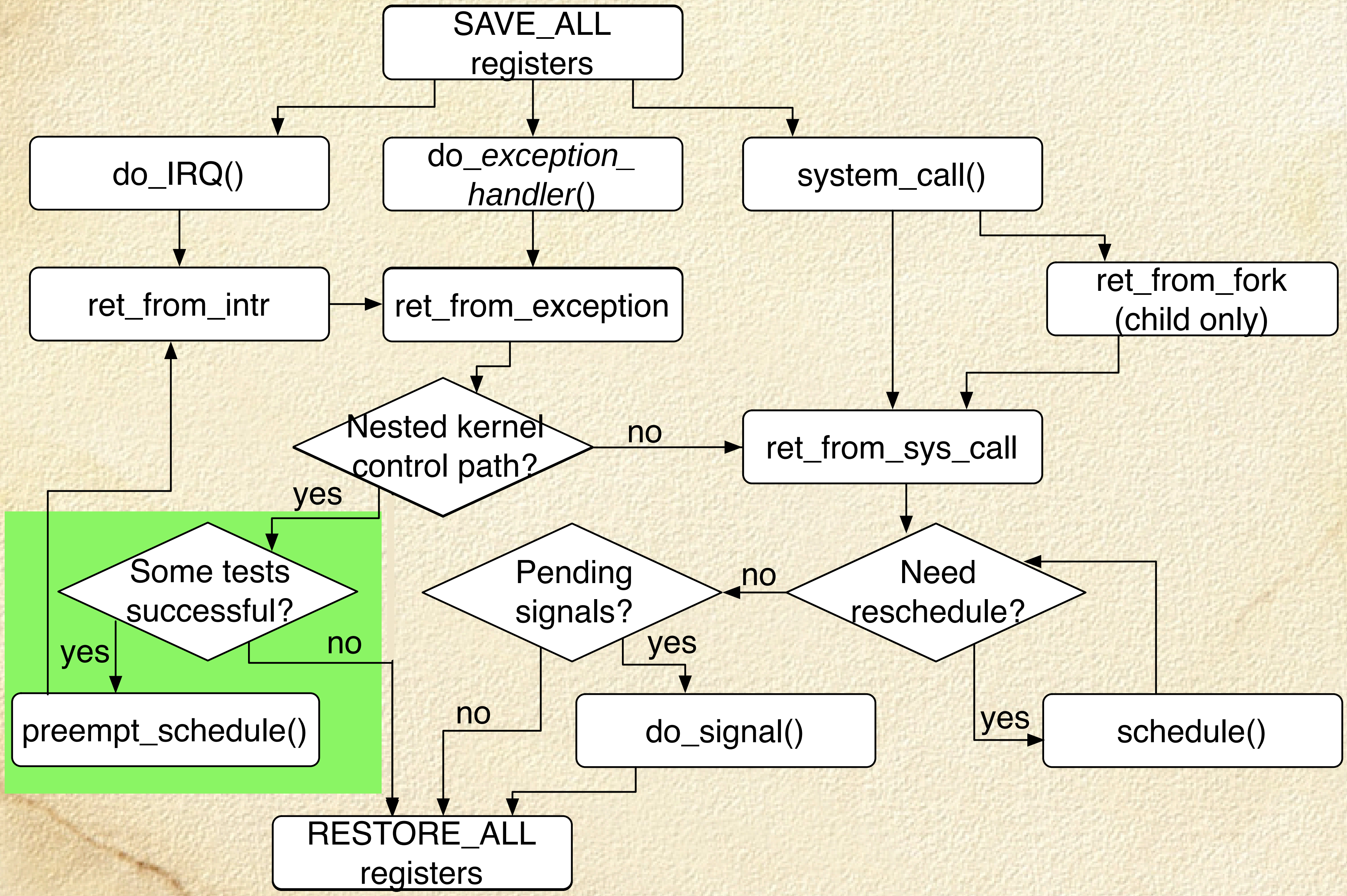
III. Comparison to other efforts and appraisal

IV. References

Hardware handling of interrupts and exceptions


interrupt / exception occurs
store ss, esp, and eflags in the kernel stack
exception carries hardware error code?
yes
save it on the stack
no
load cs & eip from IDT entry ≈ jump to handler
execute handler code
iret: load eflags, cs, eip, ss, esp from stack

… and software handling



Call of preempt_schedule in ret_from_exception

```
ret_from_exception:
        movl EFLAGS(%esp),%eax
            # mix EFLAGS and CS
        movb CS(%esp),%al
        testl $(VM_MASK I 3),%eax
            # return to VM86 mode or non-supervisor?
        jne ret_from_sys_call
#ifdef CONFIG_PREEMPT
        cmpl $0,preempt_count(%ebx)
        jnz restore_all
        cmpl $0,need_resched(%ebx)
        jz restore_all
        movl SYMBOL_NAME(irq_stat)+
            irq_stat_local_bh_count CPU_INDX,%ecx
        addl SYMBOL_NAME(irq_stat)+
            irq_stat_local_irq_count CPU_INDX,%ecx
        jnz restore_all
        incl preempt_count(%ebx)
        sti
        call SYMBOL_NAME(preempt_schedule)
        jmp ret_from_intr
#else
        jmp restore_all
#endif
```

if preempt_count == 0

and need_resched != 0

and soft_irqs on local cpu on

and irqs on local cpu on

then

call preempt_schedule()

jump to ret_from_intr

What's the problem?

- Not everything can safely be preempted, these sections are called *critical.*
- examples: the scheduler, obviously, the bottom half handler (but many more...)
- So we have to locate all of these section and mark them to be not preemptible?
  - Fortunately this work has been done!

SMP spinlocks

□ As part of the SMP support Linux already has relatively fain-grained locks: the spinlocks.

□ Spinlocks ensure exclusive access to a resource.

□ Additionally they disable interrupts only for the local CPU.

Extending spinlocks

- The preemption patch uses spinlocks as "preemption marks".
- A spinlocked region is not to be preempted.
- Nice, as preemption marks for uniprocessor (UP) systems are the logical equivalent of spinlocks for SMP.

Data protection under preemption

- preempt_disable()
  increment preempt counter
- preempt_enable()
  decrement preempt counter
- preempt_enable_no_resched()
  decrement, but no immediately preempt
- preempt_get_count()
  return the counter

How to extend spinlocks?

- Old spinlock functions wrapped.
- New wrappers call the preemption functions.
- No explicit preemption prevention necessary in any locks or with disabled interrupts.
- Any other code can be preempted at any point.
- {spin|read|write}_{un|try}lock() call preempt_enable() ⇒ preempt_schedule() !

Consequences of preemption - example #1

- Per-CPU data is not “implicitly locked” anymore.

- in linux/kernel/softirq.c

```
int cpu = smp_processor_id();
unsigned long flags;
local_irq_save(flags);
```

- replaced by

```
int cpu;
unsigned long flags;
local_irq_save(flags)
cpu = smp_processor_id();
```

Consequences of preemption - example #2

- CPU state must be protected:
- e.g. on x86 FPU mode is now critical
- What happens if the kernel executes a floating-point instruction and is then preempted?
- Remember, kernel does not save FPU state except for user mode processes.

Overview

I. Introduction

II. The kernel preemption patch

III. Comparison to other efforts and appraisal

IV. References

Counter arguments

- preemption introduces complexity
  ⇒ bad for throughput
- Tests have shown: It even improves throughput in nearly all situations.
- hypothesis:
  When I/O data becomes available, the user process (if important) can process it immediately — as soon as the interrupt that set the need_resched returns, in fact!

Why is TimeSys' Patch better?

- Basically a similar approach altering spin-lock calls, but using a mutex instead of a counter.
- Mutexes ensure mutally exclusive access to a resource.
  - counter approach: Any spinlock-held critical section prevents preemption.
  - mutex approach: A high priority process can preempt a lower priority process that holds a mutex for a different resource.
- The mutex also employs priority inheritance to avoid the Priority Inversion Problem.

Why isn't TimeSys patch merged into Linux? #1

- *TimeSys just seems not to be as commited to open source as MontaVista.*
- *Free version called "TimeSys's Linux GPL" exists, but*
  - *apparently you have to register yourself in order to get it and*
  - *other additions (incl. real-time scheduling and resource allocation) are realized as non-free modules that provide extra system calls.*
- *Sourceforge project page for MontaVista's patch*

Why isn't TimeSys patch merged into Linux? #2

- *MontaVista engaged Robert Love who since then is "getting to work on a lot of projects in the community" (acc. to his words).*

- *MontaVista feels itself responsible to the linux community to innovate and to release early and often (acc. to their words).*

- *Robert Love sent the patch to Linus Torvalds ("please apply") and Linus liked the patch.*
  It corresponds to the first design outline he did in discussions during kernel 2.3.

Conclusion

- MontaVista's / Robert Love's kernel preemption patch...
  - reduces the average latency of Linux and
  - makes it generally more responsive.
  - It does not guarantee a *maximum* latency.
  - Explicit scheduling points are still useful to break up long-held locks (only in spin-lock-held regions, of course).

Overview

I. Introduction

II. The kernel preemption patch

III. Comparison to other efforts and appraisal

IV. References

References 1

- OS design background:
  - Andrew S. Tanenbaum, Moderne Betriebssyteme, 2. Auflage
  - William Stallings, Operating Systems, Fourth Edition
- Linux specific background:
  - Tigran Aivazian, Linux Kernel 2.4 Internals, Aug. 7th, 2002 (The LKI is part of the Linux Documentation Project.)
  - Daniel O. Bovet & Marco Cesati, Understanding the Linux Kernel, First Edition (Kernel 2.2) and 2nd Edition (Kernel 2.4)
- Source codes of...
  - the Linux kernel versions 2.4.22 and 2.4.23,
  - several versions of MontaVista's / Robert Love's Kernel Preemption Patch, and
  - the low latency / lock-breaking patches by Ingo Molnar and Andrew Mortan, respectively.

References 2

online resources in order of application

- http://www.linuxdevices.com/articles/AT5503476267.html
  ELJOnline: “Real-Time and Linux, Part 2: the Preemptible Kernel”
- http://www.linuxdevices.com/articles/AT5997007602.html
  ELJOnline: “Real-Time and Linux, Part 1”
- http://people.redhat.com/mingo/lowlatency-patches/
  low-latency-patches by Ingo Molnar
- http://www.zipworld.com.au/~akpm/linux/schedlat.html
  Linux scheduling latency by Andrew Morton
- http://www.gardena.net/benno/linux/audio/
  scheduling latency tests by Benno Senoner

References 3

- http://seclists.org/linux-kernel/2000/Jul/0123.html
  Linux Kernel mail: “a joint letter on low latency and Linux,”
  75 signees, started a thread of 218 mails
  - http://seclists.org/linux-kernel/2000/Jul/0157.html
    Torvalds: “Badly written code will be a problem. The approach that the patches so far have taken is to just add scheduling points all over the map.”
  - http://seclists.org/linux-kernel/2000/Jul/0214.html
    Torvalds: “I refuse to have a kernel that is bogged down with random crap all over the place. It’s wrong. It’s distasteful. And it leads to more and more crap over time. That’s how you get a BAD operating system. “

References 4

- http://www.ussg.iu.edu/hypermail/linux/kernel/0110.0/1215.html
  mail “low-latency patches” by Bob McElrath
  starts a discussion between Robert Love and Andrew Morton
  - http://www.ussg.iu.edu/hypermail/linux/kernel/0110.0/1216.html
    Morton: “[My patch] also reorganises various areas of the kernel which can traverse very long lists when under spinlocks.”
  - deliberate responses by Robert Love:
    http://www.ussg.iu.edu/hypermail/linux/kernel/0110.0/1314.html
    http://www.ussg.iu.edu/hypermail/linux/kernel/0110.0/1338.html
    http://www.ussg.iu.edu/hypermail/linux/kernel/0110.0/1319.html
- http://www.linuxdevices.com/news/NS7572420206.html
  “MontaVista unveils fully preemptable Linux kernel prototype”
- http://www.mvista.com/news/2000/montavistafirst.html
  “MontaVista First to Deliver Hard Real-Time Linux”, Sep. 7th, 2000

References 5

- http://lwn.net/2001/0830/a/preempt.php3
  Robert Love: “Updated Linux kernel preemption patches”,
  mentiones Nigel Gamble (of MontaVista) as original author
- http://www.kernel.org/pub/linux/kernel/v2.5/testing/patch-2.5.4.log
  “Summary of changes from v2.5.4-pre5 to v2.5.4-pre6”
  “[PATCH] Preemptible Kernel for 2.5” merged
- http://www.linuxdevices.com/news/NS3989618385.html
  “Preemptible kernel patch makes it into Linux kernel v2.5.4-pre6”,
  Feb. 10, 2002
- http://www.linuxdevices.com/articles/AT8267298734.html
  “An interview with preemptible kernel patch maintainer, Robert Love”, Jan. 18th, 2002

References 6

- http://www.linuxdevices.com/news/NS4265889552.html
  “Update: Real-time Linux sub-kernels, benchmarks,
  and . . . contention”, Responses and “clarifications”
  by people of MontaVista, TimeSys, FSMLabs, etc.

- http://www.linuxdevices.com/articles/AT6106723802.html
  “ A TimeSys perspective on the Linux preemptible kernel”

- http://kerneltrap.org/node/view/336
  “Interview: Robert Love”, July 16, 2002

- http://www.mvista.com/dswp/PreemptibleLinux.pdf

Questions?

Thank you
for your attention.