Real-Time performance comparisons and improvements between 2.6 Linux Kernels
Outlines

- Preemptibility effects in Latency reduction and Performance (Analysis)

- Approaches to Real-Time under Linux

- Perceptions and definitions to improve the preemption of the Kernel

- Experiments, benchmarks, estimations
Preemptibility effects

• Throughput and the associated time constrains

• Hard real time & soft real time tasks

• When the number of task grows, the scheduler manage the situation expanding within the limits the period of SRT tasks

• Scheduler overhead , the Big O

• Synchronization and priority inheritance via mutexes not with spin – locks

• Planning high resolution timers(reduce error scale during period estimation).
Outlines

• Preemptibility effects in Latency reduction and Performance (Analysis)

• Approaches to Real-Time under Linux

• Perceptions and definitions to improve the preemption of the Kernel

• Experiments, benchmarks, estimations
Approaches to Real-Time under Linux

- **Non-CONFIG_PREEMPT** — for best effort & for soft real time tasks. - poor responsiveness for real time environment, low-level interrupt-handling code

- **CONFIG_PREEMPT** - Kernel code preemptible, except the spinlock section and RCU read side.
  - Performance penalty very small
  - The amount of code more complicated
Approaches to Real-Time under Linux

- **CONFIG_PREEMPT_RT** – priority inheritance (prevent priority inversion), mutexes, RCU read side.
  - To implement the priority inheritance for read-write locks is very difficult.

- Is not fair since generally the real time systems are not fair.

- the shortest the period of execution time, higher priority is given by the scheduler policy.

**Nested OS** – Linux as user mode, over RTOS.
• Preemptibility effects in Latency reduction and Performance (Analysis)
• Approaches to Real-Time under Linux
• Perceptions and definitions to improve the preemption of the Kernel
• Experiments, benchmarks, estimations
Improve the preemption of the Kernel

- Defining the non-preemptible zones in the Linux kernel
- Interrupt off paths
- Lowest-level interrupt management
- Scheduling Code
- Context switching code
Improve the preemption of the Kernel

- Introducing the preemption points
- Calls to the disk buffer cache
- Memory page management
- Calls to the file system
- VGA and console management
- The forking and exits of large processes
- The keyboard driver
Improve the preemption of the Kernel

Trying to make kernel preemptible in general

- Minimized interrupt disable times
- Interrupt handling via schedulable threads
- Preemptible kernel
  - Short critical sections
- Perform synchronization via mutexes (not spin locks)
  - Allows involuntary preemption
- Mutex support for priority inheritance
- High Resolution timers
• Preemptibility effects in Latency reduction and Performance (Analysis)
• Approaches to Real-Time under Linux
• Perceptions and definitions to improve the preemption of the Kernel
• Experiments, benchmarks, estimations
• Motivation

• Preliminary conditions
  • 2.6.20 – 2.6.26 – 2.6.31
  • patch-2.6.20-rt8 – patch-2.6.26-rt1 – patch-2.6.31-rt10
  • GenuineIntel, Intel(R) Pentium(R) 4 CPU 2.40GHz
  • Total memory: 757 MB - Total swap: 2212 MB
  • disk - Model: SAMSUNG SP1213N - Capacity: 117.2 GB - Cache: 8.192 MB
  • VGA controller - nVidia Corporation NV18 [GeForce4 MX 4000 AGP 8x] (rev c1) (prog-if 00 [VGA])
Experiments, benchmarks, estimations

Stream 0
- Codec: XVID
- Language:
- Type: Video
- Resolution: 512x320
- Frame rate: 25.000000

Stream 1
- Codec: mpga
- Language:
- Type: Audio
- Channels: 2
- Sample rate: 48000 Hz
- Bitrate: 130 kb/s

- VLC player
- Play time = 60 sec
- Benchmark = Realfeel
- Gnome = Off
- VLC = Terminal mode
- Plotter = Gnuplot
- SSH sessions opened = 3
- ACPI = disabled for 2.6.26 & 2.6.31 rt-patch(RTC)
Experiments, benchmarks, estimations

- Measurement method
  - RealFeel benchmark
  - RTC (real time clock) driver with a particular frequency
  - GCC
  - Timelines
  - Estimated Value
Experiments, benchmarks, estimations

- Results
Experiments, benchmarks, estimations

• Results

<table>
<thead>
<tr>
<th>Kernel releases</th>
<th>AVG Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6.2</td>
<td>393.9</td>
</tr>
<tr>
<td>2.6.2-rt</td>
<td>128.3</td>
</tr>
<tr>
<td>2.6.6</td>
<td>217.9</td>
</tr>
<tr>
<td>2.6.6-rt</td>
<td>284.3</td>
</tr>
<tr>
<td>2.7.31</td>
<td>180.3</td>
</tr>
<tr>
<td>2.7.31-rt</td>
<td>17.3</td>
</tr>
</tbody>
</table>
Experiments, benchmarks, estimations

- Results
  - General preemption Kernel (Voluntary Preempted)
  - Responsiveness means predictability doing the expectations to be much more predictable than they are when running the respective Kernel release without rt-patch.
  - Greater performance for 2.6.31 release
Conclusions

- Responsiveness and throughput are opposite with each other.

- Reduce as much as possible interrupt off regions. Why low throughput ??

- Complexity of mutex operations vs spinlocks

- The complexity of mutex increase the cs operations during priority inheritance.
Conclusions

- High resolution Timer on 2.6.20 release
- After 2.6.20 kernel release threaded interrupt handler and sleeping spinlocks (mutexes).
- Voluntary preempted kernel (general kernel)
- 2.6.26 now possible to create a work queue running at realtime priority
- Better documentation for RT scheduling options
Conclusions

- Expanded kernel preemption to other arch (CF)
- 2.6.27 CFS called SMP-nice for group policy scheduling (Sys hibernation)
Thanks for the attention... and support!

Any questions??