Linux and Real-Time: Current Approaches and Future Opportunities

Claudio Scordino and Giuseppe Lipari

15 November 2006
Summary

• Why Linux for real-time applications?

• Problems in using Linux for real-time

• Making Linux more real-time
  – Interrupt Abstraction approach
  – Kernel Preemption

• Current and future approaches
Why Linux for Real-Time

• A raising interest from industry
  – small automation companies
  – big software companies (e.g. IBM and WindRiver)

• Reason for using Linux
  – Open Source License (GPL)
  – Standard Interface (POSIX)
  – Wide popularity and success
  – Available for almost all embedded processors (ARM, MIPS, PPC, etc.)

• Total Cost of Ownership (TCO)
  – lower than other commercial (closed source) RTOSs
Application areas

- Linux cannot be used everywhere

- Memory constraints
  - some embedded systems have limited memory
  - Linux requires at least 4-8 Mbytes (Flash and RAM)

- Criticality constraints
  - some application require certification of the kernel
  - Linux is too big to be certified
Problems

- Mainstream kernel
  - officially supported by Linux Torvalds
  - optimized for throughput and performance
  - not designed for real-time performance

- Latency and real-time
  - the main problem is the high worst-case latency of time-critical operations
  - average-case latency is quite small ...
  - ... but real-time systems need predictability!

- Scheduler
  - linux supports only Fixed Priority Scheduling
Problem: Latency

- Definition:
  - *interval of time between the arrival of an interrupt signal to the processor the start of the handler that responds to the interrupt*
  - handler = real-time task awaken by the interrupt handler
The causes of latency

• Task Latency
  – some part of the kernel cannot be interrupted (critical sections of code)
  – user-level code is scheduled only when the kernel finishes to process kernel activities
  – Kernel activities can accumulate and be very long
  – Linux 2.4.17 → max latency = 230 msec!

• Timer Resolution
  – periodic activities are wake-up by timers
  – standard timer resolution is coarse-grained (1-40 msec)
Problem: Scheduler

• Linux supports Fixed Priority Scheduling
  – standard POSIX policies SCHED_FIFO and SCHED_RR
  – both suitable for real-time
  – can only be used by process with root privileges

• Security concerns
  – a buggy process with high priority can hang the system
  – not easy to handle overload situations
Making Linux more real-time

• Two approaches

• Interrupt Abstraction
  – uses a special micro-kernel for RT and executes Linux as a low-priority task
  – Used by RTLinux, RTAI, Xenomai

• Kernel Preemption
  – The hard way....
  – Reduce Latency in Linux
Linux-RT: Interrupt Abstraction

- The abstraction layer
  - intercepts interrupts
  - forwards to Linux of to RT

- If RT task execute
  - Linux interrupts become pending
  - RT interrupt are served immediately

- When no RT task executes
  - Linux interrupts are served all at once
Interrupt abstraction

• **Advantages**
  – Effective reduction of latency
  – Use of standard Linux processes for non-real-time
  – host and target platform may coincide

• **Disadvantages**
  – different interface for RT tasks
  – RT tasks execute in kernel memory space
  – buggy RT code can block the system
  – need to re-write Linux device drivers for RT
  – difficult to use!!
LXRT and Xenomai

• LXRT
  – an interface provided by RTAI
  – allows to execute RT tasks in user space (at the price of a higher latency)
  – useful for development and debugging

• Xenomai
  – A spin-off of RTAI
  – allows to execute RT tasks in user space with very low latency
  – allows to write device driver code in user space
Linux-RT: Kernel preemption

• Reduce Linux latency
  – this approach allows to execute Linux processes with Real-Time performance

• Modification to Linux code to:
  – Improve timer resolution
  – Insert preemption points in the kernel
  – Introduce kernel threads
  – Call the scheduler more frequently
Kernel preemption in 2.4

- Linux 2.4
  - Preemption Patch
    - Introduce preemption points in long sections of code
    - in this way, very long and non-interruptible sections of code can be split in different parts
  - Low Latency Patch
    - Call the scheduler more frequently in long sections of code
  - High Resolution Patch
    - provides an API for micro-second level precision timers

- Results
  - latency reduction
  - but not yet usable for real-time
Kernel Preemption in 2.6

- **Linux 2.6**
  - introduces kernel threads
    - a device driver can use a kernel thread (scheduled with the other processes)
  - $O(1)$ scheduler
    - scalable with number of threads

- **Preempt_rt patch**
  - Support for many improvements
    - Priority Inheritance for reduction of task blocking time
Comparison

The following table is provided by Paolo Mantegazza (available on Internet)

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Aver</th>
<th>Max</th>
<th>Min</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Std Linux 2.6</td>
<td>6.8</td>
<td>555.6</td>
<td>5.6</td>
<td>7.2</td>
</tr>
<tr>
<td>Preemption patch</td>
<td>7.3</td>
<td>70.5</td>
<td>5.6</td>
<td>1.8</td>
</tr>
<tr>
<td>Adeos (RTAI)</td>
<td>7.6</td>
<td>50.5</td>
<td>5.7</td>
<td>0.7</td>
</tr>
</tbody>
</table>

*numbers are in micro-seconds*
Which flavor of Linux RT to use?

- For hard real-time applications with very small constants of time (below the milliseconds), it is still necessary to use RTLinux, RTAI or Xenomai.

- There has been a constant effort to reducing the latency of the standard Linux kernel.

- It is possible to use standard Linux with Preemption Patch for
  - soft real-time applications,
  - or even hard real-time applications with large constants of time.
Conclusions

- Linux has become very popular for supporting real-time applications

- Many projects have been proposed to make Linux more real-time

- In the next future, standard mainstream Linux will include RT support
  - low latency
  - appropriate scheduling policy