About the challenges and successes with Xenomai as RTOS for Routers with integrated Ethernet Switches

Andreas Glatz
andreasglatz@ruggedcom.com

Ruggedcom Inc.
Concord, ON, Canada
Outline

- Company facts
- Motivation for using Xenomai
- Development environment
  - IDEs
  - Debugging
  - Tracing
- Outlook
Company Facts

- Founded 2001 by Marzio Pozzuoli (President and CEO) near Toronto, Ontario, Canada
  - Public: TSX:RCM
  - ~200 employees
- Designs and manufactures **Layer2** and **Layer3** networking devices for harsh electrical and climatic environments
- Key markets
  - Electrical Utility, Transportation, Industrial, Military
- Strong customer base all around the world
  - ABB, GE, Siemens, Hydro One, National Grid, Alcoa, Chevron, Boeing, Lockheed Martin, US Navy
Company Facts (cont'd)

- Switches and Routers in a power grid substation
Company Facts (cont'd)

- RuggedSwitch (RSG2300)
- Rugged Operating System (ROS)
  - Well developed and tested **Embedded C++** code (no rtti, no exceptions, no standard C++ library!)
  - Coldfire running traditional RTOS
- Supports advanced layer2 features
  - RSTP, VLAN, Link aggregation, ...
Company Facts (cont'd)

- RuggedRouter (RX1100)
- Rugged Operating System on Linux (ROX)
  - Well tested standard (Quagga, Webmin, ...) and custom applications
  - x86 running Debian Linux
- Supports Layer3 (IP) features
  - Firewall, Routing protocols (OSPF, ...), DHCP Agent, Traffic prioritization, NTP Server, ...
Motivation for using Xenomai

- NEW: RuggedBackbone
- Router with integrated Switch
  - Should have all features of a RuggedSwitch and a RuggedRouter
  - New hardware with highest port density so far
- Obvious strategy
  - Reuse as much software as possible
  - Combine IP from previous products
Motivation for using Xenomai (cont'd)

- Problems
  - Poor hard real-time capabilities of Vanilla Linux Kernel
  - Commercial RTOSes available but without rich feature set as offered by Linux
- Our solution: Xenomai
  - RuggedRouter software runs in Linux domain
  - RuggedSwitch software runs in Xenomai domain: preemption of any Linux activity (including routing)
Motivation for using Xenomai (cont'd)

- **Goal**: Run exactly the same RuggedSwitch software on traditional RTOS and Xenomai
- **Gain**: Reduces required amount of manpower for maintenance and testing to a minimum

---

**Hardware (PowerPC)**

- IPIPE
- Xenomai (native skin)
- Linux

**Hardware (Coldfire)**

- Abstraction layer (C++)
- RuggedSwitch software
- RTOS and Libraries
- Management applications

**RuggedSwitch**

- RuggedSwitch software
- Abstraction layer (C++)
- RTOS and Libraries
- Hardware (Coldfire)

**RuggedBackbone**

- RuggedSwitch software
- Abstraction layer (C++)
- Xenomai (native skin)
- Linux
- Hardware (PowerPC)
- Secondary domain
- Management applications
Development Environment

- For the past 5 years:
  - RuggedSwitch software developers used a very mature IDE (with integrated Remote Debugger, Profiler, ...)

- Alternatives for Xenomai (Linux) Software development?
  - 'Traditional' Linux development environment
    - Vim, Emacs, Gdb, Gcc, Make, ...
  - IDEs available on Linux (most mature)
    - Eclipse, Kdevelop, ...
# Linux IDEs

## Comparison of most mature IDEs

<table>
<thead>
<tr>
<th></th>
<th>Eclipse</th>
<th>Kdevelop</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Current version</strong></td>
<td>Galileo</td>
<td>3.5.5</td>
</tr>
<tr>
<td><strong>Dependencies</strong></td>
<td>Sun's Jre</td>
<td>KDE libraries</td>
</tr>
<tr>
<td><strong>Responsiveness</strong></td>
<td>acceptable</td>
<td>very fast</td>
</tr>
<tr>
<td><strong>Gdb frontend</strong></td>
<td>good (native gdb plugin), buggy (DSF plugin)</td>
<td>primitive</td>
</tr>
<tr>
<td><strong>Indexer</strong></td>
<td>close to acceptable (build-in)</td>
<td>very good (ctags)</td>
</tr>
<tr>
<td><strong>Visual appearance</strong></td>
<td>very good</td>
<td>good to very good</td>
</tr>
<tr>
<td><strong>and configurability</strong></td>
<td>of Editor</td>
<td></td>
</tr>
<tr>
<td><strong>CVS, SVN, ...</strong></td>
<td>acceptable</td>
<td>acceptable</td>
</tr>
<tr>
<td><strong>integration</strong></td>
<td>Windriver, Abatron, Lttng, Intel, IBM, ...</td>
<td>no plugins</td>
</tr>
<tr>
<td><strong>3rd party plugin</strong></td>
<td>providers</td>
<td></td>
</tr>
</tbody>
</table>
## Remote Debugging

- Remote debuggers

<table>
<thead>
<tr>
<th>Debugger</th>
<th>Type</th>
<th>User interface</th>
<th>User-space</th>
<th>Kernel-space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>real-time</td>
<td>non real-time</td>
</tr>
<tr>
<td>gdb</td>
<td>Software</td>
<td>Text, Eclipse, Kdevelop</td>
<td>gdbserver (problems with Xenomai 2.4.9)</td>
<td>gdbserver</td>
</tr>
<tr>
<td>B DI2000</td>
<td>Hardware</td>
<td>practically not available because BDI doesn't know anything about processes/threads</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lauterbach</td>
<td>Hardware</td>
<td>Motif based GUI, Eclipse plugin (?)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windriver</td>
<td></td>
<td>Eclipse plugin</td>
<td>Problems installing it because it depends on specific RedHat version</td>
<td></td>
</tr>
</tbody>
</table>
Remote Debugging (cont'd)

- Problems with remote debuggers for Xenomai (Linux)
  - Sometimes single stepping doesn't work because gcc doesn't generate proper debug information
  - Combined kernel- and user-space debugging with BDI2000: BDI catches user-space breakpoints
  - >=Linux-2.6.30: gdbserver freezes when debugging code where a thread is created inside of another thread
Tracing

- Simple tracing
  - `rt_printf()`
  - GPIOs, Time Stamp Counter (TSC) and Oscilloscope

- Advantage
  - Most accurate way of tracing as it is tailored for a specific piece of code

- Problems
  - Many development cycles to find the right spot to place the probes
  - Often applies to only one piece of code
Tracing (cont'd)

- **Tracers**
  - Compared to commercial solutions, support for tracing Xenomai applications is very basic
  - Tracing markers/probes are placed in kernel code
  - Development is slowly moving towards markers/probes in user-space applications

<table>
<thead>
<tr>
<th>Tracers</th>
<th>Kernel version</th>
<th>Principal of operation</th>
<th>Works with Xenomai</th>
<th>Interface</th>
<th>Trace duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-pipe</td>
<td>Comes with Xenomai</td>
<td>Instrumentation of every kernel function</td>
<td>yes</td>
<td>/proc</td>
<td>Short</td>
</tr>
<tr>
<td>Lttng</td>
<td>Since 2.6.12</td>
<td>Static markers, Kprobes</td>
<td>yes</td>
<td>Trace gets written to a file by a User-space daemon</td>
<td>Long</td>
</tr>
<tr>
<td>Ftrace</td>
<td>Since 2.6.27</td>
<td>Instrumentation of kernel functions</td>
<td>N/A</td>
<td>/proc</td>
<td>Short</td>
</tr>
<tr>
<td>SystemTap</td>
<td>Relies on Kprobes</td>
<td>Kprobes</td>
<td>N/A</td>
<td>Typically printk</td>
<td>Varies</td>
</tr>
</tbody>
</table>
Tracing (cont'd)

- LTTng
  - ~10min tracing with 5 markers armed produced about 1GB of data (78.350.043 events)
    - xn_nucleus_sched_switch()
    - xn_nucleus_irq_enter()
    - xn_nucleus_irq_exit()
    - xn_nucleus_irq_enable()
    - xn_nucleus_irq_disable()
  - Analysis of textual output: No lttv version which can produce a graphical representation of the trace
  - It took some time to find the right size of the kernel tracing buffers so that no events are lost
Tracing (cont'd)

- Sometimes tracing events are not ordered by time
- Sample output:

```plaintext
xn_nucleus_sched_switch: 146.891552548 (huhu/cpu_0), 0, 0, , 0, 0x0, MODE_UNKNOWN
{ thread_out = 0xe1046848, thread_out_name = "StartTask", thread_in = 0xc03b5db8,
  thread_in_name = "ROOT" }
xn_nucleus_irq_exit: 146.891556943 (huhu/cpu_0), 0, 0, , 0, 0x0, MODE_UNKNOWN { irq = 43 }
xn_nucleus_irq_enter: 146.891566528 (huhu/cpu_0), 0, 0, , 0, 0x0, MODE_UNKNOWN { irq = 43 }
xn_nucleus_irq_exit: 146.891576218 (huhu/cpu_0), 0, 0, , 0, 0x0, MODE_UNKNOWN { irq = 43 }
xn_nucleus_irq_enter: 146.891743633 (huhu/cpu_0), 0, 0, , 0, 0x0, MODE_UNKNOWN { irq = 43 }
xn_nucleus_irq_disable: 146.891756788 (huhu/cpu_0), 0, 0, , 0, 0x0, MODE_UNKNOWN { irq = 43 }
xn_nucleus_irq_disable: 146.891759338 (huhu/cpu_0), 0, 0, , 0, 0x0, MODEUNKNOWN { irq = 43 }
```
Tracing (cont'd)

- UEC ISR duration histogram
  - derived from LTTng trace of RuggedSwitch application running in kernel-space
  - Tx ISR duration is ~10us
  - Rx ISR duration varies between 120us and 160us

- Tracing overhead:
  - 10%- 25% increase in duration
  - Double peak in RxISR duration histogram
Outlook

- Open issues
  - Possibility to have tasks which have a lower priority than Linux
  - Support for debugging distributed architectures
  - Xenomai plugin for LTTV

- Xenomai/SOLO
  - Runs on top of PREEMPT_RT patched kernel
  - Provides native Xenomai API
  - *Possible solution to first issues*
Danke fuer Ihre Aufmerksamkeit