4 Ways to Improve Performance in Embedded Linux Systems

Michael Christofferson
Director Product Marketing, Enea
Korea Linux Forum
Nov 13, 2013
Enea - Powering Communications

- Increasing data traffic in communication devices require new and innovative software solutions to handle bandwidth, performance and power requirements.
- Enea software is heavily used in wireless Infrastructure (Macro, small cell), gateway, terminal, military, auto, etc.
- More than 250M of the 325M LTE population coverage is powered by Enea Solutions.
- Enea Solutions run in more than 50% of the world’s 8.2M radio base stations.
- Enea has recently released its first commercial Linux distribution, built by Yocto, and specially tailored for networking and communications.
Overview of four approaches to enhancement of standard Linux performance in embedded **multicore** devices.

- Linux PREEMPT_RT CONFIG Patch Set
- Vertical Partitioning and User Space Runtime
- Open Event Machine
- Virtualization solutions

*Relative performance comparisons, as well as other metrics that reflect “Pros and Cons” of each approach*
What Does Performance Mean?

Many measures of “performance”

• Real-time Responsiveness
  – In embedded, often linked with the concept of “deterministic” response
  – But not always!! .... See next slide

• Throughput
  – Discrete event processing bandwidth or rates
  – Does not necessarily mean short or even deterministic real-time response

• High Performance Computing
  – Massive compute intensive applications like modeling and simulation, and mathematical related computations
  – Not the same as throughput

=> For embedded, it’s about Real-time response and Throughput
What Does “Real-time” Performance Mean?

• Real-time systems
  – Have “operational deadlines from event to system response”
  – Must guarantee the response to external events within strict time constraints

• Non-real-time systems
  – Cannot guarantee response time in any situation
  – Are often optimized for best-effort, high throughput performance

• “Real-time response” means deterministic response
  – Can mean seconds, milliseconds, microseconds.
  – I.e. not necessarily short times, but usually this is the case

• Real-time system classifications:
  – **Hard**: missing a deadline means total system failure
  – **Firm**: infrequent misses are tolerable, but result is useless. QoS degrades quickly
  – **Soft**: infrequent misses are tolerable, increased frequency degrades QoS more slowly

=> *Real-time performance OFTEN is contradictory to Throughput!!*
Examples of real-time systems

• **Hard real-time applications:**
  – Automotive: anti-lock brakes, car engine control
  – Medical: heart pacemakers
  – Industrial: process controllers, robot control

  *Throughput NOT an issue*

• **Firm real-time applications:**
  – 3G/4G baseband processing/signaling in base stations and radio network controllers
  – 3G/4G baseband processing/signaling in wireless modems (phones, tablets)
  – Many other examples in the networking space – RRU, optical transport, backhaul, too numerous to list

  *Throughput is often an issue*

• **Soft real-time applications:**
  – IP network control signaling, network servers
  – Live audio-video systems on the edge or in data centers

  *Throughput with “good enough” real time response IS the issue*
Four Ways for Better Performance in Linux:

- The PREEMPT_RT patch
  - Rework the internals of Linux:
    - Linux Kernel

- "Thin-kernel" or virtualization
  - Add a thin real-time kernel underneath Linux:
    - Realtime Kernel
    - Linux Kernel
    - RT apps

- Vertical Partitioning + User mode Runtime
  - Vertically partition Linux in two domains:
    - Linux Kernel
    - RT Runtime

- Event Machine
  - Partition Linux in two domains; one not running Linux at all
    - Event Machine

CONFIG_PREEMPT_RT Patch Set
What Problem is PREEMPT_RT Trying to Solve?

*Minimize Linux Interrupt Processing*

*Delays from external event to response*

External Interrupt Triggered → Interrupt Taken → Critical section with interrupts disabled → HW Exception → “Top Half” / ISR → Signal/ Wakeup → Exit from IRQ → Reschedule → Context Switch → Interrupt Received in User/Thread Context

- Something else is executing (probably another ISR)
- E.g. locks (xtime lock could be one example?)
- Locks, RCUs, etc.
- Softirqs, RCUs
- Priority inversion/conflict
- Cache misses, etc.

Resource Conflicts
The CONFIG_PREEMPT_RT patch set

• Started 10+ years ago
  – Before multicore evolution; uni-core optimized technology
  – Many other contributors since then
• Replaces most kernel spinlocks with mutexes with priority inheritance
• Moves most interrupt handling to kernel threads
  – This means many drivers must be modified
• Roughly, PREEMPT_RT patches 500+ locations in the kernel, with 11,500+ new lines of code in total.
• In a multicore device, is “system wide in scope”

*Improves real-time performance (interrupt latency) but AT THE EXPENSE of throughput*
### PREEMPT_RT Throughput/RT Tradeoff

#### A Very Simple Example

**Linux 3.6.4:**

```
# netperf -H localhost -t TCP_STREAM -A 16K,16K -l 120 -C -D 20
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Send</td>
<td>Send</td>
<td>Send</td>
</tr>
<tr>
<td>Elapsed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td>local</td>
<td>remote</td>
</tr>
<tr>
<td>bytes</td>
<td>bytes</td>
<td>bytes</td>
</tr>
<tr>
<td>Time</td>
<td>secs.</td>
<td>10^6bits/s</td>
</tr>
<tr>
<td>87380</td>
<td>16384</td>
<td>16384</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Linux 3.6.4-rt10 (PREEMPT_RT):**

```
# netperf -H localhost -t TCP_STREAM -A 16K,16K -l 120 -C -D 20
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Send</td>
<td>Send</td>
<td>Send</td>
</tr>
<tr>
<td>Elapsed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td>local</td>
<td>remote</td>
</tr>
<tr>
<td>bytes</td>
<td>bytes</td>
<td>bytes</td>
</tr>
<tr>
<td>Time</td>
<td>secs.</td>
<td>10^6bits/s</td>
</tr>
<tr>
<td>87380</td>
<td>16384</td>
<td>16384</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*But this is a simple example that doesn’t always apply*
Other CONFIG_PREEMPT_RT Characteristics

• **ALL Linux Solution**
  – API’s / programming paradigm
  – Including all tools
  – BUT!! Requires driver modifications for all drivers

• **Compatible with Core Isolation/Shielding techniques**
  – Can work reasonably well for both real-time and throughput in a “bare metal” environment, i.e. no multithreading on isolated cores

• **Linux SMP style load balancing, for what it’s worth 😞**

• **Standard Linux memory protection**

• **Standard Linux Power Management**
Vertical Partitioning with a User Space Run Time Environment
Vertical Partitioning Concept

- Partitioning of the system into separate real-time critical (shielded cores) and non-critical domains.
- It is often the Linux kernel itself that introduces real-time problems.
- Real-time partition does not need full POSIX/Linux API.
- A combination of partitioning, combined with a user-mode environment that avoids using the kernel can improve performance and real-time characteristics compared to a standard Linux.

“Improve performance and realtime characteristics under Linux by partitioning the system into logical domains, and by avoiding usage of the Linux kernel and its resources more than necessary”
The Vertical Partitioning Concept (2)

• Configure processes and/or interrupts to run with core affinity
• Make modifications to the kernel to avoid running unnecessary
  kernel threads/timers on real-time cores
  • The NOHZ Patch
• Avoid using/calling the kernel, and rely on a user-mode execution
  runtime environment for applications

Use Cases:

a. **When targeting interrupt latency at a 3-10 us average and 15-30 us worst case requirements**

b. **When the application requires multi-threading performance**
How does it work?

Partition the system into one realtime domain and one non-realtime domain.

Add a user-mode runtime environment with a light weight scheduler – i.e. a very light weight “RTOS like” scheduler.

Migrate some specific kernel functionality (e.g. timers) away from the realtime domain. Implement NOHZ FULL patch

Add a kernel module to catch and forward interrupts to the user-mode environment.
What are the benefits?

Low latency and high throughput. Does not depend on the PREEMPT_RT patch, and does not affect throughput negatively.

Provide optimized APIs for realtime applications, and allows the same application to use the POSIX/Linux APIs when realtime doesn’t matter.

Provide very good (i.e. low-latency) interrupt response time, all the way up to user-mode.

Still an “all-Linux” solution, based on a single Linux Kernel. Thus, almost all tools from the existing Linux ecosystem will be available.
User Space Runtime vs Linux/PREEMPT_RT Performance
Scheduling Latency – vs Pthreads

Based on an Enea Prototype

Much better performance i.e. lower scheduling latency

Much better real-time characteristics, i.e. less variance.

Number of samples measured (ideally a single peak)

Clock cycles (lower is better)
Message Passing Latency

Based on an Enea Prototype
Interrupt Latency

*Based on an Enea Prototype*
Throughput ≈ “Idle” Time
Based on a Real-world LTE Example

In our example:
“Theoretical” maximum for a system with infinitesimally little overhead is 400 μs
Idle time (Throughput)

Based on an Enea Prototype

![Graph showing idle time (throughput) comparison between Standard Linux and User-Space Linux Executive.](image-url)
Other User Space Runime Characteristics

• **NOT ALL** Linux Solution
  – User runtime environment has different API’s
  – Does include all Linux tools, except for user space thread awareness
  – BUT, doesn’t require standard Linux driver modification

• Depends on Core Isolation/Shielding (NOHZ FULL)

• Slightly better real-time response/determinism than PREEMPT_RT
  – Interrupt handling model “cleaner”

• **Better than** PREEMPT_RT for Throughput
  – But only if Multithreading in the application is necessary
  – Not for bare metal solutions for Cores

• No load balancing – the current vertical partitioning concept prohibits it

• No memory protection between threading environments on a core
  – Best implementation requires ONE pthread per core

• Not standard Linux Power Management
Open Event Machine

sourceforge.net/projects/eventmachine
What does Event Machine Look Like?

EM partitions the system into one realtime domain and one non-realtime domain, like the vertical partitioning concept.

EM is a run-to-completion model for individual “contextless” work packages. NO threading or OS model.

EM does not require kernel mods, nor core isolation, but it can use core shielding, i.e. non-essential Linux processes and interrupts are migrated away from the EM cores. THE NOHZ FULL Patch

EM does not necessarily need a special interrupt handling model. Needs a “scheduler” in either Linux partition OR in HW.
Event Machine

- An efficient (low overhead) execution model for data plane processing.
- An “event” based programming paradigm, replacing traditional threads and processes.
  - “Events are data associated with code
  - Run-to-completion model code. This means “context-less” or “state-less” code for processing
- New “first class” OS primitives: queues, events, execution objects.
  - Can work within an RTOS environment!! See next slide
- A framework for distribution and scheduling in multicore scenario.
- A standardized API.
- HW offloading friendly API.
Push versus Pull Models

• Pull model
  – Simple design
  – Passive loadbalancing.
  – Offload a majority of scheduling decisions to HW
  – Core hot-plug(powersave) easier to implement.
  – Cache cold problems on MIMO/SIMO queues.

• Push model
  – Cache prefetching can be improved.
  – Active load balancing protocols needed.
  – Offloading scheduling decisions to I/O co/processor? i.e. smart HW queues.

• Push/Pull
  – Pull whenever HW can schedule I/O.
  – Keep it simple.
OS + Event Machine Scheduling Model

Interrupt Processes

Event Scheduling (in scheduler idle)

Preemption

Priority Processes

Background Jobs
Other Event Machine Characteristics

• NOT ALL Linux Solution
  – Different API’s, programming paradigm on EM cores
    • This means tools as well
  – Requires restructuring code into simple, non-preemptive, run-to-completion models .... “Context-less” processing

• Depends on Core Isolation/Shielding (can use NOHZ Patch)

• Superior for max data plane THROUGHPUT

• Real-time response is not part of the equation
  – Time to process events is not a parameter
  – But it “could” result in good real-time response depending on use case

• Designed for best load balancing on the data plane

• No memory protection EM instances on cores

• Not standard Linux Power Management
  – But not a hard problem to solve in a “Pull” model
Virtualization Techniques
- Virtualizes Linux on top of a real-time TYPE 1 Hypervisor
- Examples include hypervisor, Xenomai, RTLinux, WR, Enea, and perhaps Xen
- Provides a highly deterministic RTOS-like environment for RT apps
- Strong security support
- Cannot completely utilize the Linux eco-system (e.g. tools) in the real-time domain.
- Suitable for very high real-time requirements, especially those inherited from classic RTOS domains
Typical Embedded Type 1 Hypervisor Characteristics

• *NOT ALL* Linux Solution
  – Different API’s, programming paradigm for real-time cores
    • This means tools as well
• Superior real-time response, except for Xen
• Excellent THROUGHPUT
• Memory protection across cores

The “takeaway”:

• *Best use case for embedded hypervisors is for legacy migration or consolidation. Embedded hypervisors really not discussed too much anymore in the embedded industry*
KVM and Xen?

- **Xen is Type 1 Hypervisor, with excellent security features, and some performance advantages over KVM (all Type 1’s do over Type 2’s). Xen is used currently by many big Cloud Providers (Amazon, etc)**

- **But Xen is starting to lose to KVM**
  - Type 2 Hypervisor or Virtualization techniques are now becoming dominant in the Linux domain, like KVM etc. The real-time aspect of Type 1 Hypervisors in the Linux community is overall losing to the ease of use of a native Linux based virtual environments
  - Especially with the “encroachment” of the Cloud in the embedded domain where “elastic” solutions, i.e. the ability to quickly launch additional computing power (with connectivity) to meet demand helps “overall system” performance more than individual Linux node performance.
    - E.G. Cloud RAN (or C-RAN)
Four Ways for Better Performance in Linux:

The PREEMPT_RT patch
Rework the internals of Linux:

“Thin-kernel” or virtualization
Add a thin real-time kernel underneath Linux:

Vertical Partitioning + User mode Runtime
Vertically partition Linux in two domains:

Event Machine
Partition Linux in two domains; one not running Linux at all

Enea supports:
• Linux with PREEMPT_RT
• Type 1 Virtualization with Enea Hypervisor, Type 2 Virtualization with KVM
• A vertical partitioning user space solution (NOT Open Source Yet) on ARM A15 architectures end November 2013, with NOHZ FULL. Benchmarks coming.
• Event Machine with Broadcom XLP in Jan 2014
Thank you from Enea – the Real Time Embedded Linux Experts

Visit us at enea.com and/or see us in our small booth here at the show