Real-Time Linux and the Xenomai system

Giuseppe Lipari
`http://feanor.sssup.it/~lipari`

Scuola Superiore Sant'Anna

May 11, 2008

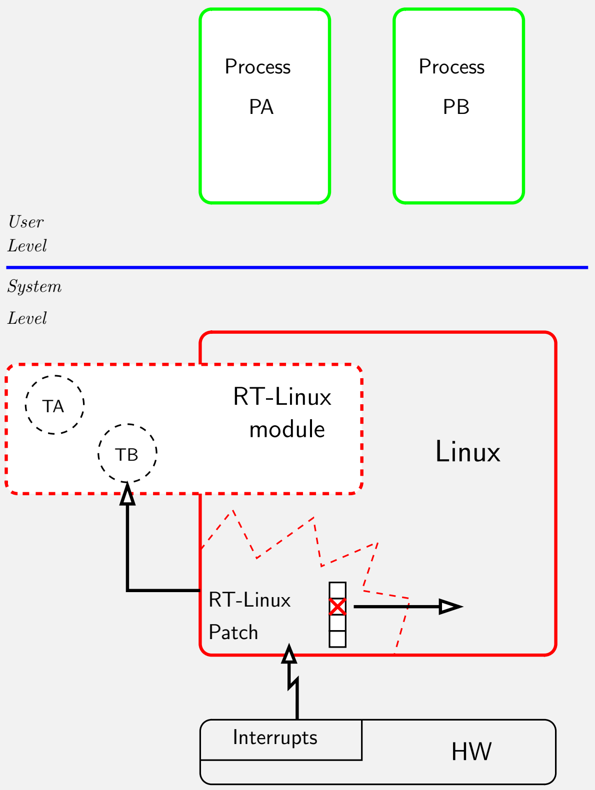
Outline

1. RT-Linux
   - Basic approach
   - RTAI-DPM
2. Adeos
   - Structure
3. Xenomai
   - Introduction
   - User-mode threads
   - Interfaces
   - User mode device drivers
   - Summary of Xenomai

RT-Linux

- RT-Linux was the first approach to making Linux more real-time
  - It consist of a patch to Linux, plus a dynamic loadable kernel module
- The patch:
  - Modifies the interrupt handling sub-system of Linux, in order to intercept and service the *real-time* interrupts
- The module contains:
  - the nano-kernel (scheduler + interrupt handler + libraries)
  - application code
- The application code executes with system privileges

RT-Linux architecture



- When an interrupt is raised
  - If it is for the RT sub-system, it is redirected to the nano-kernel and to the RT-application
  - If it is for Linux, it is simply *marked* as pending
  - and it will be served only when all RT tasks have completed execution
  - Linux must be executed with interrupts always active

Priority ordering

- In RT-Linux
  - Real-time activities (tasks and interrupt handlers) have always priority over Linux activities (tasks and interrupt handlers)
- This is made through the following two mechanisms:
  - interrupt interception
  - virtual cli/sti instructions

Linux interrupt sub-system

- We have to prevent Linux from disabling interrupts for a long time
  - otherwise, real-time activities could be delayed too much
  - Therefore, an instruction like `CLI` is substituted by a function that marks interrupt as disabled for the Linux system
  - Interrupts can still arrive and be served by the RT sub-system
  - If they are for Linux, the RT subsystem marks them as pending, if they are for the RT application, they are immediately served
  - When Linux wants to execute `STI`, a function is invoked that goes through the marked interrupts and serves them all

Advantages – disadvantages

- This approach has one big advantage
  - Minimum latency: RT interrupts cannot be blocked by Linux, but only by the RT sub-system
- But also some disadvantages:
  - Real-time applications run in kernel space →, so they can overwrite the kernel memory and crash the system (like any other single memory RTOS)
  - Communication between RT subsystem and Linux can only be non real-time;
  - Linux is scheduled in background → Linux tasks can starve or experience high delay
  - It is not possible to use Linux device drivers for RT
  - RT developers have to re-write the related device drivers on top of the RT sub-system

RTL – History

- The technique used by RT-Linux was first proposed by Yodaiken e Barabanov (University of New Mexico),
- it was initially distributed as open source software (GPL license)
- later they patented the interrupt interception method (?!)
- they founded FSM-labs to sell a professional version of RT-Linux
- they own the internet domain `http://www.rtlinux.org`
- there is still an open source version or RT-Linux, but it is not very well supported

RTAI + Adeos

- Paolo Mantegazza at Politecnico di Milano, started developing on RT-Linux
- However, he was in contrast with Yodaiken since the very beginning
- He branched off and made its own version of RT-Linux, called RTAI (Real-Time Application Interface)
- After some time, under the menace of being sued by Yodaiken, they substituted all RT-Linux code with Adeos, and re-wrote entirely the RT-nano kernel
- Recently, a very interesting approach called Xenomai branched off from RTAI
- I will present Xenomai in the rest of the presentation

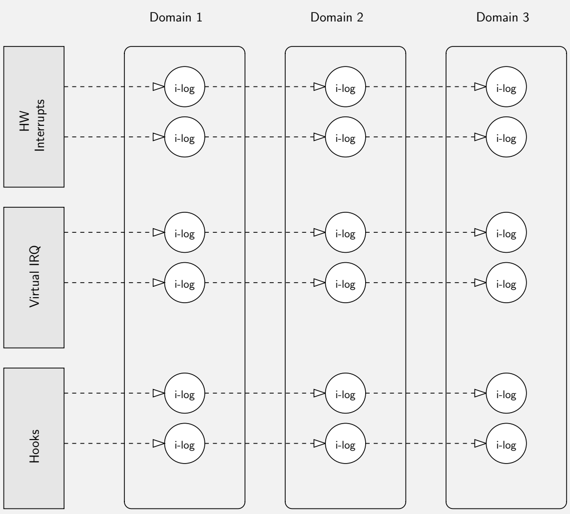
Adeos

- Adeos is a layer of software that can be used to virtualize interrupts in a general and flexible way
- It generalizes the basic concept used by RT-Linux
- But the technique is descibed in a paper published *before* the RT-Linux patent!
- Therefore, there *should not* be any problem with patents and proprietary software (with lawyers, you never know . . .)

Adeos basic concepts

- Adeos handles domains
- A domain contains an *entity* able to handle interrupts
- the word interrupt here includes both hardware and software interrupts (everything that can be trapped).
- it also includes hooks on task switches, signals, etc.

Adeos basic structure



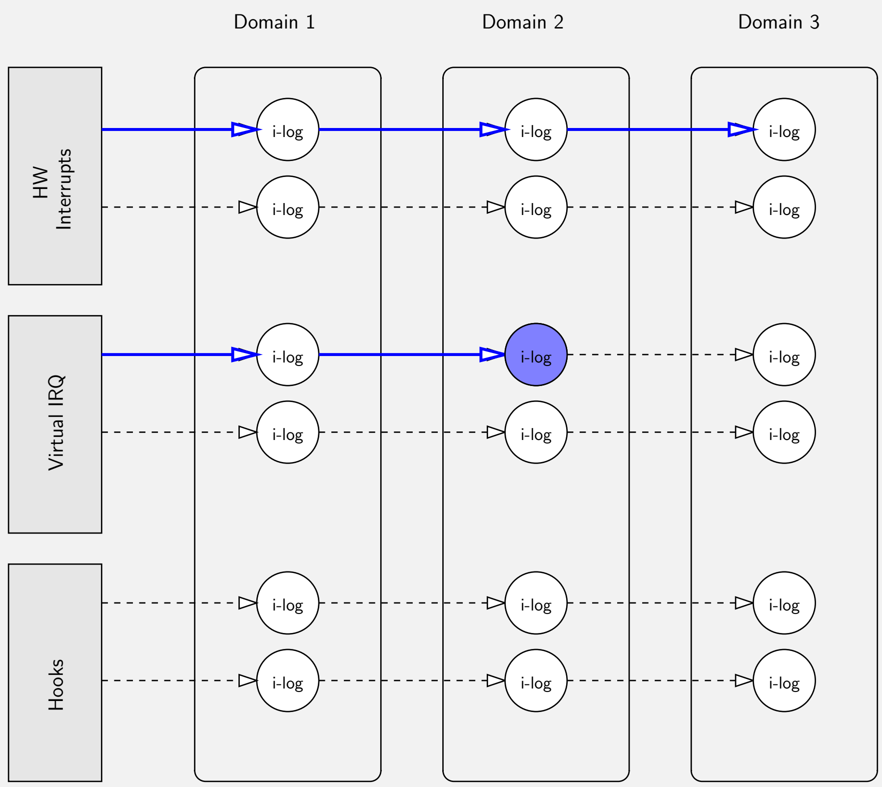
- Domains are identified by a unique number
- Domains with lower numbers have higher priority for handling events
- Event propagations happens according to a pipeline

Rules for propagation of events

- Every event goes from the first domain up to the last one (whoever generated it)
- A domain can forward the event of stop it
- A domain can also *stall* events
  - this is equivalent to disable interrupts for the subsequents domains
  - The events can be *unstalled* later, and at that point they are forwarded to the subsequent domains
  - previous domains in the pipeline are not affected by the stalling
  - Of course, it is possible to selectively decide which events to stall.

Adeos domains

Event propagation: Stalling



How to use Adeos

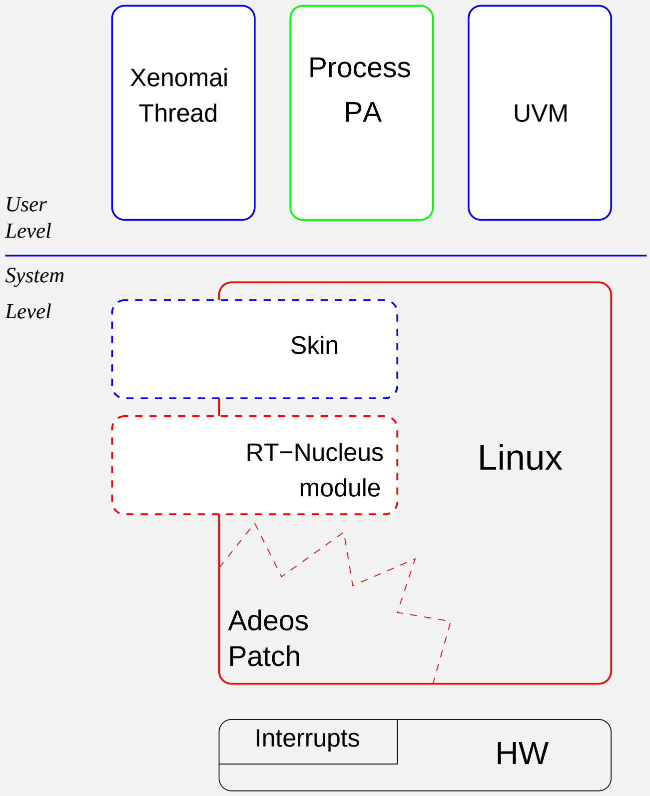
The basic idea is:

- Linux goes in domain 3
- A real-time OS goes in the first domain
- The second domain is used to stall events
- Linux operations for disabling/enabling interrupts are modified as `stall` and `unstall`

Xenomai history

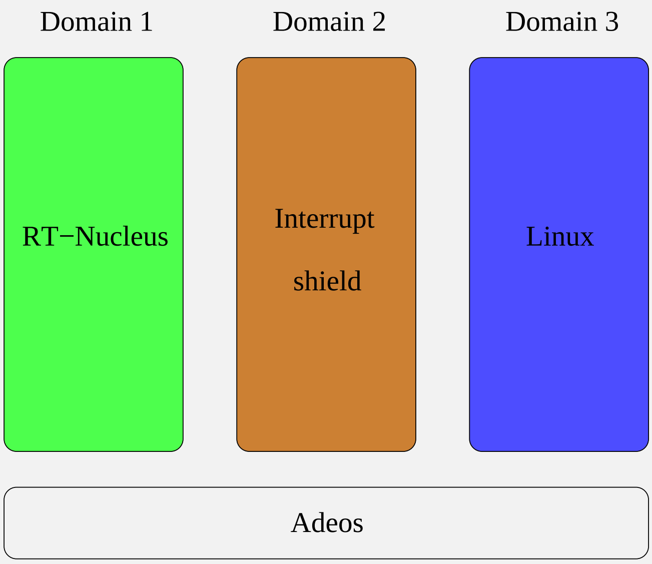
- Xenomai is a new project
- it is a banch of RTAI
- The maintainer (Philippe Gerum) was RTAI maintainer, then branched off
- Differently from other Linux-RT projects, it has a lot of documentation!

Xenomai architecture



- The basic structure is similar to RTAI;
- However, Xenomai provides a better integration with Linux
- New features: Xenomai threads, Skins, UVMs

The structure of domains under Xenomai



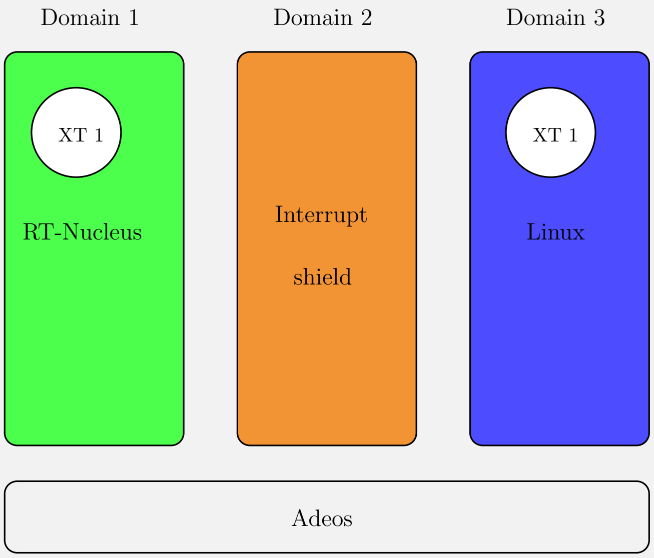
- the primary domain (1) runs a real-time kernel
- the secondary domain (3) runs Linux
- an intermediate domain (2) is used as *interrupt shield*

Xenomai threads

A real-time thread can execute

- always in the primary domain
  - In this case, it is very similar to a RT-Linux or RTAI rt-thread.
  - Its memory space is the same as the kernel memory space
  - they have very short response time
- In the secondary and in the primary
  - We call these Xenomai Threads
  - Their memory is in user space
  - They are memory-protected from other processes, and cannot crash the kernel
  - They can be executed both by the primary or by the secondary domain

Jumping from primary to secondary



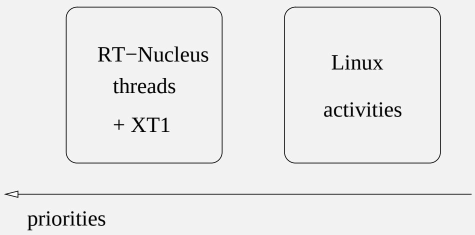
- A Xenomai thread starts in primary mode
- When it invokes a non-rt syscall, it *jumps* in secondary mode

Xenomai Threads

- The priorities used in the primary (RT-Nucleus) are *compatible* with those used in the secondary (Linux)
- In particular, RT-Nucleus provides 100 RT priorities like Linux
- When a Xenomai thread *jumps* across domains, it maintains its own priority
- Therefore, if a thread has priority 96 in RT-Nucleus, when it goes in secondary mode, Linux will serve it with RT priority 96

Xenomai threads in primary mode

- A Xenomai thread (or a set of threads) usually starts in primary mode.
- When the thread is in primary mode:
  - It is removed from Linux ready queue
  - It is served by RT-Nucleus scheduler
  - Has always precedence over any other Linux process (even when the Linux process has an higher priority).
- In primary mode, the Xenomai thread contends with the RT-threads of RT-Nucleus



Jumping to secondary mode

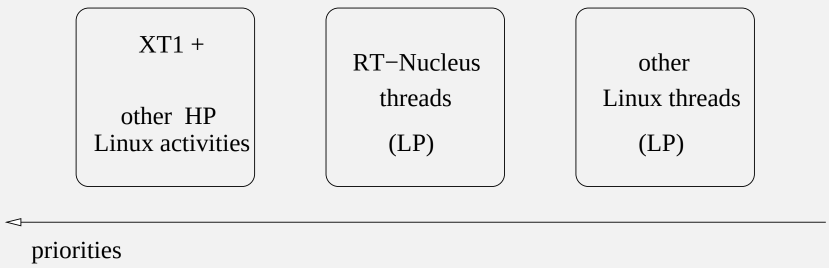
- A Xenomai thread remains in primary mode until is invokes a non-RT primitive.
  - For example, suppose that the thread invokes a `printf()` or a writes to a file.
  - In such a case, the thread is moved to secondary mode
- When a Xenomai thread is moved to secondary mode:
  - RT-Nucleus inserts the Xenomai thread in the Linux queue
  - RT-Nucleus invokes the Linux scheduler
  - All Linux is scheduled with the same priority of the Xenomai thread

Priority relationship

When

- at least one Xenomai thread (XT1) runs in secondary mode,
- and XT1 has highest priority among RT-Nucleus threads;

the priority ordering is as follows:



Priorities in secondary mode

- If a Linux process with priority higher that the Xenomai thread arrives, it is executed by Linux
- If a RT-thread with priority lower than the Xenomai thread arrives while it is in secondary mode, it cannot make preemption
- If a RT-thread with priority higher than the Xenomai thread arrives, it makes preemption

Mixing Linux RT processes and Xenomai threads

- No need to say that this is a dangerous thing to do
- The Linux RT processes could preempt Xenomai threads while these are in secondary mode.
- Suggestion: never mix the two.

Linux interrupts

To reduce latency:

- If an interrupt arrives while a Xenomai thread executes in secondary mode, it is not forwarded to Linux!
- The mechanism works as follows:
  - When a Xenomai thread is executed in secondary mode, the *interrupt shield domain* is activated
  - All Linux interrupts are stalled, until the Xenomai thread completes execution
  - The interrupts will be served again when Linux goes back to execute in background mode.

Latency

The worst-case latency could happen when

- Xenomai thread goes to secondary mode
- Linux scheduler is invoked
- However, Linux was in the middle of completing an interrupt processing, and preemption was disabled.
- Therefore, we have to wait until Linux re-enables preemption

Due to improvement to latency reductions in latest versions of Linux, this latency will be reduced even further in the future

Shadow thread

- This is implemented through the shadow threads in RT-Nucleus;
- Each Xenomai thread has a corresponding shadow thread in RT-Nucleus;
- When the RT-Nucleus schedules the shadow thread, the Xenomai thread is executed instead, *wherever the tread is located in that moment*.

Xenomai Interfaces

- Xenomai provides several API to the user
- The internal API (core) is the interface used internally by RT-Nucleus. Should not be used directly
- many skins built on top of the internal interface
- One of the skins is the Native Xenomai interface (again built on top of the internal interface)
- A skin is a loadable kernel module

Interfaces currently available

- native
- POSIX
- PSOS
- RTAI
- μ-Itron
- VRTX
- VxWorks
- rtdm (rt driver model)

Basic idea

- All RTOS have a very similar behavior, in particular regarding the APIs
- but different internal implementations
- the differences in the APIs are somewhat non-essential
    - priority ordering
    - semaphore queues
    - etc.
- Moreover, task states are very similar

Task states

The developers of Xenomai found that:

- by supporting the POSIX states,
- and the μ-Itron states,
- they cover all possible task states in all interfaces (until now)

The core interface:

- It is very essential
- every syscall can be implemented as a sequence of core APIs calls

The RTDM skin

- By using the mechanism of Xenomai threads (jumping between secondary and primary),
- it is possible to write device drivers in user space, with good response time
- the idea is that a thread can wait for interrupts
- basically, the entire handler is embedded into a Xenomai thread
- the thread waits for an interrupt line (blocks)
- when the corresponding interrupt line is raised, RT-Nucleus unblocks the thread

Advantages

- Xenomai enables RT response times in user space, allowing
  - memory protection
  - easy of debugging
- It provides an emulation layer (UVM) for trying out how the code works in user mode and with GDB
- It provides skins to support many different RTOS interfaces