Project
Current and future

Pierre FICHEUX - pierre.ficheux@smile.fr

June 2023

whoami

- Embedded Linux expert
- Not a flight software (nor space industry) specialist !
- CTO for Smile (ECS), a french software service provider
- Teacher for Embedded Linux (Yocto), Linux RT, AOSP

Some RTOS

- Most of “embedded OS” are RTOS
- Proprietary (VxWorks, pSOS, VRTX, LynxOS, µC/OS-II)
- Open source (RTEMS, FreeRTOS, eCOS, Zephyr)
- homemade OS
- UNIX (and so Linux) was not designed as a RTOS
- Linux real-time scheduling needs kernel patches (PREEMPT_RT, RTAI, Xenomai)

POSIX scheduling policy

- The Linux kernel is POSIX compliant provides the following policies:
  - SCHED_OTHER (priority 0, no real-time)
  - SCHED_FIFO (priority from 1 to 99)
  - SCHED_RR (same as SCHED_FIFO with “round-robin”)
  - SCHED_DEADLINE (EDF + CBS algorithms for the Linux kernel)
- Policy and priority can be statically defined in the source code
- The “chrt” command is useful to define the policy and priority for a process
- Policy and priority must be defined as “attributes” for a pthread

Measuring the kernel latency

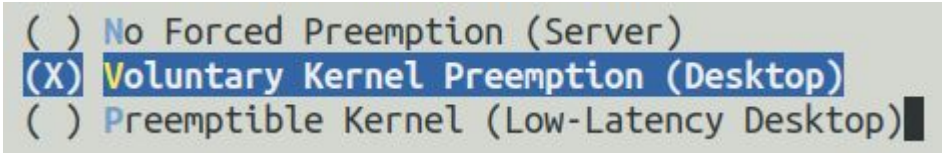
- A real-time application is based on periodic tasks
- One can evaluate the performance with the following procedure:
  - Start a real-time periodic task (SCHED_FIFO, SCHED_RR, SCHED_DEADLINE)
  - Increase the system load
  - Compare the measured deadline with the theoretical one (the difference is called “jitter” or “latency”)
- Several tools are available for testing, such as the “rt-tests” package (available in standard distros + Yocto, Buildroot)
  - cyclictest (create real-time tasks)
  - hackbench (increase system load with *non real-time* tasks)
- Use “Ftrace” for a real application

Improving the performances

- Using kernel preemption options (obsolete)
- Using the PREEMPT_RT patch
- Using the co-kernel approach (RTAI / Xenomai)

Preemption options

- Designed for 2.4
  - “preempt-kernel” by Robert Love (MontaVista)
  - “low-latency” by Andrew Morton
- Similar features integrated to mainline in 2.6 and still available in 6.x !
  - CONFIG_PREEMPT_NONE
  - CONFIG_PREEMPT_VOLUNTARY (“explicit” preemption points added by Ingo Molnar in 2005, default option)
  - CONFIG_PREEMPT (original preemption option, Robert Love)

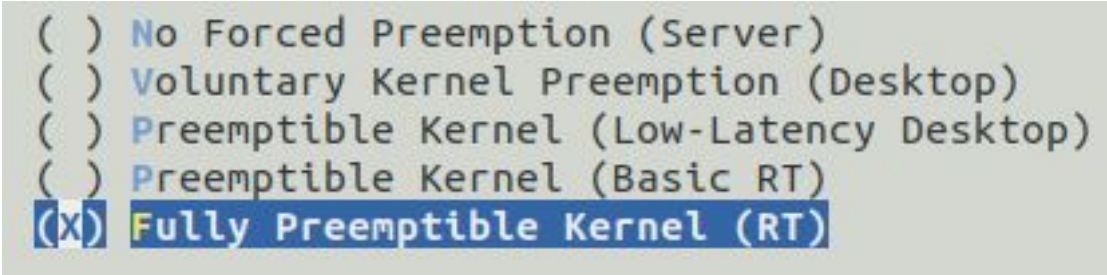


PREEMPT_RT

- Started as an experimental branch by Ingo Molnar in 2004
- First patch published for 2.6.9
- Designed for the mainline kernel
- Maintained by Ingo Molnar, Steven Rostedt and Thomas Gleixner
- Finally became an official project for the Linux Foundation in 2015 named the “Real Time Linux Collaborative Project”
- Mainlining the PREEMPT_RT has been around since 2008 but it’s still not achieved today
- Easy to install (one kernel patch), same user/kernel API
- Good performance
- Ready-to-run distributions available (RHEL and Ubuntu)

Some PREEMPT_RT features

- High resolution timers support (hrtimer)
- Dynamic ticks (CONFIG_NO_HZ, CONFIG_NO_HZ_FULL)
- Priority inheritance
- Threaded interrupt model
- The maximum jitter for the Raspberry Pi 3 is < 100 µs
- Select CONFIG_PREEMPT_RT_FULL

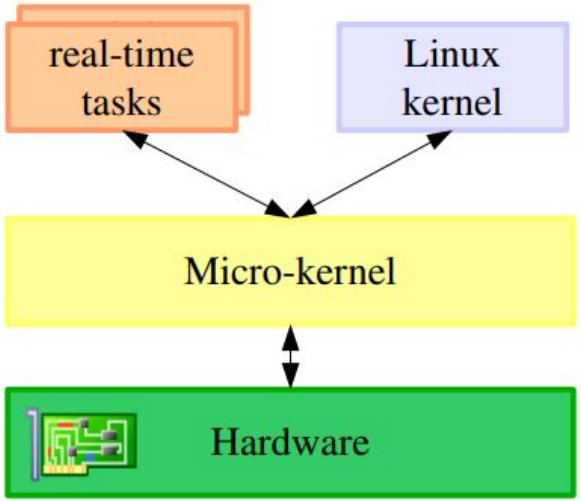


Co-kernel (the truth is elsewhere ?)

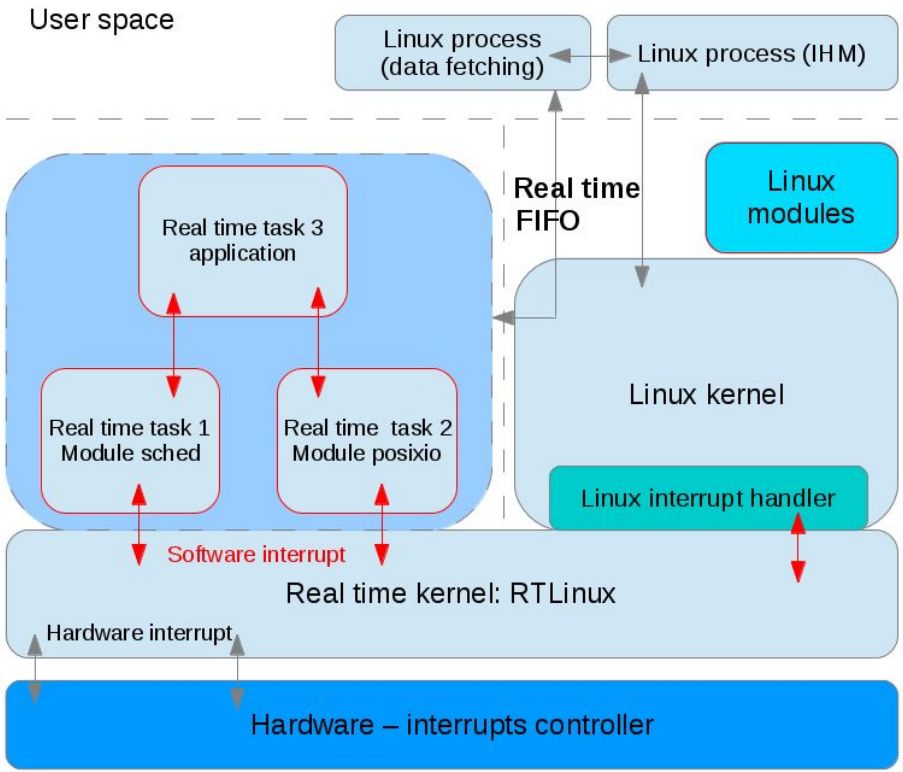
- A very different approach !
- Adding a dedicated real-time kernel to the Linux kernel
- A real-time sub-system based on kernel modules
- Two “domains” for the application
  - Real-time domain for RT threads
  - Linux domain for NRT threads
- Several models
  - Kernel only (RTLinux, the ancestor !)
  - Kernel and (partial) user space (RTAI, a “fork” of RTLinux)
  - Kernel and (full) user space (Xenomai)
- User space support is very important for the industry regarding licensing (GPL vs LGPL) !

Co-kernel principle

- Use a real-time specific scheduler (not the Linux scheduler)
- Interrupt handling virtualization by a “micro kernel”
  - The kernel does not not mask interrupts
  - real-time interrupts have higher priority
- Linux is an “idle task” for the real-time kernel



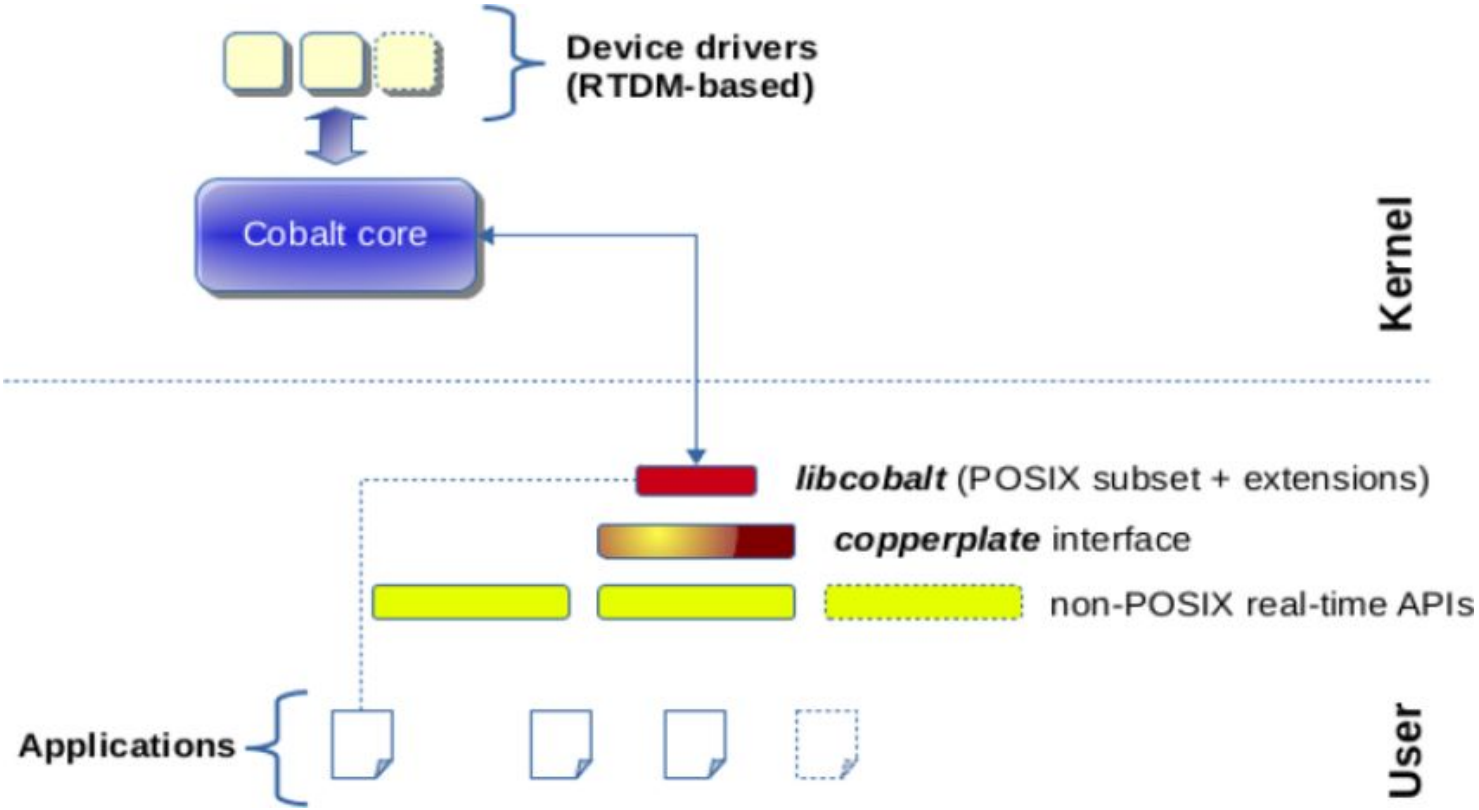
RTLinux architecture (old)



Xenomai

- Designed by Philippe Gerum in 2001 for RTOS API emulation (aka “skins”) such as VxWorks, VRTX, etc.
- RTAI collaboration to work around problems with the RTLinux patent (RTAI/fusion in 2004)
- Since 3.x, two choices for the architecture:
  - co-kernel (Cobalt), mostly used
  - single kernel (Mercury) → skins over PREEMPT_RT
- Interrupt handling virtualization
  - “Dovetail” (from the EVL project) for kernel >= 5.10
  - I-pipe for a older kernels
- The current stable version is 3.2.1
- Xenomai 4 is EVL - https://evlproject.org/ (using “Dovetail”)

Xenomai 3 / Cobalt



The RTDM API

- A RTDM driver is a Linux module
- A RTDM driver runs in the Xenomai domain
- Close to the Linux kernel API but one needs to adapt the drivers to the RTDM API !

Xenomai “skins”

- Alchemy (aka “native”)
- POSIX
- pSOS
- VxWorks
- Smokey (test API)
- RTDM (kernel space)
- Several skins are usable at the same time
- Very nice feature as it provides POSIX source compliance

Xenomai 😁

- The most efficient real-time extension for the Linux kernel (better than PREEMPT_RT)
- A real-time task runs in user space (LGPL)
- Provides “skins” for RTOS API (POSIX, VxWorks, VRTX, etc.)
- Used (and maintained) by big companies such as SIEMENS
- Yocto and Buildroot integration

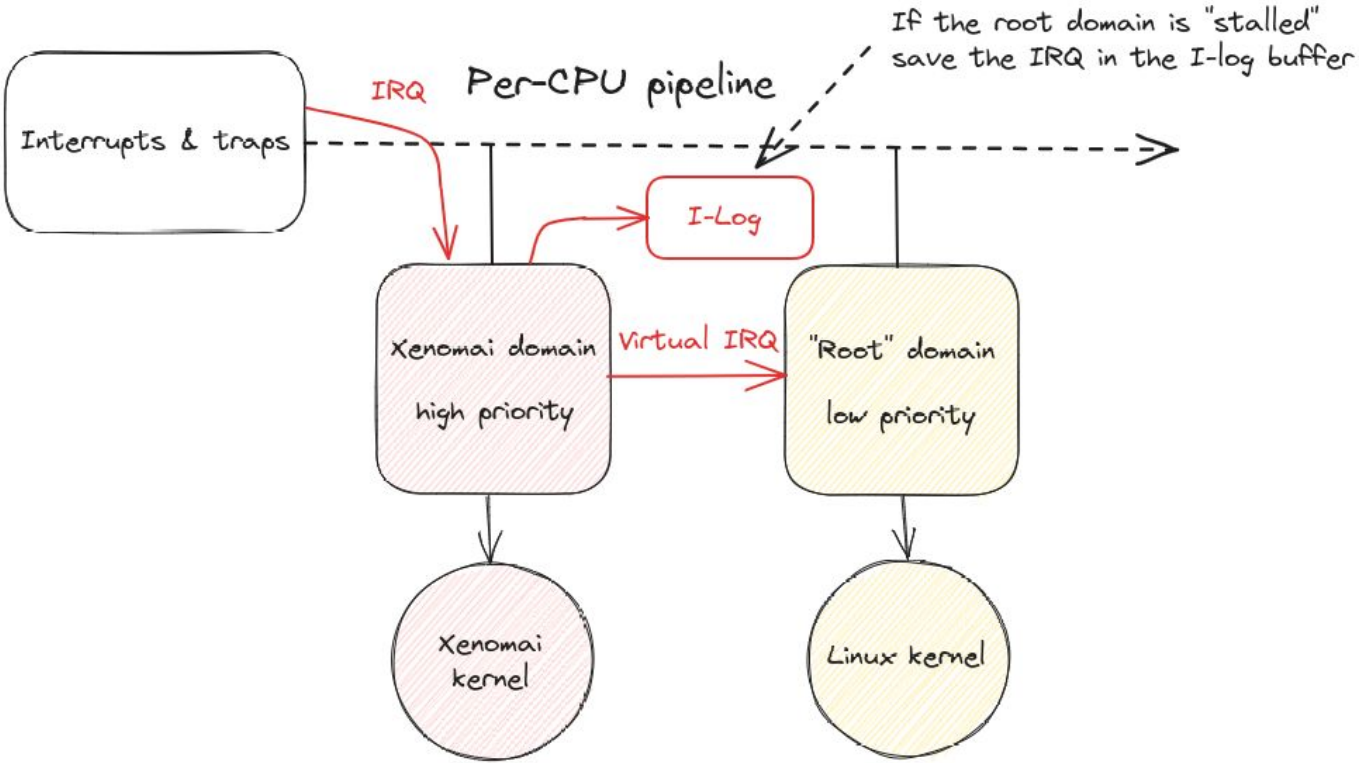
Xenomai 😢

- Small community
- Lacks of examples (except from some users)
- Cobalt is difficult to set up (and use) because of:
  - Most of the time, a mainline kernel is mandatory
  - User / kernel installation
  - RTDM (Real Time Driver Model, specific driver API)
- Not an official project from the Linux Foundation
- No “Xenomai ready” distribution

Domains and I-pipe

- Guest OS (Linux, Xenomai) run in prioritized “domains”
- Xenomai runs in the highest priority domain
- Linux (aka “root”) runs in the lowest priority domain
- For each event (interrupts, exceptions, syscalls, etc.), the domains may handle the event or pass it down the pipeline
- Calls to standard IRQ handlers should be replaced by calls to I-pipe functions
- Read the article “Life with ADEOS”

Interrupt dispatching principle



Dovetail

- Initially a fork of I-pipe for the EVL project !
- Introducing a high-priority execution stage enabling all device interrupts to behave like NMIs
- Increasing the possibility to maintain Dovetail with common kernel development knowledge (porting is simpler)
- Tracking the most recent kernel releases
- Available from Xenomai 3.2 (no backport to 3.1)
- Instead of patching the kernel we use the “Dovetail ready” kernel (based on the mainline)

Installing Xenomai

- More complex than PREEMPT_RT, as you need:
  - A compatible kernel (Dovetail ready of I-pipe compatible)
  - The Xenomai sources
- Build the kernel
  - `$ ./scripts/prepare-kernel.sh --linux=<kernel-path> --arch=<arch> [--ipipe=<ipipe-path>]`
  - `$ make`
- Build the user space libraries, demos, etc. (Autotools)
  - `$ configure --host=<cross-toolchain-name> --enable-smp`
  - `$ make`
- Use Yocto (or Buildroot) !
  - Add the provided layer for Xenomai support (BSP dependent)
  - Use `BR2_PACKAGE_XENOMAI` and `BR2_LINUX_KERNEL_EXT_XENOMAI` for Buildroot

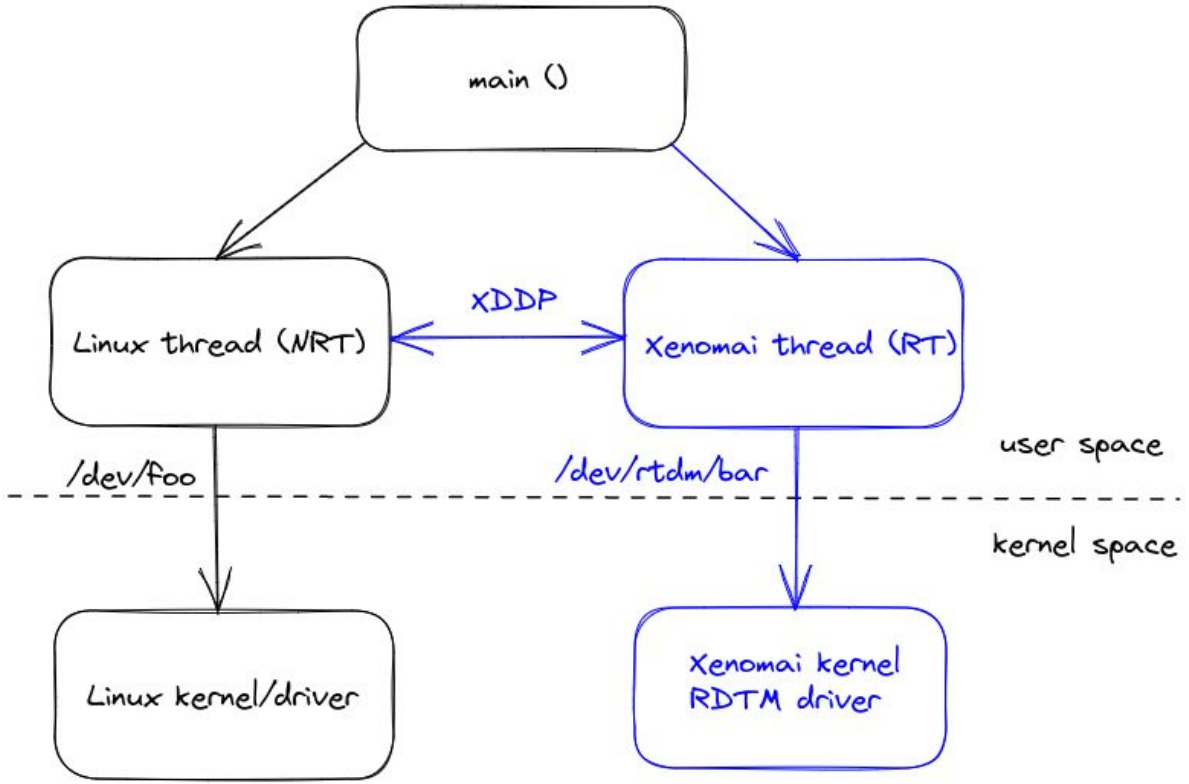
Testing Xenomai

- Very close to the PREEMPT_RT one:
  - Periodic task (RT domain)
  - System stress (Linux domain)
- Xenomai provides “latency” and its own “cyclictest”
- Start latency (the default period is 1 ms)
  - `# latency`
- Use “dohell” (or hackbench) to stress the system
  - `# dohell 600`
- The maximum jitter for the Raspberry Pi 3 is about 30 µs

Xenomai application basics

- A Xenomai application is a Linux program using Xenomai libraries
- The application runs on two “domains”:
  - Linux for non real-time threads
  - Xenomai for real-time threads
- Most of the time, the “main” function is managed by the Linux domain
- RT and NRT domains communicate with the XDDP protocol

Xenomai application basics



Designing a Xenomai application

- “Segregate” RT / NRT code
  - RT threads are managed by Xenomai
  - NRT threads (i.e. SCHED_OTHER) managed by Linux
- Select an API (aka “skin”)
  - POSIX is the best for portability
  - Xenomai native/alchemy skin is usable too
  - Other skins (such as VxWorks) are useful for porting
- You can force using Linux system calls with `__real_` prefix such as `__real_pthread_create()`
- Lock the allocated memory with:
  - `mlockall (MCL_CURRENT | MCL_FUTURE)`

Compiling a Xenomai application

- Based on the “xeno-config” script:
  - Created when building Xenomai
  - Defines path and options depending on skins
- Compilation options examples:
  - `$ xeno-config --cc`
  - `$ xeno-config --posix --cflags`
  - `$ xeno-config --posix --ldflags`
  - `$ xeno-config --native --cflags`
  - `$ xeno-config --native --ldflags`
- Some external projects for CMake support

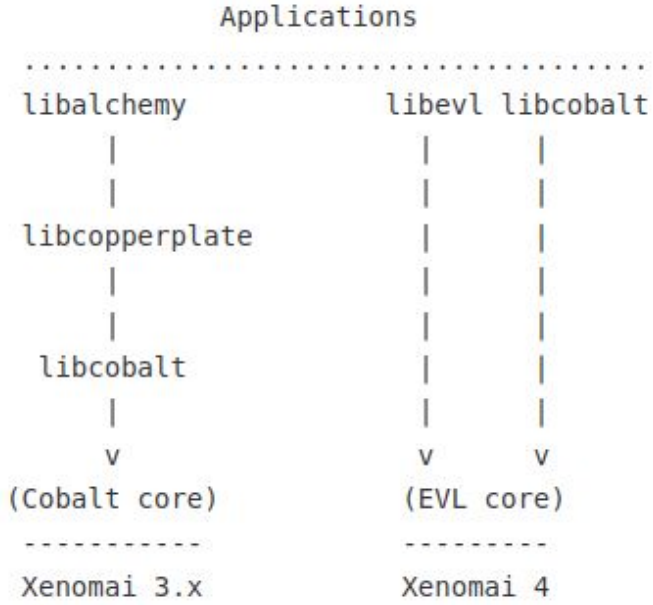
Running / debugging an application

- Use the `/proc/xenomai` pseudo-filesystem
- Be careful about domain migration (MSW) !
  - MSW occurs when using Linux system call from Xenomai domain
  - Thanks to the SIGDEBUG signal, it’s possible to catch a migration with a signal handler
- Ftrace is available for Xenomai (Cobalt events)
  - `# trace-cmd record -e "cobalt_*"`
  - `# trace-cmd report`

EVL (Xenomai 4)

- EVL was originally a fork of the Xenomai 3 Cobalt core
- Based on Dovetail
- Small footprint (just like before)
- An application is (currently) based on “libevl”
- Xenomai 4 will provide two direct interfaces to the underlying EVL core:
  - “libevl” which is readily available
  - POSIX from “libcobalt” (Xenomai 3)

Common Xenomai Platform (CXP)



Xenomai 4 demo

- Based on Yocto 4.1
- Designed by Lukasz Majewski (DENX) for the Raspberry Pi 4
- The “latmus” benchmark is used instead of “latency”
- The “hackbench” tool is usable

Questions ?

References

- https://source.denx.de/Xenomai/xenomai/-/wikis/home
- https://source.denx.de/Xenomai/xenomai/-/wikis/Common_Xenomai_Platform
- Xenomai 3.x over Dovetail https://www.mail-archive.com/xenomai@xenomai.org/msg18635.html
- https://evlproject.org/overview
- EVL application https://evlproject.org/core/user-api
- Xenomai 4 benchmarks https://evlproject.org/core/benchmarks
- Xenomai 4 on Pi 4 https://source.denx.de/lukma/meta-xenomai-demo