针对evl的posix和ethercat生态构建

Building the Ecosystem of POSIX and EtherCAT for libevl

邱奇琛

RROS group

北京邮电大学

2024.11.26

Agenda

POSIX wrapper

- Introduction
- Xenomai Implementation
- Our works
- Key Challenges

EtherCAT

- Overview
- IgH Ethercat and Xenomai
- Porting to EVL

Why we need a POSIX wrapper in evl4?

- The current EVL API (libevl) is simple, elegant, and well-documented, but…
  - Not fully compatible with POSIX API.
    - Some usages may differ for users accustomed to vanilla Linux.
    - e.g: attach_thread, timer, signal, etc.

- Facilitating the smooth migration of Xenomai3 applications to EVL.
  - Xenomai2&3 already includes a POSIX wrapper.
  - Some users may want to transport it to EVL
    - newer version of Linux kernel
    - Better SMP support
    - Support native OOB network stack, ebpf, valgrind etc

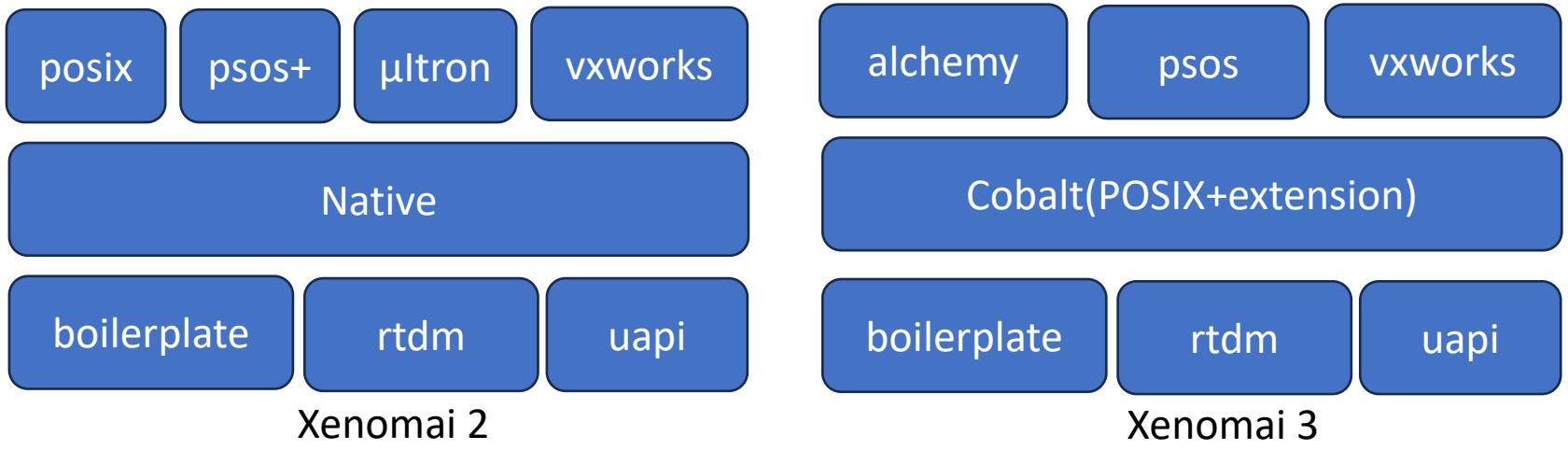
Native wrapper with better performance

POSIX wrapper

Previous works – Xenomai 2&3

- Both Xenomai 2&3 have a POSIX wrapper.
  - The wrapper uses the compiler's --wrap mechanism to reassign symbols during linking.
  - In Xenomai 2, the POSIX wrapper is based on the native API.
  - In Xenomai 3, libcobalt serves as the native API.

- Since we already have the libevl, the situation is more similar to Xenomai2



Agenda

POSIX wrapper

- Introduction
- Xenomai Implementation
- Our works
- Key Challenges

EtherCAT

- Overview
- IgH Ethercat and Xenomai
- Porting to EVL

Internal of Xenomai wrapper
linking

- The C/C++ compiler provides the --wrap option
  - -Wl,@/usr/xenomai/lib/cobalt.wrappers
  - It redirects all references to foo to __wrap_foo, while mapping the original foo symbol to __real_foo.
  - By implementing a custom __wrap_foo, users interact with the custom wrapper function instead of the original foo.

- Xenomai offers specific macros to simplify managing these wrapper functions:

```
#define COBALT_DECL(T, P)    \
    __typeof__(T) __RT(P);   \
    __typeof__(T) __STD(P);  \
    __typeof__(T) __WRAP(P)
```

```
#define COBALT_IMPL(T, I, A)                                    \
__typeof__(T) __wrap_ ## I A \
__attribute__((alias("__cobalt_" __stringify(I)), weak));   \
__typeof__(T) __cobalt_ ## I A
```

Internal of Xenomai wrapper
linking

- Inside the POSIX wrapper, we use `__STD` macro to avoid recursion

```
void __wrap_foo(){
    // ...
    // foo() // recursion
    __STD(foo())
    // ...
}
```

- But things can be a mass sometimes if we use static linking.
  - Example:

```
// foo is a common POSIX API,
// which may be called by others functions
void __wrap_foo(){
    // ...
    bar() // static linking
    // ...
}
```

```
// In other library:
void bar(){
    // ...
    foo() // will be replaced
}
```

Internal of Xenomai wrapper

Two stages compiling

- Solution:
  - Two stages compiling:
    - Firstly compile the source file with wrap option enabling and **without** stdlib.
    - Then compile and link the results with common library.
      - This time do not use wrap option. So that the symbol inside std library will not be changed.

- For further details, refer to the script: scripts/wrap-link.sh.

```
$1 -v gcc -o foo foo.o -Wl,@/usr/xenomai/lib/cobalt.wrappers -L/usr/xenomai/lib -lcobalt -lmodechk -lpthread -lrt
will print and run:
+ gcc -o foo.tmp -Wl,-Ur -nostdlib foo.o -Wl,@/usr/xenomai/lib/cobalt.wrappers -Wl,@/usr/xenomai/lib/modechk.wrappers -L/usr/xenomai/lib
+ gcc -o foo foo.tmp -L/usr/xenomai/lib -lcobalt -lmodechk -lpthread -lrt
+ rm foo.tmp
```

Internal of Xenomai wrapper

wrap main & bootstrap

- By default, the Xenomai wrapper automatically wraps the main function.
  - Initialization (bootstrap and init) occurs before the program's entry point.

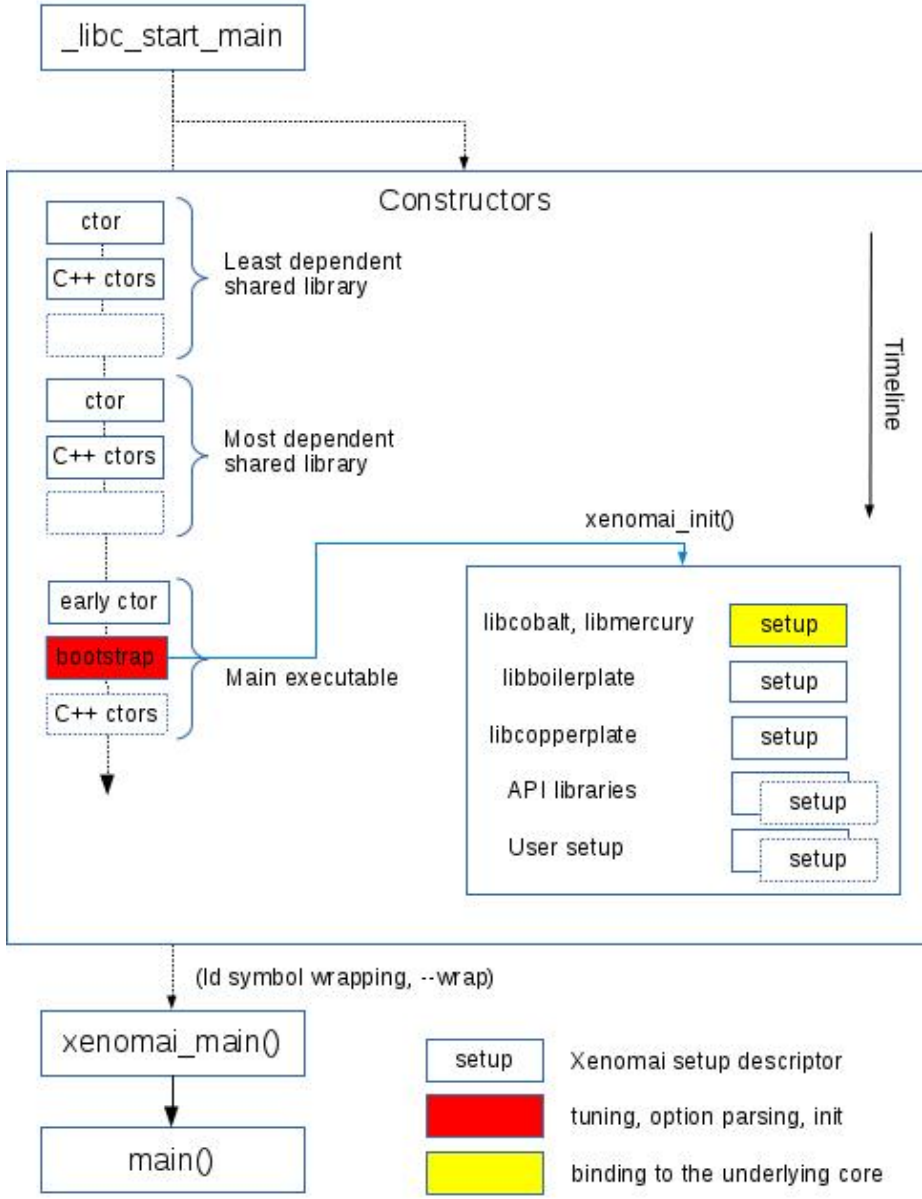
- Static linking: Uses the --wrap option to replace the original main and initialization logic.

- Dynamic linking: enable the macro __BOOTSTRAP_DSO__,
  - Initialization is handled by the same function (xenomai_init)
  - But in this case, the main function is not wrapped.

Internal of Xenomai wrapper
setup

- Wrappers are registered as modules.
  - Each wrapper includes a setup descriptor.
  - The setup descriptor is added to a private list, sorted by priority.
  - Wrappers are registered only when their corresponding library is linked.

- In Xenomai 3, cobalt(POSIX) is firstly loaded while in Xenomai2, native wrapper is loadded first.



New cobalt wrapper

- This month, Florian Bezdeka post a new method of refactoring cobalt wrapper[1].

- Replace all the posix symbols with a prefix of `__cobalt_trmp`.
  - Trumpoline just simply jumps to the wrapper function.

```
#define COBALT_SERVICE_WRAPPER(fn)                                          \
        asm(".text \n"                                                      \
            ".weak " #fn "\n"                                               \
            ".set " #fn ", __cobalt_trmp_" #fn "\n"                         \
            ".globl __cobalt_trmp_" #fn "\n"                                \
            ".type __cobalt_trmp_" #fn ", @function \n"                     \
            "__cobalt_trmp_" #fn ":\n"                                      \
            "jmp __wrap_" #fn "\n"                                          \
            ".size __cobalt_trmp_" #fn ", .- __cobalt_trmp_" #fn "\n")
```

Example:

```
0000000000408120 <__cobalt_trmp_pthread_create>:
  408120: e9 6b 09 00 00        jmp    408a90 <___cobalt_pthread_create>
```

[1] https://gitlab.com/Xenomai/xenomai-hacker-space/-/commit/017f1e4ee2e83cd1102c22c10c5c35d59cde13a6

New cobalt wrapper

- Original symbols are stored with the prefix __real_
  - The macro STD is used to automatically add this prefix to symbol names.

```
#define COBALT_DECL(T, FN, I)                                    \
        extern __typeof__(T) __cobalt_trmp_##FN I;               \
        extern __typeof__(T)(*__STD(FN)) I;                      \
        __typeof__(T) __RT(FN) I;                                \
        __typeof__(T) __WRAP(FN) I
```

- The real address of the original symbol is resolved at runtime using dlsym.
  - dlsym(RTLD_NEXT, name);
  - Note that the new bootstrap should run earlier than any other funtions

```
static __attribute__ ((constructor(250))) void bootstrap(void)
{
        void *p;

        printf("bootstrap\n");

        p = dlsym(RTLD_NEXT, "pthread_create");
        if (p == NULL)
                error(1, EINVAL, "%s", dlerror());

        __real_pthread_create = (__typeof__(__real_pthread_create))p;
}
```

- Simplifies the linking script and reduces the complexity of the wrapper implementation

POSIX wrapper functions in Xenomai3

sched_setscheduler

pthread_create

pthread_join

Thread & Sched

pthread_yield

pthread_attr_*

sleep/nanosleep

timer_*

Clock & Timer

timerfd_*

Clock_gettime

pthread_mutex_*

Thread synchronization

pthread_sem_*

pthread_cond_*

pthread_kill

Signal

kill

pthread_sig*

read

ioctl

write

I/O (rtdm)

mmap

poll

select

...

mq_open

mq_close

Message queue

mq_send

mq_receive

Agenda

POSIX wrapper

- Introduction
- Xenomai Implementation
- Our works
- Key Challenges

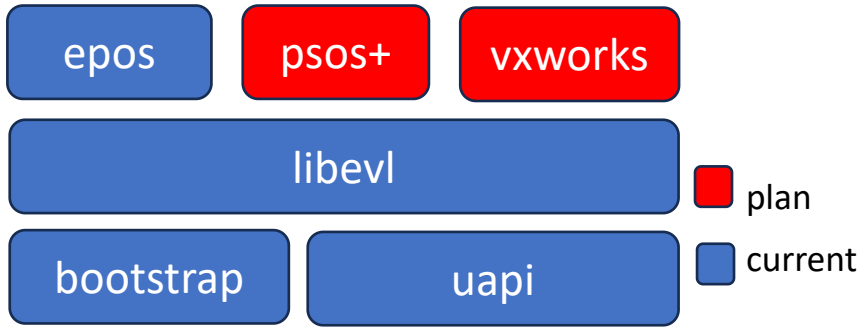
EtherCAT

- Overview
- IgH Ethercat and Xenomai
- Porting to EVL

Our works

- Building the POSIX wrapper linking macros and scripts.
  - Following a similar approach as Xenomai 2 and Xenomai 3.
  - The boilerplate layer has not been moved yet.

- Implementing some POSIX wrapper functions.
  - Including thread, synchronization and the clock related
  - Benefited from libevl due to its similar interfaces.



Xenomai 4

- Developed tests and migrated simple applications from Xenomai 3.
  - Examples include clocktest and latency.

- Issues with the current design or missing syscalls.
  - Plans to discuss these with the community
  - Share details later.

- Currently, the work is under libevl r27
  - Will be backported to r50 sooner or later

Our works

sched_setscheduler

pthread_create

pthread_join

Thread & Sched

pthread_yield

pthread_attr_*

sleep/nanosleep

timer_*

Clock & Timer

timerfd_*

Clock_gettime

pthread_mutex_*

Thread synchronization

pthread_sem_*

pthread_cond_*

pthread_kill

Signal

kill

pthread_sig*

read

ioctl

write

I/O (rtdm)

mmap

poll

select

...

mq_open

mq_close

Message queue

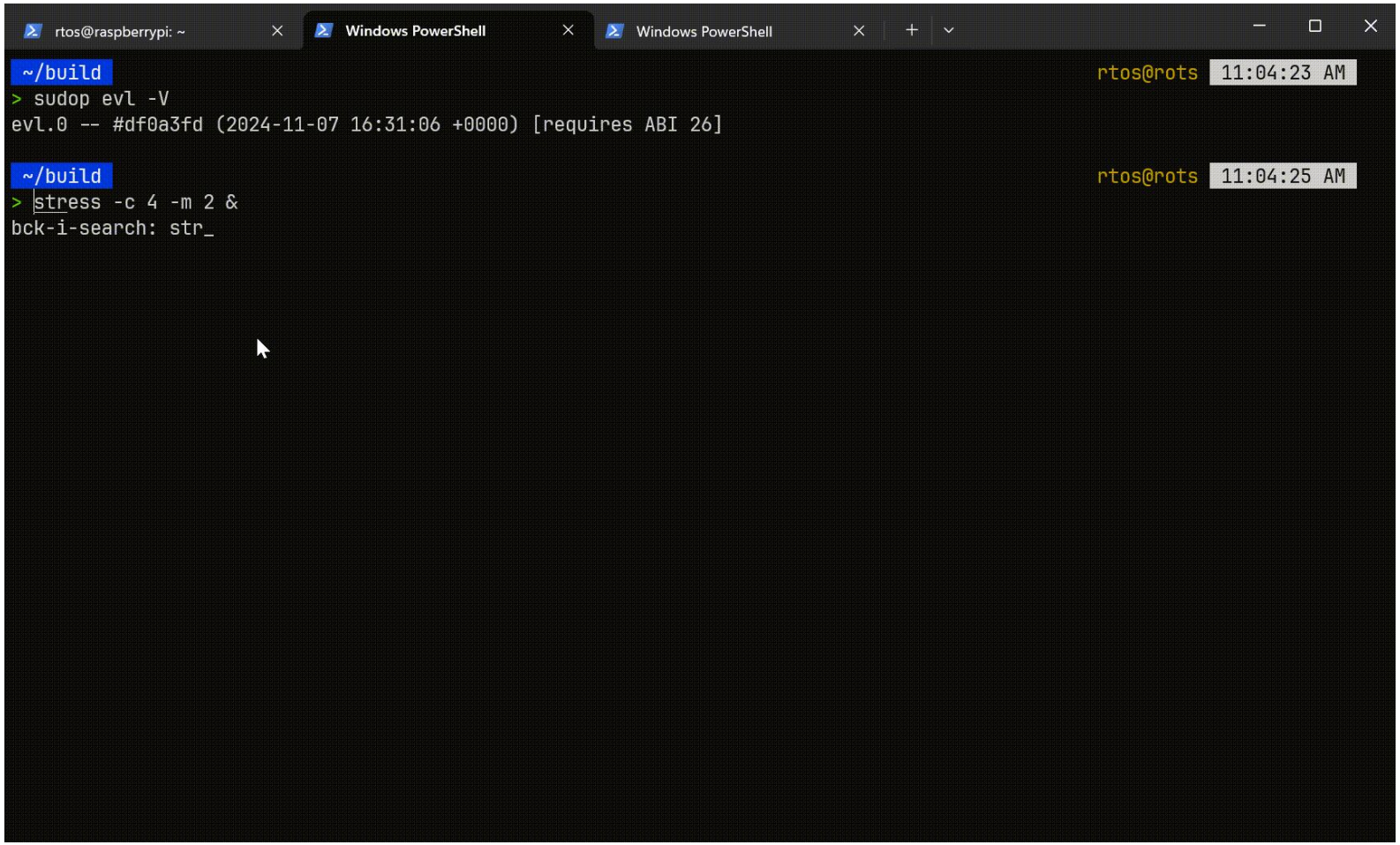
mq_send

mq_receive

Xenomai3: clocktest

Clock test:

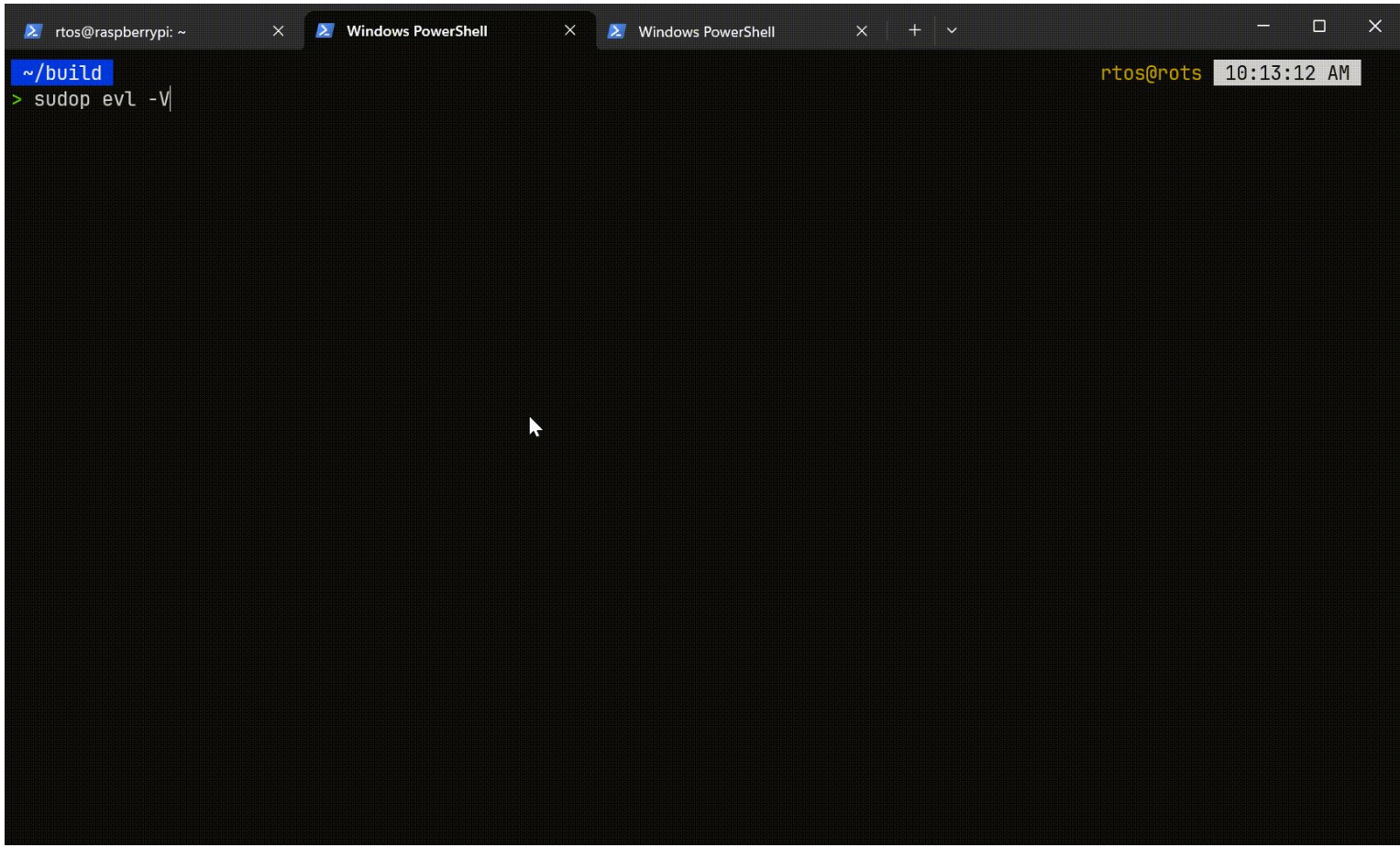
- Run in qemu-kvm on RPI 5b
- With stress cpu=4,memory=2



Xenomai3: latency

- Run in qemu-kvm on RPI 5b
- Remove some codes that related to Xenomai kernel config or structs.
- made some changes:
  - read -> oob_read
  - sem_open -> sem_init
  - timerfd_set with flag.

- With stress cpu=4,memory=2



Agenda

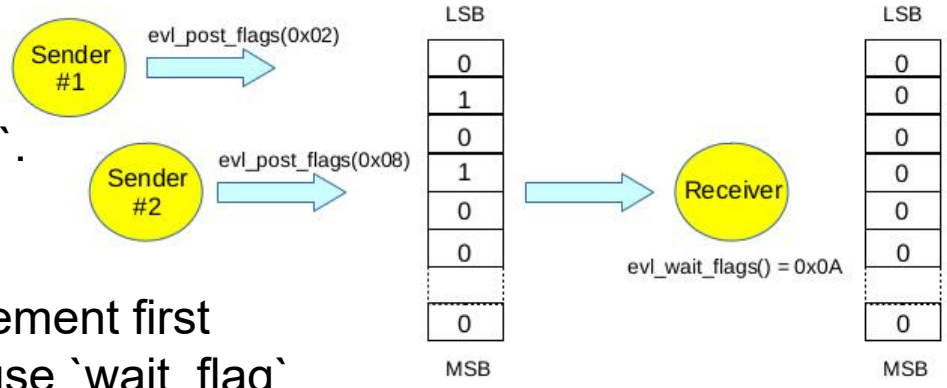
POSIX wrapper

- Introduction
- Xenomai Implementation
- Our works
- Key Challenges

EtherCAT

- Overview
- IgH Ethercat and Xenomai
- Porting to EVL

Challenges
signal



- Unlike Xenomai, EVL does not have a OOB signal mechanism.
  - Instead, EVL provides the `flags` which is based on `monitor`.

- Flags is similar to real-time signal.
  - To use the flag, the sender and receiver need to open the element first
  - Sender use `post_flag` to send some signal, while receiver use `wait_flag`
  - Some gaps between flag based signal with POSIX signal:

	signal	flag
Initialization	-	Open/Create element first
Target	Use pthread_t or pid	Use pointer to struct flag
Sigqueue	Take data or pointer with signal	Can only send data or signal

  - Maybe need some adoptions in kernel.

- As for non real-time signals, like SIGINT, SIGTRAP, SIGSTOP etc,
  - Dovetail convert them as `inband_event`. Handling directly in kernel.

Challenges
other

- RTDM
  - RTDM is the core I/O component in Xenomai 3.
  - EVL currently uses only three syscalls (oob_*).
  - A decision is needed on whether to transplant RTDM to EVL.

- Thread
  - Most thread services in EVL use pthread_t as the identifier.
  - A new syscall may be required to map evl fd to pthread_t.

- Timer
  - EVL timers resemble timerfd for synchronous operations.
  - POSIX timers rely on SIGALERT signals for asynchronous notifications.

- Overall
  - **libevl** follows a more synchronous API style.
  - **POSIX** incorporates more asynchronous API elements.

Agenda

POSIX wrapper

- Introduction
- Xenomai Implementation
- Our works
- Key Challenges

EtherCAT

- Overview
- IgH Ethercat and Xenomai
- Porting to EVL

EtherCAT: overview

- EtherCAT is an open, high-performance Ethernet-based fieldbus system.
  - EtherCAT has become widely adopted in industries.
  - Widely adopted in robotics, manufacturing, and motion control.
- IgH Ethercat Master is an open-source implementation
  - Performance similar to or better than TwinCAT[1]
  - Among the most widely used EtherCAT master implementations.
- IgH Ethercat has the RTAI and Xenomai support.

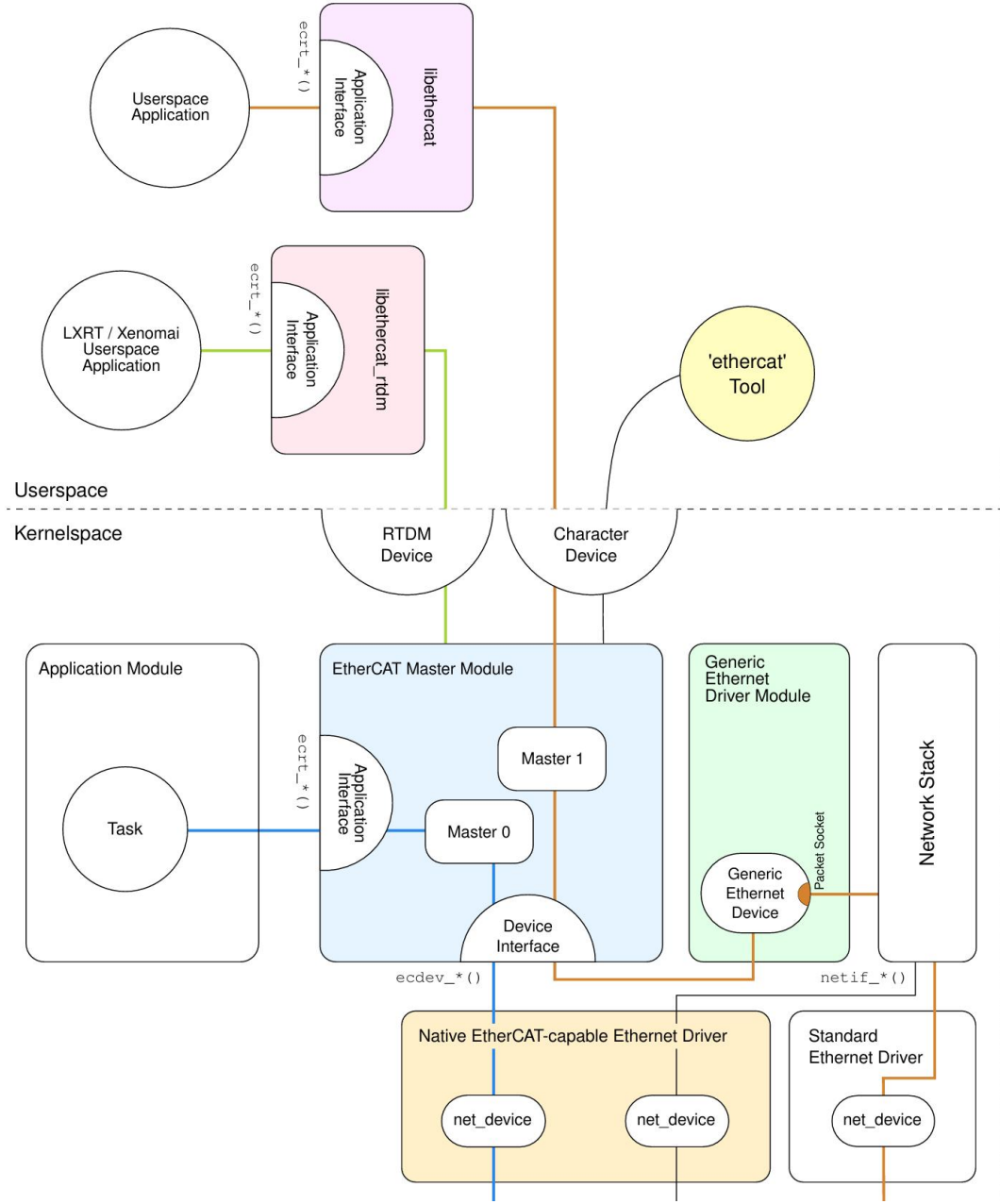
[1]: https://avestia.com/CDSR2017_Proceedings/files/paper/CDSR_128.pdf

How IgH EtherCAT works?

- The architecture of Igh Ethercat can be divided into 3 parts[1]
- **Application interface**
  - User space library and API.
  - Handling syscall and call the master module.

- **EtherCAT master module**
  - Main logic of EtherCAT protocol
  - Realtime date exchange, configuration management, state maintainence etc.

- **Device interface**
  - A unfied interface for net deivce operations.
  - Adopt common NIC to have real-time capacity
    - Remove netif_* call.
    - Disable interrupt
    - Fixed size socket buffer.



[1] https://docs.etherlab.org/ethercat/1.5/pdf/ethercat_doc.pdf

IgH EtherCAT Xenoami support

- To achieve realtime data exchange,
  - Few changes in the master module layer.
  - No changes in the device layer.
  - Thanks to good decoupling

- In user space, use the Cobalt POSIX wrapper to wrap standard library functions for real-time support.

- In kernel space, write a RTDM device interface.
  - Implement an RTDM device interface.
  - Redirect rtdm_ioctl commands to the master module.
  - Components like the EoE thread and others do not require real-time behavior.

```
#ifdef EC_IOCTL_RTDM
# include "rtdm_details.h"
/* RTDM does not support locking yet,
 * therefore no send/receive callbacks are set too. */
# define ec_ioctl_lock(lock) do {} while(0)
# define ec_ioctl_unlock(lock) do {} while(0)
# define ec_ioctl_lock_interruptible(lock) (0)
# define ec_copy_to_user(to, from, n, ctx) \
    rtdm_safe_copy_to_user(ec_ioctl_to_rtdm(ctx), to, from, n)
# define ec_copy_from_user(to, from, n, ctx) \
    rtdm_safe_copy_from_user(ec_ioctl_to_rtdm(ctx), to, from, n)
```

- Xenomai does not supports cross-domain mutexes;
  - locks are disabled as a result.
- Because of that, the Xenomai version does not have the callback interfaces.[2]

[1] https://gitlab.com/etherlab.org/ethercat/-/commit/9faaa83212d6dc79b6fbc9c022c8eff5b1c6badd
[2] https://gitlab.com/etherlab.org/ethercat/-/issues/64

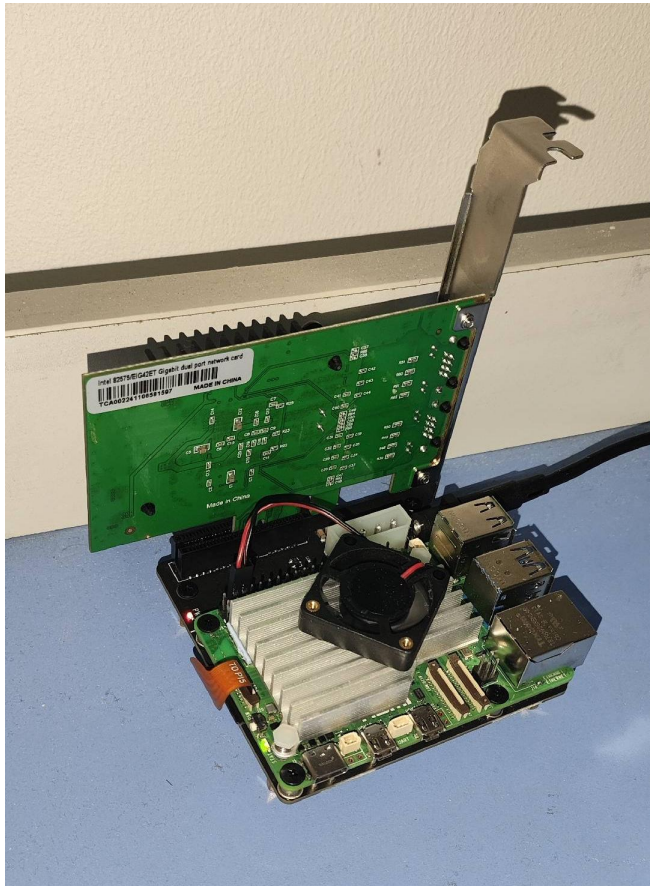
Porting to EVL

```
struct file_operations {
        ...
        ssize_t (*oob_read) (struct file *, char __user *, size_t);
        ssize_t (*oob_write) (struct file *, const char __user *, size_t);
        long (*oob_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_oob_ioctl) (struct file *, unsigned int, unsigned long);
        __poll_t (*oob_poll) (struct file *, struct oob_poll_wait *);
        ...
} __randomize_layout;
```

- Instead of using RTDM, we plan to use EVL native file operations.
  - Implement the RTDM facilities in EVL can cost a lot of time.
  - An EVL driver is a regular Linux driver
- Take use of the ordinary linux version, except for the verbs we use.
  - EVL POSIX wrapper can help us do the stuffs in user space.
  - In kernel space, implement the oob_* interfaces.
- The problem of the share mutex issue may resolve in evl.
  - Evl has stax(Stage exclusion lock)
    - serializes in-band vs out-of-band thread activities for accessing an arbitrary resource.
  - But might affect the real-time performance.
  - Put as next stage work.

Current status

- We are still at very early stage.
  - The software development environment and hardware setup are newly prepared.
  - The first-step device driver is still under modification.
- Testing Plan:
  - Initial tests will be conducted in QEMU.
  - Later, testing will move to Raspberry Pi OS with a custom kernel.
- Hardware:
  - Raspberry Pi 5 equipped with:A PCIe expansion board.
  - Igb Nic
- Linux-evl:
  - v6.12-evl-rebase
- Ethercat:
  - 1.6



Thanks!

libevl/posix_dev in RROS

我的微信

群聊：RROS社区

该二维码7天内(12月2日前)有效，重新进入将更新