Xenomai Real-Time nanoKernel

Ricardo Correia Pinto
ricardo.pinto@ciencias.ulisboa.pt

Faculty of Sciences - University of Lisboa

Sistemas Embebidos e de Tempo-Real
2014/2015

Outline

1. Real-Time Linux
2. ADEOS
3. Xenomai
    - Architecture
    - Threads
4. Xenomai Programming
    - Native Skin
    - POSIX Skin
5. Bibliography

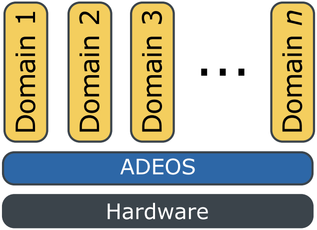
Real-time Co-kernel in Linux

- RT-Linux patches/extensions providing a co-kernel solution
  - RT Scheduler for RT tasks, Linux Scheduler for NRT
  - RT coexists with native Linux, but has higher priority
  - Interception of events going to Linux, to guarantee that RT has higher priority than NRT
- Event Management
  - Events are interrupts, traps, etc.
  - Events create disturbances in the execution of programmes
  - RT Linux approaches intercept the events generated, processing them prior to Linux
  - Adaptive Domain Environment for Operating Systems (ADEOS) is a generic framework for event management

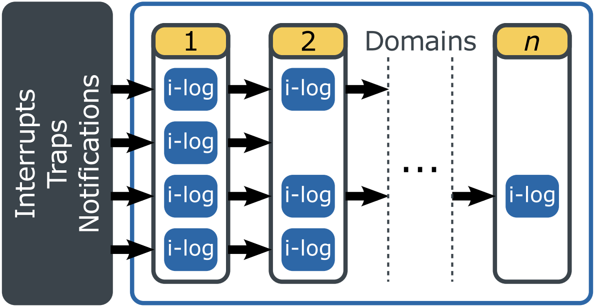
ADEOS

→ Architecture I

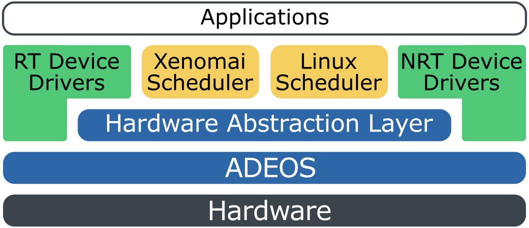
- Offers a resource virtualization layer - interrupts & traps
- Provides support priority-based *domains*
  - A domain can be a simple application, or a kernel
  - A domain can be aware of the existence of other domains, i.e. have access to them
- ADEOS manages the flow of events, cascading them through domains



- ADEOS is also known as I-Pipe, since it provides an interrupt pipeline
- Events flow from the highest to the lowest priority domains
  - A domain can process events and passe them through to other domains
  - Events can be stalled (delayed) by domains



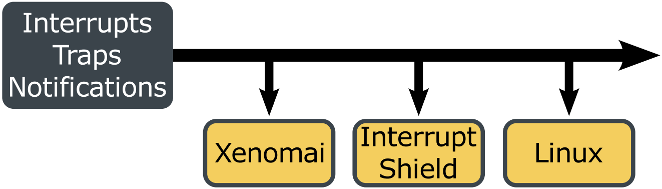
Xenomai-enabled Linux System Architecture



- Xenomai follows the co-kernel philosophy
  - Linux kernel is patched with Xenomai and ADEOS
  - Both RT and NRT applications can exist on a Xenomai-enabled system
- ADEOS provides event management
- Xenomai is aware of Linux

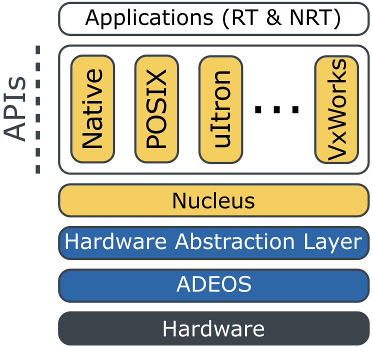
Xenomai and Linux Integration with ADEOS

- ADEOS ensures interrupts are first treated by Xenomai
  - Xenomai has the higher priority
  - Linux has the lower priority
  - Between them there is an Interrupt Shield domain
- Interrupts to be propagated to Linux can be stalled by Interrupt Shield domain
- Once there are no more runnable tasks in the Xenomai domain the interrupts are propagated to Linux and its scheduler runs



Xenomai Architecture

→ Overview



- **Skins**, offering the APIs of several *flavours* of RTOS
- **Nucleus**, an abstract RTOS interface specialized by *skins*
- **Hardware Abstraction Layer**, conferring portability
- **ADEOS**, with the interrupt pipeline

Xenomai Architecture

→ Modules

- Modular components, loaded by the Linux kernel
  - **xeno_hal**, the hardware abstraction layer
  - **xeno_nucleus**, the abstract RTOS interface including scheduler
  - **xeno_<skin>**, the API skins
    - native
    - POSIX
    - uITRON
    - ...
  - **xeno_rtipc**, mechanisms for IPC between RT-RT and RT-NRT
- xeno_hal and xeno_nucleus mandatory, skins optional
  - Keep the memory footprint low, good especially in low-resource embedded platforms as ARM or PowerPC
- These modules can also be part of the kernel itself
  - Supplied VM and lab PCs

Xenomai Architecture

→ Nucleus and skins

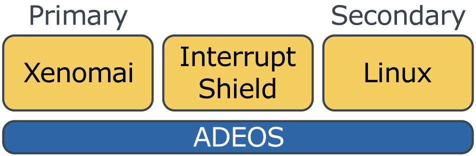
- Nucleus provides the abstract RTOS entity
  - Contais basic RT services, entities and primitives
    - Scheduler and Thread/Task state-keeping
    - Interrupt management and Timer services
    - Dynamic memory allocation services
    - Real-time shadow services - more on this later
  - Contains the hooks for skins to invoke services

- Skins specialize the abstract RTOS
  - API for building RT applications
  - Provide portability, e.g. POSIX
  - Skins are wrappers for Native API functions

Xenomai Threads

→ Concepts

- Xenomai provides the concept of *primary* and *secondary* mode of thread scheduling and execution
  - **Primary** mode corresponds to the Xenomai scheduler
  - **Secondary** mode corresponds to the Linux scheduler
- A thread running only in primary mode is equivalent to a **real-time task** in other RTOS, i.e. it has timeliness guarantees
- Xenomai threads are created in user-space, using regular POSIX threads
  - Attached to the Xenomai scheduler as a *shadow thread*
  - Possibility to migrate between primary and secondary modes



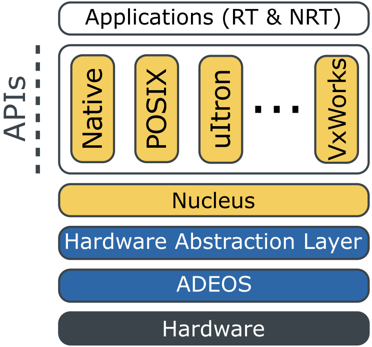
Xenomai Threads
→ Mode Switch

- A Xenomai Thread can migrate from primary to secondary mode
  - If it makes a non-RT system call, e.g. `printf`
- Thread retains its priority during migration
  - Xenomai and Linux have the same number of priority levels and semantic
  - Priority levels range from 0 to 99, with 99 being higher than 0
- Migration mechanism mitigates priority inversion and ill-effects of RT preempting NRT during syscalls

- Mode switch
  1. Xenomai intercepts the system call via ADEOS
  2. Preemption of the thread
  3. Scheduled in Linux, with the same priority
  4. System call executed in Linux
  5. Thread re-scheduled in Xenomai

Xenomai Programming

- The APIs provided by the skins wrap nucleus functions
  - `xeno-config` script outputs the necessary include directives and link flags to the compiler, based on the skin
  - Choice of skin only depends on the user/application



Xenomai Programming - Native Skin

- Xenomai offers a native set of primitives for real-time application programming
  - Task management
    - `rt_task_create`
    - `rt_task_start`
    - `rt_task_set_periodic`
    - `rt_task_wait_period`
  - Counting Semaphores
    - `rt_sem_p`
    - `rt_sem_v`
  - Mutexes
    - `rt_mutex_acquire`
    - `rt_task_release`
- ... and much more, like I/O and IPC/Message Passing

Native skin

→ Task Creation

- Tasks are functions
- Initialization creates and starts tasks
- Create
  - `rt_task_create`
  - Initialize the data structures
- Start
  - `rt_task_start`
  - Bind handler to task function
  - Release the task, to be scheduled

Code

```
#include <native/task.h>

RT_TASK tsk_handle; /* Task */

void* task_example(void* cookie){
  /* Task body */
}

int main(int argc, char* argv[]){

  /* Avoid page allocation/swapping */
  mlockall(MCL_CURRENT|MCL_FUTURE);

  rt_task_create(&task_example, /* &task */
                 str, /* Task name */
                 0, /* Stack size, 0=default*/
                 50, /* Priority, max 99 */
                 0); /* mode (FPU, start suspended, ...) */

  rt_task_start(&tsk_handle, /* &task*/
                &task_example, /* Function with task body */
                NULL); /* "Cookie", function arguments */

}
```

Native skin
→ Periodic task

- Periodic Task Execution
  1. Initialize variables
  2. Create periodic timer
  3. Enter loop and execute code
  4. Sleep until the next period

- Period
  - Defined in nano-seconds
  - Set by `rt_task_set_periodic`

- `rt_task_set_periodic` Args
  1. Task - `NULL` if self
  2. Start time - `TM_NOW` for current
  3. Period

Code

```
#include <native/task.h>
#include <native/timer.h>

void task_example(void *arg){
  /* Variables, Initialization, etc */
  int period = 1000000; /* 1 ms */

  /* ... */

  rt_task_set_periodic(NULL, TM_NOW, period);

  while (1) {

    /* Task code
      .
      .
    */

    rt_task_wait_period(NULL);
  }
}
```

Xenomai Programming - POSIX Skin

- Xenomai POSIX skin aims at supporting POSIX 1003.1b
- Semaphores
  - Counting semaphores
  - Primitives
    - `sem_wait`
    - `sem_post`
- Mutexes
  - Primitives
    - `pthread_mutex_lock`
    - `pthread_mutex_unlock`
  - Priority inheritance
    - `pthread_mutexattr_setprotocol`
    - Protocol can be either *NONE* or *Priority Inheritance*

Xenomai Programming - POSIX Skin

→ Thread Management

- Thread Primitives
  - `pthread_create`
  - `pthread_exit`
  - `pthread_join`
- Thread Attributes
  - `pthread_attr_init`
  - `pthread_attr_setschedparam`
  - `pthread_attr_setschedpolicy`
- Scheduler Types
  - **SCHED_FIFO** - Priority-based
  - **SCHED_RR** - Round-Robin
  - **SCHED_OTHERS** - Linux
    - The priority in Linux will be the one defined in the attributes

POSIX skin

Periodic Task

```
#include <time.h>

void∗ task_example(void∗ cookie){

  unsigned int period = PER_TASK_EXAMPLE;
  struct timespec next_period;

  /∗ ... ∗/
  /∗ Initialization code ∗/

  clock_gettime(CLOCK_MONOTONIC, &next_period);

  while(1){
    /∗ Calculate the time for the execution of this task∗/
    next_period.tv_nsec += period ∗ 1000;
    while (next_period.tv_nsec >= NSEC_PER_SEC) {
      next_period.tv_nsec −= NSEC_PER_SEC;
      next_period.tv_sec++;
    }

    /∗ Loop user code here ∗/

    /∗ Sleep until the next execution∗/
    clock_nanosleep(CLOCK_MONOTONIC,
                    TIMER_ABSTIME,
                    &next_period, NULL);
  }

}
```

- Linux Process - No RT
  - A channel is seen as a device: **/dev/rtp0..63**
  - Channel is FIFO
  - Linux processes access the channel through `read()` and `write()` operations

- Xenomai Threads - RT
  - A channel is seen as a socket
  - The Cross-Domain Datagram Protocol (XDDP) is used
  - I/O is performed through `sendto()` and `recvfrom()`

Xenomai POSIX Skin
→ RT to NRT IPC Code Example

RT - Producer

```
#include <rtdk.h>
#include <rtdm/rtipc.h> /* XDDP */

void task_rt(void * cookie){

  int ret, s, N = 0; /* N => /dev/rtpN on NRT side*/
  double data;
  size_t streamsz;
  struct sockaddr_ipc saddr;

  /** Initialization *************************/
  streamsz = 16*sizeof(double);
  ret = setsockopt(s, SOL_XDDP, XDDP_BUFSZ,
                   &streamsz, sizeof(streamsz));

  memset(&saddr, 0, sizeof(saddr));
  saddr.sipc_family = AF_RTIPC;
  saddr.sipc_port = N;
  ret = bind(s, (struct sockaddr *)&saddr, sizeof(saddr));

  /** Execution **********************/
  ret = sendto(s, &data, sizeof(double), 0, NULL, 0);

}
```

NRT - Consumer

```
#include <rtdk.h>
#include <rtdm/rtipc.h> /* XDDP */

void task_nrt(void* cookie){

  int fd;
  double data;

  /* ... */

  fd = open("/dev/rtp0", O_RDWR);

  /* ... */

  read(fd, &data, sizeof(data));

  /* ... */
}
```

Xenomai POSIX Skin

→ NRT thread

- A NRT thread known by Xenomai is a thread which executes always in secondary mode
- The selection of secondary mode is done by the scheduler type
  - `SCHED_OTHER` instead of `SCHED_FIFO`

RT - Producer

```
pthread_t task_nrt;


void nrt(void * cookie){
  /* NRT code */
}

int
main(int argc, char * argv[]){

  pthread_attr_t attr;
  int rc;

  /* NRT thread parameters */
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
  pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
  pthread_attr_setschedpolicy(&attr, SCHED_OTHER);

  rc = pthread_create(&task_nrt, &attr, &nrt, NULL);

  /* ... */

}
```

References

- *Life with ADEOS* - `http://www.xenomai.org/documentation/xenomai-2.6/pdf/Life-with-Adeos-rev-B.pdf`
- *Xenomai 2.6.2.1 Programming Manual* - `http://www.xenomai.org/documentation/xenomai-2.6/pdf/`