Real-time Linux Basics

How does Real-time Linux work?

The general idea of Real-time (RT) Linux is that a small real-time kernel runs beneath Linux, meaning that the real-time kernel has a higher priority than the Linux kernel. Real-time tasks are executed by the real-time kernel, and normal Linux programs are allowed to run when no real-time tasks have to be executed. Linux can be considered as the idle task of the real-time scheduler. When this idle task runs, it executes its own scheduler and schedules the normal Linux processes. Since the real-time kernel has a higher priority, a normal Linux process is preempted when a real-time task becomes ready to run and the real-time task is executed immediately.

How is the real-time kernel given higher priority than Linux kernel?

Basically, an operating system is driven by interrupts, which can be considered as the heartbeats of a computer:

- All programs running in an OS are scheduled by a scheduler which is driven by timer interrupts of a clock to reschedule at certain times.
- An executing program can block or voluntary give up the CPU in which case the scheduler is informed by means of a software interrupt (system call).
- Hardware can generate interrupts to interrupt the normal scheduled work of the OS for fast handling of hardware.

RT Linux uses the flow of interrupts to give the real-time kernel a higher priority than the Linux kernel:

- When an interrupt arrives, it is first given to the real-time kernel, and not to the Linux kernel. But interrupts are stored to give them later to Linux when the real-time kernel has finished.
- As first in row, the real-time kernel can run its real-time tasks driven by these interrupts.
- Only when the real-time kernel is not running anything, the interrupts which were stored are passed on to the Linux kernel.
- As second in row, Linux can schedule its own processes driven by these interrupt.

Hence, when a normal Linux program runs and a new interrupt arrives:

- it is first handled by an interrupt handler set by the real-time kernel;
- the code in the interrupt handler awakes a real-time task;
- immediately after the interrupt handler, the real-time scheduler is called ;
- the real-time scheduler observes that another real-time task is ready to run, so it puts the Linux kernel to sleep, and awakes the real-time task.

Hence, a special way of passing of the interrupts between real-time kernel and the Linux kernel is needed. Each flavor of RT Linux does this is in its own way. Xenomai uses an interrupt pipeline from the Adeos project. For more information, see also Life with Adeos.

Linux kernel system calls versus Real-time kernel system calls

If a program wants something to be done, but does not have the right privileged mode of operation, the general approach is to generate a special software interrupt (called a system call).

The program is interrupted, and the interrupt is given to the kernel which will do the operation on behave of the program. The message given along in the interrupt determines which operation, or which system call has to be executed. The collection of system calls can be seen as the user program interface to the kernel.

In the case of real-time Linux we have two kernels running, the normal Linux kernel and the real-time kernel. For each specific kernel a unique software interrupt is assigned. So depending on the unique interrupt number we know if it is an system call for the Linux kernel or a system call for the real-time kernel. Thus we have:

- a set of Linux system calls which allows you to call a part of the kernel API
- a set of real-time system calls which allows you to call a part of the real-time API

Note: because the real-time kernel gets the interrupts first, the real-time system calls will always be run first.

RT Linux Tasks are not Linux Programs

Observe the difference between real-time tasks and standard Linux programs:

- In real-time Linux we can make a real-time program by programming real-time threads. None-real-time programs in real-time Linux just use the conventional Linux threads.
- A real-time task can be executed in kernel space using a kernel module, but it can also be executed in user space using a normal C program.
- Running in user space instead of in kernel space gives a little extra overhead, but with the following advantages :
    - A bug in a kernel module can crash the whole system, however a bug in a user space program can only crash the program.
    - In a kernel model one can only use the real-time API and the limited kernel API, but in a real-time user space program the real-time API and the whole Linux API can be used. However when using the Linux API in user space, the program cannot be scheduled by the real-time scheduler (HARD real-time) but must be scheduled by the Linux scheduler (SOFT real-time). So calling a Linux API call from a real-time user space task will degrade its performance from HARD to SOFT real-time. After the call the task will return to the real-time scheduler.
    - In user space you can use a debugger to debug realtime programs. (E.g. gdb for C programmers).

Overhead user-space Xenomai tasks compared to kernel space

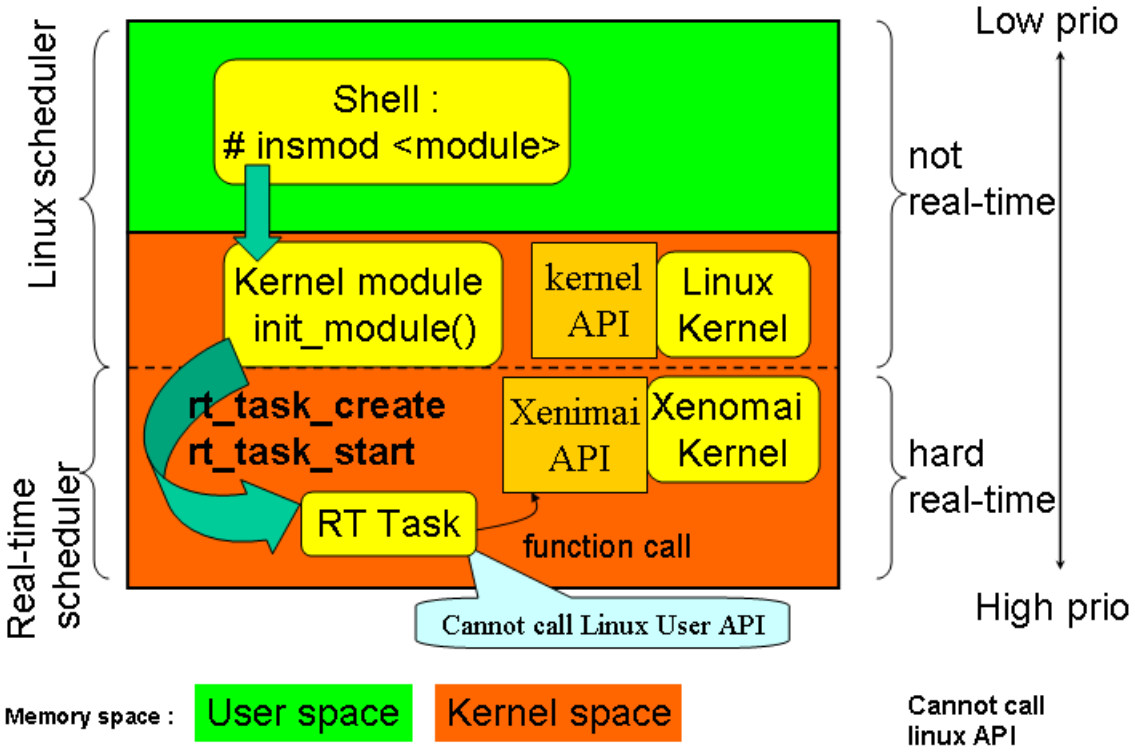
Comparing Xenomai tasks in user space with those in kernel space, observer that there is no overhead by swapping out virtual memory:

- Virtual memory is used by both kernel and user space, so both have to lookup in the page tables where the actual memory
- Both Xenomai kernel-space task and Xenomai user-space task are memory locked:
    1. The kernel part of memory is locked memory, meaning that it never can be swapped to disk. Thus automatically kernel memory used by kernel modules can never be swapped to disk.
    2. Also memory of Xenomai tasks in kernel space cannot be swapped to disk. Xenomai task in user space are memory locked by the programmer on startup.

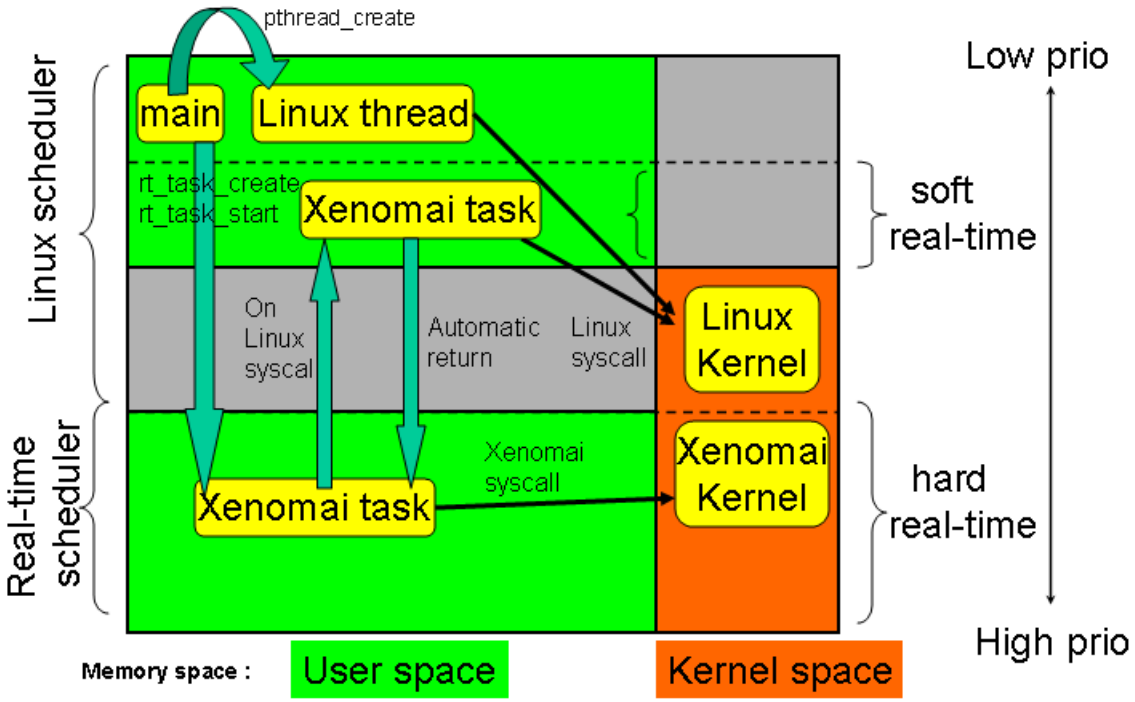
There is some overhead for Xenomai API functions:

- A user-space Xenomai task needs to use a Xenomai system call to execute a Xenomai API function which is located in kernel space.
- A kernel-space Xenomai task can just do a function call to a Xenomai API function.

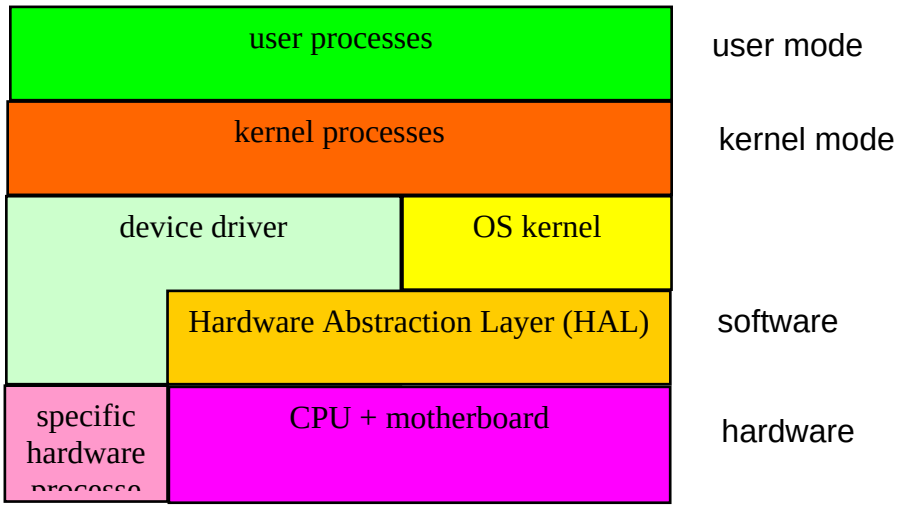
Xenomai kernel space tasks



Xenomai user space tasks



Linux architecture



Realtime Linux - Xenomai architecture