Troubleshooting a dual kernel configuration

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

This page is a troubleshooting guide enumerating known issues with dual kernel Xenomai configurations.

Tip
If running any release from the Xenomai 2 series, or a Xenomai 3 release using the **Cobalt** real-time core, then you are using a dual kernel configuration, and this document was meant for you. Xenomai 3 over the **Mercury** core stands for a single kernel configuration instead, for which you can find specific troubleshooting information here.

1 Kernel-related issues

1.1 Common kernel configuration issues

When configuring the Linux kernel, some options should be avoided.

CONFIG_CPU_FREQ
: This allows the CPU frequency to be modulated with workload, but many CPUs change the TSC counting frequency also, which makes it useless for accurate timing when the CPU clock can change. Also some CPUs can take several milliseconds to ramp up to full speed.

CONFIG_CPU_IDLE
: Allows the CPU to enter deep sleep states, increasing the time it takes to get out of these sleep states, hence the latency of an idle system. Also, on some CPU, entering these deep sleep states causes the timers used by Xenomai to stop functioning.

CONFIG_KGDB
: This option should not be enabled, except with x86.

1.2 Kernel hangs after "Uncompressing Linux... done, booting the kernel."

This means that the kernel crashes before the console is enabled. You should enable the `CONFIG_EARLY_PRINTK` option. For some architectures (blackfin, x86, arm), enabling this option also requires passing the `earlyprintk` parameter on the kernel command line. See *Documentation/kernel-parameters.txt* for possible values.

For the ARM architecture, you have to enable `CONFIG_DEBUG_KERNEL` and `CONFIG_DEBUG_LL` in order to be able to enable `CONFIG_EARLY_PRINTK`.

1.3 Kernel OOPSes

Please make sure to check the "Kernel configuration" section first.

If nothing seems wrong there, try capturing the OOPS information using a *serial console* or *netconsole*, then post it to the xenomai mailing list, along with the kernel configuration file (aka `.config`) matching the kernel build.

1.4 Kernel boots but does not print any message

Your distribution may be configured to pass the `quiet` option on the kernel command line. In this case, the kernel does not print all the log messages, however, they are still available using the `dmesg` command.

1.5 Kernel log displays Xenomai or I-pipe error messages

1.5.1 I-pipe: could not find timer for cpu #N

The most probable reason is that no hardware timer chip is available for Xenomai timing operations.

Check that you did not enable some of the conflicting options listed in the Kernel configuration section.

With AMD x86_64 CPUs

You will most likely also see the following message:

```
I-pipe: cannot use LAPIC as a tick device
I-pipe: disable C1E power state in your BIOS
```

The interrupt pipeline outputs this message if C1E option is enabled in the BIOS. To fix this issue, disable C1E support in the BIOS. In some Award BIOS this option is located in the `Advanced BIOS Features`→ menu (`AMD C1E Support`).

Warning

Disabling the `AMD K8 Cool&Quiet` feature in the BIOS will **NOT** solve this problem.

With other CPU architectures

The interrupt pipeline implementation may lack a registration for a hardware timer available to Xenomai timing operations (e.g. a call to `ipipe_timer_register()`).

If you are working on porting the interrupt pipeline to some ARM SoC, you may want to have a look at this detailed information.

1.5.2 SMI-enabled chipset found, but SMI workaround disabled

You may have an issue with System Management Interrupts on your x86 platform. You may want to look at this document.

1.6 Xenomai and Linux devices share the same IRQ vector

This x86-specific issue might still happen on legacy hardware with no MSI support. See this article from the Knowledge Base.

1.7 Kernel issues specific to the Xenomai 2.x series

1.7.1 system init failed, code -19

See this entry.

1.7.2 system init failed, code -22

On the ppc64 platform, check whether `CONFIG_PPC_64K_PAGES` is defined in your kernel configuration. If so, then you likely need to raise all Xenomai parameters defining the size of internal heaps, such as `CONFIG_XENO_OPT_SYS_HEAPSZ`, `CONFIG_XENO_OPT_GLOBAL_SEM_HEAPSZ` and `CONFIG_XENO_OPT_SEM_HEAPSZ`, so that (size / 64k) > 2. The default values for these parameters are currently based on the assumption that PAGE_SIZE = 4k.

1.7.3 Local APIC absent or disabled!

The Xenomai 2.x *nucleus* issues this warning if the kernel configuration enables the local APIC support (`CONFIG_X86_LOCAL_APIC`), but the processor status gathered at boot time by the kernel says that no local APIC support is available. There are two options for fixing this issue:

- either your CPU really has *no* local APIC hardware, in which case you need to rebuild a kernel with LAPIC support disabled.
- or it does have a local APIC but the kernel boot parameters did not specify to activate it using the *lapic* option. The latter is required since 2.6.9-rc4 for boxes which APIC hardware is disabled by default by the BIOS. You may want to look at the file *Documentation/kernel-parameters.txt* from the Linux source tree, for more information about this parameter.

2 Application-level issues

2.1 --enable-x86-sep needs NPTL and Linux 2.6.x or higher

or, === --enable-x86-vsyscall requires NPTL . . .

This message may happen when starting a Xenomai 2.x or 3.x application respectively. On the x86 architecture, the configure script option mentioned allows Xenomai to use the *vsyscall* mechanism for issuing system calls, based on the most efficient method determined by the kernel for the current system. This mechanism is only available from NPTL-enabled glibc releases.

Turn off this feature for other libc flavours.

2.2 binding failed: Operation not permitted

This is the result of an attempt to run a Xenomai application as an unprivileged user, which fails because invoking Xenomai services requires `CAP_SYS_NICE`. However, you may allow a specific group of users to access Xenomai services, by following the instructions on this page.

2.3 incompatible ABI revision level

Each major Xenomai release (e.g. 2.1.x, 2.2.x . . . 2.6.x, 3.0.x . . .) defines a kernel/user ABI, which remains stable across minor update releases (e.g. 2.6.0 → 2.6.1). This guarantee makes partial updates possible with production systems (i.e. kernel and/or user support).

For instance, any application built over the Xenomai 2.6.0 libraries can run over a Xenomai 2.6.3 kernel support, and conversely.

However, it is not possible to mix kernel and user-space supports from different major releases.

Tip

A common source of error is running a kernel with support from the Xenomai 2.6.x series, on a system with pre-installed Xenomai libraries from the 2.5.x series, shipped with a Debian-based Linux distribution (notably Ubuntu), which won't work as the two series have different ABIs. If however you did install the correct Xenomai user-space support on your target system, chances are that stale files from a previous Xenomai installation still exist on your system, causing the mismatch.

2.4 <program>: not found

Although the program in question may be present, this message may happen on ARM platforms when a mismatch exists between the kernel and user library configurations with respect to EABI support. Typically, if user libraries are compiled with a toolchain generating OABI code, the result won't run over a kernel not enabling the `CONFIG_OABI_COMPAT` option. Conversely, the product of a compilation with an EABI toolchain won't run on a kernel not enabling the `CONFIG_AEABI` option.

2.5 incompatible feature set

When a Xenomai application starts, the set of core features it requires is compared to the feature set the kernel provides. This message denotes a mismatch between both sets, which can be solved by fixing the kernel and/or user build configuration. Further details are available from this page for Xenomai 3, and this page for Xenomai 2.

2.5.1 feature mismatch: missing="smp/nosmp"

On some SMP-capable architectures, for kernel-space and user-space supports to be compatible, both should be compiled with the same setting for SMP.

SMP support in kernel-space is enabled with the `CONFIG_SMP` option.

For these architectures, SMP support in user-space is enabled by passing `--enable-smp` to the configure script, and disabled by passing `--disable-smp` (SMP is enabled by default on some platforms).

Other SMP-capable architectures may run userland code built with `--enable-smp` or `--disable-smp` over the same kernel indifferently, at no noticeable performance cost. These architectures never receive such SMP-related error.

2.6 Application-level issues specific to the Xenomai 2.x series

The following feature mismatches can be detected with the 2.x series:

2.6.1 feature mismatch: missing="kuser_tsc"

See the "ARM tsc emulation issues" section.

Note
This issue does not affect Xenomai 3.x as the latter requires modern I-pipe series which must provide *KUSER_TSC* support on the ARM architecture.

2.6.2 feature mismatch: missing="sep"

This error is specific to the x86 architecture on Xenomai 2.x, for pre-Pentium CPUs which do not provide the *sysenter/sysexit* instruction pair. See "this section".

Note
This issue does not affect Xenomai 3.x as the latter does not support pre-Pentium systems in the first place.

2.6.3 feature mismatch: missing="tsc"

This error is specific to the x86 architecture on Xenomai 2.x, for pre-Pentium CPUs which do not provide the *rdtsc* instruction. In this particular case, `--enable-x86-tsc` cannot be mentioned in the configuration options for building the user libraries, since the processor does not support this feature.

The rule of thumb is to pick the **exact** processor for your x86 platform when configuring the kernel, at the very least the most specific model which is close to the target CPU, not a generic placeholder such as *i586*, for which *rdtsc* is not available.

If your processor does not provide the *rdtsc* instruction, you have to pass `--disable-x86-tsc` option to the configure script for building the user librairies. In this case, Xenomai will provide a (much slower) emulation of the hardware TSC.

Note
This issue does not affect Xenomai 3.x as the latter does not support pre-Pentium systems in the first place.

2.6.4 ARM tsc emulation issues

In order to allow applications to measure short durations with as little overhead as possible, Xenomai uses a 64 bits high resolution counter. On x86, the counter used for this purpose is the time-stamp counter readable by the dedicated *rdtsc* instruction.

ARM processors generally do not have a 64 bits high resolution counter available in user-space, so this counter is emulated by reading whatever high resolution counter is available on the processor, and used as clock source in kernel-space, and extend it to 64 bits by using data shared with the kernel. If Xenomai libraries are compiled without emulated tsc support, system calls are used, which have a much higher overhead than the emulated tsc code.

In recent versions of the I-pipe patch, SOCs generally select the CONFIG_IPIPE_ARM_KUSER_TSC option, which means that the code for reading this counter is provided by the kernel at a predetermined address (in the vector page, a page which is mapped at the same address in every process) and is the code used if you do not pass the --enable-arm-tsc or --disable-arm-tsc option to configure, or pass --enable-arm-tsc=kuser.

This default should be fine with recent patches and most ARM SOCs.

However, if you see the following message:

```
incompatible feature set
(userland requires "kuser_tsc...", kernel provides..., missing="kuser_tsc")
```

It means that you are either using an old patch, or that the SOC you are using does not select the CONFIG_IPIPE_ARM_KUSER_TSC option.

So you should resort to what Xenomai did before branch 2.6: select the tsc emulation code when compiling Xenomai user-space support by using the --enable-arm-tsc option. The parameter passed to this option is the name of the SOC or SOC family for which you are compiling Xenomai. Typing:

```
/patch/to/xenomai/configure --help
```

will return the list of valid values for this option.

If after having enabled this option and recompiled, you see the following message when starting the latency test:

```
kernel/user tsc emulation mismatch
```

or

```
Hardware tsc is not a fast wrapping one
```

It means that you selected the wrong SOC or SOC family, reconfigure Xenomai user-space support by passing the right parameter to --enable-arm-tsc and recompile.

The following message:

```
Your board/configuration does not allow tsc emulation
```

means that the kernel-space support for the SOC you are using does not provide support for tsc emulation in user-space. In that case, you should recompile Xenomai user-space support passing the --disable-arm-tsc option.

2.6.5 hardware tsc is not a fast wrapping one

or, ==== kernel/user tsc emulation mismatch or, ==== board/configuration does not allow tsc emulation

See the "ARM tsc emulation issues" section.

2.6.6 native skin or CONFIG_XENO_OPT_PERVASIVE disabled

Possible reasons for this error are:

- you booted a kernel without Xenomai or I-pipe support, a kernel with I-pipe and Xenomai support should have a */proc/ipipe/version* and */proc/xenomai/version* files;
- the kernel you booted does not have the CONFIG_XENO_SKIN_NATIVE and CONFIG_XENO_OPT_PERVASIVE options enabled;
- Xenomai failed to start, check the "Xenomai or I-pipe error in the kernel log" section;
- you are trying to run Xenomai user-space support compiled for x86_32 on an x86_64 kernel.

2.6.7 "warning: <service> is deprecated" while compiling kernel code

Where <service> is a thread creation service, one of:

- `cre_tsk`
- `pthread_create`
- `rt_task_create`
- `sc_tecreate` or `sc_tcreate`
- `taskSpawn` or `taskInit`
- `t_create`

Starting with Xenomai 3, APIs are not usable from kernel modules anymore, at the notable exception of the RTDM device driver API, which by essence must be used from kernel space for writing real-time device drivers. Those warnings are there to remind you that application code should run in user-space context instead, so that it can be ported to Xenomai 3.

You may switch those warnings off by enabling the `CONFIG_XENO_OPT_NOWARN_DEPRECATED` option in your kernel configuration, but nevertheless, you have been **WARNED**.

2.6.8 a Xenomai system call fails with code -38 (ENOSYS)

Possible reasons for this error are:

- you booted a kernel without Xenomai or I-pipe support, a kernel with I-pipe and Xenomai support should have a */proc/ipipe/version* and */proc/xenomai/version* files;
- the kernel you booted does not have the `CONFIG_XENO_SKIN_*` option enabled for the skin you use, or `CONFIG_XENO_OPT_PERVASIVE` is disabled;
- Xenomai failed to start, check the "Xenomai or I-pipe error in the kernel log" section;
- you are trying to run Xenomai user-space support compiled for x86_32 on an x86_64 kernel.

2.6.9 the application overconsumes system memory

Your user-space application unexpectedly commits a lot of virtual memory, as reported by "`top`" or */proc/<pid>/maps*. Sometimes OOM situations may even appear during runtime on systems with limited memory.

The reason is that Xenomai threads are underlaid by regular POSIX threads, for which a large default amount of stack space memory is commonly reserved by the POSIX threading library (8MiB per thread by the *glibc*). Therefore, the kernel will commit as much as *8MiB * nr_threads* bytes to RAM space for the application, as a side-effect of calling the `mlockall()` service to lock the process memory, as Xenomai requires.

This behaviour can be controlled in two ways:

- via the *stacksize* parameter passed to the various thread creation routines, or `pthread_attr_setstacksize()` directly when using the POSIX API.
- by setting a lower user-limit for the initial stack allocation from the application's parent shell which all threads from the child process inherit, as illustrated below:

```
ulimit -s <initial-size-in-kbytes>
```

2.6.10 freeze or machine lockup

Possible reasons may be:

- Stack space overflow issue now biting some real-time kernel thread?
- Spurious delay/timeout values computed by the application (specifically: too short).
- A case of freeze is a system call called in a loop which fails without its return value being properly checked.

On x86, whenever the nucleus watchdog does not trigger, you may want to try disabling CONFIG_X86_UP_IOAPIC while keeping CONFIG_X86_UP_APIC, and arm the kernel NMI watchdog on the LAPIC (nmi_watchdog=2). You may be lucky and have a backtrace after the freeze. Maybe enabling all the nucleus debug options would catch something too.

3 Issues when running Xenomai test programs

3.1 Issues when running the *latency* test

The first test to run to see if Xenomai is running correctly on your platform is the latency test. The following sections describe the usual reasons for this test not to run correctly.

3.1.1 failed to open benchmark device

You have launched `latency -t 1` or `latency -t 2` which both require the kernel to have been configured with the `CONFIG_XENO_DRIVERS_TIMERBENCH` option.

3.1.2 the *latency* test hangs

The most common reason for this issues is a too short period passed with the `-p` option, try increasing the period. If you enable the watchdog (option `CONFIG_XENO_OPT_WATCHDOG`, in your kernel configuration), you should see the "watchdog triggered (period too short?)" message.

3.1.3 watchdog triggered (period too short?)

The built-in Xenomai watchdog has stopped the *latency* test because it was using all the CPU in pure real-time mode (aka *primary mode*). This is likely due to a too short period. Run the *latency* test again, passing a longer period using the `-p` option this time.

3.1.4 the *latency* test shows high latencies

The *latency* test runs, but you are seeing high latencies.

- make sure that you carefully followed the "Kernel configuration" section.
- make sure that you do not have an issue with SMIs, see the section about SMIs.
- if you have some legacy USB switch at BIOS configuration level, try disabling it.
- if you do not have this option at BIOS configuration level, it does not necessarily mean that there is no support for it, thus no potential for high latencies; this support might just be forcibly enabled at boot time. To solve this, in case your machine has some USB controller hardware, make sure to enable the corresponding host controller driver support in your kernel configuration. For instance, UHCI-compliant hardware needs `CONFIG_USB_UHCI_HCD`. As part of its init chores, the driver should reset the host controller properly, kicking out the BIOS off the concerned hardware, and deactivate the USB legacy mode if set in the same move.

- if you observe high latencies while running X-window, try disabling hardware acceleration in the X-window server file. With recent versions of X-window, try using the *fbdev* driver. Install it (Debian package named *xserver-xorg-video-fbdev* for instance), then modifiy the `Device` section to use this driver in */etc/X11/xorg.conf*, as in:

```
Section "Device"
        Identifier  "Card0"
        Driver      "fbdev"
EndSection
```

With olders versions of X-window, keep the existing driver, but add the following line to the `Device` section:

```
        Option "NoAccel"
```

3.2 Issues when running the *switchtest* program

3.2.1 pthread_create: Resource temporarily unavailable

The switchtest test creates many kernel threads, this means that the options `CONFIG_XENO_OPT_SYS_HEAPSZ` and `CONFIG_XENO_OPT_SYS_STACKPOOLSZ`, in your kernel configuration, should be configured to large enough values. Try increasing them, rebuilding the kernel.