Troubleshooting guide

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

This file is a troubleshooting guide about various known issues regarding Xenomai.

1 Kernel configuration

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_CC_STACKPROTECTOR
: This option must be disabled on all platforms except x86_64: it requires changes to the context switches currently only implemented for x86_64.

CONFIG_KGDB
: This option can not be enabled with current versions of the I-pipe patch.

For x86 specific options see also this page.

2 Xenomai or I-pipe error in the kernel log

If the Xenomai and I-pipe messages do not appear in the kernel log as:

```
I-pipe: head domain Xenomai registered.
Xenomai: hal/<arch> started.
Xenomai: scheduling class idle registered.
Xenomai: scheduling class rt registered.
Xenomai: real-time nucleus v2.6.1 (Light Years Away) loaded.
Xenomai: debug mode enabled.
Xenomai: starting native API services.
Xenomai: starting POSIX services.
Xenomai: starting RTDM services.
```

Where <arch> is the architecture you use, check the following sections, they describe the usual error messages you may encounter.

2.1 The kernel stops 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`.

2.2 The kernel stops with an OOPS

Please make sure that you have followed the "Kernel configuration" section. Then, try capturing the oops text (using a serial console or netconsole) post the oops to the xenomai mailing list, with the kernel configuration you used to compile the failing kernel.

2.3 The 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.

2.4 I-pipe: could not find timer for cpu #x

See code -19.

2.5 Xenomai: Local APIC absent or disabled!

See code -19.

2.6 Xenomai: SMI-enabled chipset found, but SMI workaround disabled

First you should run the latency test under some load and see if you experience any pathological latency ("pathological" meaning more than, say, 100 micro-seconds). If you do not observe any such latency, then this warning is harmless, and if you find it annoying, you may disable "SMI detection" in Xenomai's configuration menu. You can skip the rest of this section.

If you observe any high latency then you have a problem with SMI, and this warning was intended for you. But the Xenomai configuration menu allow you to enable two workarounds which may help you. These workarounds may be found in the Machine/SMI workaround sub-menu of Xenomai configuration menu.

The first workaround which you should try is to disable all SMI sources. In order to do this, in the Xenomai configuration menu, select the options "Enable SMI workaround" and "Globally disable SMI". This option is the safest workaround, because when enabled, no SMI can interfere with hardware interrupt management behind your back and cause high latencies. Once this workaround enabled, you should run the latency test again, verify that your high latency disappeared but most importantly, verify that every peripheral you intend to use with Xenomai is working properly. If everything is working properly, then you are done with SMI.

If some peripheral is not working properly, then it probably needs SMI, in which case you can not simply disable SMI globally, you will need to disable all SMI sources on your system except the SMI needed by your peripheral. This is a much less safe choice, since Xenomai has to know all SMI sources to disable them, one by one. In order to choose this second workaround, unselect the option "Globally disable SMI", and select the option corresponding to your peripheral. For example, if you need legacy USB emulation to get your USB mouse working, select the option "Enable legacy USB emulation". You should then run the latency test again and verify that you do not observe any high latency and that all your peripherals are functioning correctly. If you can not find your peripheral in the list proposed in the Xenomai configuration menu, drop a mail to the Xenomai mailing list, we will try and possibly add the proper option if needed. If when running the latency test again, your peripheral is working properly and you still observe high latencies, then you are out of luck, the peripheral you want is likely to be the cause of such latencies.

Important
On some systems, SMI may be involved in thermal throttling of the CPU. Thus, switching it off **can cause hardware damage** in overheat situations. Do not disable SMIs if you are in this case.

2.7 Xenomai: system init failed, code -19

The most probable reason is that Xenomai could not find a timer.

Check that you have not enabled one of the options in the "Kernel configuration" section.

2.7.1 On x86

You will most likely also see the following message:

```
Xenomai: Local APIC absent or disabled!
Disable APIC support or pass "lapic" as bootparam.
```

Xenomai sends this message if the kernel configuration Xenomai was compiled against enables the local APIC support (`CONFIG_X86_L` 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 hw, then you need to rebuild a kernel with LAPIC support disabled, before rebuilding Xenomai against the latter;
- 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 boxen 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.

2.7.2 On other supported platforms

As on x86, on other platforms where Xenomai shares the timer with Linux, the timer is only used if it was not shut down by Linux. So you should check the log for messages about disabled timers. You can also check */proc/timer_list* to see which timers are enabled. For instance, Xenomai on SMP systems requires per-cpu local timers, so the local timers should be enabled. In case of doubt, post a message to the xenomai mailing list, sending:

- your kernel configuration
- the contents of */proc/timer_list* run on the exact kernel which has the issue
- the complete kernel boot log.

2.7.3 On a new I-pipe port

You will most likely also see the following message:

```
I-pipe: could not find timer for cpu #x
```

Starting with the I-pipe patch for Linux 3.2, the timers provided by the I-pipe patch to Xenomai are registered at run-time. So, you may lack a `struct ipipe_timer` definition, and its registration with `ipipe_timer_register()` or with the `ipipe_timer` member of the `struct clock_event_device` structure.

For an example on the ARM platform see this page.

2.8 Xenomai: 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`, `CONFIG_XENO_OPT_SEM_HEAPSZ` and `CONFIG_XENO_OPT_SYS_STACKPO` so that (size / 64k) > 2. The default values for these parameters are currently based on the assumption that PAGE_SIZE = 4k.

3 Problems 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 Xenomai: --enable-x86-sep needs NPTL and Linux 2.6.x or higher

On the x86 architecture, the configure script option `--enable-x86-sep` allows Xenomai to use the SYSENTER/SYSEXIT mechanism for issuing system calls.

However, this mechanism requires support from the libc. Currently, we know the glibc with NPTL has this support, other libraries will cause Xenomai applications to fail with this error message.

3.2 latency: 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.

3.3 Hardware tsc is not a fast wrapping one

See the "ARM tsc emulation issues" section.

3.4 Xenomai: incompatible ABI revision level

Each Xenomai branch (2.1, 2.2, 2.3, 2.4, 2.5, 2.6,...) defines a kernel/user ABI, so that it is possible to mix kernels and user-space supports of different versions in the same branch. So, for instance, after having build a system with a kernel and user-space support using Xenomai 2.6.0, it is possible to update the user-space support to Xenomai 2.6.1 without changing the kernel.

However, it is not possible to mix kernel and user-space supports of different branches.

A common reason for this error is when you run a kernel compiled with Xenomai 2.6.1 support on a system where you have a user-space installed by your Debian based Linux distribution (notably Ubuntu) from the 2.5 branch, this can not work, the two branches use different ABIs. See README.INSTALL for details on how to compile a user-space support, or to build a new `xenomai-runtime` Debian package.

If you compiled and installed the correct Xenomai user-space support, there are probably files on your system remaining from a previous installation.

3.5 Xenomai: incompatible feature set

Since kernel-space support and user-space support are compiled separately, each Xenomai application checks, at startup, whether the kernel and user-space supports have been configured with compatible options. If you see this message, it means they have not. See README.INSTALL for further details. The following sections detail the most frequent reasons for this message.

3.5.1 missing="kuser_tsc"

See the "ARM tsc emulation issues" section.

3.5.2 missing="sep"

On the x86 architecture, the configure script option `--enable-x86-sep` allows Xenomai to use the SYSENTER/SYSEXIT mechanism for issuing system calls.

However, this mechanism requires a recent kernel (2.6 or higher).

3.5.3 missing="smp/nosmp"

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.

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).

3.5.4 missing="tsc"

This error is specific to the x86 architecture. You enabled tsc in user-space by passing the `--enable-x86-tsc` option, but you selected a processor when configuring the kernel which has no tsc.

So, if your processor has a tsc (all Intel processors starting with some Pentium and Pentium Pro have a tsc), you probably mis-configured your kernel and should select the exact processor you are using in the kernel configuration and recompile it.

If your processor does not have a tsc, you should not pass the `--enable-x86-tsc` option to the configure script.

3.6 Xenomai: kernel/user tsc emulation mismatch

See the "ARM tsc emulation issues" section.

3.7 Xenomai: 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.

3.8 latency: not found

On the ARM platform this message happens when there is a mismatch between kernel and user for the EABI setting: for instance you compiled the user-space support with a toolchain generating OABI code, and are trying to run the result on a kernel with `CONFIG_AEABI` but without `CONFIG_OABI_COMPAT`. Or vice versa, when running user-space compiled with an EABI toolchain on a kernel without `CONFIG_AEABI`.

3.9 Xenomai: Your board/configuration does not allow tsc emulation

See the "ARM tsc emulation issues" section.

3.10 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.

3.11 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. Note that if you have an Intel chipset and you do not see the message:

```
Xenomai: SMI-enabled chipset found, but SMI workaround disabled
```

in the boot logs, it may mean that your chipset is not detected.

- 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; in the `Device` section of */etc/X11/XF86Config-4* add the following line:

```
        Option "NoAccel"
```

3.12 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, with its "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:

```
Xenomai: 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 (to this date the only in-tree SOC family not using this option is ixp4xx).

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:

```
Xenomai: 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:

```
Xenomai: 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.

4 switchtest fails with "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_ should be configured to large enough values. Try increasing them and recompiling.

5 Problem with my code (not Xenomai code)

5.1 "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, the skins will not export their interface to 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.

The reason for this is fully explained in the project Roadmap document, see "What Will Change With 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**.

5.2 High latencies when transitioning from primary to secondary mode

Such transition requires to wake up the Linux task underlying your real-time thread when running in secondary mode, since the latter needs to leave the Xenomai domain for executing under the control of the regular Linux scheduler. Therefore, it all depends on the Linux kernel granularity, i.e. its ability to reach the next rescheduling point as soon as such wakeup has been requested. Additionally, the task wakeup request is performed from a virtual interrupt handler which has to be run from the Linux domain upon request from the Xenomai domain, so the time required to handle and dispatch this interrupt outside of any critical kernel section also needs to be accounted for. Even if the kernel granularity improves at each new release, there are still a few catches:

- Although the use of DMA might induce additional interrupt latency due to bus bandwidth saturation, disabling it for disk I/O is a bad idea when using mixed real-time modes. This is due to the fact that using PIO often leads to lengthy non-preemptible sections of kernel code being run from e.g. IDE drivers, from which pending real-time mode transitions could be delayed. In the same vein, make sure that your IDE driver runs in unmasked IRQ mode. In any case, a quick check using the "hdparm" tool will help:

```
# hdparm -v /dev/hda

/dev/hda:
 ...
 unmaskirq    =  1 (on)
 using_dma    =  1 (on)
 ...
```

- Even if your application does not directly request disk I/O, remember that the kernel routinely performs housekeeping duties which do, like filesystem journal updates or VM commits to the backing store, so latencies due to improper disk settings may well trigger apparently randomly. Of course, if your application only operates in primary mode during all of its time critical duties, i.e. never request Linux syscalls, it will not be adversely affected by DMA deactivation or IDE masking, since it will remain in the Xenomai domain, and activities from such domain can preempt any activity from the Linux domain, including disk drivers.

5.3 Any Xenomai service 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_PER` 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.

5.4 My application reserves a lot of memory

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

The Xenomai tasks are underlaid by native POSIX threads, for which a huge default amount of stack space memory is reserved by the native POSIX support, usually 8MiB per thread, so the overall allocated space is about 8MiB * `nr_threads`, which are likely to be locked using the `mlockall()` service, which in turn even commits such space to RAM.

Unfortunately, this behaviour cannot be controlled by the "`stacksize`" parameter passed to the various thread creation routines, i.e. the latter is about limiting the addressable stack space on a per-thread basis, but does not affect the amount of stack memory initially reserved by the POSIX library. A work-around consists of setting a lower user-limit for initial stack allocation, like calling:

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

in your parent shell before running your application (defaults to 8192).