Building and running applications with Xenomai 3.x

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

The latest version of this document is available at this address.

Make sure to read the installation document before going through this one.

For questions, corrections and improvements, write to the mailing list.

1 Compiling a Xenomai application

You should use the `xeno-config` script to get the proper compilation and linker flags related to Xenomai, in order to build your application for whichever *Cobalt* or *Mercury* core.

Here is a trivial Makefile fragment retrieving the compiler and flags for building the single-file application `vxapp.c`, over the VxWorks emulation API:

```
XENO_CONFIG := /usr/xenomai/bin/xeno-config
CFLAGS := $(shell $(XENO_CONFIG) --vxworks --cflags)
LDFLAGS := $(shell $(XENO_CONFIG) --vxworks --ldflags)
CC := $(shell $(XENO_CONFIG) --cc)

EXECUTABLE := vxapp

all: $(EXECUTABLE)

%: %.c
        $(CC) -o $@ $< $(CFLAGS) $(LDFLAGS)
```

2 Running a Xenomai application

For *Cobalt*, you will need the real-time core built into the target Linux kernel as described in this document.

For *Mercury*, you need no Xenomai-specific kernel support so far, beyond what your host Linux kernel already provides. Your kernel should at least provide high resolution timer support (`CONFIG_HIGH_RES_TIMERS`), and likely complete preemption (*PREEMPT_RT*) if your application requires short and bounded latencies.

An application recognises a set of options that may be passed on the command line, namely:

--<api>-clock-resolution=<ns>
: The clock resolution available with the real-time API, given as a count of nano-seconds, i.e. HZ=(1000000000 / ns).
<api> is the name of one of the existing Xenomai APIs your application can be linked against, e.g. *psos*, *vxworks* or *alchemy*. When your application combines multiple APIs, you may pass several clock-resolution switches to set them all.
The default value depends on the API being considered. For instance, the VxWorks ™ and pSOS ™ emulators default to 1 millisecond clock rates. The Alchemy API is tickless by default, i.e. `--alchemy-clock-resolution=1`.

Caution
Specifying a resolution greater than 1 nanosecond requires the low resolution clock support to be available from the Xenomai libraries (see the `--enable-lores-clock` configuration switch).

--mem-pool-size=<kb>
: The initial size in Kilobytes of the main memory pool. This option only makes sense when the TLSF allocator is being used (i.e. `--enable-debug` is not specified when compiling the Xenomai libraries). This is only a hint, since TLSF will increase the main pool size dynamically as needed, if needed.

However, this option may be used to pre-allocate the specified amount of memory to the application process, thus avoiding costly system calls to extend the data segment of such process while operating in time critical mode.

--no-mlock
Tells the Xenomai libraries not to lock the process memory while initializing. The application will have to handle this task when and how it sees fit, in order to avoid the extra latency induced by virtual memory paging. Otherwise, `mlockall(MCL_CURRENT | MCL_FUTURE)` is automatically invoked as part of the API initialization duties.

This flag only applies to the *Mercury* core. Memory must be locked when invoking dual kernel services, therefore this switch is a nop over *Cobalt*.

--registry-root=<path>
Tells Xenomai to root the object registry at the given path, instead of `/mnt/xenomai` by default (see the `--enable-registry` switch from the configuration options).

--no-registry
This switch disables registry support at runtime. No real-time objects will be exported to `/mnt/xenomai/<session>.<pid>`, despite the registry code was compiled in.

--session=<label>
Name of the session the new process will be part of (or create if not present). If `--enable-pshared` was given when configuring the Xenomai libraries, this label allows multiple processes giving the same label at startup to operate on the same set of objects.

For instance, a process can post a semaphore created by another process from the same session. This is done using a common heap area, shared by all processes within the same session.

This label is also used to form the registry mount point for each process, e.g. `/mnt/xenomai/<session>.<pid>`. See `--enable-registry` from the build options.

By default, *anon* is used as the session label.

--reset
Forces removal of an older session. This only works if the process which initiated the former session has exited, otherwise an error is raised.

--cpu-affinity=<cpu[,cpu]...>
Sets the CPU affinity of threads created by the Xenomai libraries within the new process.

--version
Writes the Xenomai version information to stdout. The program immediately exits with a success code afterwards.

--dump-config
Dumps the configuration settings to stdout. Those settings are defined as a result of running the configure script. The program immediately exits with a success code afterwards.

3 Valgrind support

Running Xenomai applications over *Valgrind* is currently available to the *Mercury* core only.

When the Valgrind API is available to the application process, the configuration symbol CONFIG_XENO_VALGRIND_API is defined at build time, and may be tested for existence by the application code. See the tool documentation at this address.

The Xenomai autoconf script will detect the Valgrind core header on the build system automatically, and define this symbol accordingly (i.e. /usr/include/valgrind/valgrind.h).

Note
You may need to install the Valgrind development package on your build system to provide for the core header files. For instance, such package is called *valgrind-devel* on Fedora.

---

4 Available real-time APIs

Alchemy This is a re-implementation from scratch of Xenomai's 2.x *native* API, fully rebased on the new RTOS abstraction interface.

pSOS pSOS ™ is a registered trademark of Wind River Systems, Inc.

VxWorks VxWorks ™ is a registered trademark of Wind River Systems, Inc.