Exercise 6: BRTT and Xenomai

In this exercise we will do the same things as in the previous, just that we will be using Xenomai which improves the real-time properties of Linux. Using Xenomai a thread/task can run with a higher priority than the rest of the Linux system, including Linux itself. Xenomai is available when you boot in Linux Mint.

There are alternatives to Xenomai e.g. RTAI and RT Linux. Both RTAI and Xenomai allows for non-root user to run tasks in hard real time. Previous years this exercise was done using RTAI. If you are interested in looking at RTAI yourself, the exercise can be made available.

The following links are to some Xenomai documentation that you will need to use to complete the exercise. The cs.ru.nl link is a **great** tutorial and parts of it is **highly** relevant for this exercise. The other link is to the Xenomai API reference. This exercise expects you to use the native skin.

http://www.xenomai.org/documentation/xenomai-2.6/pdf/Native-API-Tour-rev-C.pdf

http://www.cs.ru.nl/lab/xenomai/exercises/

http://www.xenomai.org/documentation/xenomai-2.6/html/api/index.html

1. Xenomai

Xenomai introduces two classes of scheduling, primary and secondary mode. In primary mode a task is running with a higher priority than the OS. The operating system and services provided by it are ran in secondary mode. A task will switch between these two modes dependent on what calls it is making. Xenomai calls puts the task in primary mode, and non-xenomai calls puts the task in secondary mode. Primary mode tasks are scheduled in front of secondary tasks.

If your computer becomes sluggish after starting a RT application, you can kill the program with ctrl+c. If the computer freezes completely, you have probably started an infinite loop with higher than Linux priority. Ctrl-c will not work; you have to power off and on the computer. Needless to say, you should avoid this.

1.1. Makefile

To use Xenomai you should include the following in your makefile:

```
XENO_DESTDIR:=
XENO_CONFIG:=$(XENO_DESTDIR)/usr/bin/xeno-config
XENO_CFLAGS:=$(shell DESTDIR=$(XENO_DESTDIR) $(XENO_CONFIG) --skin native --cflags)
XENO_LIBS:=$(shell DESTDIR=$(XENO_DESTDIR) $(XENO_CONFIG) --skin native --ldflags)
```

And include XENO_LIBS and XENO_CFLAGS in your LIBS and CFLAG variables. These lines will set the correct path to the Xenomai libraries.

1.2. Memory and Xenomai

To avoid a real time task to be swapped to disk, Xenomai provides the `mlockall()` function. Calling this, with the argument `MCL_CURRENT|MCL_FUTURE`, will lock the current memory allocations and future

memory allocations to the main memory. This is important because when executing under Hard real time constraints we want to avoid all unnecessary time delays. This call should be the first thing your program does.

1.3. Creating real time tasks

A real time task is created by a call to rt_task_create().

Int rt_task_create (RT_TASK *task, const char *name, int stksize, int prio, int mode)

- task: Adress of the task descriptor. Pointer to a RT_TASK.
- name: Descriptive name string for the task.
- stksize: Stack size for the task. 0 sets the stack size to a predefined value.
- prio: Base priority. 0-99 where 0 is the lowest.
- mode: Task creation mode. Flags can be OR'ed together:
  - T_CPU(cpuid): Sets the cpu affinity of the task.
  - T_JOINABLE: Allows another task to wait for the termination of the task by using `rt_task_join()`. User space only flag.

`rt_task_create()` returns 0 on success and negative values on error.

1.4 Starting real time tasks

After creating a task it must be assigned a function to execute. This is done with the rt_task_start():

int rt_task_start (RT_TASK *task, void(*entry)(void *cookie), void *cookie)

- task: Adress of the task descriptor.
- entry: pointer to a function.
- cookie: void pointer to the arguments passed to entry.

This function also returns 0 on success and negative values on error.

1.5 Periodic execution

The real time clock is already running when you start a Xenomai program. To set the period of a rt task:

int rt_task_set_periodic (RT_TASK *task, RTIME idate, RTIME period)

See the API reference for how this function work. period is set in nanoseconds(ns). Use the API reference to find out how to invoke a wait within a task.

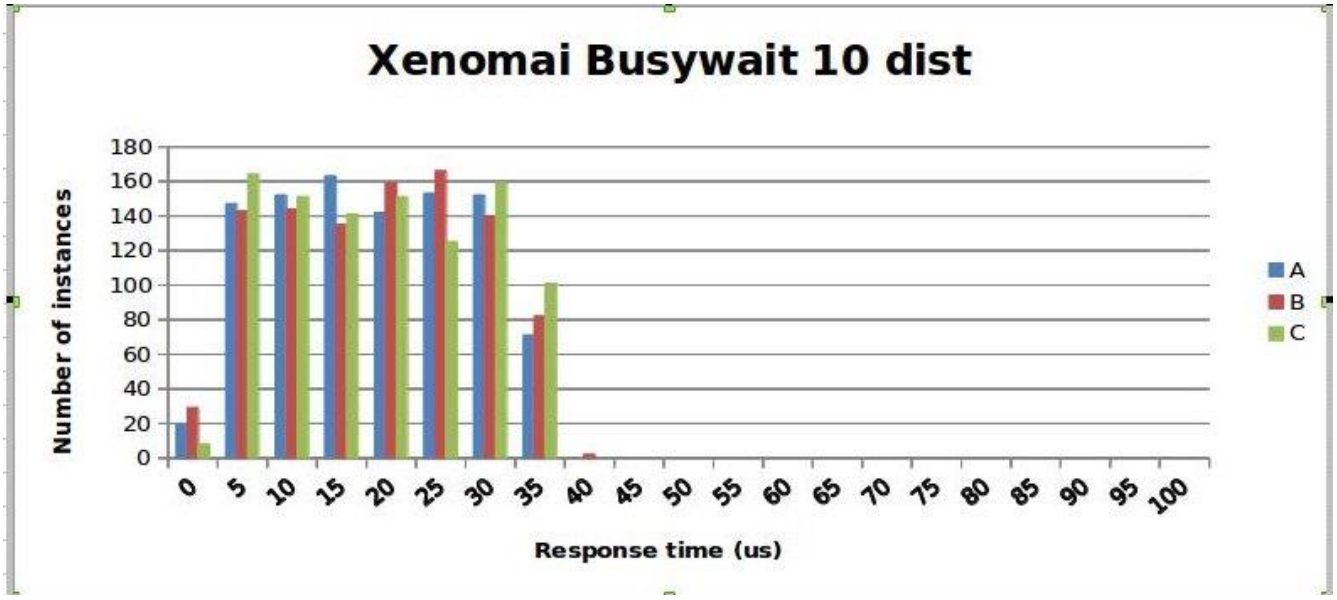
To make sure you got everything right so far, you should create a periodic task which periodically prints to the screen.

2 BRTT Reaction Test

Last week we used periodic POSIX threads to do a reaction test. In this exercise we will do the same test with Xenomai tasks instead of normal pthreads. To get a fair comparison we must do the same things as last week, meaning that we must start 10 pthreads in infinite loops and make sure everything happens on the same CPU core. You can do this the same way you did last week. DO NOT create these tasks as Xenomai tasks as this will most likely lock you computer.

For this exercise you will only run a periodic test. Because in the process of creating a functional busy wait test you will most likely crash the computers multiple times.

How does the figure below compare to the busy wait results you got last week?



2.2 Testing Xenomai

Use what you got so far to create a similar test to last week. 3 periodic responding tasks executing in primary mode and 10 disturbance tasks running in secondary mode. All executing on the same CPU.

Assignment A:

Run the following reaction tests:

1. 1000us no disturbance
2. 1000us with 10 disturbance threads
3. 100us with 10 disturbance threads
4. Try lower than 100us down towards 10us with 10 disturbance threads.

3. Approving the exercise

For this exercise you need to show the following to a student assistant to get the exercise approved.

1 How does the provided Xenomai busy-wait test compare to the same test you did last week, and why is it so?

2 Run a reaction test with the lowest period that you got good results

- What happens when the period is lowered further?

3 Show results from tests with period of 1000 us, 100 us, 10 us and any others that you got that was interesting.

- How did the results compare with POSIX Linux?
- Explain what happened as 10 us?

4 Hand in the equipment.

4 Appendix

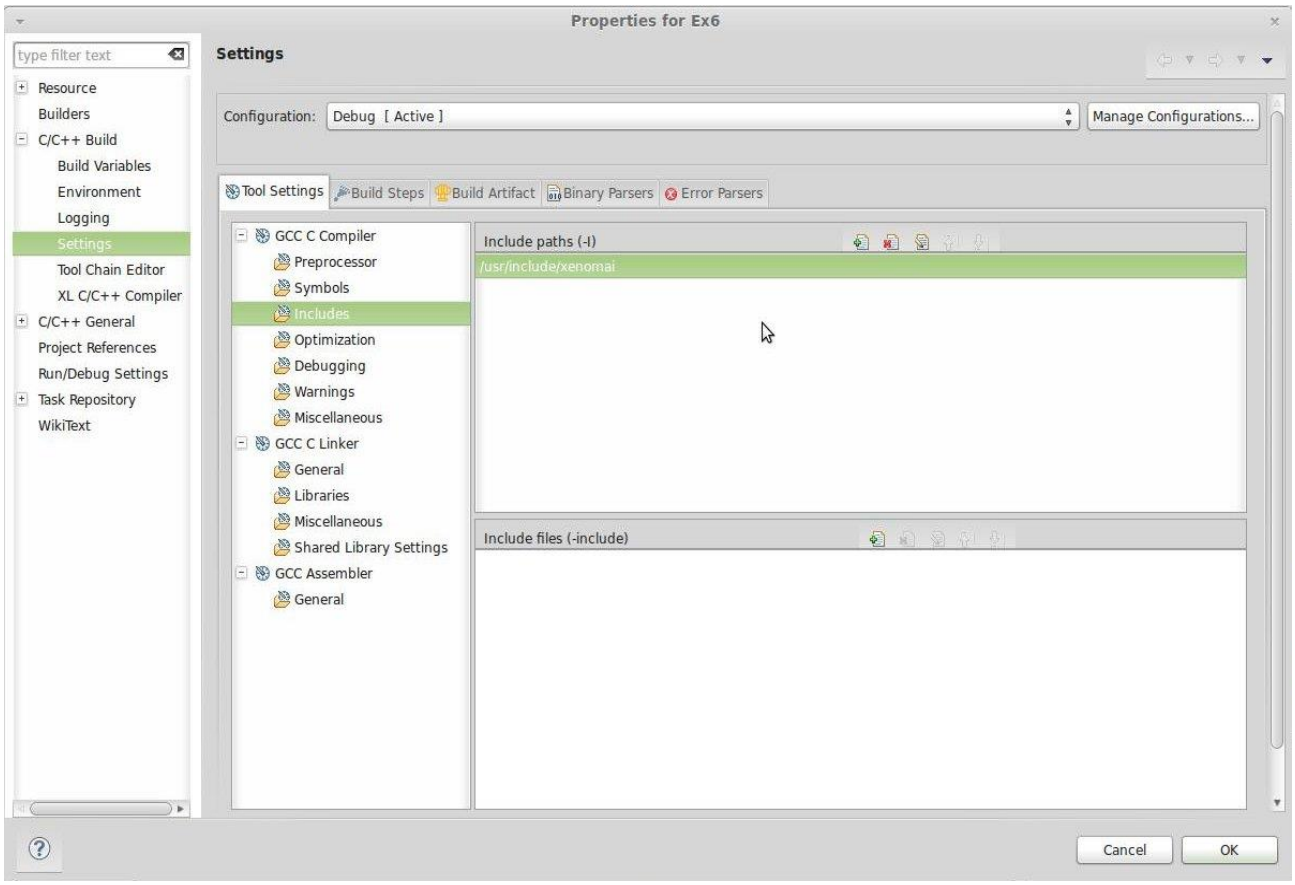
To use the Xenomai libraries with eclipse you must set the paths and flags returned from xeno-config.

Run:

```
$xeno-config --skin native --cflags

$xeno-config –skin native --ldflags
```

Input the information in the project properties window.



Cflags: Compiler flags, GCC C Compiler

- -I (capital i): Includes (remove the -I)
- -D: Symbols (remove the -D)

Ldflags: Linker flags, GCC C Linker

- -l: libraries (remove the -l)
- -L: librarie search path (remove the -L)

4.1 POSIX Skin

This exercise can also be done with the POSIX skin. This allows the use of pthreads as rt tasks. This is a bit more auto-magical, but if something does not work, it can be harder to figure out. The student assistants will not assist with the POSIX skin. The documentation and tutorials for native skin is a lot better. Therefore it is recommended to use the native skin.

Porting a linux application to Xenomai dual kernel

4.2 Printing in rt tasks

The Linux syscall `printf()` will force your task to enter secondary mode. To avoid this unwanted behavior, you should use `rt_printf()` these functions are located in *rtdk.h*. Before these functions can be used the program must issue a call: `rt_print_auto_init(1)`.

Printf prints to the terminal and printk prints to the kernel print output, which is available with:

```
$dmesg
```