SYSC 3303 Real-Time Concurrent Systems
Lab #3

Getting Started

1. Re-boot your machine into Xenomai by selecting **RT** when prompted.

Objective: Experiments with Real-Time Linux Threads

To begin, you are going to repeat the same exercise as in Lab 2, except using Xenomai and C. As before, two threads will be created, each with two arguments: a unique character that will be used to identify the thread (i.e. a 1-letter name) and the number of loops it shall run. You shall experiment with their parameters and observe their behaviour.

- Note: I'm trying to get you to learn *command-line* arguments (namely, args in Java and argv[] in C). Do not hard-code values. Take the time to learn these arguments now

Part A: Learning Xenomai's Documentation.

1. Nagivate to the online documentation: In a browser, go to **www.xenomai.org**
    1. Open the API reference material: On the left side bar, under **Documentation**, click on **Programming Interfaces**
    2. Open the Native API reference Material:
        1. Under Programming **Reference Manuals**, under **Previous Releases**, under v2.4.x (or just any version), click on **html**
        2. On the left side bar, open **Modules**, open **Native API**: view the list of libraries
        3. Today's library is the **Task Management Library**
    3. Scan through the list of functions - rt_task_create, rt_task_start, rt_task_join, rt_task_delete
2. There are also example programs that you can use to copy-paste code, or learn through reverse-engineering how to use the C syntax to pass the needed parameters.
    1. On the left side bar, scroll down to the bottom to find **Examples**.
    2. For this lab (on task creation), click on **trivial-periodic.c**

You do not need to submit or demo anything. This is just to help you become an **active learner**.

Part B: Printing

Using the same class diagram from Lab 2, implement the **second version** of the two threads, namely, that both *Sleepy* and *Yappy* should **log** its name each time through its loop.

- Suggestion: Copy-paste the code from a Xenomai example (i.e. trivial-periodic.c), so that you get the boiler-plate code. Name the file **lab3b.c**
- Your file should contain three functions:
    - main() for initializing the threads and the log,
    - Two "run" functions with the required call signature (view the sample code): one containing code for *yappyRun* and one containing code for *sleepyRun*
- The log should be a simple array, declared as a global, outside the scope of any functions (i.e. at the top)
- In the two threads, as in Lab 2, only log 1-character **names** ('y' and 's'), with no linefeeds (\n).
    - The **names** and **nLoops** must be passed as an argument (i.e. cookie) to the threads (so that you practice casting)
    - Read the Implementation Instructions below for notes about sleeping and printing in Xenomai
- main() should print out the contents of the log before terminating.
    - In C, strings must be null-terminated. Before printing, make sure that the last element in the character array is \0.
    - Question: Can you use rt_printf() in main() or must you use printf()? Why or why not?
- You will have to edit your Makefile too! Make a copy of the Makefile from Lab 1, but change the name of the file to build to *lab3b*

Run your program. Did it run as expected?

Implementation Instructions

- Remember that the Xenomai is a real-time kernel buit on top of a "normal" Linux kernel. Consequently, at times, you have the choice of using basic Linux services of real-time Xenomai services, for example, for printing, sleeping and spinning.
    - Objective: Have you explore the Xenomai API as well as the **tips on the Resource page.**
    - Do not use the Linux sleep() procedure. It will lead to bugs when running Real-Time Tasks. Find rt_**task**_sleep().
    - Consider the use of rt_**task**_join() and rt_**task**_delete()
- **There are idiosyncrasies with the sleep and print functions. Sometimes, in the past, we have had problems. If you do have troubles, consider:**
    - Do not use printf(). Use rt_printf(). Caution! This function is not in the API; you must read the XENOMAI TIPS on the COURSE RESOURCE page! Find the example code! This function requires initialization before using, by calling rt_print_init_auto(**?**) by one and only one thread before the first rt_printf().
    - Worst case, use Linux sleep() not rt_task_sleep().

Lab Reflection and Analysis:

Run your program. What got printed out first? Which of the two created threads ran first? What was the pattern of the numbers printed out?
Run your program several more times. Is the same thread always run first?
Run your program several more times, **changing the sleep time and the spin time**. What changes do you see?

Part C : Xenomai Threaded Server

We are going to combine your code from Lab 1 with the threading code from this lab to make a threaded TCP server: When a TCP server receives a connection request, it creates a new thread that will then exchange TCP data/messages with the requesting client while the TCP server goes back to listen for further connection requests.

- As in all Lab exercises, do not worry about making the server do anything meaningful.
  - If you're in a rush, just hard-code messages to send back-and-forth.
  - If you have some time to learn: How about we just send the date-and-time back and forth between clients and servers?
    - Google: Linux how to get current time in C
- **Focus on the server code that creates a new "worker thread" to actually read/write data over the TCP connection.**
  - **What parameter must be passed as a cookie to the worker thread?**
- Testing will require that you run more than one client.
- As in all lab experiments, you may have to create "conditions" for overlapping execution
  - Slow down the server by spinning or sleeping
  - Loop the clients so that they execute more than once, so you have more than one try to see concurrent execution.

Submitting Your Lab Work

- Submit your files (lab3b.c, lab3c.c).

Submit your work (even if it's not quite complete) by the end of the lab period in which you are registered.
Late submissions will **not** be accepted by the submit program or the instructors!