Creating simple applications with Xenomai

Objectives:

Become familiar with a full-featured real-time operative system (RTOS).

1. Get familiar with the RTOS API, task model, RT modules, creation and termination of tasks and basic IPC mechanisms
2. Develop simple multi-task real-time applications using Xenomai

Procedures:

1. Analysis of the supplied source code

Download the supplied source code form the course website (file "xenomai-class-sources.zip").

1. Observe the application source code("sample_task.c"). Watch the system calls to define, create and activate tasks. Study the structure of the real-time tasks.
2. Compile and execute the sample application. Note that it is supplied a "makefile" to simplify the compilation process. Watch the periodic execution of the task.

2. Tasks to execute:

Complete the following tasks. You are required to deliver a short report with the observed results and a summary of the code changes carried out, so please annotate all the relevant data as you complete these tasks.

1. Modify the task code so that it shows the maximum and minimum time between successive jobs. Execute the application and, concurrently, execute other Linux processes. Compare these results with the ones obtained with the real-time services on Linux.

2. Add two other tasks to the application, with distinct priorities. Force these tasks to share the same CPU core. Test several priority allocations and observe the impact on the regularity of the task's jobs. Correlate these results with the priority.

3. Many applications are composed by diverse modules/tasks with a chained execution (precedence constraints). Modify the application in such a way that the first task remains periodic and the other ones become sporadic. The first task should, at the end of its execution, activate the second task, and so on and so forth. Each task should pass a sequential number to the following task. This allows to check if the execution order is correct and if the inter-task communication operates properly.

    - Suggestion: use semaphores (see rt_sem_create(), rt_sem_p(), rt_sem_v()) combined with shared memory (a simple global variable, in this case).

4. There are mechanism for inter-process communication that allow communication between user and real-time domains. Use one of these mechanisms to make each task communicate its state changes (start and finish) to the user space. Then build a user-space program that prints this information on a terminal.