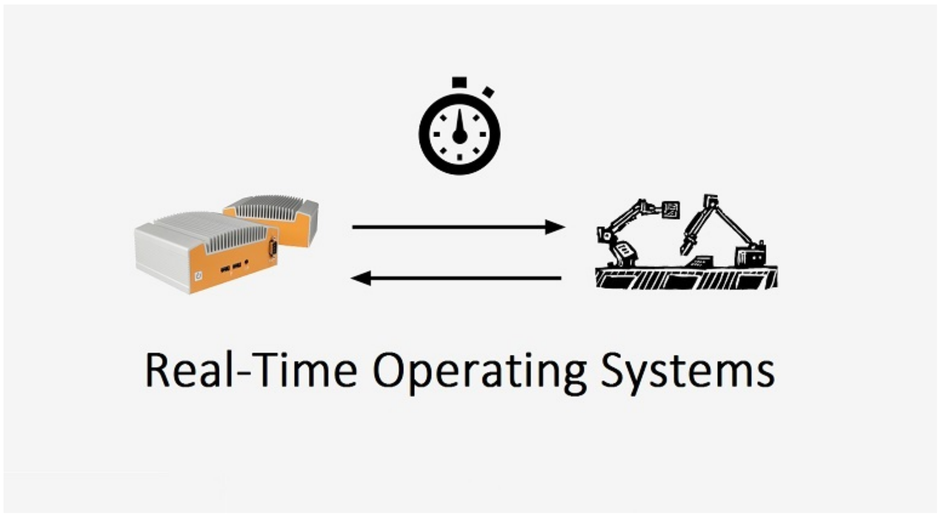

Real-time Operating Systems

Tutorial 2: Xenomai introduction
2022/2023

Paulo Pedreiras
DETI/UA/IT
pbrp@ua.pt

October 17, 2022

1 Objectives

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

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

2 Procedure

- Download the sample code provided at the course webpage
- Carry out the steps indicated at Sections 2.1 and 2.2
- Where appropriate, take note on your notebook of the code modifications and results observed.

2.1 Installation and analysis of the sample code

- Install Xenomai, Mercure "flavor", following the indications provided in the Xenomai installation tutorial slides.
- Analyze the provided source code.
  Pay special attention to the overall program structure, including task structure, real-time API for task creation, etc.
- Compile and execute the Xenomai task and then other normal Linux processes, concurrently, in different terminals, and observe the behavior.
  - As for the case of the previous tutorial (Linux Real-Time services), use processes that generate intensive I/O operations (e.g. backup a big folder with tar, watch youtube videos).
  - Repeat the operation several times and annotate the results.
  - Note that the impact depends on the specific HW. Launch at least as many load processes as CPU cores present on the testbed computer.

2.2 Using Xenomai

- [**A1**] Modify the source code provided so that it shows the maximum and minimum time between successive jobs of the same task
- [**A2**] 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
- [**A3**] Many real-time applications are composed of diverse modules/tasks with a chained execution. Each one of such tasks carries out a specific function, such as data acquisition, data stream processing, data analysis, data storage, alarm generation, etc. In this assignment it is proposed to develop a real-time data acquisition and processing application of this type, with the following characteristics:
  - The application is composed of the following three tasks: Sensor, Processing and Storage;
  - **Sensor task:** the sensor task is periodic (period is defined by macro ACK_PERIOD_MS, with a value at your choice). In every job this task acquires a sensor value and sends it to the processing task, via a message queue.
  - **Processing task:** this task takes every sensor reading generated by the sensor task and applies a filter. The filter output is then sent to the storage task, again via a message queue.
  - **Storage task:** the storage task takes every value generated by the processing task and stores it in permanent storage.
  - To avoid having a physical sensor attached to the system, the sensor task reads samples from a text file named "sensordata.txt", which contains a 16 bit unsigned integer number per row that emulates one sensor reading.
  - The processing task implements a simple 5 point moving-average filter. Despite being very simple, moving-average filters are common in DSP, among other reasons, because they are optimal for reducing random noise while retaining a sharp step response. You should implement a non-recursive version, in which each output value is computed as the average of M samples, as follows:

$$y[i] = \frac{1}{M} \cdot \sum_{j=0}^{M-1} x[i+j]$$

  - The Storage task could, e.g., upload the data to a Cloud system. To keep it simple, the task should take each filtered data point and store it in a file named “sensordataFiltered.txt”, as a 16 bit unsigned integer.
  - The code should provide a generic solution, i.e., cannot depend on the number of CPUs, number and/or phasing of tasks, etc.. Moreover, task’s internal operation should be transparent to the outside. E.g., if the processing task is modified to implement a different filter, the sensor and storage tasks should not need to be modified (concept of modular software).
  - When available, Xenomai native functions must be used (e.g. semaphores, queues, etc.).

3 Deliverables

The following elements should be uploaded for evaluation:

- A compressed file with a Makefile and the source code corresponding to Assignments A2 and A3 (two independent “.c” files)
- A **one page** report, in “pdf” format, with your analysis of:
  - Analysis of the correlation of task’s priority with its regularity, as observed in Assignment A2.
  - Time diagram associating relevant RTOS events to task execution in Assignment A3.

4 Notes

4.1 Note 1

The execution of programs with real-time priorities requires superuser privileges. The command “sudo” allows a user to execute certain commands with this kind of privileges.

4.2 Note 2

In some distributions it is necessary to configure the environment. To do that:

```
$export PATH=$PATH:/usr/xenomai/bin/
$export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/xenomai/lib/
```

For a permanent solution update the configuration scripts (".profile", ".bash_profile","/etc/environment") accordingly.
You ca issue the command "source SetXenoEnv.txt" to set environment.