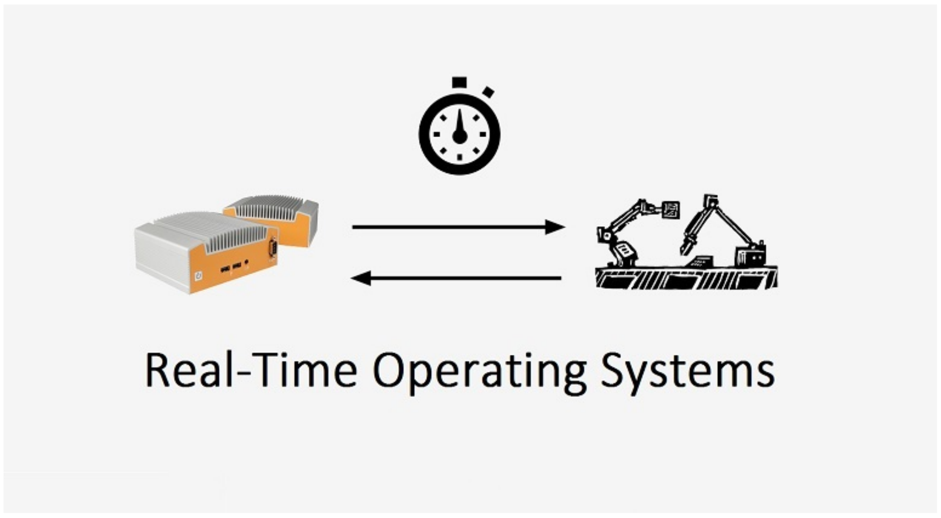

Real-time Operating Systems

Tutorial 2: Xenomai introduction
2021/2022

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

October 1, 2021

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 applications are composed of diverse modules/tasks with a chained execution (precedence constraints). Modify the code developed in Assignment A2 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.

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

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
- 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.