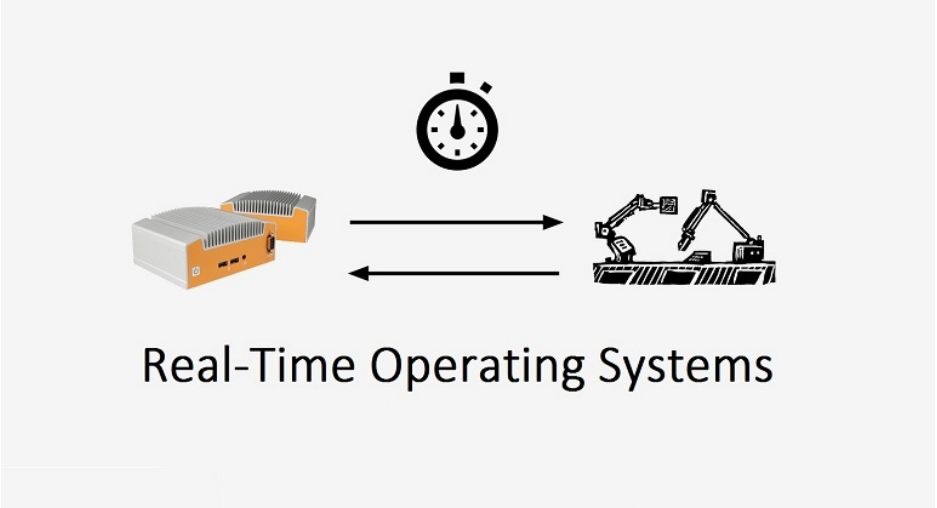

Real-time Operating Systems

Tutorial 1: RT Services on Linux
2021/2022

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

October 1, 2021

1 Objectives

- Creating periodic processes and observe its temporal behavior on Linux
- Utilization of real-time services of Linux

2 Procedure

- Download the sample code provided at the course webpage ("LinuxRT-Services.tar.gz")
- 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, in order to evaluate the impact of the RT services.

2.1 Observation of the behavior of a periodic task in Linux

- Analyze the provided source code.
  Pay special attention to the technique used to generate periodic activations and to the structure of a "task".
- Tune the "Heavy_Load" function to take an execution time of around $20\,ms$.
- Launch the "task" and then other processes, concurrently, in different terminals, and observe the behavior.
  - 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 Applying the RT services of Linux

- **[A1]** Modify the program in such a way that the periodic thread receives a fixed real-time priority, hard-coded in the source code. Repeat

now the experiments above, making sure that it is used a similar interfering load (i.e., the same applications are used).

- [**A2**] Modify the code developed in **A1** to allow passing priorities via command line, as an argument. Execute several instances of the application, in different terminals, with different priorities. Analyze the results.
- [**A3**] Modify the code developed in **A3** so that tasks are all assigned to CPU 0. Execute several instances of the application, in different terminals, with different priorities. Analyze the results.

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:
  - The improvements observed in A1 with respect to the execution of the task using the standard Linux scheduling services
  - The impact of priorities in Assignment A3
  - Note: your analysis should be supported by numerical values.

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

To allow the observation and interpretation of the interference pattern between tasks in multi-core CPUs, it necessary to force all processes to execute on the same CPU (aim of Assignment A3). That can be done with the following code, which should be placed at the beginning of the program.

```
...
#define _GNU_SOURCE /* Required by sched_setaffinity */
...
/* Variables*/
cpu_set_t cpuset;
...
/* Forces the process to execute only on CPU0 */
CPU_ZERO(&cpuset);
CPU_SET(0,&cpuset);
if(sched_setaffinity(0, sizeof(cpuset), &cpuset)) {
    printf("\n Lock of process to CPU0 failed!!!");
    return(1);
}
...
```