REAL-TIME INDUSTRIAL SYSTEMS
PROJECT ASSIGNMENT PROPOSALS
(a.a. 2021/22)

1. **rt-cgroup analysis**. The rt-cgroup is a new feature in Linux to enable the creation of real-time containers. The assignment requires to analyze the code of rt-cgroups on Linux kernel to derive the server model adopted and (optionally) the reasons of conflict with PREEMPT_RT. Then requires to perform experiments to test the functioning of rt-cgroups with arbitrary FIFO periodic/aperiodic task sets (or taking advantage of the cyclictest utility)

2. **Simultaneous multithreading for Monitoring** . Usually, on the modern Intel micro-architectures (new ARM too), we can execute two hardware threads simultaneously (SMT) on the same physical core. Average private resource utilization is 30% better with only 5% more hardware. The two hardware threads share functional units and memory access opening new interesting throttling possibilities. Furthermore, we can read each other's performance events adding new monitoring opportunities.
   Can we use one hardware thread to monitor a real-time execution task on the other virtual core? I.e., emulating a hardware controller forecasting eventual deadline miss. We need to design and quantify the monitoring overhead deterministically for the WCET analysis. The advantage is a fine-grained control without context switch or modifying the real-time program. The assignment requires a deterministic evaluation of SMT temporal contention (it should be fair the policy) when the virtual core is executing distinct kinds of workload (We provide the measurement tools) on a Bare metal Linux on x86.

3. **Simultaneous multithreading for Throttling (memory regulation)** In HPC platforms critical cores and no-critical cores could share hardware resources. While critical tasks need timing guarantees, no-critical tasks are interested in the average response time. Hence, we need an efficient memory bandwidth regulation algorithm: a monitoring policy to detect the interference and a throttling policy to prevent memory access. Until now, the core execution stop seems the only possible throttling solution without hardware support. Can we use one hardware thread (usinig the SMT technology) to throttle the no-critical virtual thread? I.e., can we use one thread to limit the memory accesses of the other virtual core? As for assignment 2, also in this case is required a deterministic evaluation of SMT temporal contention (it should be fair the policy) when the virtual core is executing distinct kinds of workload on a Bare metal Linux on x86.

4. **POSIX Sporadic server implementation.** Implement a Sporadic Server to manage aperiodic or periodic tasks using RT_POSIX primitives and SCHED_FIFO tasks over a PREEMPT_RT patched Linux kernel. The functioning of the Server has to be demonstrated with experiments on arbitrary FIFO periodic/aperiodic task sets

5. **Kubernetes orchestration of real-time containers**. Recompile the Linux kernel in the low-latency mode, setup an Mixed-Criticality System with both normal containers and real-time containers upon Docker. Can they be orchestrated with Kubernetes? If yes, how is it handled?

6. **Isolation test on an embedded board (requirement: Raspberry-PI board).** Setup a Mixed-Criticality System with Linux on 2 cores and FreeRTOS on other two cores, create real-time tasks in linux (preempt-rt), and in FreeRTOS (maybe sampling data from a sensor if you have) and compare their activation latencies. Do they interfere? How? Try in isolation and with different stress tests.

7. **Real-time Monitor implementation with RT_POSIX**. Implement a real-time monitor (MONITOR_TASK) for deadline miss detection using RT_POSIX primitives over a PREEMPT_RT patched Linux kernel. Given a number of rt-tasks, the monitor shall detect if a task misses its deadline, and report the detected deadline to an Analysis task (ANALYSIS_TASK). The ANALYSIS TASK periodically reports statistics on the detected deadline misses, e.g., total and per-task number of deadline misses. Another task (CONTROL_TASK) allows to set the recovery strategy that the MONITOR_TASK shall perform in case of deadline miss.

8. **Isolation test on the Jailhouse partiotioning hypervisor**. The development of real-time embedded systems is increasingly requiring the use of virtualization techniques. In particular, partitioning hypervisors are those that can theoretically guarantee the highest level of isolation.
   Jailhouse is an example of the most popular hypervisors at the moment, which also allows the use of inter-partition communication mechanisms. The activity intends to explore the isolation mechanisms provided by this type of hypervisor, and to develop robustness approaches that supports classical testing activities:
   a. Test temporal isolation of critical cells against one or more stressful cells in MCSs
   b. Test isolation of inter-cell communication mechanisms (IVSHEM) in MCSs
   References:
   [1] https://github.com/siemens/jailhouse/tree/master/Documentation
   [2] https://lwn.net/Articles/578295/
   [3] https://lwn.net/Articles/578852/
   [4] https://github.com/ldesi/virtualization_technologies_course/tree/master/5_real_time_virtualization

9. **Analysis of the Bao hypervisor.** Bao is a lightweight, open-source embedded hypervisor which aims at providing strong isolation and real-time guarantees. Bao provides a minimal, from-scratch implementation of the partitioning hypervisor architecture.
   a. Understanding architecture and capabilities for temporal and fault isolation mechanisms for MCSs
   b. Test isolation capabilities for MCSs
   References:
   [1] https://lwn.net/Articles/820830/
   [2] https://github.com/bao-project/bao-hypervisor

10. **Analysis of Dom0less Xen and Hyperlaunch.** The design enables seamless transition for existing systems that require a dom0, and provides a new general capability to build and launch alternative configurations of VMs, including support for static partitioning and accelerated start of VMs during host boot, while adhering to the principles of least

privilege. It incorporates the existing dom0less functionality, extended to fold in the new developments from the DomB project, with support for both x86 and ARM platform architectures, building upon and replacing the earlier 'late hardware domain' feature for disaggregation of dom0.

a. Understanding architecture and capabilities for temporal and fault isolation mechanisms
b. Test isolation capabilities for MCSs

References:
[1] https://wiki.xenproject.org/wiki/Hyperlaunch

11. **Analysis of the sel4 microkernel**. Despite the offering of fully fledged bloated rtos systems, when dealing with industrial standards and certification minimality and independence is the key. Microkernels are incredibly good with this. Install seL4, a spread out L4 microkernel that claims to have a sound timing analysis (https://github.com/seL4/seL4), with two partitions: one non critical and non real time partition, and another one with narrow timing requirements (bare metal or free rtos). Analyze it: can Linux do intensive networking activity without interfering with critical activities?

12. **EDF on FreeRTOS**. FreeRTOS is one of the most used Real-Time OS for small embedded systems. It implements a static priority-based scheduler because of its simplicity and low overhead on low-end processors (like ARM M4 etc..). The possibility to deploy freeRTOS on RPUs, that are between low-end and application-class processors, could be an opportunity to use a more sophisticated scheduler; implement Earliest Deadline First (EDF) on freeRTOS and test it. The functioning of the scheduler has to be demonstrated with experiments made on RPUs. Is the overhead caused by the scheduler too high also for RPUs?

13. **Isolation tests on Xen**. Install Xen on a PREEMPT_RT-patched Linux kernel, and then realize a mixed criticality system in which one guest OS has a PREEMPT_RT-patched Linux kernel while one or more other gest OSes are simple standard Linux. Verify with an adequate number of tests the isolation achieved by XEN using RTDS and NULL SCHEDULER: in the first case the guests may use the same physical processors, while obviously with NULL SCHEDULER the physical processors are pinned to guest OSes.

14. **Implementation of a Sporadic Server on FreeRTOS.** On an MPSoC architecture, the RPUs can execute one or more periodic/aperiodic tasks required by APUs through OpenAMP messages. Currently, the tasks running on RPU are managed by a simple Polling Server on a fixed priority scheduler (Rate Monotonic). To improve the response time for aperiodic requests without degrading the processor utilization factor, implement a Sporadic Server (SS) on freeRTOS for the management of the running tasks. Demonstrate server operation by highlighting differences with server polling. The tasks must be requested via OpenAMP and the APU should check the feasibility before to run them.

15. **Analysis of KataContainers**. Install KataContainers (https://github.com/kata-containers), stripped down VMs compliant with container standards, on PREEMPT_RT patched Linux system. Make a latency analysis and a deadline miss analysis. Make a comparison wrt the FIFO tasks in a bare metal linux PREEMPT_RT distribution. Are they significantly higher?

https://wiki.linuxfoundation.org/_media/realtime/events/rt-summit2016/kvm_rik-van-riel.pdf

16. **Analysis of RunX**. Install RunX (https://github.com/lf-edge/runx), stripped down VMs compliant with container standards, on a PREEMPT_RT patched Linux system. Make a latency analysis and a deadline miss analysis. Make a comparison wrt the FIFO tasks in a bare metal linux PREEMPT_RT distribution. Are they significantly higher?

17. **Analysis of the VxWorks SDK.** (https://forums.windriver.com/t/vxworks-software-development-kit-sdk/43). WindRiver has recently released an evaluation SDK to protptype real-time systems based on VxWorks. Make a short analysis of the main API available for tasks programming and latency analysis of VxWorks RTPs on QEMU.

18. **Analysis of the OSEK implementation on ERIKA**. ERIKA enterprise is one of the few representative open source implementions of OSEK. Install the ERIKA system on Linux following guidelines (http://erika.tuxfamily.org/wiki/index.php?title=Tutorial:_Installing_ERIKA_and_RT-Druid_on_a_Linux_host) and make a latency test taking advantage of the available emo code.