U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Embedded Systems

Master in Electrical and Computers Engineering

Embedded and Real-Time Systems

Master in Informatics and Computing Engineering

Ideas for projects

(2020-04-05)

Based on RaspberryPi

1- **Install and characterize an RTOS on the RaspberryPi**
   - **Features to evaluate/test:**
     - Scheduling algorithms supported
     - Measure context switch time
     - jitter caused by OS overhead
     - Measure memory use and overhead
     - Behavior under overload conditions
     - Hardware support
     - etc…
   - **Some of the available RTOS:**
     - ChibiOS
       https://www.stevebate.net/chibios-rpi/GettingStarted.html
     - RISC OS Open
       https://www.riscosopen.org/content/downloads/raspberry-pi
     - FreeRTOS
       https://www.osrtos.com/rtos/freertos/
       https://github.com/jameswalmsley/RaspberryPi-FreeRTOS
       (RaspberryPi2 -> NOTE: Last commit seems to have been in 2017)
       https://github.com/Forty-Tw0/RaspberryPi-FreeRTOS
       (forked from https://github.com/jameswalmsley/RaspberryPi-FreeRTOS)
     - eCos
       https://www.ecoscentric.com/news/press-170314.shtml
       https://www.ecoscentric.com/ecos/ecospro.shtml
       https://www.ecoscentric.com/rpi/index.shtml
     - uC/OS
       https://www.micrium.com/rtos/
       https://easychair.org/publications/open/VPzR
       (Last commit 2018)
       https://github.com/samj3sus/Real-Pi

- **The following are not guaranteed to work...**
    - FuschiaOS

      (I have some doubts whether this is feasible, Google seem to have removed support for RaspberryPi)

      https://fuchsia.dev/
      https://fuchsia.googlesource.com/
      https://github.com/faku99/fuchsia-rpi

      (The following links seem to be to code for RaspberryPi3)

      http://www.mediafire.com/file/9zalt9f90ydfl8v/Fuchsia86x64x08262016.zip/file
      http://www.mediafire.com/download/mhgy7cyedfdp7xs/FuchsiaRaspberryPi308262016.zip

      (I don't know whether they are secure, but since they will be installed on a RaspberryPi that can be isolated from a network, it should be OK to give it a whirl)

Based on Arduino

2- Simple ukernel on Arduino
- Fully implement the ukernel shown on the guided lab classes.
- Extend the ukernel from the guided lab classes to support EDF.
- Implement a basic program/project that uses said capabilities with minimal external hardware.

3- Advanced ukernel on Arduino
- Develop a tick based ukernel for the Arduino, supporting fixed priority multi-threading, full pre-emption, with independent stacks for each thread.
  (requires some assembly programming skills. Assembly can be embedded in C code)

4- Install and characterize an RTOS on the Arduino
- **Features to evaluate/test:**
    - Scheduling algorithms supported
    - Measure context switch time
    - jitter caused by OS overhead
    - Measure memory use and overhead
    - Behavior under overload conditions
    - Hardware support
    - etc...
- **Some of the available RTOS:**
    - DuinOS
      https://code.google.com/archive/p/duinos/
      https://code.google.com/archive/p/duinos/downloads
    - RIOT-OS
      https://doc.riot-os.org/group__boards.html
    - ARTe – Arduino Real-Time extension
      https://create.arduino.cc/projecthub/104900/getting-started-with-real-time-multi-tasking-in-arte-86494b

5- Carry out a project on Arduino
- The examples are numerous, but you need to have the materials needed with you. It is enough that one group member handles the HW. The SW development can be divided among the group. The member with the HW has to integrate and test…
- Check the youtube videos of SEMB/SETR and also the Arduino project hub
- **Requirements and features to test:**
    - Describe the main functionality + HW to be used + simplified SW architecture

- The SW design should use multitasking, at least three tasks (can be ISRs)
  - Typical functions include Control, User Interface, Communication and Logging
- Measure tasks execution and response times of tasks
- Compare with analysis
- Measure jitter of periodic signals
- Measure the OS overhead (tick execution time, memory footprint…)
- Behavior under overload conditions
- etc...

Based on ESP32 / ESP8266

6- Characterize the performance of FreeRTOS

- **Features to evaluate/test:**
  (Remember these devices come with an SDK based on FreeRTOS)
  - Scheduling algorithms supported
  - Measure context switch time
  - jitter caused by OS overhead
  - Measure memory use and overhead
  - Behavior under overload conditions
  - Hardware support
  - etc...
- **Other possible RTOS that can be installed and assessed**
  - TizenRT
    https://github.com/Samsung/TizenRT
    https://github.com/Samsung/TizenRT/blob/master/build/configs/esp32_DevKitC/README.md

7- Carry out a project on an ESP

- The examples are also numerous, but it will depend on what the specific project is and the materials you have available. As I referred above, it is enough that one group member handles the HW. The SW development can be divided among the group. The member with the HW has to integrate and test…
- Check the youtube videos of SEMB/SETR and also ESP32 / ESP8266 projects online
- **Requirements and features to test:**
  - Describe the main functionality + HW to be used + simplified SW architecture
  - The SW design should use multitasking, at least three tasks (can be ISRs)
    - Typical functions include Control, User Interface, Communication and Logging
  - Measure tasks execution and response times of tasks
  - Compare with analysis
  - Measure jitter of periodic signals
  - Measure the OS overhead (tick execution time, memory footprint…)
  - Behavior under overload conditions
  - etc...

Based on a Personal Computer

8- **Build a simulator of a CPU executing tasks**
   - **Features to evaluate/test:**
     - Read the descriptions of the tasks from a file
     - Simulate the execution in time according to a scheduling policy
     - Mark/log activations, start of executions, preemptions, terminations
     - Use a graphical interface to observe the execution timeline as a Gantt chart
     - Determine automatically the worst-case response times and deadline violations
     - Log distributions of start times and response times
     - Generate configurable random task sets

9- **Build a schedulability analysis tool**
   - **Features to evaluate/test:**
     - Create a configurable random task set generator that saves in it a file
     - Create an automatic procedure to execute different schedulability analysis
     - Run the analysis for a large number of tasks sets
     - Observe and plot the results of the analysis as function of total utilization

10- **Install and characterize an RTOS on a PC**
   - **Features to evaluate/test:**
     - Scheduling algorithms supported
     - Measure context switch time
     - jitter caused by OS overhead
     - Measure memory use and overhead
     - Behavior under overload conditions
     - Hardware support
     - etc...
   - **Some of the available RTOS:**
     - Linux
       Evaluate fixed priority scheduling, with impact of RT_PREEMPT flag
       Compare fixed priorities with EDF
       Implement the SRP resource sharing method
     - FreeRTOS
       https://www.freertos.org/index.html
       (running on the PC)
       https://www.freertos.org/RTOS_ports.html (See Intel/x86 port)
       (A windows port)
       https://www.freertos.org/FreeRTOS-Windows-Simulator-Emulator-for-Visual-Studio-and-Eclipse-MingW.html
     - On Time RTOS-32
       http://www.on-time.com/rtos-32-eval.htm
     - RTX64 (Windows based RTOS)
       (seems to be free to download)
       https://www.intervalzero.com/
       https://www.intervalzero.com/rtx-downloads/rtx64-downloads/rtx64-3-0-downloads/#tab-1
     - RTAI
       https://www.rtai.org/?Documentation
     - RTEMS
       https://www.rtems.org/