Introduction to Real Time Operating System

Fabrice Muller

Polytech Nice Sophia

Fabrice.Muller@univ-cotedazur.fr

Outline

- Courses
  - Introduction
  - Scheduling
  - FreeRTOS
- Labs
  - Part 1- Theory
  - Part 2- Espressif Framework
  - Part 3- FreeRTOS Labs
    - 1- Task & Scheduling
    - 2- Message Queue & Interrupt service
    - 3- Semaphore & Mutex
    - 4- Direct task notification
    - 5- Application

Planning

2 groups for TD and TP (Lab)

- 1. CM (2h): Introduction, Scheduling
- 2. CM (2h): Scheduling
- 3. TD (2h): Theory LAB, Espressif Framework
- 4. CM (2h): FreeRTOS
- 5. TP (3h): LAB1, Task & Scheduling
- 6. CM (2h): FreeRTOS, **Evaluation**
- 7. TP (3h): LAB2, Message Queue & Interrupt service, LAB3.1/3.2, Sem & Mutex
- 8. TP (3h): LAB3.3, Sem & Mutex, LAB4, Direct task notification
- 9. TP (3h): LAB5, Application
- 10 (3h). **Lab evaluation** (1h30 per student)

Why using an Operating System ?

- Human-machine direct interaction
- Easy access all the features of the computer
  - Processors
  - Memories
  - Storage, files
  - Peripherals (screen, mouse, printer …)
  - Network …
- Hides the complexity of the operations (files, peripherals …)
- Allows access to privileged resources without having special rights (could be dangerous)

Variety of OS

- According to the context of use, an OS can be
  - Single/multi user
  - Mono/multi tasks (cooperative/preemptive scheduler)
  - Distributed (managing the resources of multiprocessor systems or multi-cores systems)
  - Embedded
  - Realtime
- Can be
  - Generalist (often based on time sharing)
  - Specialized (developed for a limited domain of applications)

Examples of OS

- Generalist
  - Windows, Linux
- Embedded
  - µCLinux, Windows CE
- Embedded & Real time
  - Amazon-FreeRTOS
  - RTLinux (Linux + path PREEMPT-TR)
- Embedded & Distributed & Real time
  - RTEMS, VxWorks
- Specialized
  - Android, iOS
  - OSEK/VDX (automotive)

Why using a Real Time Operating System ? Example

- The behavior of a computer system is called real-time when the operation is subject to the dynamic evolution of an industrial process connected to it.
- The process is controlled, managed or supervised by the system that reacts to the state changes of the process.

Industrial process to be control

Robots assembling (2)

Machine tools (1)

hydraulic regulation

climatisation

Continuous time

measurement

events

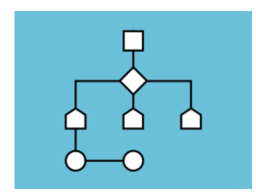
commands

Management with computer systems

sensors

actuators

Control algorithms



(1) https://www.damatomacchine.com/

(2) Robots assembling a Ford factory in Kansas. (Source: Ford)

Why using a Real Time Operating System ?

- Simple cases
  - No kernel (standalone)
  - Writing good embedded software without the use of a kernel

- Complex cases
  - Using a kernel is preferable
  - Task prioritization can help ensure an application meets its processing deadlines
  - But other less obvious benefits

Why using a Real Time Operating System ?

- Abstracting away timing information
  - Responsible for execution timing
  - Time-related API
  - Allows code to be simpler and the code size to be smaller
- Maintainability/Extensibility
  - Fewer interdependencies between modules
  - Allows the software to evolve in a controlled and predictable way
  - Application performance is less dependent of the hardware
- Modularity
  - The tasks are independent modules
  - Each of tasks has a well-defined purpose

Why using a Real Time Operating System ?

- Team development
  - Tasks have well-defined interfaces
  - Allows easier development by teams
- Easier testing
  - If tasks are well-defined independent modules with clean interfaces, they can be tested in isolation (i.e., non-regression tests)
- Portability / Code reuse
  - Application can be easily performed on different boards
  - Greater modularity and fewer interdependencies results in code that can be reused with less effort

Why using a Real Time Operating System ?

- Improved efficiency
  - Kernel allows software to be completely event-driven
  - No processing time is wasted by polling for events
  - Code executes only when there is something that must be done
  - The need to process the RTOS tick interrupt and to switch execution from one task to another
- Idle time, for what ?
  - Idle task is created automatically when the scheduler is started
  - To measure spare processing capacity
  - To perform background checks
  - To place the processor into a low-power mode

Why using a Real Time Operating System ?

- Power Management
  - RTOS allows the processor to spend more time in a low power mode
- Flexible interrupt handling
  - Interrupt handlers can be kept very short (execution, code) by deferring processing to either a task or the daemon task
- Mixed processing requirements
  - Mix of periodic, continuous and event-driven processing/tasks within an application
  - Hard & Soft real-time requirements can be met by selecting appropriate task and interrupt priorities

Real-time is NOT equal to go fast

- Examples
  - Need a short reaction time (10 ms) to control an airplane
  - Need for a shorter reaction time (1s) for an HMI (Human-Machine Interface)
  - Need for a shorter reaction time (1 min) for the control of a production line
  - Need a shorter reaction time (1h) to control a chemical reaction
  - Need a time latency of a few hours for the establishment of a weather forecast
- The important thing is to respect the deadline
- A fair result after the deadline may be unusable and considered a fault

Services

- Activity management
  - Create
  - Suspend
  - Destroy
- Time management
  - Clock, Tick
  - Timers
- Communication & Synchronization
  - Message queue
  - Shared memory
  - Semaphore, mutex, conditional variable

Services

- Notification
  - Interrupt, signal
  - Event
- Memory management
  - Virtual memory
  - Memory lock
  - Memory protection
- Inputs/Outputs
  - Open, Read, Write
  - (Asynchronous) Direct I/O
- Network management

POSIX Standard

- **P**ortable **O**perating **S**ystem **I**nterface = **POSIX**
- Standards specified by the IEEE
- Compatible with variants of the Unix operating system (although the standard can apply to any operating system)
- Portability: a compliant POSIX application should be ported on several systems
- RTOS compliant with POSIX
  - VxWorks, FreeRTOS (partially)
  - QNX, RTEMS, RTAI
  - RTLinux (partially)
  - …

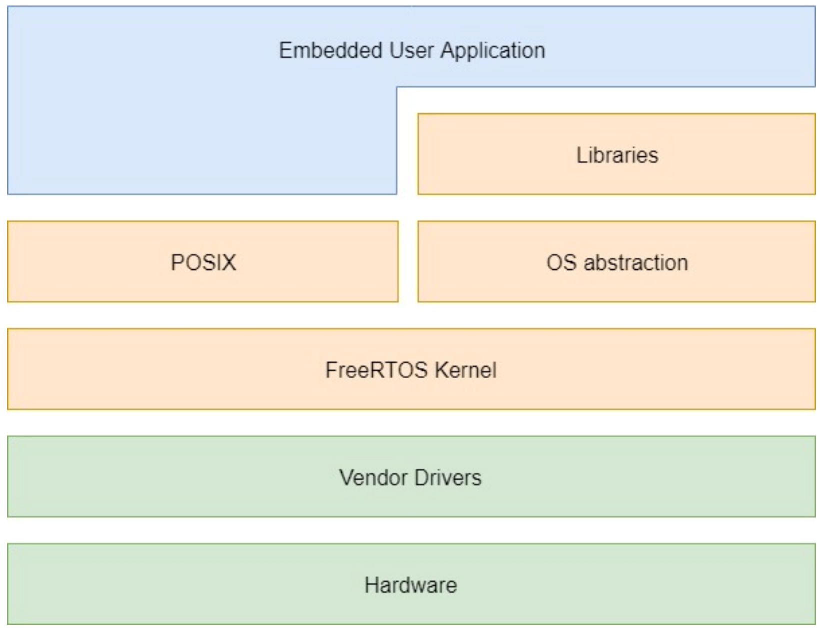
POSIX - Overview

POSIX.1 **Core Services** **IEEE Std 1003.1**	**Process Creation and Control, Signals, Timers ...**	**Define API with the RTOS**
POSIX.1b **Real-time extensions** **IEEE Std 1003.1b**	**Priority Scheduling, Real-Time Signals, Clocks and Timers, Semaphores, Message Passing …**	**Define the extensions for real-time**
POSIX.1c **Threads extensions** **IEEE Std 1003.1c**	**Thread Creation, Control, and Cleanup; Thread Scheduling …**	**Define the API for the threads management**

POSIX - FreeRTOS Example

- Implement partially the IEEE STD 1003.1-2017
- 20% of the POSIX API
- Use of a wrapper

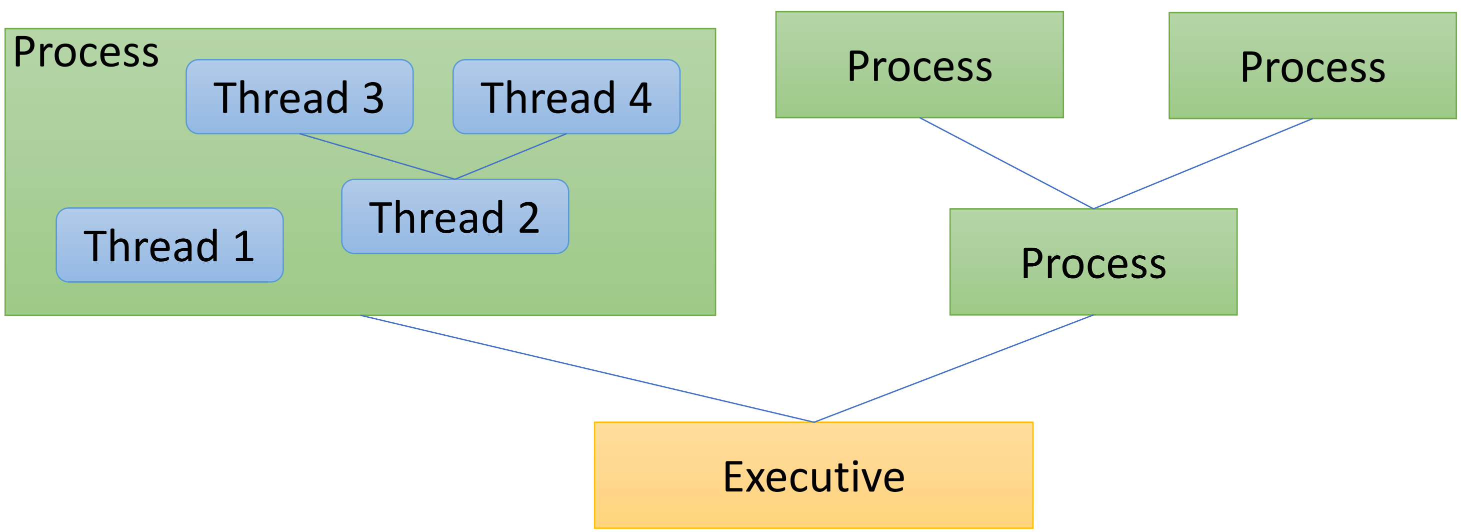
- Open source
- Royalties free



https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_POSIX/index.html

POSIX - Process & Thread

- Process
  - Entity with its own memory
- Thread
  - Internal entity of a process.
  - The threads of a process share the memory



Conclusion

- Lot of RTOS
- POSIX standard or not
- Control of scheduling is essential in the design of real-time applications
  - Sharing of the CPU between the different tasks
  - Management of dependencies between tasks (sharing of exclusive resources)
  - Management of overload situations
- In addition to functionalities of a classic OS, a RTOS allows
  - Set up the scheduling algorithms
  - Synchronize tasks
  - Communicate and exchange data between tasks