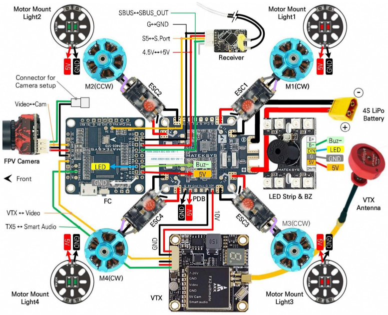
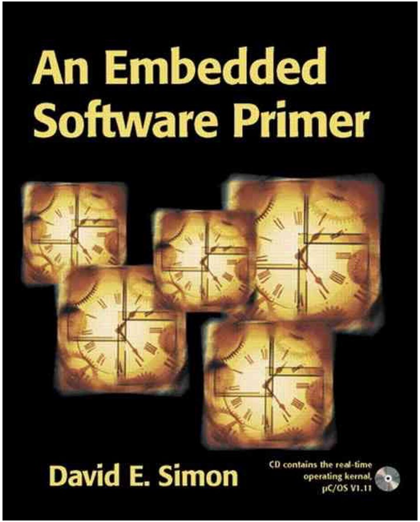


Introduction to RTOS

Kizito NKURIKIYEYEZU, Ph.D.

Readings

- Read Chap 6 of Simon, D. E. (1999). An Embedded Software Primer



[1] Readings are based on Simon, D. E. (1999). An Embedded Software Primer.

Real-time operating system

- A real-time operating system (RTOS) is a program that schedules execution in a timely manner, manages system resources, and provides appropriate developing application code[1].
- RTOS are complex software architecture needed to handle multiple tasks, coordination, communication, and interrupt handling
- Desirable RTOS properties: use less memory, application programming interface, debugging tools, support for variety of microprocessors, already-debugged network drivers
- Contiki source code, FreeRTOS, Zephyr Project[2, 3]

---

1 https://en.wikipedia.org/wiki/Real-time_operating_system

2 Wikipidia provide an extensive list of existing RTOS at https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

3 The The Zephyr Project provides a promising RTOS for IoT devices. It is designed for connected resource-constrained devices, built to be secure and safe. An interested reader can read more at https://www.zephyrproject.org/

TAB 1. Desktop Operating systems vs Real-time operating systems

Desktop OS	**Real-time OS**
At boot-time, the OS takes control and sets up environment	At boot, application takes control and tarts the RTOS
Application run under the OS and independently of each other	Applications are linked with the RTOS and are tied together
Multiuser, thus need more security and protection	Usually single user and no sacrifice security for performance
Limited configuration	Extensive configuration: allow to leave out all what you don't need, e.g. file managers, I/O drivers, utilities, and even memory management
OS and application run in different address space	Both the RTOS and applications run in the same address space. Thus, the RTOS is less protected
Require large memory	Usually use little memory
Big or Large User Interface Management	Limited No. of User Interface
Time response of OS is not deterministic	The time respond of TROS is deterministic

The need for an RTOS

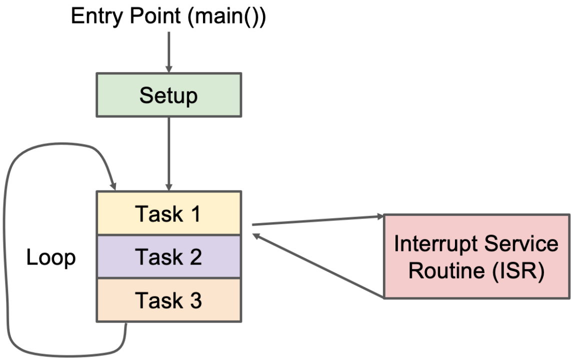


FIG 1. Architecture without an RTOS

The need for an RTOS

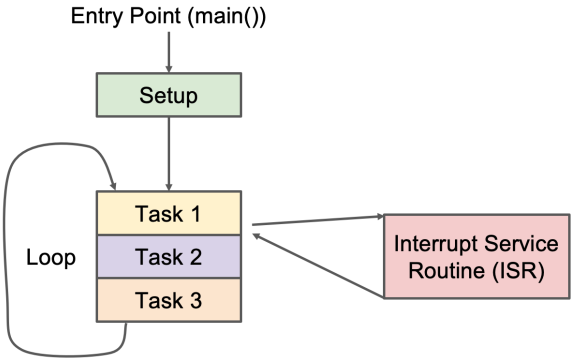


FIG 1. Architecture without an RTOS

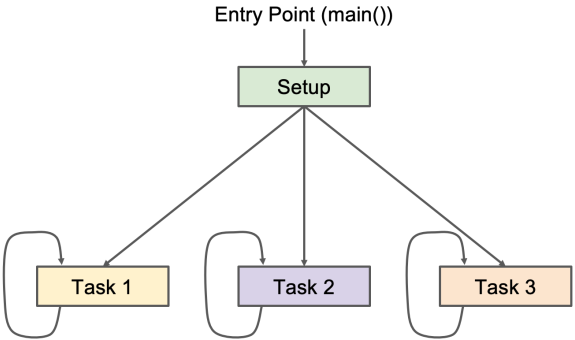


FIG 2. Architecture with an RTOS

TAB 2. Bare Metal vs. Operating Systems

	Bare metal	RTOS	Desktop OS
Good for small devices (i.e., small MCU)	✓	✓	✗
Level of application hardware control	Complete	Medium	None
Ability to multitask	None	Fair	Excellent
Overhead	None	Minimal	High
Efficient memory usage	✓	✓	✗
Community support	✗	✓	✓
Scalability and portability	✗	Medium	Excellent

[1] Bare-metal programming is a term for programming that operates without various layers of abstraction e.g. without an operating system supporting it.
[2] https://www.nabto.com/bare-metal-vs-rtos-vs-os/
[3] https://www.embedded.com/why-a-bare-metal-developer-moved-to-operating-systems/

Why use an RTOS in your project?

- Resource managment—Maximum utilization of devices and systems. Thus more output from all the resources.
- Easy coding—maintainability/extensibility, modularity, easy testing, code reuse
- Abstracting timing information—helps not worry about calculating timers
- Priority-based scheduling—automatically decide which task should be executing at any particular time
- Reduce errors—Commercial (or open source) RTOS well-debugged and have fewer bugs compared to writing your own scheduler
- Background tasks—Background tasks are performed when the system is idle. This ensures that things such as CPU load measurement, background CRC checking etc will not affect the main processing

Why use an RTOS in your project?

- Task prioritization can help ensure an application meets its processing deadlines
- Abstracting away timing information from applications
- Well-defined interfaces help in team development
- Easier testing with well-defined independent tasks
- improved efficiency with event-driven software
- Flexible interrupt handling
- Easier control over peripherals
- Power Management—allow the processor to spend more time in a low power mode.

Why NOT to use an RTOS

- simple systems—always use the simplest architecture when possible
- Limited resources—If the MCU is limited (e.g., in RAM, stack memory, processor capabilities), do not use an RTOS
- Functionality—The decision will be based on what your device will do:
  - Does the application have multiple concurrent tasks?
  - Does your application's tasks need to communicatewith each other, or to synchronise with each other?
  - Does the application include stacks such as Bluetooth, USB, WiFi, TCP/IP, etc.?
  - Will the systems time management be simplified by using an RTOS?
  - Is deterministic behavior needed?
  - Do program tasks need the ability to preempteach other?
  - Does the MCU have at least 32 kB of code space and 4 kB of RAM?

The end