EEE499 - Real-Time Embedded System Design

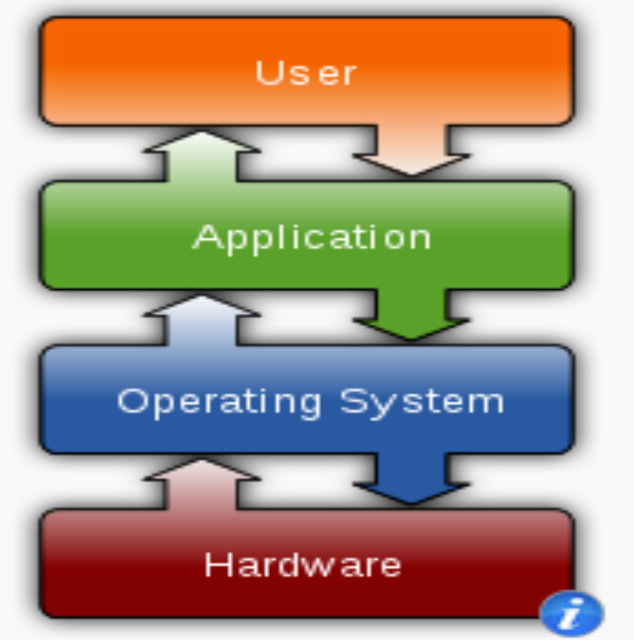
Introduction to Real-Time Operating Systems

Outline

- Operating Systems VS Real-time Operating Systems ?
- Basic Requirements of an RTOS
- Characteristics of RTOS
- Examples of RTOS

Operating Systems?

OS is set of system software that manages hardware and software resources and provides common services for other applications.



Why Operating Systems?

- Operating system provides a layer of abstraction between the users and the system
  - it hides the complexities of the system's resources from the programmer
  - Free the applications programmer from writing code for task scheduling and dispatching and etc.
  - it allows the computer to be treated as a virtual machine

Real-Time OS

- An RTOS is a class of operating systems that are designed to meet real time-applications requirement. It means it must be predictable and guarantees the timing constraints.
- RTOS usually directly deals with hardware, whereas the general purpose OS which use drivers to access the hardware.

Typical Requirements of a OS

- multi-tasking
  - single processor -> quasi-concurrent tasks
  - typical # of tasks - 32, 64, 128, 256 or unlimited
- scheduling
  - creation/deletion and scheduling policy of tasks
  - time slice/round robin (equal priority)
  - static priority versus dynamic priority
- multiprocessor support
  - more advanced features
  - non-traditional

Typical Requirements of a OS

- control of shared resources
  - mutual exclusion mechanisms
  - semaphores, monitors
- inter-task communication/synchronization
  - synchronous and asynchronous data transfer
  - mailboxes and queues
- memory management
  - minimal for diskless systems

Typical Requirements of a RTOS

- ROMable
  - embed into product
- scalable
  - conditional compilation, optimization
  - add-ons, plug-ins
- reliable
  - robust, well established, well tested
- deterministic
  - execution time of all services and functions known and published

Typical Requirements of a RTOS

- source code support
  - traditionally Assembly, C, Ada
  - today C++, Java
- target support
  - micro-controller market versus DSP versus PC
  - 60k, PowerPC, … / TI DSPs … / x86, SPARC
- tool support
  - debuggers, compilers, linkers, 3rd party
  - visibility tools (MicroC- Probe)
- TCP/IP support
  - distributed RTS support

Characteristics of RTOS

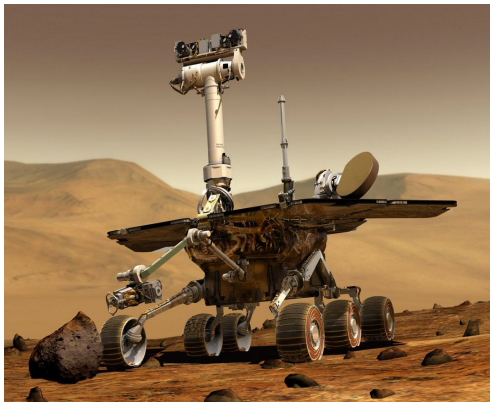
- small kernel ***footprints (this varies a lot)***
  - 2.5k - 400k ROM / 0.5k - 30k RAM
- RAM per thread / queue
  - 50/30 bytes - 1k/200 bytes
- scheduling policies
  - round-robin, fixed priority, dynamic priority
  - priority inversion support
- thread switching times
  - 10μsec - 1000 clock cycles (~350 nsec on a 2.8GHz processor)
- costs (US$)
  - $69.95 (source included) - $25,000 per seat

Advantages & Disadvantages of using a RTOS

- Advantages
  - Simplifies design
  - Facilitates application expansion (scalability)
  - Provides a set of commonly used “built-in” services
  - Deterministic (hopefully)
- Disadvantages
  - Extra overhead (2-4% is typical)
  - Cost
  - Potential increased complexity
    - There are cases where an RTOS is “over-kill”

RTOS Examples - VxWorks

- It is commercial
- 1.5 billion embedded devices use it
  - world's most widely deployed proprietary RTOS
- Supports lots of CPU architectures: ARM, PowerPC, Intel, etc
- Support for equal priority
    - Uses both priority-based preemption or round-robin scheduling
- Basic set of task communication
- No memory protection

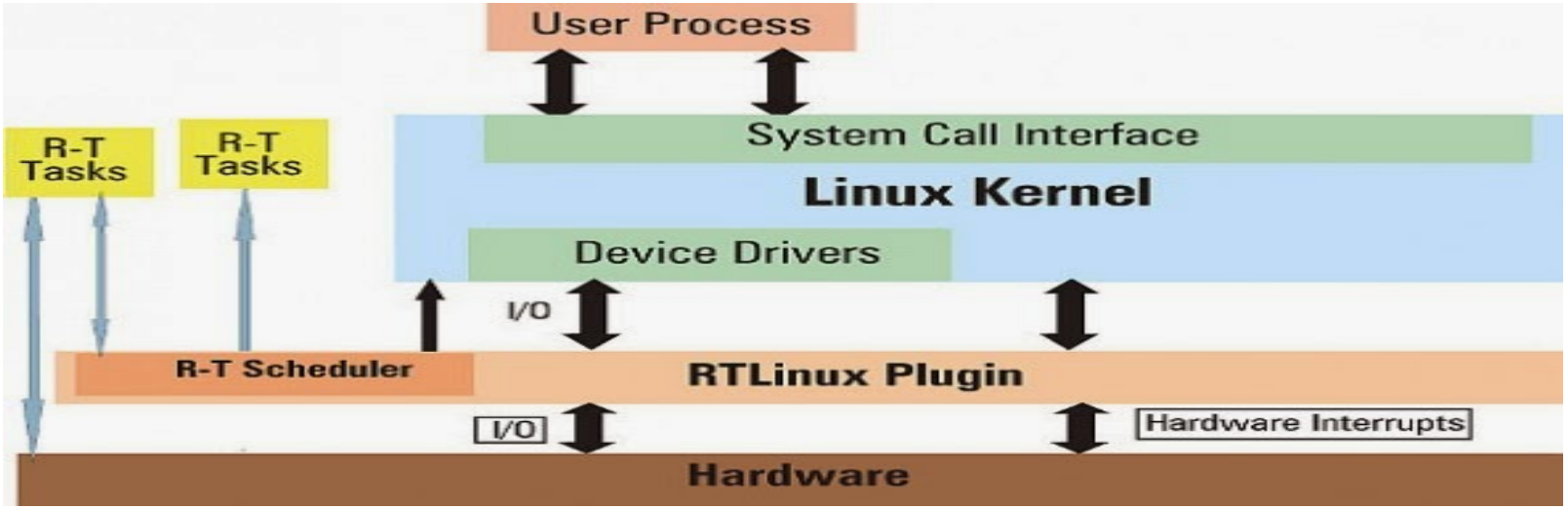


RTOS Examples - QNX

- Commercial, was acquired by BlackBerry in 2010
- Supports lots of CPU architectures: ARM, PowerPC, Intel, etc
- QNX was one of the first commercially successful microkernel operating systems used in cars and mobile phone.
- Small memory footprint
- Dynamically start & stop filesystems, network, serial, etc.
- Bootable from ROM

RTOS Examples- RTLinux

- Is Linux
- Runs on anything, even toasters
- Hardware support for anything
- Isn't technically real-time, but can be



RTOS Examples - FreeRTOS

- Leading open source RTOS
- Key features:
  - Preemptive and co-operative scheduling, Multitasking, Services, Interrupt management, MMU; Supports stacks for TCP/IP, USB, & basic file systems
- Highly portable C, 24 architectures supported, Ports are freely available in source code
- Scalable:
  - Only use the services you need by specifying in FreeRTOSConfig.h
  - Minimum footprint = 4KB

References

[1] Cooling, J.E., “Software Design for Real-time Systems”, Chapters 1 & 9.

[2] Labrosse, J.J., “MicroC/OS-II”, Chapter 2.

[3] Furht et al, “Real-Time UNIX Systems”, Chapter 2.

[4] Greenfiled J.D., “The 68HC11 Microcontroller” Chapter 3.

[5] http://www.realtime-info.be/encyc/...