Real Time and Embedded Data Systems and Computing

Real-Time Operating Systems
VxWorks in particular
What is a Real-Time Operating System (RTOS)?

- In general, an operating system (OS) is responsible for managing the hardware resources of a computer and hosting applications that run on the computer. An RTOS performs these tasks, but is also specially designed to run applications with very precise timing and a high degree of reliability. This can be especially important in measurement and automation systems where downtime is costly or a program delay could cause a safety hazard.

- To be considered "real-time", an operating system must have a known maximum time for each of the critical operations that it performs (or at least be able to guarantee that maximum most of the time). Some of these operations include OS calls and interrupt handling. Operating systems that can absolutely guarantee a maximum time for these operations are commonly referred to as "hard real-time", while operating systems that can only guarantee a maximum most of the time are referred to as "soft real-time".

The holy grail of RTOS’s: correct timing:

• If programmed correctly (!!), an RTOS can guarantee that a program will run with very consistent timing. Real-time operating systems do this by providing programmers with a high degree of control over how tasks are prioritized, and typically also allow checking to make sure that important deadlines are met. An example: the release of a car’s airbag

• In contrast to real-time operating systems, the most popular operating systems for personal computer use (such as Windows) are called general-purpose operating systems. While more in-depth technical information on how real-time operating systems differ from general-purpose operating systems is given in a section below, it is important to remember that there are advantages and disadvantages to both types of OS. Operating systems like Windows are designed to maintain user responsiveness with many programs and services running (ensuring "fairness"), while real-time operating systems are designed to run critical applications reliably and with precise timing (paying attention to the programmer's priorities).
RTOS: Important Terminology and Concepts

- **Determinism**: An application (or critical piece of an application) that runs on a hard real-time operating system is referred to as deterministic if its timing can be guaranteed within a certain margin of error.

- **Soft vs Hard Real-Time**: An OS that can absolutely guarantee a maximum time for the operations it performs is referred to as hard real-time. In contrast, an OS that can usually perform operations in a certain time is referred to as soft real-time.

- **Jitter**: The amount of error in the timing of a task over subsequent iterations of a program or loop is referred to as jitter. Real-time operating systems are optimized to provide a low amount of jitter when programmed correctly; a task will take very close to the same amount of time to execute each time it is run.
RTOS: Jitter in a control loop

Figure 1. Jitter is a measure of how much the execution time of a task differs over subsequent iterations. Real-time operating systems are optimized to minimize jitter.
RTOS: setting Priorities (1)

• When programming an application, most operating systems (of any type) allow the programmer to specify a priority for the overall application and even for different tasks within the application (threads). These priorities serve as a signal to the OS, dictating which operations the designer feels are most important. The goal is that if two or more tasks are ready to run at the same time, the OS will run the task with the higher priority.

• In practice, general-purpose operating systems do not always follow these programmed priorities strictly. Because general-purpose operating systems are optimized to run a variety of applications and processes simultaneously, they typically work to make sure that all tasks receive at least some processing time. As a result, low-priority tasks may in some cases have their priority boosted above other higher priority tasks. This ensures some amount of run-time for each task, but means that the designer's wishes are not always followed.
RTOS: setting Priorities  (2)

• In contrast to (GPOS), real-time operating systems follow the programmer's priorities much more strictly. On most real-time operating systems, if a high priority task is using 100% of the processor, no other lower priority tasks will run until the high priority task finishes. Therefore, real-time system designers must program their applications carefully with priorities in mind.

• In a typical real-time application, a designer will place time-critical code (e.g. event response or control code) in one section with a very high priority. Other less-important code such as logging to disk or network communication may be combined in a section with a lower priority.
RTOS: Interrupt Latency

• Interrupt latency is measured as the amount of time between when a device generates an interrupt and when that device is serviced. While general-purpose operating systems may take a variable amount of time to respond to a given interrupt, real-time operating systems must guarantee that all interrupts will be serviced within a certain maximum amount of time. In other words, the interrupt latency of real-time operating systems must be bounded.

• The latency comprises two deterministic components and at least two non-deterministic:

  • Deterministic:
    - Hardware switching and entering the Interrupt handler
    - Execution of handler code. While the execution time depends on the number of instructions, the time in a specific handler is constant (if it is not interrupted!)

  • Non-deterministic
    - An interrupt will not be served is i) the interrupt system is disabled (permitted by a RTOS) or ii) during servicing of an interrupt with higher hardware priority.
RTOS: Performance

- One common misconception is that real-time operating systems have better performance than other general-purpose operating systems. While real-time operating systems may provide better performance in some cases due to less multitasking between applications and services, this is not a rule. Actual application performance will depend on CPU speed, memory architecture, program characteristics, and more.

- Though real-time operating systems may or may not increase the speed of execution, they can provide much more precise and predictable timing characteristics than general-purpose operating systems.
Lecture 8
Real-Time Operating Systems

M O’Boyle

February, 2010

School of Informatics, University of Edinburgh.
Prof. Michael O’Boyle, director of Institute for Computing Systems
OS for Embedded Systems

- Embedded Systems, like traditional ones, need OS functions & services to manage h/w & s/w resources of computer

- Two ways of achieving this functionality:
  1. provide it within the programming language (e.g. some Ada implementations code this into the RTSS)
  2. provide it through a separate Real-time Operating System (RTOS)

- In both approaches, OS acts as interface between h/w below & application programs (periodic + sporadic processes) above

- Both have arguments in favour of & against
Requirements of RTOS Predictability and Control

- **Predictability** - most critically w.r.t. time
  - all services executed *within* bounded & known times & *at* controlled & known times
  - other resources, such as, files, I/O devices, etc. as well as fault management, should be predictable

- **Visibility & Control** for system components - RTOS user must be able to access & control h/w & system behaviour:
  - necessary to guarantee predictability
  - at the same time, level of abstraction for handling these should be convenient
Requirements of RTOS - Flexibility

- **Openness** - RTOS should be an *open* system
  - should define a flexible set of mechanisms without forcing a particular policy on a user
  - e.g. should allow choice of different policies for task scheduling
RTOS Functions & Services

- Functions to access & control both absolute & relative time -
  - many clock-related operations execute at highest priority & not-interruptible
  - OS timers also need to return values of a fine granularity

- Process & Thread management
  - operations to create, initialise, activate, terminate, communicate & synchronise between tasks
  - possible support for periodic & sporadic processes
  - facilities for task scheduling with a specifiable scheduling policy

- Operations for generating & handling s/w interrupts & context switches - to implement exception handling services
RTOS Functions & Services: Time

- Device management utilities must be accessible at the application-RTOS interface:
  - required to control & access sensors, timers & conventional I/O devices
  - e.g. to initiate an I/O operation, read state of a device, defining & connecting interrupt handlers
  - also facilities for attaching new real-time devices

- RTOS cannot be given the responsibility of performing main memory management transparently:
  - for control purposes, users must do it themselves
  - RTOS needs to provide suitable primitives to manage memory
Synchronisations & Communications in RTOS

- Lowest kernel functions must often be executed atomically or as locked critical sections

- For single-processor systems, indivisibility can be achieved by disabling interrupts during that function execution

- RTOS provide range of synchronisation facilities, such as, locks, semaphores, signals, messages, etc. - predictability of timing behaviour a key feature

- Different types of timeout mechanisms also provided
Additional wish-list for RTOS’s

- The possibility of building user specific RT kernels to match the wide range of processing machines for RT/Embedded, from microcontrollers and soft core processors to large distributed systems
- Quality assured
- An excellent Development Environment
- Debugging and test tools
- RTOS support of that particular computer architecture you want to use
  - See following pages
- Portability, that is, POSIX compliant
- And of course, "at the end of the day", the price is important!
Real-Time Operating Systems (RTOS)

The ARM architecture is supported by all major vendors of Real-Time Operating Systems (RTOS).

Many embedded systems require software to respond to inputs and events within a defined short period of time. Such systems can be categorised as **hard real-time**, where missing a response deadline is unacceptable (for example an anti-lock braking system), and **soft real-time**, where missing a deadline is desirable but not critical. In both types of systems, a degree of determinism is important.

RTOSes are designed to control an embedded system, and to deliver the real-time responsiveness and determinism required by the controlled device. Applications run under the control of the RTOS, and their allocated CPU time is scheduled by the RTOS kernel.

In modern systems, a RTOS consists not only of a real-time kernel, but also higher-level functions such as device management (USB, UART, Ethernet, LCD etc.), file systems, protocol stacks (CAN, TCP/IP, HTTP etc) and graphical user interfaces (GUI).

See the "RTOS vendors" tab below to see a table of [ARM Connected Community Partners](http://www.arm.com) whose RTOSes support the ARM architecture.
The ARM architecture is supported by all popular RTOS vendors in the embedded market. Below is a table showing the RTOS companies, their products, and an indication of which ARM processor families they currently support:

<table>
<thead>
<tr>
<th>Company</th>
<th>RTOS</th>
<th>Cortex-A5</th>
<th>Cortex-A7</th>
<th>Cortex-A9</th>
<th>Cortex-A15</th>
<th>Classic A</th>
<th>Classic R</th>
<th>Classic M</th>
<th>Classic ARM1</th>
<th>Classic ARM9</th>
<th>Classic ARM7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abetronic</td>
<td>OpenConRTOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>American Megatech</td>
<td>MegaTRAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMX Systems</td>
<td>CMX-RTX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eCosCentric</td>
<td>eCos</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eForce</td>
<td>µC3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enea</td>
<td>OSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eSOL</td>
<td>eT-kernel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Express Logic</td>
<td>ThreadX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FreeRTOS.org</td>
<td>FreeRTOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Green Hills Software</td>
<td>INTEGRITY, VelOSity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Huson Inc</td>
<td>IdenIOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hytron Electronic</td>
<td>EmbeddedIOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IAR Systems</td>
<td>PowerPAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KADAK</td>
<td>ARM RTOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Krell</td>
<td>Krell RL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LynxWorks</td>
<td>LynOS, Blue Cat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mentor Graphics</td>
<td>Nucleus OS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microm</td>
<td>µCOS-III</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Micro Digital Inc</td>
<td>SMX RTOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenSynergy GmbH</td>
<td>CoOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overo Systems</td>
<td>RTXC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quantum Leaps</td>
<td>QP-nano</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radsoft</td>
<td>Middleware OS-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RISC OS Ltd</td>
<td>RISC OS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rovebots</td>
<td>Unicorn v4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rt-labs AB</td>
<td>rt-kernel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCIOPTRA</td>
<td>SCIOPTRA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sagner</td>
<td>EmbOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semihalf</td>
<td>FreeBSD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSGO</td>
<td>PixOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wind River Systems</td>
<td>VxWorks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wittenstein</td>
<td>OpenRTOS, SafeRTOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Real-Time OS for the ARM architecture
The NIOS II soft core embedded processor

• **Nios II** is a 32-bit embedded-processor architecture designed specifically for the Altera family of FPGAs

• Nios II is comparable to MicroBlaze, a competing softcore CPU for the Xilinx family of FPGA

• Key features:
  – the Nios II architecture is a RISC soft-core architecture which is implemented entirely in the programmable logic and memory blocks of Altera FPGAs. The soft-core nature of the Nios II processor lets the system designer specify and generate a custom Nios II core, tailored for his or her specific application requirements
  – By using custom instructions, the system designers can fine-tune the system hardware to meet performance goals and also the designer can easily handle the instruction as a macro in C/C++
  – Introduced with Quartus 8.0, the optional MMU enables Nios II to run operating systems which require hardware-based paging and protection, such as the Linux kernel. Without an MMU, Nios is restricted to operating systems which use a simplified protection and virtual memory-model: e.g., μClinux and FreeRTOS

### Embedded Operating System Support

Altera’s embedded software partners provide an array of operating systems for use with the Nios II processor. Table 1 shows the operating system support available for the Nios II processor.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Supplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>eCos</td>
<td>eCosCentric</td>
</tr>
<tr>
<td>eCos</td>
<td>Zylin</td>
</tr>
<tr>
<td>embOS</td>
<td>Segger</td>
</tr>
<tr>
<td>Erika Enterprise</td>
<td>Evidence</td>
</tr>
<tr>
<td>Euros RTOS</td>
<td>Euros</td>
</tr>
<tr>
<td>Linux</td>
<td>Timesys</td>
</tr>
<tr>
<td>Linux</td>
<td>Wind River</td>
</tr>
<tr>
<td>Linux</td>
<td>SLS</td>
</tr>
<tr>
<td>Linux</td>
<td>CodeSourcery</td>
</tr>
<tr>
<td>Linux</td>
<td>Open Source Community</td>
</tr>
<tr>
<td>MicroC/OS-II (f)</td>
<td>Micrium</td>
</tr>
<tr>
<td>osCAN (z)</td>
<td>Vector</td>
</tr>
<tr>
<td>ThreadX</td>
<td>Express Logic</td>
</tr>
<tr>
<td>uCLinux</td>
<td>SLS</td>
</tr>
<tr>
<td>uCLinux</td>
<td>Open Source Community</td>
</tr>
</tbody>
</table>
Choice of OS for the Real-Time lab exercises

• Why use VxWorks
  – VxWorks from Wind River is an industry leader in Real-Time systems
  – It offers a very wide choice of software components (system calls)
  – Based on host – target configuration. The development and cross compilation are done on a host computer, typically a PC. The executable code is downloaded to the target processor and linked with the target resident (mini)kernel
    • The Eclipse Platform is an open and extensible platform.
  – However, VxWorks is a rather expensive solution, however, free as a University program
  – Very good tech support in my experience
  – For more info on Wind River products, see http://www.windriver.com/
  – However, nobody is perfect, see next two pages page

• VxWorks Application Programmers Guide
  – Selected sections presented in the following
    • The VxWorks-6.2_Application_Programmers Guide can be downloaded from FYS 4220 home page
  – However, the perfect system does not exist, see following pages
Flaws Uncovered in Popular RTOS

Recently presented at the Security B-Sides and DEFCON conferences in Las Vegas, two critical vulnerabilities have been discovered in vxWorks, Wind River’s popular embedded OS that is used in tens of thousands of designs for “smart devices” from organizations including Cisco, Apple and even NASA.

As reported in SC Magazine, one of the vulnerabilities allows hackers to leverage the RTOS’s embedded debugging services to take (unauthorized) control of the device.

VxWorks has a service enabled by default that provides read or write access to a device’s memory and allows functions to be called.... The vulnerable service, called WDE agent, is a “debugger” for the VxWorks operating system that is used to diagnose problems and ensure code is working properly when a product is being developed. [...] 

These two bugs are "just the tip of the iceberg," Moore wrote in a blog post on Monday, August 2nd, 2010
Android security flaws uncovered

By Eic Brown
2010-11-04

Covently found 88 "high-risk" defects in Android 2.2's kernel, but noted that Android's defect density is lower than the industry average for mobile OSes. Meanwhile, Lookout Mobile Security announced Lookout Premium for Android, featuring advanced security and privacy features, says eWEEK.

Software integrity firm Coventry discovered 399 security defects, including 88 high-risk defects, in the source code for the Android 2.2 kernel. Coventy tested only the kernel used in HTC's Droid Incredible (pictured), sold by Verizon Wireless. However, the tests provide a good indication of the general state of Android security, says the company.

"There are many more vendors than Google and HTC that contributed code into the kernel," stated Coventy co-founder Andy Chou. Coventy did not publicly list specific defects, but informed HTC of the details.

Cited issues in the Android kernel, which is a custom version of the Linux kernel, include problems such as memory corruptions, NULL pointer dereferences, and resource leaks. All these could potentially lead to security vulnerabilities or system crashes, said the company.

The company notes, however, that the Android kernel has a defect density of 6.47 defects per 1,000 lines of code, said to be better than the industry average for similar operating systems of one defect per 1,000 lines of code.

"The Coventry Scan results for the Android kernel we tested show a better than average defect density, meaning this specific kernel is shipping with fewer defects than the industry average for software of this size," stated Chou. "However, a significant number of these defects are the high risk types that our customers typically fix before shipping their products to market."

The finding was released as part of the "2010 Coventry Scan Open Source Integrity Report," which was originally initiated between Coventry and the U.S. Department of Homeland Security in 2005. The study includes analysis of more than 60 million lines of code from 291 widely used open source projects, including Firefox and Apache. All told, some 15,278 defects were found in these open source kernels, says Coventry.

"We are hoping that this report will shed some light on this issue and show that ultimately, for consumers, defects are defects, no matter where the code comes from," Chou said.
Note! This release of VxWorks runs under Workbench Development platform. An earlier release of VxWorks is also installed in the FYS4220 lab, and runs under the Tornado Development platform.