Overview of RTOS and VxWorks

Session Speaker

Deepak V.
Session Objectives

- To understand the necessity of embedded OS
- To understand the issues related to real time systems
- To study the important features of RTOS’s
- To know the reason for using the Cross-Platform Development Environment
- To study the multiprocessor support for RTOS
- To study the POSIX compliance of the RTOS
- To define fault tolerance of a real time system
- To study the VxWorks and its Features
- To understand the Tornado Development tool
Session Topics

- Embedded systems
- Languages for Programming Embedded Systems
- Issues related to RTOS
- Scheduling and Interrupts
- Cross development environment
- POSIX
- Multiprocessor support
- Fault tolerance
- Introduction to VxWorks and its features, applications
- Tornado tool introduction
What is an Embedded System?

- Most embedded systems perform specific tasks. The simplest embedded system contains input and output capability as well as control logic stored in system firmware.

- Embedded systems are everywhere in our lives, from mobile phones to medical equipment, including air navigation systems, automated bank tellers, MP3 players, printers, cars, and a slew of other devices about which we are often unaware.

- Embedded system: “A microprocessor based system that does not look like a computer”

- Embedded systems (ES) = information processing systems embedded into a larger product.

**Embedded system**

<table>
<thead>
<tr>
<th>Single/simple task</th>
<th>Multiple/complex task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directly implemented as hardwired</td>
<td>Implemented using OS which is Embedded OS</td>
</tr>
<tr>
<td>Eg. Traffic light</td>
<td>Eg. Mobile phone</td>
</tr>
</tbody>
</table>

© M.S Ramaiah School of Advanced Studies - Bengaluru
Application Areas
1. Mobile Phones and Base Stations

- Multiprocessor
  - 8-bit/32-bit for UI; DSP for signals
  - 32-bit in IR port; 32-bit in Bluetooth
- 8-100 MB of memory
- All custom chips

- Massive signal processing
  - Several processing tasks per connected call
- Based on DSPs
  - Standard or custom
  - 100s of processors
2. Sewing Machine

• User interface
  – Embroidery patterns
  – Touch-screen control

• “Smart”
  – Sets pressure of foot depending on task
  – Raise foot when stopped

• New functions added by upgrading the software
3. Cars

- Multiple processors
  - Up to 100
  - Networked together

- Large diversity in processor types:
  - 8-bit – door locks, lights, etc.
  - 16-bit – most functions
  - 32-bit – engine control, airbags

- Multiple networks
  - Body, engine, telematics, media, safety

- Functions by embedded processing:
  - ABS: Anti-lock braking systems
  - ESP: Electronic stability control
  - Airbags
  - Efficient automatic gearboxes
  - Theft prevention with smart keys
  - Blind-angle alert systems
  - ... etc ...
4. Extremely Large

- Functions requiring computers:
  - Radar
  - Weapons
  - Damage control
  - Navigation
  - basically everything

- Computers:
  - Large servers
  - 1000s of processors
5. Inside Your PC

- Custom processors
  - Graphics, sound
- 32-bit processors
  - IR, Bluetooth
  - Network, WLAN
  - Harddisk
  - RAID controllers
- 8-bit processors
  - USB
  - Keyboard, mouse
Challenges for Embedded Systems

- **Limited OS support for programming:** OS is part of application code and it closely co-ordinate with the OS to support a majority of the features that a desktop OS may provide.
- **Limited secondary memory:** many embedded systems depend on ROM/FLASH memory instead of secondary memory devices.
- **Limited RAM:** No swapping. Virtual memory. While programming the embedded system we must be very careful about memory leaks because these programs tends to run forever.
- **Limited processing power:** choose microprocessor that clock 10-100 MHZ or microcontroller with less powerful configuration.
- **Interaction with hardware:** embedded programmer cannot afford hardware independence since his code directly interact with the underlying hardware.
- **Absence of standard input/output devices:** The programmer have no direct way of knowing what is happening within the system.
Soft Parameters of Embedded System

- **Cost** – requires the designer to be extremely conscious about memory, peripherals etc
- **Reliability** – may require the designer to opt for some level of redundancy
- **Lifetime** – longer lifetime product must be built with robust and proven components
- **Power consumption** – should be as minimum as possible. Some of part of the system can be shut down whenever not required. Power saving features must be implemented in programming platform (peripherals and processors)
Languages for Programming Embedded Systems

- Lingua franca: assembly language use till recently
- C, C++, java with its new avatar J2ME, Ada (45% in C)
- Tools to model software (UML, SDL) – indicate the maturity of embedded software programming
- C is very close to assembly programming and it allows very easy access to underlying hardware. A huge number of high quality and debugging tools are available for C.
- C++ compilers are buggy due to huge size of the language which may create a buggy executable in some situation
- Some features of C++ cause a lot of code to bloat
- There is ongoing effort to identify a subset of C++ that can be used in embedded system called as embedded C++
What is an Embedded OS?

- An **embedded OS** is an operating system which runs on any embedded platform.
- An Embedded OS can be defined as an OS with a small footprint.
- Embedded platforms are generally required to function without human intervention.
- A typical embedded system consists of a **single-board microcomputer with an OS** and some software loaded in **ROM**.
- **Embedded applications** start running some special purpose program as soon as it is turned on and will not stop until it is turned off (if ever).
- It will not usually have any of the normal peripherals such as a keyboard, monitor, serial connections, mass storage, etc. or any kind of user interface software unless these are required by the overall system of which it is a part.
- Often an embedded OS must provide **real-time response** to perform its requirements.
- Linux running as embedded OS is **embedded Linux**.
What makes a Good Embedded OS?

- Modular
- Scalable
- Configurable
- Small footprint
- CPU support
- Device drivers
- etc, etc, etc...

Generation of embedded OS – need cross development environment
Defining a Real-time System

- A real-time system is a system whose specification includes both logical and temporal correctness requirements
  - **Logical**: produce correct outputs
  - **temporal**: produce correct outputs at right time
- Any system where the timely response by the computer to external stimuli is vital
- **Hard real time systems**
  - Guaranteed execution of tasks
  - Absolute deadline
  - Catastrophic failure if a deadline is missed
  - Entire system failure resulting serious harm
  - System design is based on strict policies and rules
- **Soft real time system**
  - Execution of tasks may not be guaranteed
  - Approximate deadline
  - Nothing catastrophic happens if a deadline is missed
  - Repeated miss of deadlines results in gradual degradation of system performance
What Makes OS that are Real-time so Important?

Nuclear Plant Example

• Lets say there is a power plant
• And let's say that the temperature is rising
• And that there is 10 seconds to cool it down before it is too late
• But…..

Source: Stephen Ferzetti
Nuclear Plant

• Since the plant was running its anti-virus program at the time it missed its window

• Meltdown

Source: Stephen Ferzetti
The main reasons for having a RTOS

- Meeting deadlines
- Being able to break out of lower priority processes so that more important tasks can be accomplished
- When you break out of these processes, the data collected remains intact
Issues Related to Real-Time Systems

• Architecture
  – Processor architecture
  – Network architecture
  – Architectures for clock synchronization
  – Fault tolerance and reliability evaluation

• Operating system

• Programming languages: Ada 95, Esterel, PEARL (Process and Experiment Automation Real-time Language), Java, C, C++

• Databases: Databases are a structured and convenient way to manage the sharing of large quantities of data among multiple tasks

• Performance measures: Conventional measures like throughput and reliability are not sufficient for real time systems
Operating System Issues

- Task assignment and scheduling
- Communication protocols
- Failure management and recovery
- A real-time operating system focus on deterministic response and data/program corruption prevention
- Scalability, scheduling and support for embedded system are required
- RTOS must support
  - multi-threading/multitasking with multiple thread/task priority
  - Preemption
  - predictable thread / task synchronization mechanisms
  - scheduling must support priority inheritance
What makes an OS an RTOS

- Pre-emptive Kernel
- Short Interrupt Latency
- Short Dispatch Latency (Fast Context Switch)
- Proper Scheduling algorithms
- Control of Memory Management
  - Task, thread or process are OS object which needs some memory space for the object definition
  - more complex the object, the more attributes it will have and bigger the definition space will be
  - If MMU exists, mapping tables are extra attributes for the task which require more memory
- Fine Granularity Time Services
- Rich Set of API's
  - More system calls, more complex they are, the fewer lines of codes the application will have
  - Code executed in the OS is certainly more efficiently executed (and better debugged) in the OS than the same code in the application
- Small Size
Scheduling

- If more than one thread wants to use the processor simultaneously, an algorithm is needed to decide which thread will run first.
- For hard real time response it is essential to make the system predictable under all circumstances.
- Deadline-driven scheduling mechanism would be ideal. (current state of technology does not allow this). Pre-emptive priority scheduling offers a substitute.
- Pre-emption - used all the time to ensure that a high priority event can be dealt with before any other lower priority event. This should be supported with interrupt based preemptive scheduling.
- Good RTS Design:
  - The list is organized taking into account the thread priorities
  - The new element added to the list (task or thread) is always put in a proper place
  - when a dispatch occurs, the first thread in the list can be taken
Interrupts

- Each OS needs to disable the interrupts from time to time to execute critical code that should not be interrupted.
- Number of lines of code executed should be limited to a minimum to have minimum interrupt latency.
- Should be bound under all circumstances.
- Problems with Interrupts
  - Possibility of blocking tasks with blocking system calls.
  - Using stack of task/process which lead to stack overflow.
  - Generating errors during task execution, which are difficult to diagnose.
Features of RTOS’

- **Predictability**: Meet its timing constraints
- **Reliability**: Probability that the system will not undergo failure over any part of a prescribed interval
- **Fault tolerance**: The failure rates of a real time system must be extraordinarily small, since failure may lead to loss of life. Be able to continue operate despite the failure of a limited subset of their software or hardware
- **Availability**: Is the fraction of time for which the system is up
- **Throughput**: Average number of instructions per unit time the system can process
- **High degree of concurrency**: Support multitasking
- **Responsiveness**: Close connections to real world entities
- **Distribution**: Processing is often distributed across several processors
Cross Development

- Cross-development is defined as a development paradigm where the applications are *developed and debugged on a host* other than the target where they are deployed.
- Cross development is required in situations where targets do not have the necessary *development resources*.
- The building and debugging of the applications is conducted on a host, connected to a target via a network or serial port.

**Diagram:**
- Cross-platform development environment
- Host
- Target
- Bootloader
- Kernel
- Root Filesystem
- Serial/Ethernet
Cross-Platform Development Environment

• A cross-compiler runs on a processor but generates executable code for a different processor. Eg ARM cross-compiler (arm-linux-gcc) running on an x86 processor generates code for an ARM processor and PowerPC cross-compiler (powerpc-linux-gcc) generates codes for a PowerPC

• Why cross-compiler? – Typically embedded systems don’t have the RAM or storage resources to compile their own executables. So a host processor can cross-compile code to create an executable which then transferred and executed on the target board

• Eg. BlueCat Linux / Platform creation suite is a cross development product. It allows for software development on a host and provides the necessary tools for transferring software on the target board
POSIX

- An acronym for Portable Operating System Interface
- POSIX is a family of standards developed by the Portable Applications Standards Committee (PASC) of the IEEE Computer Society
- POSIX is a set of books specifying APIs
- It is neither a piece of code nor an operating system
- What is an API?
  - Application Program Interface
  - A written contract between system developers and application developers
  - It is not a piece of code, it is a piece of paper defining what the two sets of developers are guaranteed to receive and are in turn responsible for providing
Motivation for the Profiles Standard

• The POSIX 1003.1 Standard:
  – Allows writing portable real-time applications
  – Very large: inappropriate for embedded real-time systems

• POSIX 1003.13:

• Defines four real-time system subsets (profiles)
  – Minimal: Small embedded systems
  – Controller: Industrial controllers
  – Dedicated: Large embedded systems
  – Multi-Purpose: Large general-purpose systems with real-time requirements

• C and Ada language options
Real-time System Subsets

- **Minimal**: Small embedded systems
  - Platform: Small embedded system
  - No MMU, no disk, no terminal
  - Model: controller of a “Toaster”

- **Controller**: Industrial controllers
  - Platform: Special purpose controller
  - No MMU, but with a disk containing a simplified file system
  - Model: industrial robot controller
Real-time System Subsets

• **Dedicated:** Large embedded systems
  – Platform: Large embedded system with file system on disk
  – an MMU
  – software is complex and requires memory protection and network communications
  – Models: avionics controller, cellular phone cell node

• **Multi-Purpose:** Large general-purpose systems with real-time requirements
  – Platform: Large real-time system with all the features
  – including a development environment
  – network communications
  – file system on disk
  – terminal and graphical user interfaces, etc.
  – Model: workstation with real-time requirements:
    • air traffic control systems
    • telemetry systems for Formula One racing cars
POSIX 1003.13 Profiles

- Networking
- Asynchronous I/O
- Multiple Processes
- Shell & Utilities
- Multiple Users
- Full File System
- Others

PSE52
- Simple File System
- Message Queues
- Tracing

PSE51
- Core

PSE53
- Minimal
- Controller

PSE54
- Dedicated
Multiprocessor Support

• A multiprocessor operating system is typically large and complex.
• Its maintainability, expandability, adaptability and portability strongly depend on its internal structure.
• Operating system *kernel* is the basic operating system functionality permitting use of the processors, the main memory, the interconnection network and the other devices of the parallel machine.
Fault Tolerant System

• Definition: Ability of a system to continue operating despite the failure of a limited subset of its hardware and software

• Failure rates of real time systems must be very small; smaller than the failure rates of components from which they are built
Performance Characteristics

- Performance of real time systems must be gracefully degradable - as the size of faulty set increases, the system must not suddenly collapse but continue executing part of its work load.
- As the extent of failure increases, the operating system must begin shedding the less critical tasks first and still be able to carryout the critical core tasks.
- The goal of the system designer is to ensure that probability of system failure is acceptably small.
Performance Degradation of a Fault Tolerant System

Catastrophic failure region

Failure Extent
VxWorks
Introduction to VxWorks

• VxWorks is the high performance real time operating system (RTOS) unlike other operating systems like Windows, Linux etc.
• Most widely used RTOS is the VxWorks
• Here the operating system performs the proceeding in the real time instead of latter time with storing it in some memory
• Development environment for this OS is Tornado, Wind River Systems Inc, Alameda, CA, USA
• The main advantage of RTOS is less context switching time, accuracy and the predictable response of the system
• VxWorks is generally compliant with the IEEE’s POSIX (Portable Operating System Interface)
• VxWorks requires a host workstation for the program development

• Unlike systems such as UNIX and QNX, VxWorks development is done on a “host” machine running UNIX or Windows, crossing-compiling target software to run on various target CPU architectures

• It includes integrated networking facilities and a complete software development environment for Windows and UNIX hosts

• It is the popular multitasking and single user operating system and the RTOS is response even for external events
Layered Structured

<table>
<thead>
<tr>
<th>Real-Time Embedded Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics</td>
</tr>
<tr>
<td>Java Support</td>
</tr>
</tbody>
</table>

WindNet Networking

Core OS: Wind Microkernel

The diagram shows the different layers of the system
Features of VxWorks

- A fast, multitasking kernel with pre-emptive scheduling and fast interrupt response
- It has extensive intertask communications and synchronization facilities
- Efficient UNIX-compatible memory management
- Various Multiprocessor facilities
- A shell for user interface
- Symbolic and source level debugging capabilities
- Performance monitoring and an I/O file system
Applications of Vxworks

Some usage of RTOS

- Flight simulators
- Radio and optical telescopes
- Navigation systems
- Deep sea instrumentation
- Traffic control systems
- Modems

... any systems where rigid time requirement have been placed on the operation of a processor or the flow of the data.

- Printers
- Digital Cameras
- Hand-held Computing devices
- Routers, Switches and other Network devices
TORNADO
What is TORNADO?

- **Tornado** is an integrated environment for software cross-development
- It can provide an efficient way to develop real-time and embedded applications with minimal intrusion on the target system
- Tornado comprises:
  - VxWorks, a high-performance real-time operating system
  - Application-building tools (compilers and associated programs)
  - An integrated development environment (IDE) that facilitates managing and building projects, establishing and managing host-target communication, and running, debugging, and monitoring VxWorks applications
Key Features of the Tornado IDE

• An integrated source-code editor
• A project management facility
• Integrated C and C++ compilers and **make**
• The browser, a collection of visualization aids to monitor the target system
• **CrossWind**, a graphically enhanced source-level debugger
• **WindSh**, a C-language command shell that controls the target
• An integrated version of the VxWorks target simulator, **VxSim**
• Customization options for many features, including integration of alternate editors and configuration management tools, as well as the entire Tornado GUI itself
Tornado Development Environment

• The tornado environment is designed to provide this full range of features regardless of whether the target is resource-rich or resource-constrained

• Tornado facilities execute primarily on a host system, with shared access to a host-based dynamic linker and symbol table for a remote target system

• The following figures above illustrates the relationships between the principal interactive host components of Tornado and the target system. Communication between the host tools and VxWorks is mediated by the target server and target agent
Summary

• Most embedded systems perform specific tasks. The simplest embedded system contains input and output capability as well as control logic stored in system firmware.

• A real-time system is a system whose specification includes both logical and temporal correctness requirements.

• Pre-emptive kernel, short interrupt latency, short dispatch latency and proper Scheduling algorithms are some of the requirements of a good RTOS.

• Each OS needs to disable the interrupts from time to time to execute critical code that should not be interrupted.

• Cross-development is defined as a development paradigm where the applications are developed and debugged on a host other than the target where they are deployed.
Summary

• Operating system *kernel* is the basic OS functionality permitting use of the processors, the main memory, the interconnection network and the other devices of the parallel machine.
• The ability of a system to continue operating, despite the failure of a limited subset of its hardware and software is known as Fault Tolerant system.
• Vxworks is the **real-time operating systems** (RTOS) unlike other operating systems like Windows, Linux etc.
• Most widely used RTOS is the VxWorks and the development environment for this OS is **Tornado**.
• It performs the proceeding in the real time instead of latter time with storing it in some memory.
• Tornado is an integrated environment for software cross-development.