Real Time Group LTD

Real Time & Embedded Linux Solutions

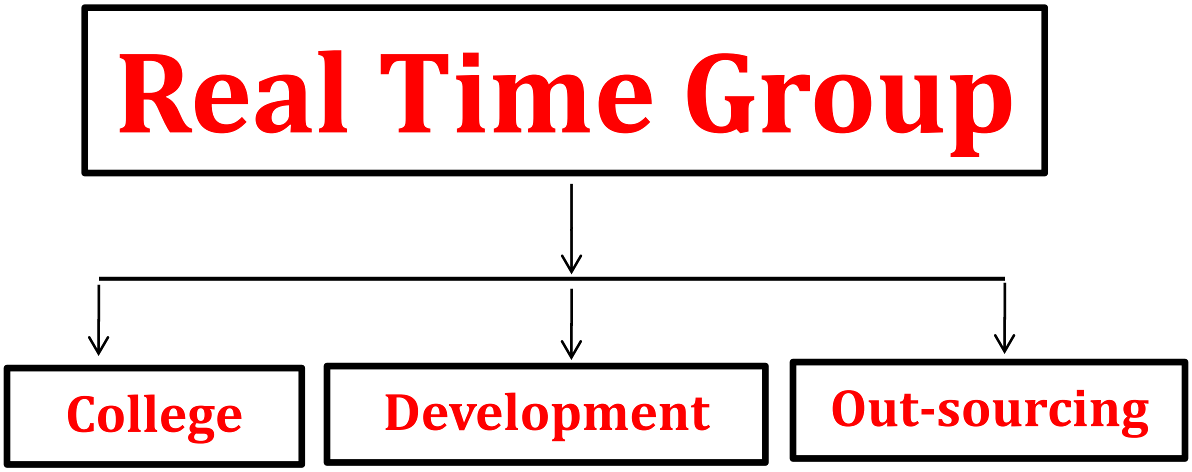
Linux Kernel Device Drivers

Duration: 90 Hours
Hands-On-Training: 75%

Real Time Group is a multi-disciplinary dynamic and innovative Real-Time O.S. and Embedded Software Solutions Center, established in 2007.

Providing Bare-Metal and Embedded Linux solutions, professional services and consulting, end-to-end flexible system infrastructure, outsourcing, integration and training services for Hardware, Software and RT-OS \ Embedded Systems.

The company is divided into the following three Divisions:



rt-ed.co.il rt-dev.com rt-hr.co.il

Training Division:

Professional Training Services for Hardware, Software, RT-OS and Embedded systems industries.

We provide the knowledge and experience needed to enable professional engineers to Develop, Integrate and QA Hardware, Software and Networking Projects.

In order to insure experience, all courses are practical – hands-on-training. The latest Development, QA and Automation equipment which are adopted by the industry are used.

All students are supplied with Development-Boards for home-work and course projects.

Course Overview:

The Linux Kernel and Device Drivers targets engineers who wish to develop Linux based Kernel modules and device drivers.
It's followed by dozens of class and home hands-on exercises and practices.

The course starts by introducing the development tools \ environment, essentials of kernel development: kernel architecture, the main APIs, integration of device drivers with other parts of the kernel and with user applications, it goes through different device driver types starting from Character and Block devices and ending with PCIe drivers.

** An ARM Evaluation card is used for class demonstrations and homework exercises.

Who should attend:

- HW C programmers who need to program on Linux operating systems.
- Embedded Linux Developers.
- Linux system programmers \ Engineers.

Prerequisite:

- Knowledge of Linux administration (Linux Fundamentals).
- Knowledge in C programming language.

Linux Kernel and Device Drivers (40 AH)

1. **Main characteristics**

   (a) Getting the sources (Class Exercise-1)
   (b) Source Organization
   (c) Where is everything?
   (d) Browsing the Kernel source codes(cscope, Kscope, Linux Cross Reference (LXR))
   (e) IDE's used for developing the Kernel source codes .

2. **Linux Basic data structures**

   (a) Doubly Linked Lists .
   (b) RB-trees structure
   (c) Radix tree

3. **Linux Kernel overview**

   (a) Why use Linux Kernel.
   (b) Linux Kernel features and constraints.
   (c) Supported Architectures.
   (d) Embedded Linux Distributions.
   (e) Linux File-System structure.
   (f) Kernel Version management.
   (g) What's new in 4.x.
   (h) Kernel development process
   (i) System calls.
   (j) Kernel space and user space interface

   - /proc
   - /sys

4. **Kernel configuration (Class Exercise-2)**

   (a) Utilities used to configure the kernel
   (b) The output configuration file - .config
   (c) How does the kernel know which packages to build.

5. **Compiling and loading the kernel (Class Exercise-3)**
   (a) Using the Native gcc
   (b) The architecture dependent cross compiler for embedded devices.
   (c) Files created during the Kernel compilation process .
   (d) booting parameters passed by the boot-loader to the kernel.
   (e) Loading the Kernel using TFTP
   (f) Loading the rootFS from NFS (Network Files System)

6. **Scheduling in Linux**
   (a) Process Scheduling in Linux
   (b) Processes, Threads and Tasks
   (c) Linux scheduling policies,
   (d) Scheduler Classes
   (e) RT ver Normal policies.
   (f) The O(1) scheduler.
   (g) The CFS scheduler
   (h) Nice command effecting CFS

5. **Developing Kernel Modules (Class Exercise-4)**
   (a) What are Kernel Modules
   (b) Compiling a module
   (c) Kernel loggers
   (d) Passing arguments to a Modules

6. **Character Device Drivers as LKMs (Class Exercise-5)**
   (a) Simple Char Driver
   (b) Passing parameters to the module
   (c) Exporting symbols
   (d) File Operations

7. **Kernel object model**
   (a) Kobjects,
   (b) Ktypes,
   (c) Ksets,
   (d) Attributes,
   (e) Kernel Events

8. **Sysfs Class Exercise-6)**
    - (a) Devices,
    - (b) buses,
    - (c) drivers,
    - (d) subsystems

9. **Udev – unified device model**
    - (a) What is it used for
    - (b) Uploading messages from kernel to user mode
    - (c) Calling the right driver

10. **Memory management (Class Exercise-7)**
    - (a) Linux Memory model
    - (b) Physical and virtual Memory
    - (c) What is the MMU
    - (d) Different Memory exceptions (page faults)
    - (e) kernel and user space addressing
    - (f) GFP Memory Allocation – kmalloc() (Atomic \ Kernel)
    - (g) Working with page frames
    - (h) Working with vmalloc()
    - (i) Contiguous virtual memory Allocation
    - (j) Different allocators (SLAB \SLOB\SLUB)
    - (k) Memory Caches
    - (l) User space memory access

11. **Platform Devices Drivers**

    - (a) Platform drivers
    - (b) Platform devices
    - (c) Resources and platform data
    - (d) Devices, drivers, and bus matching

12. **Linux Device Tree**

(a) Before the Device Tree
(b) What is the Device Tree and why does the kernel needs it.
(c) Kernel with device tree
(d) Device Tree source files and tools
(e) Compiling a Device Tree Blob
(f) Connecting the driver to the DT
(g) How do drivers read properties from DTS

13. **I/O memory and ports (Class Exercise-9)**

(a) What are I/O Ports
(b) What is Memory mapped I\O
(c) Accessing I/O ports \ memory mapped I\O
(d) Requesting I/O ports and I/O mapped memory
(e) Caching issues
(f) Managed APIs
(g) Mapping I/O memory in virtual memory
(h) Compiler Optimization and Hardware Reordering

14. **Wait Queues (Class Exercise-10)**

(a) What are waitqueues
(b) Declaring a waitqueue
(c) Waking up from a waitqueue
(d) Exclusive sleeps

15. **Interrupts and Exceptions (Class Exercise-11)**

(a) Behind the scene- do_irq()
(b) low level handling
(c) Interrupt disabling
(d) Top & Bottom halves
(e) Differed work
(f) software interrupts
(g) Tasklets
(h) Work queues

16. **Kernel Synchronization**
    - (a) How to avoid race conditions
    - (b) Mutex
    - (c) Spin Locks
    - (d) Reader\Writer locks.
    - (e) What to use when .
    - (f) Deadlock situations.
    - (g) Atomic variables.

17. **Network Device Drivers**
    - (a) Allocation the network device
    - (b) Registration the net_device structure
    - (c) Probing the MAC & PHY chip
    - (d) Working with Socket buffers
    - (e) NIC Device initialization
    - (f) Packet reception
    - (g) Packet transmission
    - (h) NAPI
    - (i) Initializing device operation
    - (j) Network queues
    - (k) Implementing the Davicom MAC and PHY (100Mbps)

18. **Debugging & Tracing techniques**
    - (a) Using strace for application debugging
    - (b) Debugging with printk
    - (c) Native GDB debugging
    - (d) Cross GDB debugging
    - (e) **Kernel memory debugging**
        - (i) Kmemcheck
        - (ii) Kmemleak
        - (iii) Kasan - kernel address sanitizer
    - (f) Using JTAG for debugging
    - (g) Analyzing kernel core dumps
    - (h) kernel's performance counters
    - (i) static kernel tracing
    - (j) Kernel TAP

Appendix:

1. **User space device drivers**
2. **Kernel Subsystems:**
    - **(a) PCI Subsystem, Bus (Class Exercise-13)**
    - **(b) Working with DMA Controller**
    - **(c) USB Subsystem (Class Exercise-14)**
    - **(d) Input Subsystem (Class Exercise-15)**
3. **Polled input subclass**
4. **Serial drivers**
5. **Kernel Booting Sequence (What happens after the boot-loader)**
6. **Timing Measurements**
7. **RCUs**

הערות :

- ✓ פתיחת המסלול מותנה במספר נרשמים.
- ✓ המכללה מביאה ידעתם של המשתתפים שיתכנו שינוים בתוכן הקורסים ובמועדם.
- ✓ המכללה מתחייבת להודיע המשתתפים על כל שינוי.
- ✓ המכללה שומרת לעצמה את הזכות לשנות את תכני המסלול בהתאם לשיקול דעתה הבלעדית.