Red Hat Enterprise Linux Kernel Internals

Duration: 5 Days Course Code: RHD361

Overview:

Red Hat® Enterprise Linux® Kernel Internals (RHD361) is a hands-on course providing experienced developers an intensive, low-level examination of the Linux kernel architecture.
Topics include kernel compilation, debugging tools and techniques, and internal kernel APIs, including synchronization, process management, and memory management. These topics provide a solid understanding of the kernel's architecture, providing a useful base from which more specialized topics, like those presented in Red Hat Enterprise System Monitoring and Performance Tuning (RH442) or Red Hat Enterprise Linux Kernel Device Drivers (RHD362), can be addressed.

Target Audience:

Experienced developers who want to gain a thorough understanding of the Linux kernel architecture.

Objectives:

- Working with the Developer Community
- User Mode and Kernel Mode
- Kernel Compilation and Tools
- Modules
- Kernel API Overview
- Synchronization
- Kernel Debugging 1: Tools and Techniques
- Interrupts
- Device Driver Overview
- Memory Management
- Processes
- The Scheduler
- Kernel Timing
- SystemTapSystem and Kernel InitializationKernel Debugging 2: Crash Dumps
- Unit 17 - Red Hat Enterprise Linux Realtime Kernel

Prerequisites:

- Experience in C programming
- Knowledge of systems programming in a UNIX or Linux environment
- Register-level hardware programming knowledge is recommended but not required
- Familiarity with basic tools, such as vi, Emacs, and ?le utilities
- Familiarity with UNIX development tools, such as gcc and make

Follow-on-Courses:

- RHD362, Red Hat Enterprise Linux Kernel Device Drivers

Content:

Working with the Developer Community

- Community Linux Kernel Development
- Why Contribute Kernel Code Upstream?
- Licensing
- Copyright
- Submitted Work
- The Kernel Development Process
- Creating Patches for the Merge Window
- Staging Trees

User Mode and Kernel Mode

- The Linux Kernel - An Overview
- The Role of the Kernel
- Kernel Contexts
- Four Milliseconds in the Life of the Kernel
- System Ring Levels
- Kernel Mode
- User Mode
- Mode Switching Example: System Calls
- x86 System Call Interface
- x86 System Call Interface (cont.)
- Mode Switching Example: IRQ Event
- Kernel Mode Linux

Kernel Compilation and Tools

- Kernel Packages
- Kernel Version
- Kernel Documentation
- Kernel Source Layout
- Kernel Source Layout (cont.)
- Recompiling the Red Hat Kernel
- Install Kernel Development Packages
- Kernel Source Package
- Preparing Source Code for Compilation
- Customizing Kernel Name (Optional)
- Choosing Compilation Options
- Compiling the Kernel and Modules
- Installing the Kernel Modules
- Installing the Compiled Kernel and Related Files
- Kernel Application Binary Interface (kABI)
- cscope
- LXR
- git
- git Documentation

Modules

- Kernel Modules
- Kernel Module Utilities
- Mapping Modules to Attached Devices
- Kernel Module Essentials
- modinfo Macros
- printk()
- /proc/kmsg and klogd
- printk() Loglevels
- Rate Limiting printk()
- Putting It All Together: A Simple Module
- Compiling a Module
- Integrating A New Module with the Kernel
- Makefile and Kconfig

Kernel Debugging 1: Tools and Techniques

- Debugging Preparations
- kernel-debuginfo Warnings
- Kernel vs. User Space
- Live vs. Postmortem Debugging
- Crashes vs. Hangs
- Debugging Device Drivers
- User Space Debugging Tools
- /proc Kernel Information
- kernel.panic Tunable and Kernel Crashes
- /sys Filesystem
- debugfs Filesystem
- Printing from the Kernel
- Kernel Oops Messages
- SysRq Mechanism
- sosreport
- The crash Tool
- crash Requirements
- crash Installation
- crash Invocation
- crash Invocation Output
- crash Help
- crash Command Input
- crash Command Output
- crash Command Overview
- crash Default Context

Interrupts

- Interrupts
- Nature of Interrupts
- Types of Interrupts
- Interrupt Specific Hardware
- Interrupt Descriptor Table (IDT)
- IDT Initialization
- IDT Initialization Functions
- Exception Handling
- Asynchronous Interrupt Handling
- Interrupt Handler Considerations
- irq_desc Structure
- irqaction Structure
- Interrupt Handler Registration
- Performing Deferred Work
- Softirqs
- Using Softirqs
- Tasklets
- Using Tasklets
- Work Queues
- Work Queue Data Structures
- Using Work Queues

Device Driver Overview

- Device Drivers
- Device Types
- Device Nodes
- Creating a Device Node
- Dynamic Loading of Driver Modules
- Major and Minor Numbers
- Dynamic Major and Minor Numbers
- Dynamically Created Device Nodes
- Dynamically Created Device Nodes Made Easy

Kernel Timing

- The Need for Timing
- Timing Hardware
- Timing Source Selection
- Wall/Real Time: xtime
- Wall Clock System Calls
- Kernel Ticks: jiffies
- Software Timers
- POSIX Timers
- Interval Timers and alarm()
- High-Resolution Timers
- Timer Interrupt Handler
- TIMER_SOFTIRQ Softirq
- Delay Functions

SystemTap

- Introduction to SystemTap
- SystemTap's Main Components
- Monitoring the Kernel with SystemTap
- The stap Command
- Flow of Data in SystemTap
- Common Tapset Probe Points
- SystemTap Script Examples

System and Kernel Initialization

- Boot Sequence Overview
- BIOS Initialization
- Bootloader
- Starting the Boot Process: GRUB
- Bootloader Components
- The Chicken/Egg Module Problem and the Initial RAM Disk
- GRUB and grub.conf
- Kernel Initialization Overview
- __init and __initdata
- Initialization Subsections and Ordering
- Kernel Initialization
- init/main.c: start_kernel()
- init/main.c: rest_init()
- init/main.c: init()
- init/main.c: do_basic_setup()
- init/main.c: init_post()
- init Initialization
- Run Levels

Kernel Debugging 2: Crash Dumps

- Introduction to Crash Dumps
- Netdump/Diskdump Challenges
- Kdump
- Kdump Solution
- Kexec
- Relocatable Kernel
- In-place Kernel Decompression
- Starting Kdump
- Kdump Initrd Image
- Configuring Kdump
- Kdump Core Dumps to the Local System
- Kdump Core Dumps to NFS Mount Points
- Kdump Core Dumps to SSH Servers
- Dump File Size

- Module Parameters
- Example: Module with Parameter

Kernel API Overview

- Multitasking, Stacks, and Task-Descriptors
- Contents of a Program's Stack
- Kernel Mode Switch and the Stack
- Task Structures
- What Is a Process?
- thread_info Structure
- task_struct: Process Identifiers
- task_struct: Process State
- task_struct: Scheduling Information
- Doubly Linked Lists
- Doubly Linked Lists: Manipulation
- Doubly Linked Lists: Iteration
- Doubly Linked Lists: Processes
- task_struct: Related Processes
- task_struct: Statistics
- Allocating Kernel Memory: kmalloc()
- Memory Cache Optimizations: Branch Prediction
- Memory Cache Optimizations: Binding Structures
- Generating Kernel Errors

Synchronization

- Critical Sections
- Mutual Exclusion Devices
- Linux Mutex Toolbox
- Atomic Bit Operations
- Atomic Integers
- Spinlocks
- Spinlocks and Local Interrupts
- Read-Write Spinlocks
- Mutexes
- Semaphores
- Spinlock/Mutex Example
- Alternatives to Locking
- Sequential Locks
- Read-Copy-Update (RCU)
- Linux RCU Implementation
- Per-CPU Variables
- Completions
- The Big Kernel Lock

- Device Driver Essentials
- Character Device Registration
- Device Driver File Operations
- Driver Methods
- The file Structure
- The inode Structure
- The open and release Methods
- The read and write Methods
- Module Usage Count
- Simple Character Driver Example

Memory Management

- Virtual Memory and Paging
- x86 Memory Architecture
- Memory Segmentation in Linux
- x86 Segmentation
- x86 Segmentation in Linux
- Memory Paging
- Page Tables
- Mapping Virtual Addresses (x86)
- Mapping Virtual Addresses (x86-64)
- Memory Zones
- Arranging the Virtual Address Space
- ZONE_NORMAL
- ZONE_HIGHMEM
- ZONE_DMA
- Kernel Memory Allocation
- Memory Management
- Buddy Allocator
- Requesting and Releasing Page Frames
- Slab Allocator
- Slab Allocator (cont.)
- Non-Contiguous Memory Area Management
- Memory Flags: gfp_mask
- __get_free_pages()
- kmalloc()
- vmalloc()

Processes

- Creating Processes
- Sharing Resources
- do_fork()
- Process Memory Maps
- Memory Areas
- vm_flags
- pmap
- Kernel Threads
- Process 0
- Destroying Processes
- Context Switches
- When Does Context Switching Occur?
- When Is need_resched Set?
- When Is schedule() Called?
- Kernel Preemption

The Scheduler

- Priority
- Priority for Normal Processes
- Priority for Real-Time Processes
- Time Slices
- The O(1) Scheduler: Run Queues
- The O(1) Scheduler: Priority Arrays

- Customizing the Dump Capture Method: makedumpfile
- Dump Filtering
- Dump Compression
- Future Challenges

Unit 17 - Red Hat Enterprise Linux Realtime Kernel

- Realtime (RT) Linux
- Benefits of a Realtime Kernel
- Response Time Comparisons
- Wake-Up Response Time Example
- Changes in the Kernel
- Changes in the C Library
- RT Measurement Tools
- RT Tuning Tools
- RT Tuning Methods
- Loading the RT Kernel

- The O(1) Scheduler: How it works
- Wait Queues
- The O(1) Scheduler: Load Balancing
- The O(1) Scheduler: load_balance()
- Problems with the O(1) Scheduler
- O(1) Scheduler vs. CFS
- Overview of CFS
- Details of CFS
- CFS Task Scheduling
- CFS Scheduler Policies
- CFS Scheduler Classes
- CFS fair_sched_class
- CFS Tuning
- CFS Group Scheduling
- CONFIG_FAIR_GROUP_SCHED
- CONFIG_FAIR_CGROUP_SCHED

Further Information:

For More information, or to book your course, please call us on Head Office 01189 123456 / Northern Office 0113 242 5931

info@globalknowledge.co.uk

www.globalknowledge.co.uk

Global Knowledge, Mulberry Business Park, Fishponds Road, Wokingham Berkshire RG41 2GY UK