L-5_Corp Linux Debugging Techniques *: User-mode*

Duration: 5 days	
Pre-requisites	
Mandatory	***Preferable***
Course L-0 “Linux CLI and Shell Scripting” *plus*	Experience developing Linux kernel code with 'C', Course L-3 “Linux Device Drivers” strongly recommended.
Course C-1 “Programming in C” *plus*	
Course level L-1 “Linux Systems Programming”	
Course L-2 “Linux Kernel Internals”	
Below is the Outline TOC (Table Of Contents) document: it presents the (approximate) Day-wise Coverage.	

Part I – Introduction

Day 1

Module 1 : Introduction

Debugging [1]
- Origin
- Tools
- Basic Steps
- Steps to Reduce Debugging

Actual Cases

[1] “Debugging” from Wikipedia.org

Part II – Essential Application-Space Debugging

Module 2 : Useful Tools and Skills

Browsing Source
- By Hand with find & grep
- With ctags
- With cscope

Coding Style
Vim plugins for coding style
Using the indent utility
- Makefile target

Resources

Static Code Analysis with [sp]Lint
Assignments

Userspace Runtime Tracing Tools
- strace

ltrace

Creating and Applying Patches
Assignments

Module 3 : Memory

Internals: An Introduction to Virtual Memory Concepts
A Peek at the Internals Side
View of Process Image layout as seen by the Kernel

Memory Allocation
The malloc() API family
(Optional: Inspecting and Changing the program break)

Day 2

Module 3 : Memory (contd.)

Common Memory Issues
Memory Leaks
malloc() and fork()
Memory overcommit
Resource Limits
Manipulating resource limits with system calls
Using ulimit

Module 4 : Using the GNU Debugger gdb

Instrumentation with [f]printf
The MSG macro

gdb Commands
Compiling a Program That Is to Be Debugged with gdb
A Typical gdb Session

Useful gdb commands
Alternatives
gdb in TUI mode / ddd / Eclipse CDT

Core Dump Analysis
What it is
Generation and Nomenclature
Procfs settings
Analysing a Core Dump

Assignments: Debugging with gdb

Stack Analysis

x86 (IA-32) Stack Layout in detail
Looking up the stack with gdb
[Optional] ARM-32 stack layout
[Optional] x86_64 stack layout

Attempting to Analyze a Core Dump generated without symbolic information
Techniques 1 and 2

Assignments: Debugging with gdb

Module 5 : Bug Prevention Guidelines

Some Important Guidelines for Robust API usage
General Issues
Signal Issues
Sleeping correctly

"The Ten Commandments for 'C' Programmers" – Henry Spencer
"The Top 10 Ways to get *crewe* by the 'C' Programming Language" – D Dyer

Day 3

Module 6 : System Information via procfs and Performance Monitoring

What is the proc filesystem ?
A Map of /proc
Shell scripts to monitor procfs – system and process

Performance Monitoring
vmstat, dstat
perf
top, iotop, iftop
More tools (blog article)

Module 7 : Memory Management Debugging

Available opensource tools
Valgrind
Introduction
Valgrind Quick Start
Installation
Debugging session
Comparison
Assignments

Part III – Essential Kernel Internals and Programming

Module 8 : Building the 2.6 / 3.x Linux Kernel from Source

Version Nomenclature
Download and extraction
Applying Patches
Configuration
    Kernel Config Debug options
Compile and build
Bootloader setup

Case Study: Building and installing a custom kernel with key debugging features enabled.

Module 9 : Using QEMU

What is QEMU?
Installation
Usage with a root filesystem and kernel image
    Preparing the Kernel
    Preparing the Root Filesystem (RFS)
    Preparing an Initrd image

Lab: Build a functioning system (kernel + initrd-based root filesystem) using QEMU
(Optional: Full System Emulation)

Module 10 : A Quick Introduction to writing Loadable Kernel Modules for 2.6 Linux

Introduction
The "Hello, World" kernel module
Compilation – the Makefile
Module Utilities
    lsmod, insmod, rmmod, modinfo
Lab Assignment: Write and verify running a simple Loadable Kernel Module running on the emulator.

Day 4

Part IV – Kernel Debugging Techniques

Kernel-Space Code-Based Debugging Techniques

Module 11 : Code-based Techniques for Debugging

Debugging by Printing with the printk
    Loglevels
    How Messages Get Logged
    Turning Messages On and Off

Rate Limiting
Where you can and cannot use printk

Ftrace functionality
Basics
Tracing using ftrace
Code-Based tracing

A Header of Convenience

Using **debugfs**
Debugging with ioctl

Module 12 : Debugging System Faults

Oops Messages
Generating a (trivial) Oops
Analyzing an Oops dump
Article: “The foggy crystal ball: Understanding Oopses”
Generating a mixed source-assembly-machine-language dump
Kernel Mailing List Oops
Lab Assignment

System Hangs
Using the Magic SysRq Facility
Kernel Panic
Setting up your own panic handler using the kernel panic chain notifier mechanism

Module 13 : Kernel-Level Debuggers

Using **gdb** for kernel-space
gdb and loadable kernel modules

Day 5

KGDB
Introduction
kgdb setup
Initiating a debugging session on the target system
Using the kernel debugger kgdb
Useful gdb Macros
Debugging kernel modules

Using **QEMU and [k]gdb** for source-level kernel debugging
Stand-alone kernel debug
Building a root filesystem for Qemu VM

Sample Debug Session

KDB

kdb Kernel Patching, build and installation
kdb Activation
kdb Commands
kdb Debug Session

KDB / KGDB live session demo on the Raspberry Pi hardware board

objdump
gcc : setting debug flags
For debug symbolic information
For mixed source-assembly-machine language listing

Module 14 : Kernel-Level Tools

Dynamic Probes – Kprobes & Jprobes

Introduction
Kernel build and setup
Kprobe Interfaces

Probe example with sys_open
Probing at an offset within a function

Kprobes and Loadable Kernel Modules
A basic framework
Usage

Using Jumper Probes (jprobes)

Kdump, kexec and crash

The kexec with kdump feature
Tools and kernel installation
Usage Procedure
crash utility
Case Study whitepaper

Kernel Hacking Config Options

Wrap Up.