Linux Kernel Debugging - Tools & Techniques

Covers kernel version 3.x to 6.x

Duration: 3 days

Authored and Compiled by: Kaiwan N Billimoria, kaiwanTECH.
<kaiwan@kaiwantech.com>

Level:

BASIC INTERMEDIATE [**ADVANCED**]

Course Code: L5-K

Courseware version 24.02

Contact Information
Email : kaiwan@kaiwantech.com
Reach us online at http://kaiwantech.in/

Brief Description

This training is targeted primarily at software professionals - tech leads, system programmers / developers, maintainers and testers whose work on the Linux OS requires of them the ability to professionally debug (both) applications and/or kernel-space code. Often, a training like this one emphasizes tasks that are carried out on a more-or-less daily basis by the participant; obviously, this goes a long way to greatly increasing productivity.

The training begins with an introduction to debugging (the story behind the first bug is an interesting one!) and the debugging process in general.

Memory management and memory leakage are a common source of hard-to-find bugs; several very useful memory checker tools are covered. The participant will work on assignments designed to simulate the debugging process using the various tools learned.

This training then changes course to tackle kernel-space debugging problem areas and techniques. Priniting (instrumentation) techniques, using the powerful ftrace facility, procfs, sysfs and debugfs are covered. Several tools are covered in this regard; analyzing an Oops dump is covered in depth. The kernel debugger (kdb) is covered. Kprobes, kexec/kdump and crash round off useful kernel-space tools.

Throughout, professional / industry best practices are taught and encouraged.

Level:

BASIC INTERMEDIATE **[ADVANCED]**

Course Prerequisites

It is very important that the prerequisite(s) marked as Mandatory below be met by all participants intending to attend this training, either by having successfully attended a training program (mentioned below), or having the equivalent knowledge / skill sets.

Mandatory:

You have attended the “LINUX Kernel Internals” and “Linux Device Drivers” trainings -*or*- have the equivalent knowledge. The prerequisites in detail are as follows:

- A solid understanding of user-level UNIX or LINUX and some experience using the OS. Participants *should* be able to / understand and use:
  - use the shell command line
  - basic shell scripting
  - use editors like vi, vim, etc
  - know how to use tools/utilities such as find, du, cut, redirection operators, grep, etc
- Strong / working knowledge of 'C' programming skills
  - write good C code
  - compile the same
  - (basic) Makefile usage
- Application development exposure on UNIX/Linux platform using 'C'
  - know what library and system calls are
  - usage of file I/O (open, read, write, close, lseek) system calls
    - Application development experience and usage of system calls on any UNIX or LINUX platform using 'C'
  - working knowledge regarding the system calls execve, fork, wait*, fcntl, sigaction family, ioctl, select.
    - Working knowledge of multithreading on Linux (pthreads)

Preferable (Strongly Recommended):

Working knowledge of >=3.x Linux Kernel Internals

Should include: process descriptor (task structure), building the kernel from source.

- ○ Should have some (at least minimal) previous exposure to kernel code, being able to write (simple) kernel code in the form of LKMs (Loadable Kernel Modules) for the 2.6 kernel.
- ○ Should understand the essential character device driver framework (includes the FTE, the file_operations structure and their setup in the driver registration, as well as the switching operations of the driver).

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

Miscellaneous

- The need for a debug kernel
- Exploring various Kernel Hacking config options.

Kernel-Space Code-Based Debugging Techniques

Module 2 : Code-based Techniques for kernel Debugging

- **Debugging by Printing with the printk**
- Loglevels
  - Programatically turning Messages On and Off
  - Rate Limiting
  - Where you can and cannot use printk
  - trace_printk as an alternative.

- A Header of Convenience
- **Leveraging the powerful *dynamic printk* feature**

- Using **debugfs** programatically
  - Introduction
  - Debugfs pre-requisites
  - The debugfs API (ABI)
  - Full example walk-through

- Debugging with the ioctl.

Module 3 : Kernel Tracing with Ftrace, LTTng and front-ends

- A quick note - userspace tracing with ltrace, strace

- **Ftrace**
  - The Need
  - Kernel configuration
  - Usage

Example Usage

Manually

Automated with shell scripts.

Using the **trace-cmd** front-end to ftrace

The trccmd front-end to trace-cmd

Tracing the complete kernel flow with **LTTng**

Viewing with TraceCompass GUI

The **perf-tools** package – a quick overview.

eBPF – an introduction.

Day 2

Module 4 : Debugging Oops'es and System Faults

Oops Messages

Generating a (trivial) Oops

Analyzing an Oops dump

Article: "The foggy crystal ball: Understanding Oopses"

Investigating kernel Oops on the Aarch32 (ARM-32) and AArch64

"ARM Oops Kernel Messages" Table

objdump

gcc : setting debug flags: for debug symbolic information, and for mixed source-assembly-machine language listing.

Lab Assignment

System Hangs

Using the Magic SysRq Facility

WARN*(), BUG*(), etc macros

Kernel Panic

Setting up your own panic handler using the kernel panic chain notifier mechanism.

Module 5 : Static and Dynamic Probes (Kprobes)

Introduction

Kernel build and setup

Using static Kprobes

Kprobe Interfaces

Static kprobes – demos

Using kretprobes

Dynamic kprobes / kprobe-based event tracing

Setting it up

Dynamic kprobe on a kernel module

kprobe-perf

Perf and eBPF approaches

Demo - trapping into the execve().

Day 3

Module 6 : Kernel Memory Debugging

- Address Sanitizer for the kernel (KASAN)
    - Configuration
    - Usage
- SLUB memory Corruption Debug options
    - Via slab poisoning
    - Boot and runtime debug flags
- Memory leakage debug with kmemleak
    - Configuration
    - Usage

Module 7 : Essentials of Kdump, kexec and crash

- Motivation
- The kexec with kdump feature
- Tools and kernel installation
- Triggering kdump.

Using crash

- Running crash
    - Prerequisites
- Crash context
    - Viewing and Changing (set)
- Common Commands
    - ps, bt, log, struct, whatis, files, vm, net, dis, eval
- Running crash in 'batch' mode
- Symbolic display
    - Data Structures
    - System State
- Module debugging.

Module 8 : Kernel-Level Debuggers - KGDB

- Using **GDB** for kernel-space
    - GDB and loadable kernel modules.
- **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

KGDB live session demo.