




Linux debugging, profiling, tracing and performance analysis training

Course duration _____

 3 days – 24 hours

Language _____

Materials English


Oral Lecture English
French
Italian


Trainer _____

One of the following engineers

- Alexis Lothoré
- Luca Ceresoli

Contact _____

 training@bootlin.com

 +33 484 258 097

Audience

Companies and engineers interested in debugging, profiling and tracing Linux systems and applications, to analyze and address performance or latency problems.

Training objectives

- Be able to understand the main concepts of Linux that are relevant for performance analysis: process, threads, memory management, virtual memory, execution contexts, etc.
- Be able to analyze why a system is loaded and what are the elements that contributes to this load using common Linux observability tools.
- Be able to debug an userspace application using *gdb*, either live or after a crash, and analyze the contents of ELF binaries.
- Be able to trace and profile a complete userspace application and its interactions with the Linux kernel in order to fix bugs using *strace*, *ltrace*, *perf* or *Callgrind*.
- Be able to understand classical memory issues and analyze them using *valgrind*, *libfence* or *Massif*.
- Be able to trace and profile the entire Linux system, using *perf*, *ftrace*, *kprobes*, *eBPF* tools, *kernelshark* or *LTTng*
- Be able to debug Linux kernel issues: debug kernel crashes live or post-mortem, analyze memory issues at the kernel level, analyze locking issues, use kernel-level debuggers.

Prerequisites

- **Knowledge and practice of UNIX or GNU/Linux commands:** participants must be familiar with the Linux command line. Participants lacking experience on this topic should get trained by themselves, for example with our [freely available on-line slides](#).
- **Minimal experience in embedded Linux development:** participants should have a minimal understanding of the architecture of embedded Linux systems: role of the Linux kernel vs. user-space, development of Linux user-space applications in C. Following [Bootlin's Embedded Linux course](#) allows to fulfill this pre-requisite.
- **Minimal English language level: B1**, according to the *Common European Framework of References for Languages*, for our sessions in English. See the [CEFR grid](#) for self-evaluation.

Pedagogics

- Lectures delivered by the trainer: 40% of the duration
- Practical labs done by participants: 60% of the duration
- Electronic copies of presentations, lab instructions and data files. They are freely available [here](#).

Certificate

Only the participants who have attended all training sessions, and who have scored over 50% of correct answers at the final evaluation will receive a training certificate from Bootlin.

Disabilities

Participants with disabilities who have special needs are invited to contact us at training@bootlin.com to discuss adaptations to the training course.



Onsite
training

Required equipment

For on-site session delivered at our customer location, our customer must provide:

- Video projector
- One PC computer on each desk (for one or two persons) with at least 16 GB of RAM, and Ubuntu Linux 24.04 installed in a free partition of at least 30 GB
- Distributions other than Ubuntu Linux 24.04 are not supported, and using Linux in a virtual machine is not supported.
- Unfiltered and fast connection to Internet: at least 50 Mbit/s of download bandwidth, and no filtering of web sites or protocols.
- PC computers with valuable data must be backed up before being used in our sessions.

For on-site sessions organized at Bootlin premises, Bootlin provides all the necessary equipment.

Hardware platform for practical labs

STM32MP1 Discovery Kit

One of these Discovery Kits from STMicroelectronics:

STM32MP157A-DK1,
STM32MP157D-DK1, STM32MP157C-
DK2 or STM32MP157F-DK2

- STM32MP157, dual Cortex-A7 processor from STMicroelectronics
- USB powered
- 512 MB DDR3L RAM
- Gigabit Ethernet port
- 4 USB 2.0 host ports
- 1 USB-C OTG port
- 1 Micro SD slot
- On-board ST-LINK/V2-1 debugger
- Arduino compatible headers
- Audio codec, buttons, LEDs
- LCD touchscreen (DK2 kits only)



Day 1 - Morning

Lecture	Linux application stack	<ul style="list-style-type: none"> Global picture: understanding the general architecture of a Linux system, overview of the major components. What is the difference between a process and a thread, how applications run concurrently. ELF files and associated analysis tools. Userspace application memory layout (heap, stack, shared libraries mappings, etc). MMU and memory management: physical/virtual address spaces. Kernel context switching and scheduling Kernel execution contexts: kernel threads, workqueues, interrupt, threaded interrupts, softirq
Lecture	Common analysis & observability tools	<ul style="list-style-type: none"> Analyzing an ELF file with GNU binary utilities (<i>objdump</i>, <i>addr2line</i>). Tools to monitor a Linux system: processes, memory usage and mapping, resources. Using <i>vmstat</i>, <i>iostat</i>, <i>ps</i>, <i>top</i>, <i>iotop</i>, <i>free</i> and understanding the metrics they provide. Pseudo filesystems: <i>procfs</i>, <i>sysfs</i> and <i>debugfs</i>.

Day 1 - Afternoon

Lab	Check what is running on a system and its load	<ul style="list-style-type: none"> Observe running processes using <i>ps</i> and <i>top</i>. Check memory allocation and mapping with <i>procfs</i> and <i>pmap</i>. Monitor other resources usage using <i>iostat</i>, <i>vmstat</i> and <i>netstat</i>.
Lecture	Debugging an application	<ul style="list-style-type: none"> Using <i>gdb</i> on a live process. Understanding compiler optimizations impact on debuggability. Postmortem diagnostic using core files. Remote debugging with <i>gdbserver</i>. Extending <i>gdb</i> capabilities using python scripting
Lab	Solving an application crash	<ul style="list-style-type: none"> Analysis of compiled C code with compiler-explorer to understand optimizations. Managing <i>gdb</i> from the command line, then from an IDE. Using <i>gdb</i> Python scripting capabilities. Debugging a crashed application using a coredump with <i>gdb</i>.

Day 2 - Morning

Lecture	Tracing an application	<ul style="list-style-type: none"> Tracing system calls with <i>strace</i>. Tracing library calls with <i>ltrace</i>. Overloading library functions using <i>LD_PRELOAD</i>.
Lab	Debugging application issues	<ul style="list-style-type: none"> Analyze dynamic library calls from an application using <i>ltrace</i>. Overloading library functions using <i>LD_PRELOAD</i>. Analyzing an application system calls using <i>strace</i>.
Lecture	Memory issues	<ul style="list-style-type: none"> Usual memory issues: buffer overflow, segmentation fault, memory leaks, heap-stack collision. Memory corruption tooling, <i>valgrind</i>, <i>libefence</i>, etc. Heap profiling using <i>Massif</i> and <i>heaptrack</i>
Lab	Debugging memory issues	<ul style="list-style-type: none"> Memory leak and misbehavior detection with <i>valgrind</i> and <i>vgdb</i>. Visualizing application heap using <i>Massif</i>.

Day 2 - Afternoon

Lecture	Application profiling	<ul style="list-style-type: none">▪ Performances issues.▪ Gathering profiling data with <i>perf</i>.▪ Analyzing an application callgraph using <i>Callgrind</i> and <i>KCachegrind</i>.▪ Interpreting the data recorded by <i>perf</i>.
Lab	Application profiling	<ul style="list-style-type: none">▪ Profiling an application with <i>Callgrind/KCachegrind</i>.▪ Analyzing application performance with <i>perf</i>.▪ Generating a flamegraph using <i>FlameGraph</i>.

Day 3 - Morning

Lecture	System wide profiling and tracing	<ul style="list-style-type: none">▪ System wide profiling using <i>perf</i>.▪ Using <i>kprobes</i> to hook on kernel code without recompiling.▪ Application and kernel tracing and visualization using <i>ftrace</i>, <i>kernelshark</i> or <i>LTtng</i>▪ Tracing with <i>eBPF</i>: core principles, usage with BCC and with libbpf
Lab	System wide profiling and tracing	<ul style="list-style-type: none">▪ System profiling with <i>perf</i>.▪ System wide latencies debugging using <i>ftrace</i> and <i>kernelshark</i>.
Lab	Tracing tool with eBPF	<ul style="list-style-type: none">▪ Python scripting with <i>bcc</i>.▪ Custom tool development with libbpf.

Day 3 - Afternoon

Lecture	Kernel debugging	<ul style="list-style-type: none">▪ Kernel compilation results (<i>vmlinux</i>, <i>System.map</i>).▪ Understanding and configuring kernel <i>oops</i> behavior.▪ Post mortem analysis using kernel crash dump with <i>crash</i>.▪ Memory issues (<i>KASAN</i>, <i>UBSAN</i>, <i>Kmemleak</i>).▪ Debugging the kernel using <i>KGDB</i> and <i>KDB</i>.▪ Kernel locking debug configuration options (<i>lockdep</i>).▪ Other kernel configuration options that are useful for debug.
Lab	Kernel debugging	<ul style="list-style-type: none">▪ Analyzing an <i>oops</i> after using a faulty module with <i>objdump</i> and <i>addr2line</i>.▪ Debugging a deadlock problem using <i>PROVE_LOCKING</i> options.▪ Detecting undefined behavior with <i>UBSAN</i> in kernel code.▪ Find a module memory leak using <i>kmemleak</i>.▪ Debugging a module with <i>KGDB</i>.