PSCG

Premium Software Consulting Group

THE LINUX FOUNDATION AUTHORIZED TRAINING PARTNER

For quote and registration: email ron@thepscg.com or call +972-54-5529466 (Course #: EMBLINUX003)

Embedded Linux Development with QEMU

This course will give you the step-by-step framework for developing an embedded Linux product. You'll learn the methods used to build and adapt cross-compilers, boot loaders, the Linux kernel, device trees and user-space libraries and utilities to particular embedded environments, such as those in use in consumer electronics, military, medical, industrial, and auto industries.

Length: 4 Days
Type: Hands-On
Target Audience: Experienced developers, programmers and engineers who are interested in learning how to adapt Linux to an embedded system, especially those who have been assigned to a team tasked with designing such a system.

Prerequisites:

- Essential:
  - The attendees should proficient with C
  - The attendees should have Linux development experience.
  - The attendees should feel comfortable with the Linux command line interface
- Recommended:
  - Experience with cross development
  - Good understanding of the different stages of compiling, linking etc.

Summary:

The course is designed to provide experienced developers a solid grounding in the methods used to adapt the *Linux* kernel and userspace libraries and utilities to particular embedded environments, such as those in use in consumer electronics, military, medical, industrial, and auto industries. This four day course includes extensive hands-on exercises and demonstrations, focused upon the tools necessary for developing embedded *Linux* devices. Upon mastering the material you will have an understanding of:

- The different hardware and software components that make systems boot and run
- The *Linux* kernel architecture, emphasizing the essential points relevant to adapting the kernel to a custom embedded platform.
- The techniques for right sizing the system to meet project constraints
- The multitude of resources available for constructing a cross development environment for embedded projects
- The options available for populating libraries and application user-spaces to meet the goals and constraints of embedded systems

- Working with different file systems and understanding the different tools and hacks available to quickly troubleshoot and repair common Embedded Linux issues
- Building and troubleshooting a handful of tools, filesystems, and
- And more.

Note: 5 and 8 day versions with in-depth discussions of build systems and particular hardware from various vendors are available from The PSCG.

Outline:

1. Before attending the course
    - Self assessment
    - Class registration
    - Hardware and software setup
2. Administrative - Methodology and of operation
    - Who we are: about your instructor and The PSCG
    - Who you are
    - Objectives
    - Overview of our theory of operation
    - Dive-in-and-out easier and harder tasks methodology
    - Multiple perspectives of what can, should, or has to be done when building an embedded system
    - Caveats and challenges
    - Developing an understanding of the motivation and understanding the choice of tools for a mission
    - Linux kernel boot in QEMU when someone else has taken care of everything for you
    - Interactive demo: verifying setup, building the PSCG-mini-linux Linux distro
3. Introduction to Embedded and Real Time systems
    - Evolution of embedded systems and operating systems
    - Microcontrollers and Microprocessors
    - Real time vs. Hard real time
    - Real time operating systems
    - General Purpose Operating Systems
    - Embedded Linux
    - Status of Real Time Linux
    - Evolution of cloud software development vs. embedded SW/HW development
    - Supertip: Planning for Hardware that is not yet available
4. Introduction to Embedded Build Systems
    - Cross build systems
    - Native builds
    - Cross building
    - Canadian cross building
    - Building Linux systems - the common steps

- Building Linux systems - the Android way
- Building Linux systems - the Yocto Project way
- Building Linux systems - the Buildroot way
- Tool arsenal: the Kbuild system
- Tool arsenal: busybox
- Tool arsenal: chroot jails

5. Cross toolchains
   - Tool arsenal: tools in GCC and Clang
   - Tuple/triplet/multiarch concepts
   - Obtaining cross-toolchains
   - Notable toolchain providers
   - Building cross-toolchains from source
6. QEMU target building and setup
   - Resources and Virtualization: Hypervisors, Namespaces and cgroups
   - The QEMU project
   - The KVM project
   - QEMU examples: Linux and Android Emulator
   - Building QEMU from source
   - QEMU system vs QEMU user
   - Running static/chroot binaries of alien architectures with binfmt_misc
   - Networking and troubleshooting of provided systems
7. Introduction to Linux Kernel Initialization
   - Obtaining the Linux kernel, git and non-git flows
   - Working with the Linux kernel
   - Linux kernel configuration
   - The Linux boot process and init
   - User space vs. Kernel space - theory and practice
   - Build time, boot time and and sizing considerations
   - Filesystem principles
   - Initial ramdisk, initramfs and initrd
   - Linux kernel special in memory file systems
   - Root filesystems
8. Embedded Linux storage
   - Creating disk images
   - Partitioning
   - Filesystems
   - Common boot time troubleshooting
   - Linux memory allocation and swap partitions/files
   - Common memory hog troubleshooting
   - Sizing considerations: build time, flashing time, runtime
   - [Encryption at rest]
9. Root filesystems
   - init troubleshooting

- init frameworks: trivial static *init*, *busybox*, *SystemV*, *systemd*, *Android*
- The Linux File System Hierarchy (FHS)
- Filesystem types
- Kernel support, kernel file systems and user space file systems (FUSE)
- Supporting alien file systems
- Compression considerations
- Read only considerations
- File system corruption considerations
- Tuning tools
- Cross building a busybox based file system (revisited)
- Cross building a Debian root file system with *debootstrap*
- [How docker works]

10. Embedded Linux Networking
    - The networking layers
    - Network stacks in the Linux kernel
    - Tool arsenal: DHCP, BOOTP, TFTP and NFS
    - Tool arsenal: understanding the ip command set
    - Tool arsenal: DNS and route
    - Tool arsenal: iptables
    - Common networking troubleshooting
    - [Security]
11. Boot Phases
    - From power-up to Kernel
    - Warm boot and cold boot
    - x86 vs. ARM
    - [Secure boot flow]
    - [Low power mode: Suspend/Resume]
    - Boot ROM examples
    - BIOS
    - UEFI
    - Linux and Android boot loaders
    - GRUB example and terminology
    - Chainloading concepts
    - Describing your hardware to the operating system I: Board Files and ACPI
    - Motivation for U-Boot
12. Das U-Boot
    - Running U-Boot in QEMU
    - The U-Boot command line interface
    - U-Boot source code organization
    - U-Boot and peripherals communication
    - Working with memory and debugging
    - TFTP boot of Linux Kernel
    - TFTP boot of Linux Kernel + initramfs/initrd

- NFS mounting of a Linux file system
- Troubleshooting common errors: boot file size changes and memory
- Troubleshooting network boot
- [Modifying U-boot code]
- [Adding new code to U-Boot]
- Revisiting board files
- Describing your hardware to the operating system II: Device Tree Blobs
- [Linux and Android device enumeration and repackaging concepts]

13. Embedded Linux Kernel essentials
    - The Linux kernel source code organization
    - Kernel configuration and building from source code
    - Initial root file system selection considerations: [in kernel] *initramfs*, compression
    - Troubleshooting *initramfs* builds
    - Using a kernel debugger: KDB and KGDB
    - Loadable Kernel Modules
    - Understanding device nodes
    - [The Unified Device Model: sysfs, kobjects]
    - [Plug and play and device enumeration: udev and mdev]
    - Simple device drivers
    - [Tracing and profiling considerations]
    - [Additional debugging and observability mechanisms]
    - [Security]
14. Busybox revisited
    - The *busybox* source code organization
    - Configuring *busybox*
    - Building busybox from source
    - [Modifying busybox applets]
    - [Adding new busybox applets]
15. Standard libraries
    - Use case: Linux kernel *libc*
    - Static vs dynamic libraries, linkers and the binutils
    - The Linux kernel loader and linker
    - *ld*, the userspace dynamic linker
    - Calling conventions and dynamic library hooking/injection
    - *glibc*
    - *uClibc*
    - *musl*
    - Planning for different devices, different libraries and writing your own library code
16. Buildroot
    - Embedded Build systems constructs revisited
    - Building *Buildroot* from source
    - Configuring *Buildroot* for building a set of Embedded Linux devices

    - Booting and testing the results
17. [Yocto Project]
    - Embedded Build systems constructs revisited
    - Building *Yocto Project* from source
    - Configuring Yocto Project for building a set of Embedded Linux devices
    - Booting and testing the results
18. [AOSP]
    - Embedded Build systems constructs revisited
    - Building the Android Open Source Project (AOSP) from source
    - Configuring AOSP for building a set of Embedded Android devices
    - Booting and testing the results
19. Best practices
    - Considerations in building an Embedded Linux System
    - Considerations in testing an Embedded Linux System
    - [COTS repackaging]
    - [Building physical products: BOM, ,manufacturing, packaging, support]
    - [Managing the SBOM]
    - [Licensing]
    - [Considerations in securing an Embedded Linux System]
    - [Considerations in designing an OTA update solution]
20. Summary
21. Evaluation Survey

Notes

- Everything between [] is optional.
- The course can be taken with additional days targeting either tracing, debugging or security hardening.
- In on-site training, it is common with selected customers to have a dedicated 5th day either addressing one of the concepts listed above or having a joint consulting session with the class, working on their actual systems and actual problems.
- The PSCG offers dedicated courses for each of the above, as well as deeper dive into *Linux kernel development*, *device drivers*, *system programming*, *secure boot frameworks*, *Linux kernel security*, *Linux hardening*, *Android internals*, *Yocto Project* and more.

List of labs

The list of labs for the course is presented here exactly as it is from the different lab documents, for your convenience. It aims to give you the feeling of what you are going to do during (and most probably after, as there are more labs than the time usually permits) the course.
Please do not click the links - they were put in this document for your convenience, but you will not receive the lab documents before enrolling in the course.

Embedded Linux Development with QEMU

- Notes
- List of labs

Part II: Cross Building Tools and Running Systems

- Building a cross-toolchain
  - Creating the meta tool ct-ng
  - Feeling the tuple (some call it triplets) for your own
  - Building the cross-toolchain for riscv64
- Building QEMU from source
  - Using a distro provided qemu-system package
  - Building QEMU from source - obtaining the source and preparing for build
  - Building qemu-system from source
    - Verifying your build
    - Getting help
  - Building qemu-user from source
  - Building qemu-system from source - revisited
  - Building qemu-user-static from source
  - Wrapping up: Building QEMU (all the previous examples combined)
  - Building qemu-user-static from source - package dependency issues
- Cross Compiling a simple application and executing it in qemu-user
  - Cross Compiling a simple program
  - Using qemu-user to run your previously built binary
- Using Debootstrap to build a foreign Debian root filesystem
  - Naively using cross debootstrap
  - binfmt prerequisites - Linux kernel support
  - Registering the binfmt
  - Retrying debootstrap after registering the binfmt
- Using QEMU to emulate an entire system
  - Obtaining the files premade for you for this lab:
    - The efs way

- The Google drive way
- Running a QEMU system image
- Inspecting the root filesystem
- Creating a partitioned image and filesystems
- Partitioning your disk and creating your own filesystems
- Formatting your partitions
- Updating the image
- Tuning and fixing the image
- QEMU networking - Virtual class network setup - TFTP boot server
- Setting up tap device on the host and running dnsmasq
- dnsmasq setup troubleshooting tips
- QEMU networking - running QEMU with a tap device
- Testing TFTP from userspace
- Helper script: runqemus
- Networking Feng Shui
- Kicking off userspace - making your own init
- Basic init application that does something
- The persistent storage device method
- The ramdisk method
- [Bonus exercise for after we cross-build busybox]
- [Bonus exercise - booting a Debian and Ubuntu filesystems]

Part III: U-Boot and Network boot

- Running U-Boot in QEMU
- Using U-Boot to boot a kernel with DHCP
- U-Boot with QEMU - without specifying any network interface
- [U-Boot with QEMU - adding Ethernet user (SLIRP) backend]
- U-Boot with QEMU - adding Ethernet tap device
- U-Boot with QEMU - setting the TFTP server files - teaser
- U-Boot with QEMU - setting the TFTP server files
- U-Boot with QEMU - setting the TFTP server files
- Extracting the Flattened Device Tree Blob from a live image
- QEMU Device Tree Blob modification
- U-Boot with QEMU - booting a Linux kernel and Device Tree Blob
- U-Boot with QEMU - tftp boot of a kernel, dtb and initial ramdisk (i.e. the ramdisk method)
- Kicking off userspace via NFS root file system mounting
- Mounting a root filesystem via NFS - verifying the NFS server is configured properly and works

- Possible nfs-kernel-server service workaround for systemd less docker or WSL2 images:
- QEMU NFS root mount
- U-Boot NFS root mount

Part IV: Cross Building and Modifying Systems

- Recap and Environment Variable Preparation
  - Recap - tools used and built in the previous labs
  - Helper script: runqemus status
  - The next chapters will be using the following environment variables to specify different architectures
  - RISC-V labs environment variables
- **Busybox Revisited - Building a more meaningful root file system using Busybox**
  - Helper script: make-simple-busybox-based-rootfs.sh
  - Fetching and building busybox from source
  - Using Busybox's init
  - Using Your own init script
- Building the Linux Kernel from source
  - Cross build and boot a Linux Kernel
  - Adding an initramfs to the kernel
  - Troubleshooting initial console issues in userspace
- Building Das U-Boot from source
  - Clone, experiment with make menuconfig, build
- Building Buildroot from source
  - Building the root filesystem, kernel, and a bootable QEMU image
  - Configuring the buildroot kernel
  - Adding systemd as the init framework
  - Updating the previous images
  - Experiment with other standard C libraries
  - Adding U-Boot
- Building and customizing the Yocto Project from source