Embedded Linux Systems Development

Objectives

- Discover Linux and its development tools
- Connect an embedded Linux system in a network
- Review the Linux boot sequence
- Boot a remote Linux kernel
- Program and Debug Linux applications
- Mastering kernel development and debug tools
- Programming IOs, interrupts
- Discovering kernel debugging techniques

Labs are conducted QEMU ARM-based board
We use a recent version of Kernel

Prerequisite

- Basic Linux user knowledge
- Good C programming skills (see our L2 course)

Course Environment

- Theoretical course
  - PDF course material (in English).
  - Course dispensed using the Teams video-conferencing system.
  - The trainer answers trainees' questions during the training and provide technical and pedagogical assistance through the Teams video-conferencing system.
- Practical activities
  - Practical activities represent from 40% to 50% of course duration.
  - Code examples, exercises and solutions
  - One Online Linux PC per trainee for the practical activities.
  - The trainer has access to trainees' Online PCs for technical and pedagogical assistance.
  - Eclipse environment and GCC compiler.
  - QEMU Emulated board or physical board connected to the online PC (depending on the course).
  - Some Labs may be completed between sessions and are checked by the trainer on the next session.
- Downloadable preconfigured virtual machine for post-course practical activities
- At the start of each session the trainer will interact with the trainees to ensure the course fits their expectations and correct if needed

Duration

- Total: 24 hours
- 4 sessions, 6 hours each (excluding break time)
- Some Labs must be completed after the session

Target Audience

- Any embedded systems engineer or technician with the above prerequisites.

Evaluation modalities

- The prerequisites indicated above are assessed before the training by the technical supervision of the traineein his company, or by the trainee himself in the exceptional case of an individual trainee.
- Trainee progress is assessed in two different ways, depending on the course:
  - For courses lending themselves to practical exercises, the results of the exercises are checked by the trainer while, if necessary, helping trainees to carry them out by providing additional details.
  - Quizzes are offered at the end of sections that do not include practical exercises to verifythat the trainees have assimilated the points presented
- At the end of the training, each trainee receives a certificate attesting that they have successfully completed the course.
  - In the event of a problem, discovered during the course, due to a lack of prerequisites by the trainee a different or additional training is offered to them, generally to reinforce their prerequisites,in agreement with their company manager if applicable.

Plan

First Session

Linux overview

- Linux
  - History
  - Version management
- Linux architecture and modularity
- Linux system components
- The various licenses used by Linux (GPL, LGPL, etc)

Linux application development

- Structure of Linux applications
  - The ELF file format
- Linux development tools
  - Compiling
  - Documentation
  - Makefiles
  - Integrated Development Environments
- Creating Linux libraries
  - Static libraries
  - Dynamic libraries

Exercise: *Writing a simple, static and dynamic, library*

Linux application debugging

- Software Debug tools
  - Gdb
  - Memory management debug using dmalloc and efence
  - Runtime checks using valgrind

Exercise: *Debug an application and its libraries using gdbserver*
Exercise: *Checking memory management using dmalloc and valgrind*

User-Level Input-Output

- Standard input-output
  - disk files
  - devices
- Network programming
  - sockets

    - UDP and TCP protocols
- Asynchronous input-output
    - Non-blocking I/O
    - Multiplexed (the select and poll APIs)
    - Notified I/O
    - Chained I/O (the aio POSIX API)

Exercise: *Understanding Non-blocking Requests*

Second Session

Signal handling

- Signal handling
    - Signal types
    - Handling a signal
    - Functions usable in a signal handler
    - Signal masking and synchronous handling

Exercise: *Manage timeouts using signals and timers*

Multitask programming

- POSIX Threads
    - Definition of a thread
    - Thread programming
- Processes
    - The process concept
    - Processes and security
    - Process states
    - Process life-cycle: the "fork" and "'exec" calls
- Inter-Process Communications
    - File Mapping
    - Shared Memory

Exercise: *Fork a process*

Exercise: *Understand mmap system call*

Linux kernel programming

- Development in the Linux kernel
    - Kernel compilation
- Modules compilation
- Memory allocation in the kernel
- Helper libraries
    - Kref and container_of
    - Linked list
    - Fifos

Exercise: *Write the typical "Hello World" kernel module*

Exercise: *Understand kernel parameters and their access through the /sys filesystem*

Third Session

Linux kernel Debugging

- Oops
- The /proc filesystem
- The DebugFS filesystem
- Traces
- Miscellaneous Kernel Debug Features
- Debugging with KGDB

Exercise: *Debugging a module using KGDB*

Kernel multi-tasking

- Kernel task handling
- Concurrent kernel programming
- Miscellaneous Locks
- Creating kernel threads

Introduction to Linux drivers

- Kernel Architecture
- Device Access
- Functional Driver registration

Exercise: *Understand the driver structure and registration*

Driver IO functions

- Kernel structures used by drivers
- Opening and closing devices
- Data transfers
- Controlling the device
- Mapping device memory

Exercise: *Understand how to transfer data to or from user space*
Exercise: *Understand the basic read and write calls*

Fourth Session

Synchronous and asynchronous requests

- Task synchronization
- Synchronous request
- Asynchronous requests
  - Non-blocking requests
  - Multiplexed I/O
  - Notified I/O

Exercise: *Understand the ioctl system call handling*

Input-Output and Interrupts

- Memory-mapped registers
- Interrupts
- Legacy features

The Linux Device Model

- Linux Device Model Architecture
  - Overview
  - Classes
  - Busses
- The platform bus
  - Platform devices
  - Platform drivers
- Classes
  - The misc class
  - Custom classes
- Writing udev rules

Exercise: *Learning how to implement a platform driver (callbacks, registration, resources)*

Renseignements pratiques

Inquiry : 24 hours