Training Linux user mode programming: Programming Embedded Linux Applications for Linux - Operating Systems: Linux

D0 - Linux user mode programming

Programming Embedded Linux Applications for Linux

Objectives

- Discover Linux and its development tools
- Connect an embedded Linux system in a network
- Review the Linux boot sequence
- Mount a remote file system
- Boot a remote Linux kernel
- Program and Debug Linux applications
  - Network programming
  - Synchronous and asynchronous input-output
  - Multi-thread programming
  - Inter-process communications

Labs are conducted on target boards, that can be:
Dual Cortex/A7-based "STM32MP15-DISCO" boards from STMicroelectronics.
Quad Cortex/A9-based "SabreLite" boards from NXP.
Quad Cortex/A53-based "imx8q-evk" boards from NXP.

Who should attend this course?

- Engineers that must create embedded Linux applications

Prerequisite

- Basic Linux user knowledge
- Good C programming skills

Course environment

- Printed course material (in English).
- One Linux PC for two trainees.
- One target platform for two trainees.

Plan

First Day

Linux overview

- Linux
- The various licenses used by Linux (GPL, LGPL, etc.)
- Linux distributions

Linux for the user

- The Linux filesystem
- The Linux shell and scripts
- The vi editor
- Basic administration of a Linux system

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

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

Second Day

Linux application debugging

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

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

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)

Exercice: *Programming a client-server application*
Exercice: *Handle several parallel connections using asynchronous I/O*

Tracing system calls

- Trace system calls tools
  - Strace
  - Ltrace

Exercice: *Understanding Strace*

Third Day

Time and signal handling

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

Exercice: *Manage timeouts using signals and timers*

Multitask programming

- Processes
  - The process concept
  - Processes and security
  - Process states
  - Process life-cycle : the "fork" and "'exec" calls
- POSIX Threads
  - User and kernel threads
  - Thread programming
  - Mutexes and condition variables
  - Barriers
  - Thread-specific data

Exercice: *Managing several clients in parallel using fork*
Exercice: *Create a remote server using fork and exec*
Exercice: *Managing several clients in parallel using threads*
Exercice: *Manage thread-static data in a library*

Fourth Day

Memory management and Scheduling

- Linux Memory Management
  - Virtual and physical memory
  - Pagination and protection
  - Swapping
  - Memory allocation
  - Caches
- Scheduling in the Linux kernel
  - Context switches
  - The Completely Fair Scheduler
  - Scheduling groups
  - The real-time scheduler
  - Scheduling and SMP (Symmetrical Multi Processors)

Inter Process Communication

- Inter Process Communication
  - File mapping of files and devices
  - Shared memory
  - Message queues
  - Pipes
- Task synchronisation
  - Semaphore
  - Mutex
  - Signals
- The System V IPC (optional, described in appendix)

Exercice: *Handle communications between processes in a multi-process client-server system*

Exercice: *Setup timeouts to close dead connections on a server*

Renseignements pratiques

Duration : 4 days
Cost : 2160 € HT