

Embedded Linux and Device Drivers using x86 and ARM9 board HandsOn Workshop

Course Description:

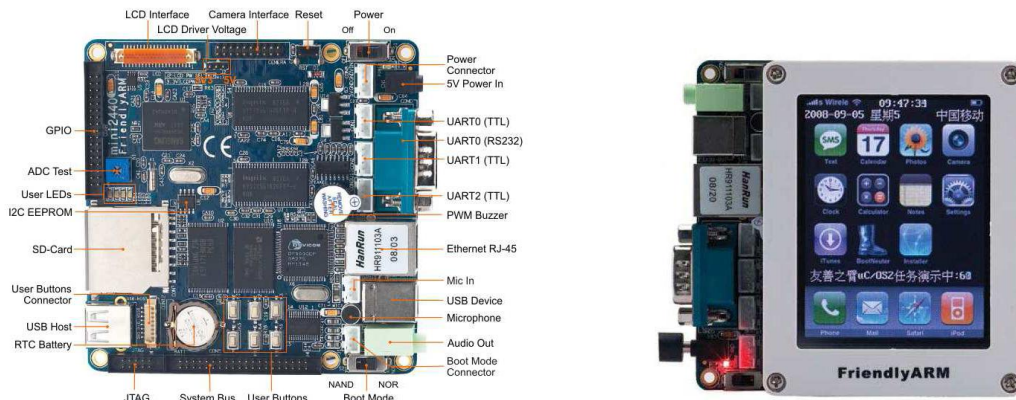
Embedded Linux is rapidly becoming an Operating System of choice for many embedded developers. According to the LINUX open source website, the significant reasons are that the source code is available, there are no runtime royalties and it is a robust reliable operating system which has excellent networking support.

This course teaches embedded skills using an embedded environment. Unlike most Linux courses, that use PC's as the target, this course uses ARM target hardware – a true embedded environment. It also covers various standard drivers for standard X86 Architecture and how it is implemented in the board used in the lab.

Overview:

Linux has a world wide support database, and probably the largest supported device driver base of any operating system. However, for the Linux beginners the choice in the market is confusing (e.g. various versions like RedHat, BlueCAT, Ubuntu, etc)

This course is unique in a number of areas like, First it is aimed at existing developers using traditional RTOS's in embedded environments. Second, it proposes that Linux is not suitable for all applications. It is as important to understand what Linux cannot do as much as what it can do. Third, and most significant, it isn't based, on traditional x86 PC hardware alone, but also include to try the same using ARM board.



This course uses a true embedded environment (a ARM PDA board) that does not have a floppy or hard disk. An environment most embedded engineers are familiar with.

This course demonstrates, through extensive hands-on, how to build small, fast applications with embedded Linux. Note, it does not promote any vendors tools.

Course Objectives:

- To get real-world exposure to embedded Linux
- To develop an application to run on an embedded Linux system
- To understand what is required to set-up a Linux cross development environment
- To understand the different approaches to making Linux "realtime" Delegates will learn:
- How to configure a standard Linux kernel for use in a cross development system.
- The steps to write, compile, download and debug an embedded Linux application with real hardware.
- How threaded applications fit into Linux.
- How drivers work in x86 and embedded environment.

Who Should Attend:

Software engineers who are developing applications for embedded or real-time Linux. Engineers wishing to assess the suitability of Linux for their next application.

Pre-Requisites:

- Good 'C' programming skills
- General knowledge of an RTOS or embedded operating systems
- Knowledge of Linux or Unix will be useful, but not essential
- Be able to use basic Linux/Unix commands (e.g. ls, cat, vi).

COURSE CONTENTS**Introduction to Linux**

History and open source
Features
Modular and monolithic vs micro
Kernel design goals

Understanding the kernel

Kernel structure
Kernel components and organization
Modes of operation
User library different from kernel

Booting and kernel initialization

Booting Basics
System Ups and Down under Linux
Boot-loaders (Various target specific boot loaders)
Understanding Linux start_kernel in brief

Linux Basic Common Set

File and directory related command
Using VI effectively [contrast w.r.t Word processor]
Commands for filters, IO redirection and etc
Process related commands
Miscellaneous Command [zip/tar/mount etc]
Shell scripts
a. Simple handy utilities
b. Major script techniques
 Compiling C codes

System calls

Understanding the Layer for Communication
Basics
Common system calls
How is it implemented in Linux?
Using a system call directly in assembly
Linux implemented methods

Day-2**Introduction Embedded OS**

What is Embedded OS
The cross development environment
Various commercial OS
Embedded Linux and other
Scalable Features: Linux Kernel
Customizing Linux kernel

Introduction to GNU toolchain

The gcc command
Size
Nm
Readelf
Strace
Ar
Ranlib
Objdump
Strings

The make utility
Gdb

Building your own kernel

Kernel source code
Configuring kernel
Building the Image
Entering the image path to the loader's script
Rebooting the system with your image

Experimenting the fresh 2.4 kernel

Applying patch for a kernel

Configuring the Kdb with patch

Experimenting the 2.6 kernel

Building the default kernel to startup a new 2.6 kernel

Setting up the host and target board

Minicom utility
Running existing kernel with the same
Also testing on Hyper Channel

Day-3

Introduction to ARM architecture

The feature of board
The board memory map
The application support on the board

Loading Boot loader

Introduction
Using DNW
Linux Download
Using H-JTAG
Bootloader download to NOR flash

Configuring the board
Test the Board without OS
Download the OS on the Board
Installing the application
Debugging an application using the board
Compiling and configuring kernel
X Applications on the Board

DAY 4

Linux Device Driver using X86

Introduction to Device Drivers: brief
Introduction to LKMS
Introduction to Linux Loadable Kernel Modules
Terminology
History of Loadable Kernel Modules
The Case For Loadable Kernel Modules
What LKMs Can't Do
What LKMs Are Used For
Making Loadable Kernel Modules
LKMs Utilities
How To Insert And Remove LKMs
Unresolved Symbols
About Module Parameters
Writing Your Own Loadable Kernel Module

Lab-session: Using X86 architecture
After the concept's presentation trainee will work on
Module Programming
Parameters passing
Stacked module

Introduction to Linux Device Drivers
Types of drivers (Other OS based discussion)

Character and Block Device Drivers
Major and Minor numbers
Dynamic Allocation of Major Numbers
Registering your driver
Removing a Driver from the system
File Operations
The file Structure
Open, release, read, and write
Copy to/from user
A sample Device Driver: simple_char

Debugging Techniques
Debugging by Printing
Debugging by Querying
Debuggers and Related Tools

Lab-session: Using X86 Architecture
After the concept's presentation trainee will work on
Write drivers programs and implement a test driver
Check applications on the same
Major Debugging Techniques
Configuring KDB based patch when needed

* Outline is subject to change during or before the program.

* * * * *