The Linux Kernel Module Programming Guide Tldp

A Beginners Guide to Linux Internals
A Top-down Approach for X86 and PowerPC Architectures
Chapter 25. Linux for Embedded Systems
Mastering Linux Kernel Development
The Linux Kernel Module Programming Guide
Create user-kernel interfaces, work with peripheral I/O, and handle hardware interrupts
Understanding the Linux Kernel
Linux Kernel and Driver Development - Practical Labs
Windows Kernel Programming
Kernel Projects for Linux
A Simpler Approach to Linux Kernel
Linux Driver Development with Raspberry Pi - Practical Labs
Develop customized drivers for embedded Linux
Linux Kernel Programming

Linux System Programming Techniques
Explore Linux system programming interfaces, theory, and practice
Linux Device Drivers
Mastering Linux Device Driver Development
Exploring Raspberry Pi
Algorithms and Structures of Version 2.4
Linux Kernel Programming
Linux Kernel in a Nutshell
Linux in a Nutshell
Tools and Techniques for Building with Embedded Linux
A comprehensive guide to kernel internals, writing kernel modules, and kernel synchronization
Write custom device drivers to support computer peripherals in Linux operating systems
Linux System Programming
Pro Linux Kernel Module Programming
A Guide for the Intrepid
FreeBSD Device Drivers
Professional Linux Kernel Architecture
Talking Directly to the Kernel and C Library

Software Engineering for Embedded Systems
Linux Kernel Development
Develop custom drivers for your embedded Linux applications
Embedded Linux
Interfacing to the Real World with Embedded Linux
Linux for Embedded and Real-time Applications
Linux Kernel Programming Part 2 - Char Device Drivers and Kernel Synchronization

SINGH COMPTON

A Beginners Guide to Linux Internals "O'Reilly Media, Inc."
Pro Linux Kernel Module Programming is your step-by-step guide to developing, debugging, and testing Linux Kernel Modules (LKMs) with ease. As LKMs and the applications that use them become more widely used, there are an increasing number of system software developers who wish to become involved in the development and maintenance of Linux-based systems. Some of these engineers are motivated purely by personal interest; some work for Linux companies, some work for hardware manufacturers, and some are involved with in-house development projects. However, all face a common problem: the learning curve for the kernel module is getting

longer and steeper. The system is becoming increasingly complex, and it is very large. This is where Pro Linux Kernel Module Programming comes in. This book takes you from downloading Linux kernel all the way to extending it by writing your own modules, and everything in between. Discover common errors people make, and best practices you can follow. Written in a free-flowing fashion, and explaining concepts first with lots of examples, you will learn the relevant kernel data structures, and the actual implementation. You will understand kernel module development, for example: device types, kernel development process, kernel objects, kernel interfaces; which will help you to understand why and how module works. You will then move onto developing LKMs with ease. Understand and demystify LKMs today using Pro Linux Kernel Module Programming. What you'll learn How Linux Kernel Modules (LKMs) work How to develop LKMs How to debug LKMs How to test LKMs Who this book is for As the Linux kernel and the applications that use it become more widely used, there are increasing number of system software developers who wish to become involved in the development and maintenance of Linux based systems. Some of these engineers are motivated purely by personal interest; some work for Linux companies, some work for hardware manufacturers, and some are involved with in-house

development projects. This book is for anyone who wants to develop Linux kernel modules in any setting.

A Top-down Approach for X86 and PowerPC Architectures "O'Reilly Media, Inc."

This book is written for students or professionals who quickly want to learn Linux Kernel programming and device driver development. Each chapter in this book is associated with code samples and code commentary so that the readers may quickly un.

Chapter 25. Linux for Embedded Systems Packt Publishing Ltd

The open source nature of Linux has always intrigued embedded engineers, and the latest kernel releases have provided new features enabling more robust functionality for embedded applications. Enhanced real-time performance, easier porting to new architectures, support for microcontrollers and an improved I/O system give embedded engineers even more reasons to love Linux! However, the rapid evolution of the Linux world can result in an eternal search for new information sources that will help embedded programmers to keep up! This completely updated second edition of noted author Doug Abbott’s respected introduction to embedded Linux brings readers up-to-speed on all the latest developments. This practical, hands-on guide covers the many issues of special concern to Linux users in the embedded space, taking into account their specific needs and constraints.

You'll find updated information on: • The GNU toolchain • Configuring and building the kernel • BlueCat Linux • Debugging on the target • Kernel Modules • Devices Drivers • Embedded Networking • Real-time programming tips and techniques • The RTAI environment • And much more The accompanying CD-ROM contains all the source code from the book's examples, helpful software and other resources to help you get up to speed quickly. This is still the reference you'll reach for again and again! * 100+ pages of new material adds depth and breadth to the 2003 embedded bestseller. * Covers new Linux kernel 2.6 and the recent major OS release, Fedora. * Gives the engineer a guide to working with popular and cost-efficient open-source code.
Mastering Linux Kernel Development Packt Publishing Ltd
"Linux internals simplified" is a book which discusses the basics of Linux kernel internals in a code driven approach. It picks the major subsystems of the kernel which are important, and tries to simplify its internal working and data structures. As such, this book is aimed at engineers who wish to start learning about the Linux kernel.This book starts with the basic steps to acquire the Linux kernel code. It then shows ways of customizing the build options and lastly kernel compilation. Next it looks at a number of hacking tools which will help one to debug and trace in a live Linux

system. Practical examples of ftrace, kprobes and crash tool are discussed. These tools are useful in trying to understand the way the Linux system works. Chapter 3 discusses the details of a running process in a Linux system. It touches topics such as address spaces of a running process, user and kernel spaces, system calls, Linux process descriptor, Linux process creation, and so on. This chapter builds a foundation of a program in execution in the Linux system.Once the reader knows about the running processes, chapter 4 discusses about the Linux process scheduling subsystem. This chapter discusses different data structures and code paths of the Linux scheduler, which controls the scheduling of processes in the Linux system. Chapter 5 discusses Interrupts, which play a significant role in the Linux operating system. The chapter discusses edge and level triggered interrupts, interrupt handlers and their registration, shared interrupt handlers, and so on. It also shows the ftrace of the do_irq function.Chapter 6 discusses the signal subsystem. It starts with a little introduction of the design of the signal subsystem. It then traces the code execution of delivering and handling of signals in the Linux kernel. The chapter then discusses signal overloading and how it is performed, while exploring the kernel code which handles this. Chapter 7 covers Linux synchronization

primitives, and why they are needed. It shows the detailed implementation of primitives like atomic variables, spinlocks, semaphores and mutexes in the Linux kernel.Chapter 8 discusses various ways of Linux kernel memory allocation. It discusses Buddy allocator, Resource map allocator and Slab allocator. It discusses various APIs used for these allocators (alloc_page/s, kmem_cache_alloc, kmalloc etc.). It also discusses how user space malloc results in memory allocation in the Linux kernel.Chapter 9 discusses the Linux dynamic modules, Linux character driver framework, internal functions which are used while creating a character driver, UDEV events and IOCTL interface. It also discusses Linux device model. It discusses example of bus, device and device_driver components. It illustrates device model when used in PCI BUS. Chapter 10 covers the subsystem related to block IOs. It starts with an introduction of filesystem and its purpose. It then traces the path an IO takes, right from the "write()" system call, to the moment it gets written to the disk. The chapter covers basic data structures and design elements while going down the IO stack.

The Linux Kernel Module Programming Guide Packt Publishing Ltd

UNIX, UNIX LINUX & UNIX TCL/TK. Write software that makes the most effective use of the Linux system, including the kernel and core system

libraries. The majority of both Unix and Linux code is still written at the system level, and this book helps you focus on everything above the kernel, where applications such as Apache, bash, cp, vim, Emacs, gcc, gdb, glibc, ls, mv, and X exist. Written primarily for engineers looking to program at the low level, this updated edition of Linux System Programming gives you an understanding of core internals that makes for better code, no matter where it appears in the stack. -- Provided by publisher.

Create user-kernel interfaces, work with peripheral I/O, and handle hardware interrupts

"O'Reilly Media, Inc."

To thoroughly understand what makes Linux tick and why it's so efficient, you need to delve deep into the heart of the operating system--into the Linux kernel itself. The kernel is Linux--in the case of the Linux operating system, it's the only bit of software to which the term "Linux" applies. The kernel handles all the requests or completed I/O operations and determines which programs will share its processing time, and in what order. Responsible for the sophisticated memory management of the whole system, the Linux kernel is the force behind the legendary Linux efficiency. The new edition of Understanding the Linux Kernel takes you on a guided tour through the most significant data structures, many algorithms, and

programming tricks used in the kernel. Probing beyond the superficial features, the authors offer valuable insights to people who want to know how things really work inside their machine. Relevant segments of code are dissected and discussed line by line. The book covers more than just the functioning of the code, it explains the theoretical underpinnings for why Linux does things the way it does. The new edition of the book has been updated to cover version 2.4 of the kernel, which is quite different from version 2.2: the virtual memory system is entirely new, support for multiprocessor systems is improved, and whole new classes of hardware devices have been added. The authors explore each new feature in detail. Other topics in the book include: Memory management including file buffering, process swapping, and Direct memory Access (DMA) The Virtual Filesystem and the Second Extended Filesystem Process creation and scheduling Signals, interrupts, and the essential interfaces to device drivers Timing Synchronization in the kernel Interprocess Communication (IPC) Program execution Understanding the Linux Kernel, Second Edition will acquaint you with all the inner workings of Linux, but is more than just an academic exercise. You'll learn what conditions bring out Linux's best performance, and you'll see how it meets the challenge of providing good system response during process

scheduling, file access, and memory management in a wide variety of environments. If knowledge is power, then this book will help you make the most of your Linux system.
Understanding the Linux Kernel John Wiley & Sons
Over 30 recipes to develop custom drivers for your embedded Linux applications. Key Features Use Kernel facilities to develop powerful drivers Via a practical approach, learn core concepts of developing device drivers Program a custom character device to get access to kernel internals Book Description Linux is a unified kernel that is widely used to develop embedded systems. As Linux has turned out to be one of the most popular operating systems used, the interest in developing proprietary device drivers has also increased. Device drivers play a critical role in how the system performs and ensures that the device works in the manner intended. By offering several examples on the development of character devices and how to use other kernel internals, such as interrupts, kernel timers, and wait queue, as well as how to manage a device tree, you will be able to add proper management for custom peripherals to your embedded system. You will begin by installing the Linux kernel and then configuring it. Once you have installed the system, you will learn to use the different kernel features and the character drivers. You will also cover interrupts in-depth and how you can manage them. Later, you will get

into the kernel internals required for developing applications. Next, you will implement advanced character drivers and also become an expert in writing important Linux device drivers. By the end of the book, you will be able to easily write a custom character driver and kernel code as per your requirements. What you will learn Become familiar with the latest kernel releases (4.19+/5.x) running on the ESPRESSObin devkit, an ARM 64-bit machine Download, configure, modify, and build kernel sources Add and remove a device driver or a module from the kernel Master kernel programming Understand how to implement character drivers to manage different kinds of computer peripherals Become well versed with kernel helper functions and objects that can be used to build kernel applications Acquire a knowledge of in-depth concepts to manage custom hardware with Linux from both the kernel and user space Who this book is for This book will help anyone who wants to develop their own Linux device drivers for embedded systems. Having basic hand-on with Linux operating system and embedded concepts is necessary.

Linux Kernel and Driver Development - Practical Labs Prentice Hall

Provides information on writing a driver in Linux, covering such topics as character devices, network interfaces, driver debugging, concurrency, and interrupts.

Windows Kernel

Programming Pearson Education India
This book contains the practical labs corresponding to the "Linux Kernel and Driver Development: Training Handouts" book from Bootlin. Get your hands on an embedded board based on an ARM processor (the Beagle Bone Black board), and apply what you learned: write a Device Tree to declare devices connected to your board, configure pin multiplexing, and implement drivers for I2C and serial devices. You will learn how to manage multiple devices with the same driver, to acces and write hardware registers, to allocate memory, to register and manage interrupts, as well as how to debug your code and interpret the kernel error messages. You will also keep an eye on the board and CPU datasheets so that you will always understand the values that you feed to the kernel.
Kernel Projects for Linux Createspace Independent Publishing Platform
Linux continues to grow as an operating system of choice in many embedded systems such as networking, wireless, and base stations. In this chapter we look at possible uses of Linux in embedded systems. The chapter covers getting a Linux kernel set up, getting started with creating your Linux baseline, and the initial steps of getting an application running on the platform. If you haven’t used Linux for an embedded system before, this chapter will cover all

of the basic steps to get you going!
A Simpler Approach to Linux Kernel "O'Reilly Media, Inc."
A guide to using Linux on embedded platforms for interfacing to the real world. "Embedded Linux" is one of the first books available that teaches readers development and implementation of interfacing applications on an Embedded Linux platform.
Linux Driver Development with Raspberry Pi - Practical Labs Addison-Wesley Professional
CD-ROM contains: Linux kernel version 2.4.4, plus sources from other programs and documents from the Linux Documentation Project.
Develop customized drivers for embedded Linux "O'Reilly Media, Inc."
With Kernel Projects for Linux, Professor Gary Nutt provides a series of 12 lab exercises that illustrate how to implement core operating system concepts in the increasingly popular Linux environment. The makeup of the manual allows readers to learn concepts on a modern operating system—Linux—while at the same time viewing the source code. This hands-on manual complements any core OS book by demonstrating how theoretical concepts are realized in Linux.Part I presents an overview of the Linux design, offering some insight into such topics as runtime organization and process, file, and device management. Part II consists of a graduated set of exercises where readers move from

inspecting various aspects of the operating systems's internals to developing their own functions and data structures for the Linux kernel.This book is designed for programmers who need to learn the fundamentals of operating systems on a modern OS. The progressively harder exercises allow them to learn concepts in a hands-on setting.

Linux Kernel Programming Packt Publishing Ltd

Discover how to write high-quality character driver code, interface with userspace, work with chip memory, and gain an in-depth understanding of working with hardware interrupts and kernel synchronization Key Features: Delve into hardware interrupt handling, threaded IRQs, tasklets, softirqs, and understand which to use when Explore powerful techniques to perform user-kernel interfacing, peripheral I/O and use kernel mechanisms Work with key kernel synchronization primitives to solve kernel concurrency issues Book Description: Linux Kernel Programming Part 2 - Char Device Drivers and Kernel Synchronization is an ideal companion guide to the Linux Kernel Programming book. This book provides a comprehensive introduction for those new to Linux device driver development and will have you up and running with writing misc class character device driver code (on the 5.4 LTS Linux kernel) in next to no time. You'll begin by learning how to write a simple and complete misc class

character driver before interfacing your driver with user-mode processes via procfs, sysfs, debugfs, netlink sockets, and ioctl. You'll then find out how to work with hardware I/O memory. The book covers working with hardware interrupts in depth and helps you understand interrupt request (IRQ) allocation, threaded IRQ handlers, tasklets, and softirqs. You'll also explore the practical usage of useful kernel mechanisms, setting up delays, timers, kernel threads, and workqueues. Finally, you'll discover how to deal with the complexity of kernel synchronization with locking technologies (mutexes, spinlocks, and atomic/refcount operators), including more advanced topics such as cache effects, a primer on lock-free techniques, deadlock avoidance (with lockdep), and kernel lock debugging techniques. By the end of this Linux kernel book, you'll have learned the fundamentals of writing Linux character device driver code for real-world projects and products. What You Will Learn: Get to grips with the basics of the modern Linux Device Model (LDM) Write a simple yet complete misc class character device driver Perform user-kernel interfacing using popular methods Understand and handle hardware interrupts confidently Perform I/O on peripheral hardware chip memory Explore kernel APIs to work with delays, timers, kthreads, and workqueues Understand kernel concurrency issues Work with key kernel

synchronization primitives and discover how to detect and avoid deadlock Who this book is for: An understanding of the topics covered in the Linux Kernel Programming book is highly recommended to make the most of this book. This book is for Linux programmers beginning to find their way with device driver development. Linux device driver developers looking to overcome frequent and common kernel/driver development issues, as well as perform common driver tasks such as user-kernel interfaces, performing peripheral I/O, handling hardware interrupts, and dealing with concurrency will benefit from this book. A basic understanding of Linux kernel internals (and common APIs), kernel module development, and C programming is required.

Linux System Programming Techniques Prentice Hall

Harness the power of Linux to create versatile and robust embedded solutions Key Features Learn how to develop and configure robust embedded Linux devices Explore the new features of Linux 5.4 and the Yocto Project 3.1 (Dunfell) Discover different ways to debug and profile your code in both user space and the Linux kernel Book Description Embedded Linux runs many of the devices we use every day. From smart TVs and Wi-Fi routers to test equipment and industrial controllers, all of them have Linux at their heart. The Linux OS is one of the foundational technologies comprising

the core of the Internet of Things (IoT). This book starts by breaking down the fundamental elements that underpin all embedded Linux projects: the toolchain, the bootloader, the kernel, and the root filesystem. After that, you will learn how to create each of these elements from scratch and automate the process using Buildroot and the Yocto Project. As you progress, the book explains how to implement an effective storage strategy for flash memory chips and install updates to a device remotely once it's deployed. You'll also learn about the key aspects of writing code for embedded Linux, such as how to access hardware from apps, the implications of writing multi-threaded code, and techniques to manage memory in an efficient way. The final chapters demonstrate how to debug your code, whether it resides in apps or in the Linux kernel itself. You'll also cover the different tracers and profilers that are available for Linux so that you can quickly pinpoint any performance bottlenecks in your system. By the end of this Linux book, you'll be able to create efficient and secure embedded devices using Linux. What you will learn Use Buildroot and the Yocto Project to create embedded Linux systems Troubleshoot BitBake build failures and streamline your Yocto development workflow Update IoT devices securely in the field using Mender or balena Prototype peripheral additions by reading

schematics, modifying device trees, soldering breakout boards, and probing pins with a logic analyzer Interact with hardware without having to write kernel device drivers Divide your system up into services supervised by BusyBox runit Debug devices remotely using GDB and measure the performance of systems using tools such as perf, ftrace, eBPF, and Callgrind Who this book is for If you're a systems software engineer or system administrator who wants to learn Linux implementation on embedded devices, then this book is for you. Embedded systems engineers accustomed to programming for low-power microcontrollers can use this book to help make the leap to high-speed systems on chips that can run Linux. Anyone responsible for developing new hardware that needs to run Linux will also find this book useful. Basic working knowledge of the POSIX standard, C programming, and shell scripting is assumed.

Explore Linux system programming interfaces, theory, and practice Packt Publishing Ltd

Twenty five years ago, as often happens in our industry, pundits laughed at and called Linux a joke. To say that view has changed is a massive understatement. This book will cement for you both the conceptual 'why' and the practical 'how' of systems programming on Linux, and covers Linux systems programming on the latest 4.x kernels.

Linux Device Drivers John

Wiley & Sons
Presents an overview of kernel configuration and building for version 2.6 of the Linux kernel.

Mastering Linux Device Driver Development
БХВ-Петербург
Learn to develop customized device drivers for your embedded Linux system About This Book Learn to develop customized Linux device drivers Learn the core concepts of device drivers such as memory management, kernel caching, advanced IRQ management, and so on. Practical experience on the embedded side of Linux Who This Book Is For This book will help anyone who wants to get started with developing their own Linux device drivers for embedded systems. Embedded Linux users will benefit highly from this book. This book covers all about device driver development, from char drivers to network device drivers to memory management. What You Will Learn Use kernel facilities to develop powerful drivers Develop drivers for widely used I2C and SPI devices and use the regmap API Write and support devicetree from within your drivers Program advanced drivers for network and frame buffer devices Delve into the Linux irqdomain API and write interrupt controller drivers Enhance your skills with regulator and PWM frameworks Develop measurement system drivers with IIO framework Get the best from memory management and the DMA subsystem Access and manage GPIO subsystems and develop

GPIO controller drivers In Detail Linux kernel is a complex, portable, modular and widely used piece of software, running on around 80% of servers and embedded systems in more than half of devices throughout the World. Device drivers play a critical role in how well a Linux system performs. As Linux has turned out to be one of the most popular operating systems used, the interest in developing proprietary device drivers is also increasing steadily. This book will initially help you understand the basics of drivers as well as prepare for the long journey through the Linux Kernel. This book then covers drivers development based on various Linux subsystems such as memory management, PWM, RTC, IIO, IRQ management, and so on. The book also offers a practical approach on direct memory access and network device drivers. By the end of this book, you will be comfortable with the concept of device driver development and will be in a position to write any device driver from scratch using the latest kernel version (v4.13 at the time of writing this book). Style and approach A set of engaging examples to develop Linux device drivers

Exploring Raspberry Pi

Addison-Wesley Professional

This book follows on from Linux Kernel Programming, helping you explore the Linux character device driver framework and enables you to write 'misc' class drivers. You'll learn how to

efficiently interface with user apps, perform I/O on hardware memory, handle hardware interrupts, and leverage kernel delays, timers, kthreads, and workqueues.
Algorithms and Structures of Version 2.4 Packt Publishing Ltd
Uncovering the development of the hacking toolset under Linux, this book teaches programmers the methodology behind hacker programming techniques so that they can think like an attacker when developing a defense. Analyses and cutting-edge programming are provided of aspects of each hacking item and its source code—including ping and traceroute utilities, viruses, worms, Trojans, backdoors, exploits (locals and remotes), scanners (CGI and port), smurf and fraggle attacks, and brute-force attacks. In addition to information on how to exploit buffer overflow errors in the stack, heap and BSS, and how to exploit format-string errors and other less common errors, this guide includes the source code of all the described utilities on the accompanying CD-ROM.

Anatomy Episode Number