

Writing A Unix Device Driver

For more than 40 years, Computerworld has been the leading source of technology news and information for IT influencers worldwide. Computerworld's award-winning Web site (Computerworld.com), twice-monthly publication, focused conference series and custom research form the hub of the world's largest global IT media network.

Offers practical, hands-on guidance in developing your own device drivers. Clearly demonstrates how to write device drivers for adding disk drives, printers, magnetic tapes and other peripherals to your Unix system. Presents procedures for developing and testing new device drivers including how to select a convenient working directory; use make-files; preserve and boot alternative kernel versions; debug driver code and much more. Packed with examples which illustrate each operation in practice.

Linux® is being adopted by an increasing number of embedded systems developers, who have been won over by its sophisticated scheduling and networking, its cost-free license, its open development model, and the support offered by rich and powerful programming tools. While there is a great deal of hype surrounding the use of Linux in embedded systems, there is not a lot of practical information. Building Embedded Linux Systems is the first in-depth, hard-core guide to putting together an embedded system based on the Linux kernel. This indispensable book features arcane and previously undocumented procedures for:

- Building your own GNU development toolchain
- Using an efficient embedded development framework
- Selecting, configuring, building, and installing a target-specific kernel
- Creating a complete target root filesystem
- Setting up, manipulating, and using solid-state storage devices
- Installing and configuring a bootloader for the target
- Cross-compiling a slew of utilities and packages
- Debugging your embedded system using a plethora of tools and techniques

Details are provided for various target architectures and hardware configurations, including a thorough review of Linux's support for embedded hardware. All explanations rely on the use of open source and free software packages. By presenting how to build the operating system components from pristine sources and how to find more documentation or help, this book greatly simplifies the task of keeping complete control over one's embedded operating system, whether it be for technical or sound financial reasons. Author Karim Yaghmour, a well-known designer and speaker who is responsible for the Linux Trace Toolkit, starts by discussing the strengths and weaknesses of Linux as an embedded operating system. Licensing issues are included, followed by a discussion of the basics of building embedded Linux systems. The configuration, setup, and use of over forty different open source and free software packages commonly used in embedded Linux systems are also covered. uClibc, BusyBox, U-Boot, OpenSSH, tftpd, tftp, strace, and gdb are among the packages discussed.

A growing concern of mine has been the unrealistic expectations for new

computer-related technologies introduced into all kinds of organizations. Unrealistic expectations lead to disappointment, and a schizophrenic approach to the introduction of new technologies. The UNIX and real-time UNIX operating system technologies are major examples of emerging technologies with great potential benefits but unrealistic expectations. Users want to use UNIX as a common operating system throughout large segments of their organizations. A common operating system would decrease software costs by helping to provide portability and interoperability between computer systems in today's multivendor environments. Users would be able to more easily purchase new equipment and technologies and cost-effectively reuse their applications. And they could more easily connect heterogeneous equipment in different departments without having to constantly write and rewrite interfaces. On the other hand, many users in various organizations do not understand the ramifications of general-purpose versus real-time UNIX. Users tend to think of "real-time" as a way to handle exotic heart-monitoring or robotics systems. Then these users use UNIX for transaction processing and office applications and complain about its performance, robustness, and reliability. Unfortunately, the users don't realize that real-time capabilities added to UNIX can provide better performance, robustness and reliability for these non-real-time applications. Many other vendors and users do realize this, however. There are indications even now that general-purpose UNIX will go away as a separate entity. It will be replaced by a real-time UNIX. General-purpose UNIX will exist only as a subset of real-time UNIX.

An authoritative guide to Windows NT driver development, now completely revised and updated. The CD-ROM includes all source code, plus Microsoft hardware standards documents, demo software, and more.

For users of the Digital UNIX (formerly DEC OSF/1) operating system, as well as for systems engineers interested in writing UNIX-based device drivers. Discusses how to write device drivers for computer systems running the Digital UNIX operating system. In addition, the volume provides information on designing drivers, UNIX-based data structures, and OSF-based kernel interfaces.

Annotation copyright by Book News, Inc., Portland, OR

Device drivers literally drive everything you're interested in--disks, monitors, keyboards, modems--everything outside the computer chip and memory. And writing device drivers is one of the few areas of programming for the Linux operating system that calls for unique, Linux-specific knowledge. For years now, programmers have relied on the classic Linux Device Drivers from O'Reilly to master this critical subject. Now in its third edition, this bestselling guide provides all the information you'll need to write drivers for a wide range of devices. Over the years the book has helped countless programmers learn: how to support computer peripherals under the Linux operating system how to develop and write software for new hardware under Linux the basics of Linux operation even if they are not expecting to write a driver The new edition of Linux Device Drivers is

better than ever. The book covers all the significant changes to Version 2.6 of the Linux kernel, which simplifies many activities, and contains subtle new features that can make a driver both more efficient and more flexible. Readers will find new chapters on important types of drivers not covered previously, such as consoles, USB drivers, and more. Best of all, you don't have to be a kernel hacker to understand and enjoy this book. All you need is an understanding of the C programming language and some background in Unix system calls. And for maximum ease-of-use, the book uses full-featured examples that you can compile and run without special hardware. Today Linux holds fast as the most rapidly growing segment of the computer market and continues to win over enthusiastic adherents in many application areas. With this increasing support, Linux is now absolutely mainstream, and viewed as a solid platform for embedded systems. If you're writing device drivers, you'll want this book. In fact, you'll wonder how drivers are ever written without it.

New requirements for UNIX device drivers arise every week. These requirements range from drivers for mice to graphical display cards, from point of sales terminals to intelligent telephone exchanges. Writing Device Drivers for SCO UNIX is based on a training course run by The Santa Cruz Operation Ltd. It is a practical guide that will equip you with the skills you need to meet the challenge of writing a variety of device drivers. You will explore: The structure and mechanisms of an operating system, the concept of device independence and computer peripheral architecture Numerous hands-on exercises. By working through these exercises you will . . . Write a device driver for a mouse Write a Stream driver Write a simple line discipline Experiment with interrupts Examples based on the best selling, most up to date version 3.2 V4 of SCO UNIX Principles that will enable you to extend your skills to writing device drivers for other operating systems. If you are a student or a professional systems programmer with some experience of using C and developing UNIX programs you will find this book an invaluable guide.

Master the art of developing customized device drivers for your embedded Linux systems Key Features Stay up to date with the Linux PCI, ASoC, and V4L2 subsystems and write device drivers for them Get to grips with the Linux kernel power management infrastructure Adopt a practical approach to customizing your Linux environment using best practices Book Description Linux is one of the fastest-growing operating systems around the world, and in the last few years, the Linux kernel has evolved significantly to support a wide variety of embedded devices with its improved subsystems and a range of new features. With this book, you'll find out how you can enhance your skills to write custom device drivers for your Linux operating system. Mastering Linux Device Driver Development provides complete coverage of kernel topics, including video and audio frameworks, that usually go unaddressed. You'll work with some of the most complex and impactful Linux kernel frameworks, such as PCI, ALSA for SoC, and Video4Linux2, and discover expert tips and best practices along the way. In addition to this, you'll understand how to make the most of frameworks such as NVMEM and Watchdog. Once you've got to grips with Linux kernel helpers, you'll advance to working with

special device types such as Multi-Function Devices (MFD) followed by video and audio device drivers. By the end of this book, you'll be able to write feature-rich device drivers and integrate them with some of the most complex Linux kernel frameworks, including V4L2 and ALSA for SoC. What you will learn Explore and adopt Linux kernel helpers for locking, work deferral, and interrupt management Understand the Regmap subsystem to manage memory accesses and work with the IRQ subsystem Get to grips with the PCI subsystem and write reliable drivers for PCI devices Write full multimedia device drivers using ALSA SoC and the V4L2 framework Build power-aware device drivers using the kernel power management framework Find out how to get the most out of miscellaneous kernel subsystems such as NVMEM and Watchdog Who this book is for This book is for embedded developers, Linux system engineers, and system programmers who want to explore Linux kernel frameworks and subsystems. C programming skills and a basic understanding of driver development are necessary to get started with this book.

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.

New for UNIX System V Release 4.2, this guide contains the latest information for writing, installing and testing UNIX System V device drivers. It provides an in-depth explanation of new SVR4.2 features such as dynamically loadable kernel modules, the new device driver installation tools and the new system configuration file formats.

A practical, hands-on guide to driver design and development. Writing UNIX Device Drivers in C contains all the information you need to design and build UNIX device drivers. Adams and Tondo introduce the concept that device drivers are the implementation of an abstract software architecture and present a template-based development process that reduces the drudgery of implementing and debugging. This approach shortens development time and allows you to focus on the problem the device driver is designed to solve.

Learn how to write high-quality kernel module code, solve common Linux kernel programming issues, and understand the fundamentals of Linux kernel internals Key Features Discover how to write kernel code using the Loadable Kernel Module framework Explore industry-grade techniques to perform efficient memory allocation and data synchronization within the kernel Understand the essentials of key internals topics such as kernel architecture, memory management, CPU scheduling, and kernel synchronization Book Description Linux Kernel Programming is a comprehensive introduction for those new to Linux kernel and module development. This easy-to-follow guide will have you up and running with writing kernel code in next-to-no time. This book uses the latest 5.4 Long-Term Support (LTS) Linux kernel, which will be maintained from November 2019 through to December 2025. By working with the 5.4 LTS kernel throughout the book, you can be confident that your knowledge will continue to be valid for years to come. This Linux book begins by showing you how to build the kernel from the source. Next, you'll learn how to write your first kernel module using the powerful Loadable Kernel Module (LKM) framework. The book then covers key kernel internals topics including Linux kernel architecture, memory management, and CPU scheduling. Next, you'll delve into the fairly complex topic of concurrency within the kernel, understand the issues it can cause, and learn how they can be addressed with various locking technologies (mutexes, spinlocks, atomic, and refcount operators). You'll also benefit from more advanced material on cache effects, a primer on lock-free techniques within the kernel, deadlock avoidance (with lockdep), and kernel lock debugging techniques. By the end of this kernel book, you'll have a detailed understanding of the fundamentals of writing Linux kernel module code for real-world projects and products. What you will learn Write high-quality modular kernel code (LKM framework) for 5.x kernels Configure and build a kernel from source Explore the Linux kernel architecture Get to grips with key internals regarding memory management within the kernel Understand and work with various dynamic kernel memory alloc/dealloc APIs Discover key internals aspects regarding CPU scheduling within the kernel Gain an understanding of kernel concurrency issues Find out how to work with

key kernel synchronization primitives Who this book is for This book is for Linux programmers beginning to find their way with Linux kernel development. Linux kernel and driver developers looking to overcome frequent and common kernel development issues, as well as understand kernel internals, will benefit from this book. A basic understanding of Linux CLI and C programming is required.

Linux Kernel Module Programming Guide is for people who want to write kernel modules. It takes a hands-on approach starting with writing a small "hello, world" program, and quickly moves from there. Far from a boring text on programming, Linux Kernel Module Programming Guide has a lively style that entertains while it educates. An excellent guide for anyone wishing to get started on kernel module programming. *** Money raised from the sale of this book supports the development of free software and documentation.

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

Over the last few years, Linux has grown both as an operating system and a tool for personal and business use. Simultaneously becoming more user friendly and more powerful as a back-end system, Linux has achieved new plateaus: the newer filesystems have solidified, new commands and tools have appeared and become standard, and the desktop--including new desktop environments--have proved to be viable, stable, and readily accessible to even those who don't consider themselves computer gurus. Whether you're using Linux for personal software projects, for a small office or home office (often termed the SOHO environment), to provide services to a small group of colleagues, or to administer a site responsible for millions of email and web connections each day, you need quick access to information on a wide range of tools. This book covers all aspects of administering and making effective use of Linux systems. Among its topics are booting, package management, and revision control. But foremost in Linux in a Nutshell are the utilities and commands that make Linux one of the most powerful and flexible systems available. Now in its fifth edition, Linux in a Nutshell brings users up-to-date with the current state of Linux. Considered by many to be the most complete and authoritative command reference for Linux available, the book covers all substantial user, programming, administration, and networking commands for the most common Linux distributions. Comprehensive but concise, the fifth edition has been updated to cover new features of major Linux distributions.

Configuration information for the rapidly growing commercial network services and community update services is one of the subjects covered for the first time. But that's just the beginning. The book covers editors, shells, and LILO and GRUB boot options. There's also coverage of Apache, Samba, Postfix, sendmail, CVS, Subversion, Emacs, vi, sed, gawk, and much more. Everything that system administrators, developers, and power users need to know about Linux is referenced here, and they will turn to this book again and again.

Networking Device Drivers Sanjay Dhawan Although device drivers are an integral component of any network, little practical information has been available on their implementation.

Networking Device Drivers fills this void by providing a wealth of in-depth information on a variety of networking device driver architectures, for a number of operating systems. The book begins with a discussion of the basics of networking, various operating systems — such as Windows 3.x, Windows NT, UNIX, DOS, OS/2, and NetWare — and their network architectures. It then provides up-to-date, specific information on a variety of device driver architectures

including: Network Driver Interface Standard (NDIS2 and NDIS3) Open Data-Link Interface (ODI) Packet Driver Data Link Provider Interface (DLPI) The definitive "how-to" book, *Networking Device Drivers* provides ample detail in an easy-to-understand format, allowing readers to use the information from specific chapters to immediately begin work on device driver implementation. Whatever your level of network involvement, this book has something for you: MIS managers will find the extensive information on network architecture and implementation very useful for configuring and maintaining their own networks. Network engineers will find the information about specific network drivers invaluable when designing network related hardware and software applications. Software engineers will find the entire book essential for writing device drivers for network interface cards. Network card manufacturers will find the book useful for teaching engineering staffs about networking device driver architectures. Computer science students will find the book offers a comprehensive introduction to the practical aspects of computer networks. No matter what your involvement with networks, *Networking Device Drivers* should be a part of your reference collection.

Writing UNIX Device Drivers Addison-Wesley Professional

Provides a definitive resource for those who want to support computer peripherals under the Linux operating system, explaining how to write a driver for a broad spectrum of devices, including character devices, network interfaces, and block devices. Original. (Intermediate). 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.

Newly updated to include new calls and techniques introduced in Versions 2.2 and 2.4 of the Linux kernel, a definitive resource for those who want to support computer peripherals under the Linux operating system explains how to write a driver for a broad spectrum of devices, including character devices, network interfaces, and block devices. Original. (Intermediate)

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 Up-to-the-Minute, Complete Guidance for Developing Embedded Solutions with Linux Linux has emerged as today's #1 operating system for embedded products. Christopher Hallinan's Embedded Linux Primer has proven itself as the definitive real-world guide to building efficient, high-value, embedded systems with Linux. Now, Hallinan has thoroughly updated this highly praised book for the newest Linux kernels, capabilities, tools, and hardware support, including advanced multicore processors. Drawing on more than a decade of embedded Linux experience, Hallinan helps you rapidly climb the learning curve, whether you're moving from legacy environments or you're new to embedded programming. Hallinan addresses today's most important development challenges and demonstrates how to solve the problems you're most likely to encounter. You'll learn how to build a modern, efficient embedded Linux development environment, and then utilize it as productively as possible. Hallinan offers up-to-date guidance on everything from kernel configuration and initialization to bootloaders, device drivers to file systems, and BusyBox utilities to real-time configuration and system analysis. This edition adds entirely new chapters on UDEV, USB, and open source build systems. Tour the typical embedded system and development environment and understand its concepts and components. Understand the Linux kernel and userspace initialization processes. Preview bootloaders, with specific emphasis on U-Boot. Configure the Memory Technology Devices (MTD) subsystem to interface with flash (and other) memory devices. Make the most of BusyBox and latest open source development tools. Learn from expanded and updated coverage of kernel debugging. Build and analyze real-time systems with Linux. Learn to configure device files and driver loading with UDEV. Walk through detailed coverage of the USB subsystem. Introduces the latest open source embedded Linux build systems. Reference appendices include U-Boot and BusyBox commands.

"The Encyclopedia of Microcomputers serves as the ideal companion reference to the popular Encyclopedia of Computer Science and Technology. Now in its 10th year of publication, this timely reference work details the broad spectrum of microcomputer technology, including microcomputer history; explains and illustrates the use of microcomputers throughout academe, business, government, and society in general; and assesses the future impact of this rapidly changing technology."

Device drivers literally drive everything you're interested in--disks, monitors, keyboards, modems--everything outside the computer chip and memory. And writing device drivers is one of the few areas of programming for the Linux operating system that calls for unique, Linux-specific knowledge. For years now, programmers have relied on the classic Linux Device Drivers from O'Reilly to master this critical subject. Now in its third edition, this bestselling guide provides all the information you'll need to write drivers for a wide range of devices. 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.

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.

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.

Master the new Windows Driver Model (WDM) common to Windows 98 and Windows 2000. You get theory, instruction and practice in driver development, installation and debugging. Addresses hardware and software interface issues, driver types, and a description of the new 'layer' model of WDM. ;

Master the techniques needed to build great, efficient embedded devices on Linux About This Book Discover how to build and configure reliable embedded Linux devices This book has been updated to include Linux 4.9 and Yocto Project 2.2 (Morty) This comprehensive guide covers the remote update of devices in the field and power management Who This Book Is For

If you are an engineer who wishes to understand and use Linux in embedded devices, this book is for you. It is also for Linux developers and system programmers who are familiar with embedded systems and want to learn and program the best in class devices. It is appropriate for students studying embedded techniques, for developers implementing embedded Linux devices, and engineers supporting existing Linux devices. What You Will Learn Evaluate the Board Support Packages offered by most manufacturers of a system on chip or embedded module Use Buildroot and the Yocto Project to create embedded Linux systems quickly and efficiently Update IoT devices in the field without compromising security Reduce the power budget of devices to make batteries last longer Interact with the hardware without having to write kernel device drivers Debug devices remotely using GDB, and see how to measure the performance of the systems using powerful tools such as `perf`, `ftrace`, and `valgrind` Find out how to configure Linux as a real-time operating system In Detail Embedded Linux runs many of the devices we use every day, from smart TVs to WiFi routers, test equipment to industrial controllers - all of them have Linux at their heart. Linux is a core technology in the implementation of the inter-connected world of the Internet of Things. The comprehensive guide shows you the technologies and techniques required to build Linux into embedded systems. You will begin by learning about the fundamental elements that underpin all embedded Linux projects: the toolchain, the bootloader, the kernel, and the root filesystem. You'll see how to create each of these elements from scratch, and how to automate the process using Buildroot and the Yocto Project. Moving on, you'll find out how to implement an effective storage strategy for flash memory chips, and how to install updates to the device remotely once it is deployed. You'll also get to know the key aspects of writing code for embedded Linux, such as how to access hardware from applications, the implications of writing multi-threaded code, and techniques to manage memory in an efficient way. The final chapters show you how to debug your code, both in applications and in the Linux kernel, and how to profile the system so that you can look out for performance bottlenecks. By the end of the book, you will have a complete overview of the steps required to create a successful embedded Linux system. Style and approach This book is an easy-to-follow and pragmatic guide with in-depth analysis of the implementation of embedded devices. It follows the life cycle of a project from inception through to completion, at each stage giving both the theory that underlies the topic and practical step-by-step walkthroughs of an example implementation. Pajari provides application programmers with definitive information on writing device drivers for the UNIX operating system. The comprehensive coverage includes the four major categories of UNIX device drivers: character, block, terminal, and stream drivers. (Operating Systems) Easy Linux Device Driver : First Step Towards Device Driver Programming Easy Linux Device Driver book is an easy and friendly way of learning device driver programming . Book contains all latest programs along with output screen screenshots. Highlighting important sections and stepwise approach helps for quick understanding of programming . Book contains Linux installation ,Hello world program up to USB 3.0 ,Display Driver ,PCI device driver programming concepts in stepwise approach. Program gives best understanding of theoretical and practical fundamentals of Linux device driver. Beginners should start learning Linux device driver from this book to become device driver expertise. Topics covered: Introduction of Linux Advantages of Linux History of Linux Architecture of Linux Definations Ubuntu installation Ubuntu Installation Steps User Interface Difference About KNOPPIX Important links Terminal: Soul of Linux Creating Root account Terminal Commands Virtual Editor Commands Linux Kernel Linux Kernel Internals Kernel Space and User space Device Driver Place of Driver in System Device Driver working Characteristics of Device Driver Module Commands Hello World Program pre-settings Write Program Printk function Makefile Run program Parameter passing Parameter passing program Parameter Array Process related program Process related program Character Device Driver Major and Minor number API to registers a device Program

to show device number Character Driver File Operations File operation program. Include .h header Functions in module.h file Important code snippets Summary of file operations PCI Device Driver Direct Memory Access Module Device Table Code for Basic Device Driver Important code snippets USB Device Driver Fundamentals Architecture of USB device driver USB Device Driver program Structure of USB Device Driver Parts of USB end points Important features USB information Driver USB device Driver File Operations Using URB Simple data transfer Program to read and write Important code snippets Gadget Driver Complete USB Device Driver Program Skeleton Driver Program Special USB 3.0 USB 3.0 Port connection Bulk endpoint streaming Stream ID Device Driver Lock Mutual Exclusion Semaphore Spin Lock Display Device Driver Frame buffer concept Framebuffer Data Structure Check and set Parameter Accelerated Method Display Driver summary Memory Allocation Kmalloc Vmalloc loremap Interrupt Handling interrupt registration Proc interface Path of interrupt Programming Tips Softirqs, Tasklets, Work Queues I/O Control Introducing ioctl Prototype Stepwise execution of ioctl Sample Device Driver Complete memory Driver Complete Parallel Port Driver Device Driver Debugging Data Display Debugger Graphical Display Debugger Kernel Graphical Debugger Appendix I Exported Symbols Kobjects, Ksets, and Subsystems DMA I/O

“Probably the most wide ranging and complete Linux device driver book I’ve read.”
--Alan Cox, Linux Guru and Key Kernel Developer “Very comprehensive and detailed, covering almost every single Linux device driver type.” --Theodore Ts’o, First Linux Kernel Developer in North America and Chief Platform Strategist of the Linux Foundation

The Most Practical Guide to Writing Linux Device Drivers Linux now offers an exceptionally robust environment for driver development: with today’s kernels, what once required years of development time can be accomplished in days. In this practical, example-driven book, one of the world’s most experienced Linux driver developers systematically demonstrates how to develop reliable Linux drivers for virtually any device. Essential Linux Device Drivers is for any programmer with a working knowledge of operating systems and C, including programmers who have never written drivers before. Sreekrishnan Venkateswaran focuses on the essentials, bringing together all the concepts and techniques you need, while avoiding topics that only matter in highly specialized situations. Venkateswaran begins by reviewing the Linux 2.6 kernel capabilities that are most relevant to driver developers. He introduces simple device classes; then turns to serial buses such as I2C and SPI; external buses such as PCMCIA, PCI, and USB; video, audio, block, network, and wireless device drivers; user-space drivers; and drivers for embedded Linux—one of today’s fastest growing areas of Linux development. For each, Venkateswaran explains the technology, inspects relevant kernel source files, and walks through developing a complete example.

- Addresses drivers discussed in no other book, including drivers for I2C, video, sound, PCMCIA, and different types of flash memory
- Demystifies essential kernel services and facilities, including kernel threads and helper interfaces
- Teaches polling, asynchronous notification, and I/O control
- Introduces the Inter-Integrated Circuit Protocol for embedded Linux drivers
- Covers multimedia device drivers using the Linux-Video subsystem and Linux-Audio framework
- Shows how Linux implements support for wireless technologies such as Bluetooth, Infrared, WiFi, and cellular networking
- Describes the entire driver development lifecycle, through debugging and maintenance
- Includes reference appendixes covering Linux assembly, BIOS calls, and Seq files

A device driver is used in the UNIX system to control specific peripheral devices, such

as floppy disks or cartridge tapes. This is the first book to deal exclusively with writing device driver software, allowing UNIX users to expand their system's flexibility by creating their own device drivers for those not supported by the company marketing the system. In clear and concise language, it provides detailed examples of driver logic, development methods, special requirements, and steps to connecting device driver programs to a variety of systems. Includes numerous sample programs, and an appendix with program listings for all examples.

Device drivers make it possible for your software to communicate with your hardware, and because every operating system has specific requirements, driver writing is nontrivial. When developing for FreeBSD, you've probably had to scour the Internet and dig through the kernel sources to figure out how to write the drivers you need. Thankfully, that stops now. In *FreeBSD Device Drivers*, Joseph Kong will teach you how to master everything from the basics of building and running loadable kernel modules to more complicated topics like thread synchronization. After a crash course in the different FreeBSD driver frameworks, extensive tutorial sections dissect real-world drivers like the parallel port printer driver. You'll learn: –All about Newbus, the infrastructure used by FreeBSD to manage the hardware devices on your system –How to work with ISA, PCI, USB, and other buses –The best ways to control and communicate with the hardware devices from user space –How to use Direct Memory Access (DMA) for maximum system performance –The inner workings of the virtual null modem terminal driver, the USB printer driver, the Intel PCI Gigabit Ethernet adapter driver, and other important drivers –How to use Common Access Method (CAM) to manage host bus adapters (HBAs) Concise descriptions and extensive annotations walk you through the many code examples. Don't waste time searching man pages or digging through the kernel sources to figure out how to make that arcane bit of hardware work with your system. *FreeBSD Device Drivers* gives you the framework that you need to write any driver you want, now.

An exhaustive technical manual outlines the Windows NT concepts related to drivers; shows how to develop the best drivers for particular applications; covers the I/O Subsystem and implementation of standard kernel mode drivers; and more. Original. (Intermediate).

[Copyright: 9a669a79db4b8e0f37724df7dc371cf1](https://www.pdfdrive.com/freebsd-device-drivers-by-joseph-kong.html)