

Linux-Treiber entwickeln

Eine systematische Einführung in die
Gerätetreiber- und Kernel-Programmierung

DAS INHALTS- VERZEICHNIS

» Hier geht's
direkt
zum Buch

Inhaltsverzeichnis

	Vorwort zur 5. Auflage	9
1	Einleitung	11
2	Theorie ist notwendig	19
	2.1 Betriebssystemarchitektur	19
	2.2 Abarbeitungskontext und Unterbrechungsmodell	34
	2.3 Quellensuche	36
3	Kernelcode-Entwicklung in der Praxis	39
	3.1 Auf der Kommandoebene entwickeln	40
	3.2 Fehler finden	50
	3.3 Techniken der Kernelprogrammierung	57
	3.4 Cross-Development	67
	3.5 Nicht vergessen: Auswahl einer geeigneten Lizenz	69
4	Treiber aus Sicht der Applikation	73
	4.1 Die Programmierschnittstelle der Applikation	73
	4.2 Zugriffsmodi	78
5	Einfache Treiber	83
	5.1 Bevor es losgeht	84
	5.2 Cross-Kompilierung	85
	5.3 Den Kernel erweitern	86
	5.4 Die Treibereinsprungspunkte	100
	5.5 Zugriff über IO-Vektoren	122
	5.6 Daten zwischen Kernel- und User-Space transferieren	127
	5.7 Hardware anbinden	131

5.8	PCI	150
5.9	Device Tree	163
5.10	Treiberinstanzen	178
5.11	Treibertemplate: Basis für Eigenentwicklungen	180
6	Fortgeschrittene Kernelcode-Entwicklung	185
6.1	Zunächst die Übersicht	186
6.2	Interrupts	187
6.3	Softirqs	200
6.4	Kernel-Threads	212
6.5	Kritische Abschnitte sichern	222
6.6	Vom Umgang mit Zeiten	257
6.7	Dynamischen Speicher effizient verwalten	265
7	Systemaspekte	277
7.1	Proc-Filesystem	277
7.2	Das Gerätemodell	290
7.3	Green Computing	311
7.4	Firmware-Interface	331
7.5	Module parametrieren	336
7.6	Systemintegration	341
7.7	Kernel Build System	347
7.8	Module automatisiert generieren (DKMS)	354
7.9	Intermodul-Kommunikation	358
7.10	Realzeitaspekte	363
8	Sonstige Treibersubsysteme	367
8.1	GPIO-Subsystem	367
8.2	I ² C-Subsystem	377
8.3	Serial Peripheral Interface (SPI)	387
8.4	USB-Subsystem	396
8.5	Netzwerk-Subsystem	412
8.6	Blockorientierte Gerätetreiber	422
8.7	Crypto-Subsystem	437
8.8	Industrial I/O	461
8.9	Watchdog-Subsystem	477

9	Über das Schreiben eines guten, performanten Treibers	485
9.1	Konzeption.	485
9.2	Realisierung	489
9.3	32 Bit und mehr: portierbarer Code	495
9.4	Zeitverhalten	499
10	Kernel generieren und installieren	503
10.1	Nativ kompilieren: PC-Plattform	505
10.2	Nativ kompilieren: Raspberry Pi	509
10.3	Cross-Kompilieren: PC als Host, Raspberry Pi als Target. . .	511
	Literaturverzeichnis	515
	Index	517