

Inhaltsverzeichnis

1	Einleitung	1
2	Theorie ist notwendig	9
2.1	Betriebssystemarchitektur	9
2.1.1	Komponenten des Kernels	10
2.1.2	Sonstige Betriebssystemkomponenten	23
2.2	Abarbeitungskontext und Unterbrechungsmodell	24
2.3	Quellensuche	27
3	Kernelcode-Entwicklung in der Praxis	31
3.1	Auf der Kommandoebene entwickeln	32
3.1.1	Fehler finden	42
3.2	Techniken der Kernelprogrammierung	52
3.2.1	Coding Style: Kernelcode lesen und Kernelcode schreiben	52
3.2.2	Kernelcode kodieren	54
3.2.3	Objektbasierte Programmierung und Entwurfsmuster im Kernel	56
3.2.4	Hilfsfunktionen	60
3.3	Cross-Development	62
3.4	Nicht vergessen: Auswahl einer geeigneten Lizenz	64
3.4.1	GPL und LGPL	65
3.4.2	MPL und BSD	66
4	Treiber aus Sicht der Applikation	69
4.1	Die Programmierschnittstelle der Applikation	69
4.2	Zugriffsmodi	74
5	Einfache Treiber	79
5.1	Bevor es losgeht	80
5.2	Cross-Kompilierung	82
5.3	Den Kernel erweitern	83
5.3.1	Kernelmodule	83
5.3.2	Vom Modul zum Treiber	88

5.3.3	Einfaches Treibertemplate	91
5.4	Die Treibereinsprungspunkte	95
5.4.1	driver_open: die Zugriffskontrolle	98
5.4.2	Aufräumen in driver_close	101
5.4.3	Lesezugriffe im Treiber	101
5.4.4	Schreibzugriffe im Treiber	111
5.4.5	Die Universalschnittstelle IO-Control	113
5.4.6	Wenn Applikationen mehrere Ein-/Ausgabekanäle überwachen	117
5.5	Daten zwischen Kernel- und Userspace transferieren	120
5.6	Hardware anbinden	124
5.6.1	Datentypen und Datenablage	125
5.6.2	Ressourcenmanagement	126
5.6.3	Direkter Hardwarezugriff	135
5.6.4	Hardware erkennen	140
5.6.5	Device Tree	144
5.6.6	PCI	150
5.7	Treiberinstanzen	163
5.8	Treibertemplate: Basis für Eigenentwicklungen	165
6	Fortgeschrittene Kernelcode-Entwicklung	171
6.1	Zunächst die Übersicht	172
6.2	Interrupts	173
6.2.1	Interruptverarbeitung klassisch	173
6.2.2	Threaded Interrupts	177
6.2.3	Interrupts, testen mit dem Raspberry Pi	181
6.3	Softirqs	189
6.3.1	Tasklets	190
6.3.2	Timer-Funktionen	193
6.3.3	High Resolution Timer	197
6.3.4	Tasklet auf Basis des High Resolution Timers	200
6.4	Kernel-Threads	201
6.4.1	kthread-Daemon	203
6.4.2	Workqueues	206
6.4.3	Event-Workqueue	211
6.5	Kritische Abschnitte sichern	212
6.5.1	Atomare Operationen	213
6.5.2	Mutex und Semaphor	219
6.5.3	Spinlocks	230
6.5.4	Sequencelocks	237
6.5.5	Interruptsperre und Kernel-Lock	240
6.5.6	Synchronisiert warten	241
6.5.7	Memory Barriers	244

6.5.8	Per-CPU-Variablen	246
6.5.9	Fallstricke	246
6.6	Vom Umgang mit Zeiten	248
6.6.1	Relativ- und Absolutzeiten	248
6.6.2	Zeitverzögerungen	254
6.7	Dynamischen Speicher effizient verwalten	257
6.7.1	Buddy-System	258
6.7.2	Objekt-Caching	260
6.7.3	Große Speicherbereiche reservieren	265
6.7.4	Speicher pro Prozessorkern	266
7	Systemaspekte	271
7.1	Proc-Filesystem	272
7.1.1	Schreibzugriffe unterstützen	277
7.1.2	Sequencefiles	280
7.2	Das Gerätemodell	285
7.2.1	Implementierungstechnische Grundlagen	289
7.2.2	Gerätedateien automatisiert anlegen lassen	290
7.2.3	Treiber anmelden	292
7.2.4	Geräte anmelden	294
7.2.5	Attributdateien erstellen	300
7.2.6	Eigene Geräteklassen erstellen	304
7.2.7	Neue Bussysteme anlegen	305
7.3	Green Computing	306
7.4	Firmware-Interface	318
7.5	Treiber parametrieren	324
7.6	Systemintegration	329
7.6.1	Modutils	331
7.6.2	Hotplug	334
7.6.3	Module beim Booten laden	335
7.7	Kernel Build System	335
7.7.1	Treiberquellen als integrative Erweiterung der Kernel- quellen	336
7.7.2	Modultreiber außerhalb der Kernelquellen	340
7.8	Module automatisiert generieren (DKMS)	342
7.9	Intermodul-Kommunikation	347
7.10	Realzeitaspekte	352
8	Sonstige Treibersubsysteme	357
8.1	GPIO-Subsystem	357
8.2	I ² C-Subsystem	362
8.3	Serial Peripheral Interface (SPI)	370
8.4	USB-Subsystem	378

8.4.1	USB programmtechnisch betrachtet	379
8.4.2	Den Treiber beim USB-Subsystem registrieren	383
8.4.3	Die Geräteinitialisierung und die -deinitialisierung ...	385
8.4.4	Auf das USB-Gerät zugreifen	387
8.5	Netzwerk-Subsystem	393
8.5.1	Datenaustausch zur Kommunikation	394
8.5.2	Netzwerktreiber initialisieren	396
8.5.3	Netzwerktreiber deinitialisieren	397
8.5.4	Start und Stopp des Treibers	397
8.5.5	Senden und Empfangen	398
8.6	Blockorientierte Gerätetreiber	403
8.6.1	Bevor es richtig losgeht	406
8.6.2	Daten kerneloptimiert transferieren	408
8.6.3	Grundlegendes zu BIO-Blöcken	414
8.6.4	Treiberoptimierter Datentransfer	418
8.7	Crypto-Subsystem	420
8.7.1	Kleines Einmaleins der Kryptografie	420
8.7.2	Dienste in der Übersicht	423
8.7.3	Eigene Algorithmen einbinden	434
9	Über das Schreiben eines guten, performanten Treibers .	441
9.1	Konzeption	441
9.1.1	Keine halben Sachen	442
9.1.2	Intuitive Nutzung durch Struktur	443
9.1.3	Sicher muss es sein	444
9.1.4	Funktional muss es sein	445
9.2	Realisierung	445
9.2.1	Sicherheitsgerichtetes Programmieren	445
9.2.2	Mit Stil programmieren	446
9.3	32 Bit und mehr: Portierbarer Code	451
9.4	Zeitverhalten	456
	Anhang	461
A	Kernel generieren und installieren	463
A.1	Nativ kompilieren: PC-Plattform	465
A.2	Nativ kompilieren: Raspberry Pi	469
A.3	Cross-Kompilieren: PC als Host, Raspberry Pi als Target	470
B	Makros und Funktionen des Kernels kurz gefasst	475
	Literaturverzeichnis	659
	Index	661