

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	22
2.2	Abarbeitungskontext und Unterbrechungsmodell	23
2.3	Quellensuche	25
3	Treiberentwicklung in der Praxis	29
3.1	Auf der Kommandoebene entwickeln	30
3.1.1	Fehler finden	41
3.2	Techniken der Treiberprogrammierung	50
3.2.1	Coding Style: Kernelcode lesen und Kernelcode schreiben	50
3.2.2	Kernelcode kodieren	52
3.2.3	Objektbasierte Programmierung im Kernel	54
3.2.4	Hilfsfunktionen	58
3.3	Nicht vergessen: Auswahl einer geeigneten Lizenz	60
3.3.1	GPL und LGPL	60
3.3.2	MPL und BSD	61
4	Treiber aus Sicht der Applikation	63
4.1	Die Programmierschnittstelle der Applikation	63
4.2	Zugriffsmodi	68
5	Einfache Treiber	73
5.1	Bevor es losgeht	74

5.2	Den Kernel erweitern	76
5.2.1	Kernelmodule	76
5.2.2	Vom Modul zum Treiber	80
5.2.3	Einfaches Treiber-Template	82
5.3	Die Treibereinsprungspunkte	86
5.3.1	driver_open: die Zugriffskontrolle	89
5.3.2	Aufräumen in driver_close	91
5.3.3	Lesezugriffe im Treiber	92
5.3.4	Schreibzugriffe im Treiber	101
5.3.5	Die Universalschnittstelle IO-Control	103
5.3.6	Wenn Applikationen mehrere Ein-/Ausgabekanäle überwachen	107
5.4	Daten zwischen Kernel- und Userspace transferieren	109
5.5	Hardware anbinden	113
5.5.1	Ressourcenmanagement	114
5.5.2	Datentypen und Datenablage	123
5.5.3	Direkter Hardwarezugriff	125
5.5.4	Hardware erkennen	131
5.5.5	PCI	135
5.6	Treiberinstanzen	148
5.7	Treiber-Template: Basis für Eigenentwicklungen	150
6	Fortgeschrittene Treiberentwicklung	157
6.1	Zunächst die Übersicht	158
6.2	Interrupt-Betrieb	159
6.3	Softirqs	166
6.3.1	Tasklets	167
6.3.2	Timer-Funktionen	170
6.3.3	High Resolution Timer	174
6.3.4	Tasklet auf Basis des High Resolution Timers	176
6.4	Kernel-Threads	178
6.4.1	kthread-Daemon	180
6.4.2	Kernel-Threads auf die rudimentäre Weise erzeugen	183
6.4.3	Workqueues	185
6.4.4	Event-Workqueue	189
6.5	Kritische Abschnitte sichern	190
6.5.1	Atomare Operationen	191
6.5.2	Mutex und Semaphor	195
6.5.3	Spinlocks	206
6.5.4	Sequencelocks	213

6.5.5	Interruptsperre und Kernel-Lock	216
6.5.6	Synchronisiert warten	217
6.5.7	Memory Barriers	220
6.5.8	Per-CPU-Variablen	221
6.5.9	Fallstricke	222
6.6	Vom Umgang mit Zeiten	224
6.6.1	Relativ- und Absolutzeiten	224
6.6.2	Zeitverzögerungen	229
6.7	Dynamischen Speicher effizient verwalten	232
6.7.1	Buddy-System	233
6.7.2	Objekt-Caching	235
6.7.3	Große Speicherbereiche reservieren	240
6.7.4	Speicher pro Prozessorkern	240
7	Systemaspekte	245
7.1	Proc-Filesystem	245
7.1.1	Der lesende Zugriff auf die Proc-Datei	250
7.1.2	Schreibzugriffe unterstützen	255
7.1.3	Sequencefiles	256
7.2	Das Gerätemodell	262
7.2.1	Implementierungstechnische Grundlagen	266
7.2.2	Geräte dateien automatisiert anlegen lassen	267
7.2.3	Treiber anmelden	270
7.2.4	Geräte anmelden	272
7.2.5	Attributdateien erstellen	277
7.2.6	Eigene Geräteklassen erstellen	281
7.2.7	Neue Bussysteme anlegen	282
7.3	Green Computing	282
7.4	Firmware-Interface	294
7.5	Treiber parametrieren	301
7.6	Systemintegration	305
7.6.1	Modutils	307
7.6.2	Hotplug	310
7.7	Kernel Build System	311
7.7.1	Treiberquellen als integrative Erweiterung der Kernel- Quellen	312
7.7.2	Modultreiber außerhalb der Kernel-Quellen	315
7.8	Intermodul-Kommunikation	318
7.9	Echtzeitaspekte	322

8	Sonstige Treibersubsysteme	327
8.1	Blockorientierte Gerätetreiber	327
8.1.1	Bevor es richtig losgeht	330
8.1.2	Daten kerneloptimiert transferieren	332
8.1.3	Grundlegendes zu BIO-Blöcken	338
8.1.4	Treiberoptimierter Datentransfer	341
8.2	USB-Subsystem	344
8.2.1	USB programmtechnisch betrachtet	345
8.2.2	Den Treiber beim USB-Subsystem registrieren	349
8.2.3	Die Geräteinitialisierung und die -deinitialisierung	351
8.2.4	Auf das USB-Gerät zugreifen	352
8.3	Netzwerk-Subsystem	359
8.3.1	Datenaustausch zur Kommunikation	359
8.3.2	Geräteinitialisierung	363
8.3.3	Netzwerktreiber deinitialisieren	365
8.3.4	Start und Stopp des Treibers	365
8.3.5	Senden und Empfangen	366
8.4	Asynchrone Ein-/Ausgabe	371
8.4.1	Applikations-Interface	372
8.4.2	Async-IO im Treiber unterstützen	375
8.4.3	Kernel-Zugriff auf den Userspace	380
9	Über das Schreiben eines guten, performanten Treibers	385
9.1	Konzeption	385
9.1.1	Keine halben Sachen	386
9.1.2	Intuitive Nutzung durch Struktur	387
9.1.3	Sicher muss es sein	388
9.1.4	Funktional muss es sein	388
9.2	Realisierung	388
9.2.1	Sicherheitsgerichtetes Programmieren	388
9.2.2	Mit Stil programmieren	389
9.3	32 Bit und mehr: Portierbarer Code	394
9.4	Zeitverhalten	398
	Anhänge	
A	Kernel generieren und installieren	405
B	Makros und Funktionen des Kernels kurz gefasst	411
C	Literatur- und Quellennachweis	565
	Stichwortverzeichnis	569