Diplomarbeit

zum Thema

Ein zusagenfähiges SCSI-Subsystem für DROPS

an der

Technischen Universität Dresden
Fakultät Informatik
Institut für Betriebssysteme, Datenbanken und Rechnernetze
Lehrstuhl Betriebssysteme

Eingereicht von: Frank Mehnert
Eingereicht am: 15. Januar 1998

Verantwortlicher Hochschullehrer:
Prof. Dr. H. Härtig

Betreuer:
Dr. Claude-Joachim Hamann
Dipl.-Inf. Sebastian Schönberg

Inhaltsverzeichnis

1 Einleitung **9**

2 Stand der Technik **11**

2.1 Verwandte Projekte . . . 11

2.1.1 Videoserver CRAS . . . 11

2.1.2 Tiger Video Fileserver . . . 11

2.1.3 Minimierung der Zugriffszeit . . . 12

2.1.4 Extrahierung von SCSI-Platten-Parametern . . . 13

2.2 Grundlagen des SCSI-Protokolls . . . 13

2.2.1 Datenübertragung zwischen SCSI-Platte und Hostadapter . . . 15

2.2.2 Gesteuertes Vorauslesen (*prefetching*) . . . 16

2.2.3 Feststellen der maximalen *prefetch*-Größe einer Platte . . . 16

2.2.4 Zeitlicher Ablauf von SCSI-Kommandos . . . 17

2.3 SCSI-Festplatten-Modell . . . 18

2.3.1 Parameter . . . 18

2.3.2 Interne Caches . . . 18

2.3.3 *Mode pages* . . . 19

2.4 Der Linux-SCSI-Treiber . . . 20

2.4.1 Der Aufbau in drei Schichten . . . 20

2.4.2 Verwaltung der SCSI-Geräte . . . 22

2.4.3 Warteschlangen im Linux-Treiber . . . 22

3 Entwurf **27**

3.1 Entwurfsziele . . . 27

3.2 Modelle zum Plattenscheduling . . . 27

3.2.1 Begriffsdefinitionen . . . 27

3.2.2 Grundmodell . . . 28

3.2.3 Periodisches Vorauslesen (*prefetching*) . . . 29

3.2.4 Periodisches Lesen ohne Vorauslesen . . . 29

3.2.5 Das Slot-Modell 31
3.2.6 Verallgemeinerung für Datenschreiben 31
3.3 Injektion der SCSI-Aufträge 31
3.4 Planungsmodelle 32
3.4.1 Admission und SCSI-Planung getrennt 32
3.4.2 Admission verbunden mit SCSI-Planung 33
3.5 Schnittstellen-Beschreibung 35
3.6 Beachtung der Nebenläufigkeiten 36

4 Implementierung **37**
4.1 Emulationsumgebung für SCSI-Treiber 37
4.1.1 *Memory mapped Input/Output* 37
4.1.2 Hardwareunterbrechungen 38
4.1.3 Systemzeit 38
4.2 Threadstruktur 38
4.3 Synchronisationsmittel 39
4.3.1 Kritische Abschnitte 40
4.3.2 Semaphore 40
4.3.3 Expliziter Aufruf des Schedulers 40
4.4 Schnittstelle 41
4.4.1 Aufbau der Aufträge 41
4.5 Stand der Implementierung 42

5 Leistungbewertung **43**
5.1 Last-Messungen 43
5.2 Performance 44

6 Zusammenfassung und Ausblick **47**

A Glossar **49**

Abbildungsverzeichnis

2.1 Synchronisation mittels *Time-Driven Shared Memory Buffer* 12
2.2 Aufbau des SCSI-III-Standards 14
2.3 Konkurrenzsituation zwischen verschiedenen SCSI-Geräten 20
2.4 Blockauftrags-Warteschlange 23
2.5 Verteilung der SCSI-Aufträge auf die einzelnen Hosts 23
2.6 Warteschlange im hardwareabhängigen Teil des SCSI-Treibers 24
2.7 Warteschlangen auf Geräteebene 25

3.1 Begriffsdefinitionen für Kapitel 3 28
3.2 Scheduling durch periodisches *prefetching* 29
3.3 Periodisches Absetzen der Plattenaufträge ohne *prefetching*, Spezialfall 1 30
3.4 Periodisches Absetzen der Plattenaufträge ohne *prefetching*, Spezialfall 2 31
3.5 Erste Variante des Planungsmodells 32
3.6 Auftragsstruktur für den SCSI-Treiber 33
3.7 Verbesserte Variante des Planungsmodells 34
3.8 Übergang von der Systemzeit auf diskrete Zeitangaben 35
3.9 Überlappung benachbarter Slots 35
3.10 Untersuchung umgebener Slots 35

4.1 Thread-Aufbau des SCSI-Treibers 39
4.2 Verbesserte Auftragsstruktur 42

5.1 Beeinflussung zwischen SCSI-Bus und CPU 43
5.2 Slotlänge t_s in Abhängigkeit der Blockgröße und des Positionierbereiches 45
5.3 Erzielbare Datenraten (*worst case*) in Abhängigkeit der Blockgröße 45

Tabellenverzeichnis

2.1 Kenngrößen bestehender SCSI-Standards 15
2.2 Kenngrößen ausgewählter SCSI-Festplatten 15
2.3 Zeitliche Dauer von SCSI-Phasen 17
2.4 Durch *mode pages* veränderbare Einstellungen bei SCSI-Geräten 19

4.1 Mögliche Zustände der Threads des SCSI-Treibers 40

5.1 Beeinflussung von SCSI- durch Hauptspeicheroperationen 44

Kapitel 1

Einleitung

Am Lehrstuhl Betriebssysteme der TU Dresden wird seit einigen Jahren an dem Projekt DROPS (*The Dresden Real Time Operating System Project*) gearbeitet, bei dem Möglichkeiten zur Unterstützung von *Qualitiy of Service (QoS)* Parametern in Betriebssystemen untersucht werden sollen. Grundlage dieses Projektes ist der Mikrokern L4. Er zeichnet sich vor allem durch sein Minimalkonzept und die gute Performance aus. Als Umgebung für nicht-zeitkritische Programme existiert ein Linux-Server auf L4 (L^4Linux ([Hoh96]).

Ein größeres Teilprojekt von DROPS stellt den Entwurf und die Implementierung eines Dateisystems dar, das neben L^4Linux auf L4 ablaufen und sowohl Aufträge mit zeitlichen Zusagen, als auch nicht-zeitkritische Aufträge ausführen können soll. In das Dateisystem soll ein SCSI-Subsystem eingebettet werden, welches Aufträge zeitgesteuert an die Hardware weitergeben kann und so unabhängig von der Systemlast Zusagen einhält.

Der SCSI-Standard sieht allerdings keine Unterstützung für Echtzeitverarbeitung vor. Die Entwicklung verlief bisher eher in Richtung Kompatibilität und Anwendertransparenz. Durch geeignetes Ansteuern kann aber unter Zugrundelegung der Extremwerte (*worst case*) eine Aussage über das zeitliche Verhalten von Festplatten getroffen werden.

Aus Zeitgründen wird auf das Neuschreiben des SCSI-Treibers verzichtet, es soll vielmehr versucht werden, den SCSI-Treiber aus Linux durch Verwendung einer Emulationsbibliothek möglichst unverändert zu übernehmen und durch Erweiterungen an die speziellen Gegebenheiten unter L4 anzupassen.

Im folgenden Kapitel werden technische Grundlagen zum SCSI-Standard erläutert, Projekte umrissen und Erkenntnisse aus dem Aufbau des Linux-SCSI-Treibers gewonnen. In Kapitel 3 werden Lösungsansätze zu verschiedenen Teilproblemen diskutiert. Kapitel 4 gibt den aktuellen Stand der Implementierung wieder, während im fünften Kapitel erste praktische Messungen durchgeführt werden. Das letzte Kapitel schließlich zeigt zusammenfassende Bemerkungen und liefert Ansätze für fortführende Arbeiten.

Kapitel 2

Stand der Technik

2.1 Verwandte Projekte

Auf dem Gebiet der Multimedia-Dateisysteme findet man eine große Anzahl an Arbeiten, hier sollen einige zu Echtzeitanforderungen bei SCSI-Subsystemen und Plattenscheduling umrissen werden.

2.1.1 Videoserver CRAS

In [TN96] wird eine Arbeit über einen Videoserver (CRAS, *Constant Rate Access Server*) basierend auf Real-Time Mach beschrieben.

Grundlage ist die Definition einer *Interval Time*. In jedem Intervall werden die Daten gelesen, die im nächsten Intervall verarbeitet werden sollen. Die Länge dieser Intervalle richtet sich nach der maximalen Anzahl der Ströme und der Anfangslatenz der auszugebenen Ströme. Eine Durchsatzoptimierung wird durch Lesen großer Blöcke bis 256kB und durch Sortieren der Zugriffe nach Zylindernummern versucht.

An Real-Time Mach selbst wurden zwei Änderungen durchgeführt, um bessere Echtzeiteigenschaften zu erhalten: Die Warteschlange des Plattentreibers wird in eine Schlange für normale Zugriffe und eine für Echtzeitaufträge aufgeteilt. Außerdem wird ein spezielles Interface für das Lesen großer Datenblöcke definiert. Während unter Real-Time Mach normalerweise für jeden Plattenzugriff Speicher im Kern reserviert werden muß, kann man mit diesem Interface Puffer im Userspace verwenden.

Um möglichst wenig Seitenfehler zu provozieren, wird der gesamte Server so klein wie möglich gehalten. Während der Lieferung konstanter Bitraten vermeidet es der Server, auf Dienste von Nicht-Echtzeit-Servern zuzugreifen.

Die Datenübertragung zwischen CRAS und der Anwendung findet über einen gemeinsamen Speicherbereich (*shared memory*) statt. Um eine Anpassung der Lieferdatenrate des Servers und der Konsumierungsdatenrate der Anwendung zu erreichen, setzt man sogenannte *Time-Driven Shared Memory Buffer* (Abbildung 2.1 auf der nächsten Seite) ein. Die Synchronisation erfolgt über logische Zeitstempel.

2.1.2 Tiger Video Fileserver

Von Microsoft stammt eine Arbeit über Verteiltes Scheduling. [WJB97] beschreibt innerhalb eines Fileservers für Videodaten einen Mechanismus zur Planung von Plattenaufträgen. Ziel ist es, sogenannte *hotspots* zu vermeiden, die entstehen, wenn eine Platte mehr Daten liefern soll, als sie in der Lage ist.

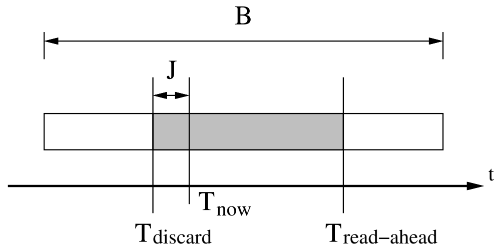

Abbildung 2.1: *Time-Driven Shared Memory Buffer* zur Synchronisation zwischen Server und Anwendung in CRAS ([TN96]).

Ein einzelner Server (*cub*) des Tiger-Systems unterstützt eine einzige Bitrate, mehrere dieser Server mit jeweils verschiedenen Bitraten können über Netzwerk (ATM) zusammengeschlossen werden.

Die Dateien werden round-robin auf allen Platten abgelegt. Die *block play time* ist die Dauer, für die ein Block angezeigt wird. Für Video-Anwendungen (1-10MBit/s) beträgt sie typischerweise eine Sekunde. Die *block play time* ist gleich für alle Dateien eines Tiger-Systems. Die *block service time* ist die benötigte Zeit, um einen Block zu lesen. Sie wird durch die Datenrate der Platten bzw. die Kapazität der Netzwerkschnittstelle bestimmt.

Als zentrale Idee wird jede Platte wird in Slots mit der Länge der *block service time* aufgeteilt. Das Produkt der Anzahl der Platten und der *block play time* ergibt die gesamte Planungsperiode. Deren Länge muß ein Vielfaches der *block play time* und der *block service time* sein, ggf. wird die *block service time* entsprechend vergrößert.

Pro Platte wird ein Zeiger auf den zentralen Plan verwaltet, der in Echtzeit bewegt wird. Erreicht der Zeiger einer Platte den Anfang eines Slots, beginnt die Platte mit dem Senden des entsprechenden Blockes. Ein Vorlauf wird erreicht, indem die Platten mindestens um eine *block service time* eher angesprochen werden.

Um den Einfluß der Positionierzeiten (*seeks*, siehe Abschnitt 2.3 auf Seite 18) zu minimieren, sollen möglichst große zusammenhängende Blöcke gelesen werden.

Ist Tiger für Fehlertoleranz konfiguriert, wird die *block service time* erhöht, damit eine zweite Platte die gespiegelten Daten verwalten kann.

2.1.3 Minimierung der Zugriffszeit

Viele Projekte befassen sich mit Algorithmen zum Plattenscheduling und suchen nach optimalen Verfahren, um die Positionierzeit zu minimieren und dabei spezielle Platteneigenschaften (z.B. *Zoning*) zu berücksichtigen. In [WGP94] zeigen Worthington, Ganger und Patt bemerkenswerte Erkenntnisse, die aus Simulationen und realen Nutzerumgebungen gewonnen wurden:

- Eine Berücksichtigung komplexer *Mapping*-Informationen im Scheduler ermöglicht nur eine sehr geringe Verkürzung der Antwortzeit bei Algorithmen zur Reduktion von Kopfbewegungen.

- Algorithmen, die *prefetch*-Caches effektiv nutzen, erlauben eine signifikante Performancesteigerung beim sequentiellen Lesen. Der C-LOOK-Algorithmus (*cyclical SCAN algorithm*), der

Aufträge nach logischen Blocknummern aufsteigend sortiert, erzielt dann die höchste Performance. Bei zufällig verteiltem Lesen schneiden die Verfahren SSTF (*Shortest Seek Time First*) und LOOK (Variation von *SCAN*) geringfügig besser ab.

- Algorithmen zum Minimieren der Gesamt-Positionierzeiten erreichen die besten Ergebnisse, vorausgesetzt, sie erkennen und nutzen vorhandene *prefetch*-Caches.
- Schreiben bei eingeschaltetem Schreib-Cache auf den Festplatten erfordert eine Sonderbehandlung.

2.1.4 Extrahierung von SCSI-Platten-Parametern

In [WGPW96] wird versucht, ein sehr genaues Modell von SCSI-Festplatten aufzustellen. Einige Parameter (fehlerhafte Sektoren, logisches Sektor-Mapping) lassen sich direkt aus der Firmware auslesen (siehe Abschnitt 2.3.3 auf Seite 19). Andere müssen aufwendiger durch spezielle Algorithmen erforscht werden, wie z.B. minimale Zeit zwischen der Beendigung zweier Aufträge (*minimum time between request completions*), Zugriffscharakteristik, Kopfumschaltzeit, Umdrehungsgeschwindigkeit, Kommandoausführungszeit und -*overhead*, Größe und Zugriffsverfahren des Plattencaches. Weil die Platte als *black-box* angesehen werden muß, kann nicht immer genau entschieden werden, wie groß der Einfluß der anderen Komponenten (Local-Bus, PCI-Bus, SCSI-Hostadapter) auf dem Datenpfad zur Platte ist.

2.2 Grundlagen des SCSI-Protokolls

Seit 1979 gibt es Bestrebungen für die Entwicklung einer allgemeinen Schnittstelle für die Anbindung unterschiedlicher Komponenten an Personalcomputer. Als der Entwurf zum SCSI-I-Standard (*Small Computer Systems Interface*) 1986 von der ANSI (*American National Standard Institution*) gebilligt wurde, war die Entwicklung von Festplatten noch nicht so weit vorangeschritten, daß sie den neuen Bus auch vollständig ausnutzen konnten. In den folgenden Jahren erfuhr der Standard aber ständig Erweiterungen, um mit der immer schnelleren Entwicklung der Speichermedien mithalten zu können. Dabei wurde stets sorgfältig auf Abwärtskompatibilität geachtet.

Der heute aktuelle SCSI-III-Standard bietet folgende Eigenschaften:

- Unterstützung verschiedener Geräte- (Festplatten, Prozessoren, Bandlaufwerke, CD-ROM) und Anschlußarten (siehe auch Abbildung 2.2 auf der nächsten Seite)
- Skalierbarkeit: Bis zu 7 bzw. 15 (*WIDE*) Geräte (zzgl. Hostadapter) lassen sich an einen SCSI-Bus anschließen
- Übertragung der Steuerbefehle asynchron (*request-acknowledge-handshake*)
- relativ geringe Beeinflussung mehrerer am Bus arbeitender Geräte, Übertragung der Daten immer mit voller Busgeschwindigkeit (synchron)
- hohe Eigenintelligenz der Geräte (Transparenz bezüglich physischer Anordnung der Datenblöcke, Umsortierung von Aufträgen möglich, automatische Behandlung fehlerhafter Blöcke)
- Optimierung für mittlere bis kleine Blockgrößen aufgrund der Anforderungen aktueller Betriebssysteme
- große Kabellängen (3m bei *single ended*, 25m bei *differential ended*) möglich

- immer nur ein Gerät kann zu einer bestimmten Zeit den Bus belegen und Daten senden – kein *interleaving*

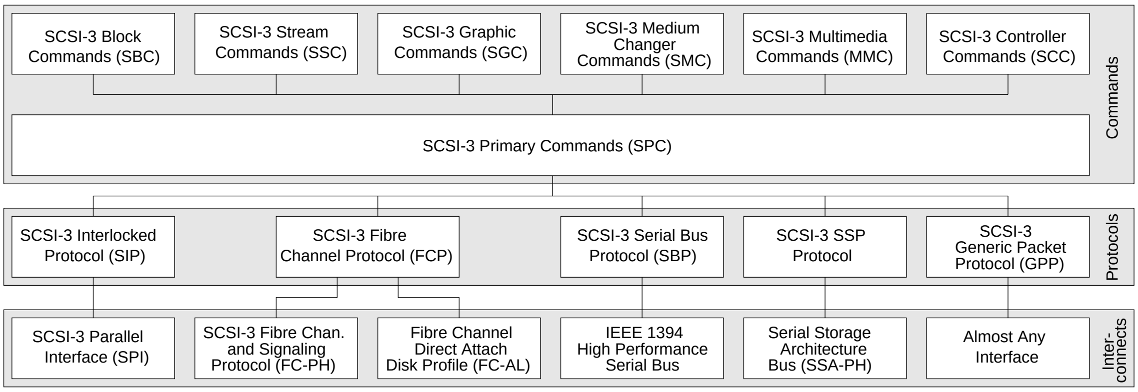

Abbildung 2.2: Der SCSI-Standard umfaßt ein vielschichtiges Modell, das für die verschiedensten Schnittstellen geeignet ist (siehe [Sch95]).

Bestimmte SCSI-Eigenschaften wirken sich negativ auf die Verwendung in Echtzeitumgebungen aus:

- Schlechte Vorhersagbarkeit der Abarbeitung von Aufträgen infolge der hohen Eigenintelligenz der Geräte
- festes, prioritätsbezogenes Scheduling
- große Unterschiede bei der Implementierung des Standards in Geräten; viele SCSI-Kommandos sind optional
- typische Effizienz (unter realen Bedingungen erreichbarer Datendurchsatz im Vergleich mit theoretischem Maximum) von 60-85% (siehe [Sch95])

Einige dieser Nachteile werden in zukünftigeren Standards zur Datenübertragung beseitigt (serielle Busse, z.B. **Fire Wire**, **FC-PH**, **FC-AL**, **SSA**).

Trotz der beim aktuellen SCSI-III-Standard bestehenden Einschränkungen in Bezug auf Echtzeitfähigkeit soll versucht werden, ein Scheduling der SCSI-Aufträge zu implementieren, das weichen Echtzeitanforderungen genügt. Harte Echtzeit wird mit gegenwärtig erhältlichen SCSI-Systemen nicht möglich sein, da sich ein Programm niemals voll darauf verlassen kann, daß ein Kommando in der vorgegebenen Zeit wirklich ausgeführt wurde, vielmehr kann man dies nur mit hoher Wahrscheinlichkeit annehmen (siehe auch [Sch93]). Diese Nicht-Vorhersagbarkeit liegt begründet in

- **Behandlung fehlerhafter Sektoren.** Bei Auftreten von physischen Plattendefekten werden die in Frage kommenden Sektoren als fehlerhaft gekennzeichnet und durch Reservesektoren ausgetauscht. Zwar kann man anhand eines (optional implementierten) Kommandos zum Auslesen der Defektliste abfragen, welche Plattensektoren betroffen sind. Allerdings würde für Echtzeitanforderungen nur in Frage kommen, besagte Sektoren nicht mehr zu verwenden. Damit ergeben sich aber negative Einflüsse dadurch, daß bisher aufeinanderfolgende Sektoren plötzlich durch unbenutzbare Bereiche unterbrochen werden. Ein weiteres Problem ist die physische Alterung von Festplatten, die ein Neuschreiben von Dateien erforderlich macht, weil von Zeit zu Zeit neue defekte Bereiche auftreten.

- **Thermische Einflüsse** machen eine Rekalibrierung der Plattenköpfe in nicht vorhersagbaren zeitlichen Abständen erforderlich. In dieser Zeit werden keine Aufträge ausgeführt. Zwar versucht die Platte, eine Kalibrierung nur dann auszuführen, wenn längere Zeit keine Aufträge an das Gerät gesendet wurden, dennoch kann nicht von außen bestimmt werden, wann dieser Zeitpunkt gekommen ist. Neueste Festplatten kommen angeblich ohne thermische Rekalibrierung aus ([Bög97]).
- **Verhalten anderer Geräte am Bus.** Insbesondere das Verhalten einer Platte im Zusammenwirken mit anderen Geräten am Bus ist nicht genau genug vorhersagbar.

2.2.1 Datenübertragung zwischen SCSI-Platte und Hostadapter

Im Allgemeinen übersteigt die Bandbreite des SCSI-Busses die maximal erreichbare Datenrate einer SCSI-Festplatte beim physischen Lesen/Schreiben um ein Vielfaches (vergleiche Tabellen 2.1 und 2.2).

Tabelle 2.1: Kenngrößen bestehender SCSI-Standards. Aktuell ist SCSI-III UW (*ultra wide*)

Standard	Marktbezeichnung	Protokoll	Periode	Datenbreite	Bandbreite
SCSI-I		asynchron [a]	-	8Bit	≈ 3MB/s
	slow	synchron	200ns	8Bit	5MB/s [b]
SCSI-II	**fast**	synchron	100ns	8Bit	10MB/s
	wide	synchron	100ns	16Bit	20MB/s
SCSI-III	**ultra**	synchron	50ns	8Bit	20MB/s
	ultra wide	synchron	50ns	16Bit	40MB/s
	ultra-2 wide [c]	synchron	50ns	16Bit	80MB/s

[a] Asynchron bedeutet Übertragung per *request acknowledge handshake*
[b] MB/s steht hier für Millionen Byte pro Sekunde
[c] LVD (englow voltage differential), nur bedingt abwärtskompatibel durch veränderte Signalpegel

Tabelle 2.2: Ausgewählte SCSI-Festplatten und deren Herstellerangaben über die wichtigsten Plattenparameter (aus [IBM97] und [Sea97])

Hersteller	Modell	Kapazität GB	UPM	Datenrate MB/s	$\mathbf{t_{seek_{max}}}$ ms
Seagate	Hawk 2LP (ST-31230N)	1,2	5411	4,0-6,7	21,4
Seagate	Cheetah 4LP (ST-34501N)	4,5	10033	11,3-16,8	19,2
Seagate	Cheetah 9 (ST-19101W)	9,1	10033	11,3-16,8	20,2
IBM	Ultrastar ES (DORS 32160)	2,2	5400	3,8-5,7	15
IBM	Ultrastar 2ES (DCAS 32160)	4,3	5400	5,0-8,0	15
IBM	Ultrastar XP (DFHS C4X)	4,5	7200	5,5-7,4	16,5
IBM	Ultrastar 2XP (DCHS 39100)	9,1	7200	6,6-10,1	18

Daraus schlußfolgernd stellt sich die Frage: Wie wird vermieden, daß eine Festplatte bei der Übertragung eines größeren Datenblocks den Bus unnötig lange blockiert?

Der Ablauf einer Datenübertragung besteht aus mehreren SCSI-Busphasen: Nach der *FREE*-Phase bewerben sich alle Geräte, die jetzt den Bus für sich beanspruchen (*ARBITRATION*). Die ranghöchste ID gewinnt (der Hostadapter hat i.d.R. die höchste ID) und belegt den Bus (*SELECTION*). Dieses Gerät ist bis zum Abschluß der gesamten Transaktion der Initiator. Gleichzeitig wird ein anderes Gerät, genannt Target, angesprochen. Ab diesem Zeitpunkt übernimmt das Target die Kontrolle über den Busablauf.

Es werden einige *MESSAGE*- und Kommandobytes zwischen beiden Geräten ausgetauscht. Das Target beginnt jetzt intern mit der Abarbeitung des SCSI-Kommandos. Meistens muß für die Erfüllung des Auftrages erst ein Sektor auf der Platte gesucht und vom Schreib-/Lesekopf angefahren werden. Während dieser *seek*-Zeit unterbricht das Target vorübergehend die Verbindung zum Bus (*disconnect*), infolgedessen er jetzt für andere Geräte verfügbar wird. Auch kann der Initiator, wenn er nicht auf das Ergebnis seines eben abgesetzten SCSI-Befehles warten muß, weitere Befehle an andere SCSI-Geräte senden, sogar an das eben beauftragte Target. Dieses beginnt in dieser Zeit mit dem Einlesen der ersten Sektoren in den internen Cache. Wenn dieser einen gewissen Füllstand erreicht hat, bemüht es sich um die Wiederaufnahme der Kommunikation mit dem Initiator. Nach erfolgreichem *reconnect* werden die gelesenen Sektoren zum Initiator übertragen. Die *STATUS*-Phase bildet den Abschluß des gesamten Ablaufes.

Da der Plattencache im Allgemeinen zu klein ist, um den gesamten angeforderten Datenblock auf einmal aufzunehmen, trennt sich das Target erneut vom Bus (*disconnect*), wenn alle Daten aus dem Cache übertragen sind. Der Bus ist wieder frei, und das Target liest weitere Sektoren in den Zwischenspeicher, um sie nach einem erneuten *reconnect* wieder an den Initiator zu übertragen.

Mit dieser Abfolge ist gewährleistet, daß eine Übertragung von Daten immer mit der maximalen zulässigen Geschwindigkeit auf dem Bus erfolgen kann und andere Geräte nicht zu lange warten müssen, um ihrerseits den Bus belegen zu können. Der offensichtliche Nachteil ist, daß nie genau vorhergesagt werden kann, wann ein Target in eine *disconnect*/reconnect-Phase geht. Heutige moderne SCSI-Geräte besitzen anscheinend sogar die Fähigkeit, am SCSI-Bus zu lauschen. Wenn kein weiteres Gerät Daten übertragen will, dann geben sie auch nicht den Bus frei, um die für Trennung und Verbindung anfallenden Zeiten einzusparen ([Ess97]). Eine einzelne Festplatte am Bus könnte dann während der ganzen Übertragungsphase auch bei größeren Datenblöcken Busmaster bleiben (und den Bus zu 100% auslasten), weil sie andere Geräte nicht beeinflußt.

2.2.2 Gesteuertes Vorauslesen (*prefetching*)

Viele SCSI-Festplatten neueren Herstellungsdatums unterstützen ein sogenanntes *prefetching*. Aufgrund einer wahrscheinlichen Lokalität der Daten auf der Platte geht deren Firmware davon aus, daß ein Zugriff auf den Block x einen Zugriff auf Block $x + 1$ zur Folge hat, auch wenn zum Zeitpunkt des Lesens noch kein expliziter Auftrag mit einem diesbezüglichem Parameter vorliegt. Aufgrund dieser Annahme liest eine Platte, einen gewissen Leerlauf vorausgesetzt, immer mehr Sektoren als angefordert in den internen Cache.

2.2.3 Feststellen der maximalen *prefetch*-Größe einer Platte

Um programmiertes Vorauslesen auf einer Platte durchführen zu können, ist die Kenntnis der maximal zulässigen Datenblock-Größe, die mit einem *prefetch*-Kommando gelesen werden kann, erforderlich. SCSI-Festplatten besitzen sogenannte *modepages*, Statusseiten, von denen Parameter gelesen und in gewissen Grenzen verändert werden können (siehe auch 2.3.3 auf Seite 19). Allerdings läßt sich beobachten, daß diese Seiten fast nie vollständig in der Platten-Firmware implementiert sind.

Als Alternative bietet es sich an, ein *prefetch*-Kommando mit einer bestimmten Datenlänge an die Platte abzuschicken und anhand des zurückgelieferten Statuswortes festzustellen, ob der Block korrekt eingelesen wurde. Die Länge kann dann schrittweise erhöht werden, bis der Block nicht mehr in den *prefetch*-Cache gelesen wird, woraufhin die Platte einen Fehlerkode zurückliefert. Die gesuchte maximal zulässige *prefetch*-Größe ergibt sich aus dem letzten zuverlässig verarbeiteten Wert.

Bei Festplatten aktueller Bauart läßt sich eine maximale *prefetch*-Größe von 64kB feststellen. Dieser Parameter kann allerdings über sogenannte *mode pages* verändert werden, sofern dies in der Plattenfirmware vorgesehen ist (Abschnitt 2.3.3 auf Seite 19).

2.2.4 Zeitlicher Ablauf von SCSI-Kommandos

Grundsätzlich erfolgt die Übertragung von Kommandos und Statusbytes asynchron. Damit wird eine Abwärtskompatibilität innerhalb des SCSI-Standards sichergestellt. Die synchrone Übertragungsart ist für Daten die bevorzugte, da sie höhere Geschwindigkeiten (bis zu 40MB/s) zuläßt.

In Tabelle 2.3 werden alle SCSI-Busphasen im Hinblick auf den zeitlichen Ablauf betrachtet. Als Beispiel dient die Übertragung von Daten mittels *READ(10)*-Kommando. Es wird bewußt auf die Verwendung von SCSI-Signal-Bezeichnungen verzichtet, um eine einfachere Verständlichkeit zu erzielen. Die Zeitangaben sind [Sch93] entnommen und geben die einzuhaltenen Höchstwerte des SCSI-Standards wieder.

Reale Zeiten lassen sich nur mit einem Logikanalysator erfassen. Umfangreiche softwareseitige Messungen an einem SCSI Hostadapter Sym53C875 mit angeschlossenen UW-Platten IBM DORS 32160 ergaben eine Ausführungszeit eines SCSI-Read-Befehls ohne Kopfbewegung, Unterbrechung (*disconnect*) und Transfer von etwa $200\mu s$. Die Ausführungszeit erhöht sich gravierend, wenn pro Auftrag mehrere Unterbrechungen der Übertragung stattfinden. Beispielsweise wird der SCSI-Befehl „Lesen“ eines 64kB Blockes etwa 3-5 mal unterbrochen. Im ungünstigsten Fall (maximale Positionierzeit, ungünstigste Lage, Konkurrenz mit mehreren Platten am Bus) werden zum Lesen eines Datenblockes dieser Größe etwa 30-40ms benötigt.

Tabelle 2.3: Dauer von SCSI Phasen bei Ausführung eines *READ*- Kommandos

Phase	Dauer	Ablauf
FREE	$0,4\mu s$	Die *FREE*-Phase ist so definiert, daß der Bus länger als 400ns nicht aktiv ist.
	$0,8\mu s$	Nachdem sich eine *FREE*-Phase eingestellt hat, muß ein Gerät mindestens 800ns warten, bis es eine Arbitrierung starten darf.
ARBITRATION	$2,4\mu s$	Für eine Zeit von $2,4\mu s$ muß jedes Gerät warten, ob sich noch ein anderes Gerät um den Bus bewirbt.
	$< 0,8\mu s$	Das Gerät mit der höchsten Priorität muß dann maximal 800ns warten, bis alle anderen Geräte, die sich beworben haben, ihre SCSI-ID und ihr $BUSY$-Signal zurücknehmen
SELECTION	$< 200ms$	Der Initiator stellt die Verbindung mit dem Target her, welches den Bus innerhalb von 200ms belegen muß.
MESSAGE	?	Eine oder mehrere Message-Bytes werden zwischen Target und Initiator mittels *request-acknowledge-handshake* übertragen (z.B. Aushandlung der Datenbreite). Zeit auch abhängig von der Kabellänge.
COMMAND	?	Die Übertragung der eigentlichen Kommando-Bytes (hier 10) vom Initiator zum Target erfolgt wie die Übertragung von Messages. Zeit auch abhängig von der Kabellänge.
DATA	$n \times 0,05\mu s$	Eine Datenphase dauert bei einer Übertragung nach SCSI-III-Standard 50n, dabei werden 8 oder 16 Bit (*wide*) übertragen.
STATUS	?	Wie *MESSAGE*

2.3 SCSI-Festplatten-Modell

2.3.1 Parameter

Eine Festplatte kennzeichnet sich durch verschiedene Parameter in Bezug auf Behandlung von Schreib-/Leseaufträgen aus. Folgend sollen die wichtigsten definiert werden:

***Interface*-Transferrate** ist die Datenrate, mit der die Platte Daten über den Bus bewegt. Festplatten nach SCSI-III übertragen 40 Millionen Byte pro Sekunde (ein Datenperiode von 50ns, siehe Abschnitt 2.2.4 auf der vorherigen Seite) bzw. 38,14 MB/s.

Rotationsgeschwindigkeit ist die Anzahl Umdrehungen pro Minute des Plattenstapels. Übliche Werte liegen bei $4500\frac{1}{min}$, $\mathbf{5400\frac{1}{min}}$, $7200\frac{1}{min}$ oder $10000\frac{1}{min}$ (siehe auch Tabelle 2.2 auf Seite 15).

Medien-Transferrate Diese Datenrate wird von der Anzahl der Sektoren pro Spur bestimmt und variiert innerhalb einer Festplatten, da nicht alle Spuren die gleiche Anzahl Sektoren enthalten (*Zoning*).

effektive Datenrate Die effektive Datenrate ist der Quotient aus der Länge des Datenblockes und der Zeit, die insgesamt benötigt wird, um den Datenblock zu übertragen. Die Übertragung eines Datenblockes erfolgt meist in kleinen Stücken, da die physische Lese-/Schreibrate der Plattendaten meist wesentlich kleiner als die Datenrate des SCSI-Busses ist (siehe Abschnitt 2.2.1 auf Seite 15). Die effektive Datenrate ist deshalb ebenfalls kleiner, als die *Interface*-Transferrate.

Kommandozeit ist die Zeit, in der die Platte auf dem SCSI-Bus selektiert wird und den Befehl vom Hostadapter in Empfang nimmt.

Positionierzeit (*seek*) In dieser Zeit wird der Schreib-/Lesekopf von der aktuellen Position auf den Beginn des zu lesenden/schreibenden Datenblocks geführt. Dabei wird zunächst elektrisch auf den richtigen Datenpfad (Kopf) umgeschaltet, dann die Spur über einen Servomotor angefahren, eine Feinjustierung anhand der Positionierdaten auf der Platte durchgeführt und solange gewartet, bis die Platte die richtige Winkelposition erreicht hat. Währenddessen werden bereits die sich unter dem Kopf vorbeidrehenden Sektoren in einen speziellen Spurpuffer gelesen, aufgrund der Wahrscheinlichkeit, diese später nutzen zu können (siehe 2.3.2).

Zugriffszeit ist die Zeit, in der ein Datenblock bestimmter Größe und Plattenposition eingelesen bzw. geschrieben wird. Sie ergibt sich aus der Summe von effektiver Datenrate, Kommandozeit und Positionierzeit. Die Positionierzeit nimmt im Normalfall die meiste Zeit in Anspruch.

Die großen Schwankungen der effektiven Datenrate, die sich aus dem physischen Aufbau der Festplatte ergeben, müssen entweder im Datenlayout oder im Scheduling berücksichtigt werden, anderenfalls kann nicht die volle Plattenbandbreite ausgenutzt werden.

2.3.2 Interne Caches

Jede SCSI-Platte besitzt einen internen schnellen flüchtigen Speicher (RAM), der aus mehreren Teilen verschiedener Funktion besteht:

Spur-Cache Dieser Cache nimmt genau soviel Sektoren auf, wieviel die längste Spur fassen kann. Erreicht der Schreib-/Lesekopf der Platte nach einer Neupositionierung die Zielspur, beginnt die

Platte sofort mit dem Einlesen der Spur, auch wenn sich der zu lesende Sektor noch nicht unter dem Kopf befindet. Aufgrund der hohen Wahrscheinlichkeit, daß nach Sektor x auch die Sektoren $x+1, x+2, \ldots, x+n$ gelesen werden, ist dieses Vorgehen sinnvoll.

***prefetch*-Cache** Ein kleinerer Abschnitt ist für die Speicherung von durch automatische oder explizit ausgelöstes Vorauslesen gewonnenen Sektoren verantwortlich.

Daten-Cache Dieser Teil ist für die Zwischenspeicherung der Daten von Sektoren verantwortlich, die schon einmal gelesen wurden. Wenn der Schreib-Cache über eine diesbezügliche *mode page* eingeschaltet wurde, werden auch Schreibzugriffe gepuffert.

Firmware Ein Teil des RAM ist für die Firmware reserviert, die nach Einschalten des Gerätes automatisch aus reservierten Sektoren geladen wird. Die Angabe der Cache-Gesamtgröße einer Platte bestätigt die teilweise Nutzung von Cache-Speicher für andere Aufgaben (z.B. 448kB statt 512kB bei IBM DORS 32160).

Die Aufteilung und die Verhaltensweise des Plattencaches ist nach außen transparent und kann nur in engen Grenzen durch *mode pages* – siehe folgender Abschnitt – beeinflußt werden.

2.3.3 *Mode pages*

SCSI-Geräte besitzen interne nichtflüchtige Speicher, die vor allem die Speicherung von defekten Blöcken (*defect list*) und von außen veränderbaren Parametern (*mode pages*) enthalten. Ein Auslesen und Verändern dieser Speicher erfolgt mittels SCSI-Kommando. Wichtige Einstellungen betreffen das Cache-Verhalten und das Verhalten bei *disconnect/reconnect* (Tabelle 2.4). Die Implementierung dieser vom Standard vorgeschriebenen Parameter in den Geräten ist meist sehr unvollständig. Der Schreib-Cache kann erfahrungsgemäß bei allen neueren SCSI-Festplatten ein- und ausgeschaltet werden.

Tabelle 2.4: Durch *mode pages* veränderbare Einstellungen bei SCSI-Geräten – Auszug (aus [SCS93])

Seite	Einstellung	Bemerkungen
read-/write error		Schreib-/Lesefehler
format	Spuren pro Zone Ersatzsektoren pro Zone Ersatzspuren pro Zone Spurversatz Zylinderversatz SURF (*surface*)	 für Behandlung fehlerhafter Sektoren für Behandlung fehlerhafter Sektoren Vorgehen bei Numerierung der Blöcke
geometry	diverse Geometrie-Daten RPL (*rotational position locking*)	 für Umdrehungskopplung bei RAID
disconnect/reconnect	Puffer-Voll-Verhältnis Puffer-Leer-Verhältnis max. Inaktivitätszeit min. Busfreigabezeit max. Verbindungszeit max. Burstlänge	ab wann erfolgt *reconnect* bei Lesen ab wann erfolgt *reconnect* bei Schreiben max. Zeit bevor Busfreigabe min. Zeit, bevor *reconnect* maximale Verbindungszeit mit Inititaor max. Sektorenanzahl bis *disconnect*
cache	RCD (*read cache enable*) WCE (*write cache enable*) Prefetch-Minimum Prefetch-Maximum *read-/write rentention priority*	Aktivierung Lese-Cache Aktivierung Schreib-Cache min. Zahl automatisch vorauszulesender Blocks max. Zahl vorauszulesender Blocks Priorität der *prefetch*-Daten zu normalen Daten
control	*queue algorithm modifier* DQue (*disable queuing*) EAENP (*error AEN permission*)	Algorithmus der *tagged queues* Abschalten der Warteschlangen Asynchrone Nachricht bei Fehler
notch		Beschreibt Bereiche (Zonen) mit konstanter Zahl von Sektoren pro Spur

2.4 Der Linux-SCSI-Treiber

Es folgt eine genauere Beschreibung des Linux-SCSI-Treibers. Dies erscheint notwendig, um die Arbeitsweise des Treibers besser zu verstehen. Der Abschnitt soll gleichzeitig als Referenz für zukünftige Arbeiten dienen.

2.4.1 Der Aufbau in drei Schichten

Der SCSI-Treiber von Linux besteht aus drei Sektionen; dabei erfolgt eine Trennung in allgemeine Gerätefunktionen (*upper level*), allgemeine SCSI-Funktionen (*medium level*) und direkte Hardwareanbindung (*lower level*).

Obere Schicht (*Upper Level*)

Dieser Teil ist gerätespezifisch und behandelt Platten, CD-ROMs, Tapes und generische Geräte. Folgend einige erwähnenswerte Funktionen für SCSI-Platten (Datei sd.c):

- `do_sd_request()` liest Aufträge der Blockverwaltung aus deren Warteschlange (siehe Datei ll_rw_block.c). Durch Aufruf von `allocate_device()` (siehe Abschnitt 2.4.1 auf der nächsten Seite) wird versucht, einen freien SCSI-Auftrag zu reservieren. Gelingt das nicht und sind am SCSI-Bus mehrere Platten angeschlossen, wird die Warteschlange der Blockverwaltung nach einem Auftrag durchsucht, der in der Zwischenzeit (wo der eigentlich an der Reihe befindliche Auftrag ausgeführt werden sollte) bearbeitet werden kann (Abbildung 2.3).

  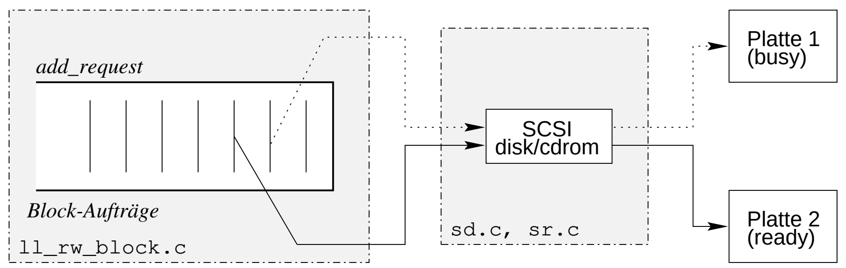

  Abbildung 2.3: Zugriff von SCSI-Platte und CD-ROM auf die Block-Auftrags-Warteschlange. Wenn eine Platte gerade keine Aufträge annehmen kann, erhält eine andere Platte Zugriff auf den Host

  Die Funktion `request_queueable()` der mittleren Schicht prüft dabei, ob eine Platte bereit zur Aufnahme eines SCSI-Auftrages ist. Dieser Mechanismus dient der besseren Auslastung des SCSI-Busses, wenn mehrere Platten an einem SCSI-Bus arbeiten. Hierbei wird davon ausgegangen, daß sich eine Platte durch *disconnect* nach Empfang eines Kommandos vorübergehend vom Bus verabschiedet, bis sie Daten liefern kann.

- `requeue_sd_request()` wird von der Funktion `do_sd_request()` aufgerufen und belegt einen SCSI-Auftrag mit den Parametern des Block-Auftrages. Außerdem wird ggf. ein *Scatter/Gather*- Array aus dem Pool des SCSI-Speichers erstellt. Der neue Auftrag wird der mittleren Schicht durch `scsi_do_cmd()` übergeben.

Mittlere Schicht (*Medium Level*)

Dieser Teil ist für das allgemeine Erkennen der angeschlossenen SCSI-Hostadapter und -Geräte verantwortlich und stellt die Verbindung zu den entsprechenden Teilen des *upper-* und *lower level* her. Für diesen Zweck sind allgemeine SCSI-Funktionen (Datei `scsi.c`) definiert:

- `request_queueable()` untersucht den SCSI-Auftragspuffer eines Gerätes und entscheidet, ob dieser ein neuer Auftrag hinzugefügt werden kann. Dazu werden alle SCSI-Aufträge des Gerätes und des zugehörigen Hosts nach freien Aufträgen durchsucht. Die Anzahl an Aufträgen für Hosts und Geräte ist konstant und wird beim Booten von `scsi_build_commandblocks()` – siehe dort – reserviert. Wenn für den Hostadapter bereits mehr als `can_queue` SCSI-Aufträge anstehen, kehrt die Funktion ergebnislos zurück. Anderenfalls wird der freie SCSI-Auftrag mit den Parametern des Block-Auftrages gefüllt. Falls dabei mehr als `sg_tablesize` Puffer physisch nicht zusammengehörig sind (also einen Eintrag in der *scatter/gather table* benötigen), wird der ursprüngliche Blockauftrag geteilt, anderenfalls freigegeben.

- `allocate_device()` gleicht der Funktion `request_queueable()`, ermöglicht aber die Übergabe eines Parameters, der bestimmt, ob die Funktion bis zur Verfügbarkeit eines Gerätes warten soll, oder nicht.

- `internal_cmnd()` wartet zuerst eine kurze Zeitspanne (`MIN_RESET_DELAY`), bis ein eventuell aufgetretener SCSI-Reset vom Hostadapter abgearbeitet wurde. Dann wird ein neuer Timeout für den aktuell zu versendenden Auftrag festgelegt. Schließlich wird der Auftrag in die Warteschlange des SCSI-Hostadapters durch Aufruf der Funktion `queuecommand()` des *lower level* eingeordnet.

- `scsi_request_sense()` erbittet vom Hostadapter Auskunft über die Ursache des zuletzt aufgetretenen Fehlers durch Erstellung eines `REQUEST_SENSE`-Auftrages.

- `scsi_do_cmd()` ergänzt die Daten eines SCSI-Auftrages um SCSI-spezifische Parameter. Zuerst wird gewartet, bis die Warteschlange des Hostadapters einen weiteren Eintrag aufnehmen kann (Anzahl der Einträge kleiner als `can_queue`). Dann werden die Rohdaten des SCSI-Befehls (bis zu 12 Byte) in eine Kommando-Struktur eingeordnet, die endlich der Funktion `internal_cmd()` übergeben wird.

- `scsi_done()` wird nach Ablauf eines SCSI-Kommandos aufgerufen (ausgelöster Interrupt durch den SCSI-Hostadapters) und ist für die Fehlerbehandlung verantwortlich.

- `scsi_abort()` unterbricht das aktuelle SCSI-Kommando (normalerweise nach Zeitüberschreitung)

- `scsi_reset()` führt einen Reset des SCSI-Busses aus.

- `scsi_main_timeout()` stellt die Behandlung von Zeitüberschreitungen im SCSI-Betrieb dar.

- `scsi_malloc()` stellt DMA-fähigen SCSI-Speicher in 512-Byte-Blöcken aus einem Pool bereit.

- `scsi_free()` gibt SCSI-Speicher wieder frei

- `resize_dma_pool()` reserviert den Pool für SCSI-Speicher während des Bootens oder Nachladens des SCSI-Treibers.

- `scsi_build_commandblocks()` reserviert für jedes vom Kern erkannte SCSI-Gerät eine Anzahl von SCSI-Aufträgen (Speicherallokation für entsprechende Strukturen), die dem Wert `cmd_per_lun` entspricht. Dieser ist abhängig von dem Hostadapter, an dessen Bus dieses Gerät arbeitet. Beim NCR-Hostadapter ist dieser Wert auf 3 festgelegt, d.h. maximal drei ausstehende Aufträge werden für jedes Subgerät an einem Hostadapter zugelassen. Siehe auch Abschnitt 2.4.3 auf Seite 25.
- `scsi_dev_init()` durchsucht alle Busse an allen erkannten Hostadaptern nach SCSI-Geräten. Danach wird der Pool an DMA-fähigem SCSI-Speicher reserviert.

Untere Schicht (*Lower Level*)

In diesem Teil des Linux-Treibers wird die Ansteuerung der Hardware vorgenommen. Für jeden unterstützten SCSI-Hostadapter existiert eine eigene Anpassung. Für die Geräteklasse NCR53c8xx gibt es zwei verschiedene Versionen: Den älteren Treiber von Drew Eckardt (Dateien 53c7,8xx.{c,h,scr}, 53c8xx_{d,u}.h) und den immer noch in Pflege befindlichen BSD-Port von Steffan Esser und Gerard Roudier (Dateien ncr53c8xx.{c,h}). Alle Aussagen dieses Textes beziehen sich auf die zweite Version.

Als Besonderheit weist der NCR-Hostadapter ladbaren SCSI-Code auf. Dabei handelt es sich um eine Sammlung von Makros, die während der Initialisierung in einen vorher reservierten Hauptspeicherbereich geladen wird. Auf diesen Bereich wird mittels des Makros `virt_to_bus` zugegriffen, welches per Definition eine 1-zu-1-Abbildung liefert (entsprechend den üblichen Gegebenheiten unter Linux 2.0.x). Zum Start des Prozessors auf dem Hostadapter wird ihm eine I/O-Anweisung mit der physischen Startadresse des Scripts übertragen, worauf er die Befehle aus dem Hauptspeicher liest und interpretiert. Dabei entsteht eine marginal zusätzliche PCI-Bus-Last. Neuere SCSI-Hostadapter erlauben das Übertragen des Codes direkt auf den Host-Chip, wodurch für SCSI-Operationen keine zusätzliche Buslast entsteht.

2.4.2 Verwaltung der SCSI-Geräte

Linux behandelt SCSI-Festplatten als Blockgeräte und weist jeder erkannten Platte eine Gerätenummer bestehend aus Haupt- und Unternummer (*Major-* bzw. *Minornumber*) zu. Die Hauptnummer ist 8 (Pseudodateien **/dev/sd***), die Unternummer wird ermittelt aus $Plattennummer \times 16 + Partitionsnummer$. Die maximale Anzahl Partitionen pro Platte beträgt somit 16.

2.4.3 Warteschlangen im Linux-Treiber

Aufgrund der relativ hohen Latenz bei der Ausführung von SCSI-Kommandos macht sich eine mehrstufige Hierarchie an Warteschlangen im SCSI-Treiber und selbst im Hostadapter sowie in den Platten (*tagged queues*, siehe Abschnitt 2.4.3 auf Seite 24) notwendig. Dadurch sollen Kommandos gesammelt und Optimierungen durch Umsortieren und Zusammenfassen ermöglicht werden. Eine grobe Beschreibung ist in [Meh96] zu finden, hier sollen aber genauer die Arbeitsweise der einzelnen Warteschlangen und die Möglichkeiten der direkten Einflußnahme untersucht werden.

Block-Auftrags-Warteschlange

Die Ansteuerung von SCSI-Geräten erfolgt üblicherweise als Blockgerät. Aufträge für alle Blockgeräte werden unter Linux in eine gemeinsame Warteschlange eingereiht (siehe Abbildung 2.4 auf der nächsten Seite). Dabei werden Aufträge zuerst für logisch aufeinanderfolgende Blöcke zusammengefaßt, dann nach dem *Elevator*-Algorithmus einsortiert und zuletzt eine kurze Zeit zurückgehalten, um ein Zusammenfassen mehrerer nacheinander eintreffender Aufträge zu erzwingen.

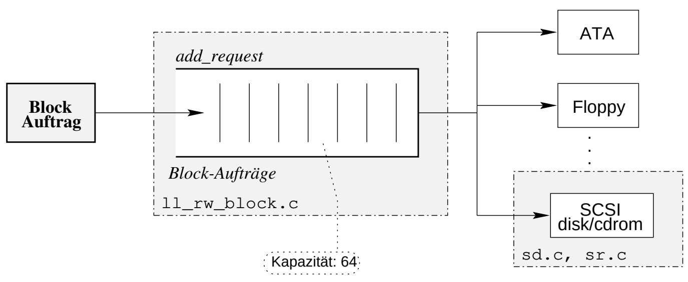

Abbildung 2.4: Alle Aufträge für Blockgeräte werden in eine gemeinsame Warteschlange, auf die jede Geräteklasse separat Zugriff hat, eingereiht.

Die Blockgeräte greifen mit einer gerätespezifischen Funktion `scsi_request_fn()` auf die Warteschlange zu.

Obere und mittlere Schicht

Der SCSI-Treiber erhält vom Linux-Blocktreiber bereits zusammengefaßte und nach *Elevator* vorsortierte Anfragen zugestellt. SCSI-Blockgeräte sind nur Platten und CD-ROMs, spezielle Geräte, wie Scanner oder CD-Recorder, werden über das *generic device interface* angesprochen.

Der jeweils nächste Auftrag für ein SCSI-Blockgerät wird mit der Funktion `scsi_request_fn()` direkt aus der Warteschlange für allgemeine Block-Aufträge gelesen und in die hostspezifische Warteschlange eingetragen (Abbildung 2.5).

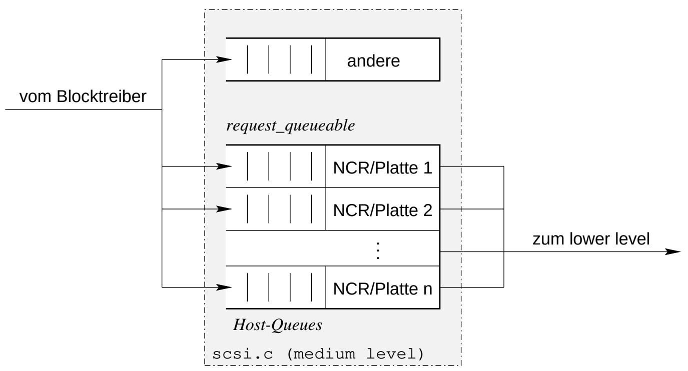

Abbildung 2.5: Die Verteilung der SCSI-Aufträge auf die Hostadapter in der mittleren Schicht des SCSI-Treibers

Untere Schicht

Die Warteschlangen dieser Schicht dienen vorrangig dazu, einen kontinuierlichen Kommandostrom an den Host zu senden (siehe Abschnitt 2.4.1 auf der vorherigen Seite).

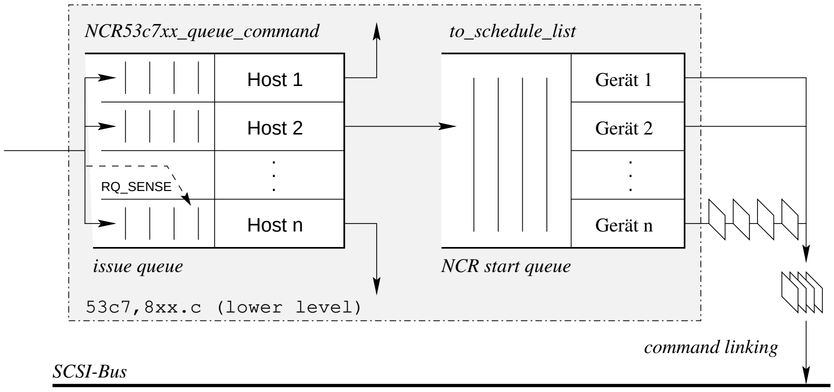

Abbildung 2.6: Warteschlangen im *lower level*. Durch *command linking* werden mehrere zusammenhängende SCSI-Kommandos übertragen

Der NCR-Hostadapter kann insgesamt bis zu 28 Einträge in seiner Warteschlange halten (Eintrag `can_queue` in der Host-Beschreibung). Nach Einreihung eines zusätzlichen Auftrages in die *issue queue* (Linux-Warteschlange) wird sichergestellt, daß der Leerungsprozeß dieser Warteschlange arbeitet (Aufruf von `run_process_issue_queue()`). Dadurch wird die Warteschlange im Hostadapter (*NCR start queue*) mit den Werten der *issue queue* gefüllt (Funktion `to_schedule_list()`). Alle diese Warteschlangen arbeiten nach FIFO mit Ausnahme der Behandlung des *REQUEST_SENSE*-Kommandos: Dieses muß unmittelbar nach einem aufgetretenem Fehler ausgeführt werden und wird deshalb an den Beginn der *issue queue* gesetzt.

Warteschlangen auf Geräteebene: *Tagged Queuing*

In [Sch93] werden geordnete Warteschlangen bei SCSI-Geräten genauer beschrieben. Als optionale SCSI-2-Eigenschaft erlaubt es *tagged queuing*, daß ein SCSI-Target mehrere SCSI-Aufträge aufnehmen kann und eventuell sogar die Ausführungsreihenfolge der Kommandos verändern darf. Ist *tagged queuing* nicht im Gerät implementiert oder wird es nicht vom Treiber unterstützt, kann ein LUN des Targets nur jeweils einen Befehl ausführen.

Wenn dieses Feature in einem SCSI-Gerät implementiert ist, dann können SCSI-Aufträge nach drei verschiedenen Verhaltensweisen in eine Warteschlange bis zu theoretisch 256 Einträge (der NCR-Treiber begrenzt diese Zahl auf 28) einsortiert werden:

- *simple queue tag* Kommandos mit dieser Kennung dürfen ausgeführt werden, wann das Target will, allerdings muß auf Kommandos mit *ordered queue tag* Rücksicht genommen werden.
- *head of queue tag* Wenn ein Kommando mit diesem Tag beim Target eintrifft, wird es sofort, wenn das aktuell ausgeführte Kommando beendet ist, abgearbeitet. Wenn alle Kommandos ein Gerät mit *head of queue tag* erreichen, stellt sich eine LIFO-Verhaltensweise des Targets ein.
- *ordered queue tag* Alle Kommandos, die vorher in der Schlange enthalten waren, sollen auch vorher ausgeführt werden. Alle späteren Kommandos (mit Ausnahme derer, die ein *head of queue tag* haben), sollen später ausgeführt werden. Wenn ein Gerät nur Kommandos mit *ordered queue tag* erreichen, stellt sich ein FIFO-Verhalten des Targets ein.

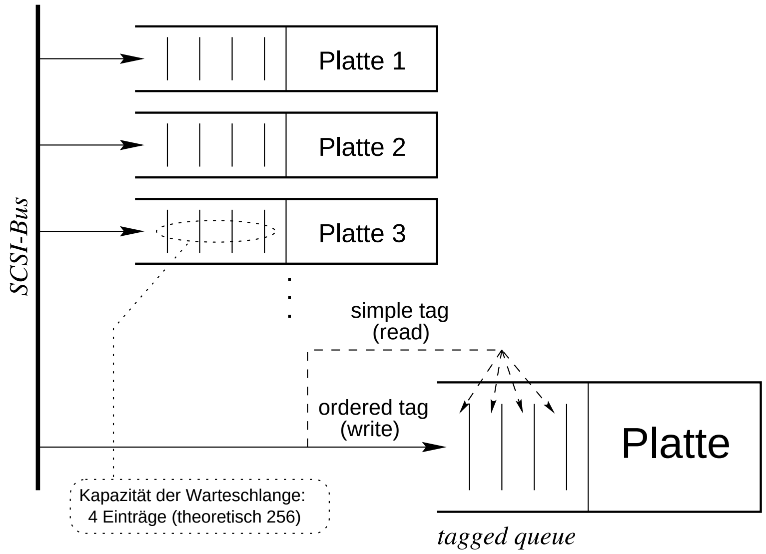

Abbildung 2.7: *Tagged queuing* bei Festplatten. Neue SCSI-Befehle werden in den Befehlspuffer auf bestimmte Art und Weise, die vom Wert eines *tags* abhängig ist, eingereiht.

Umsortieren von SCSI-Kommandos auf Geräteebene kann man verhindern, indem man entweder keine oder nur *ordered queue tags* an das Gerät sendet.

Im Gerätetreiber von Drew ist *tagged queuing* nicht implementiert. Der BSD-Port von Roudier verwendet für Schreiben *ordered tags* und *simple tags* für Lesen, vorausgesetzt, dieses Feature wurde während der Linux-Kern-Konfiguration eingeschaltet.

Command Linking

Die Abarbeitungszeit mehrerer SCSI-Kommandos für ein Gerät kann durch *command linking* verkürzt werden. Erhält `cmd_per_lun` der Hostbeschreibung einen Wert $n > 1$, so kann der Hostadapter bis zu n SCSI-Kommandos für ein Target zusammenfassen, um einige Busphasen nach Übertragung des ersten Kommandos einzusparen. Die Hauptnutzung von *command linking* besteht allerdings darin, aufeinanderfolgende Befehle „unteilbar" zu machen.

Diese Eigenschaft verlangt Unterstützung vom hardwareabhängigen Teil des SCSI-Treibers und wird noch nicht durchgehend unterstützt.

Kapitel 3

Entwurf

3.1 Entwurfsziele

Aufgrund der Erfahrungen aus dem 2. Kapitel soll ein SCSI-Subsystem entworfen werden, das folgenden Ansprüchen genügt:

- Zusagenfähigkeit, d.h. Abarbeitung von Aufträgen in einer vorherbestimmten Zeit
- gute Performance, sparsamer Umgang mit Ressourcen – insbesondere CPU-Zeit und Hauptspeicher
- lauffähig auf L4 neben L^{4}Linux (parallele Verarbeitung von zeitkritischen und nicht-zeitkritischen Aufträgen)
- keine oder nur geringfügige Änderungen am Linux-SCSI-Treiber

Als Grundvoraussetzung dafür ist eine Möglichkeit gesucht, wie SCSI-Befehle zeitlich so gesteuert abgesetzt werden können, daß sich mehrere Festplatten am selben Bus gegenseitig nicht oder nur in vorhersagbarem Maße beeinflussen.

3.2 Modelle zum Plattenscheduling

Im folgenden soll ein Modell zum Scheduling von SCSI-Aufträgen erarbeitet werden. Vorerst wird nur der lesende Zugriff auf SCSI-Festplatten betrachtet.

3.2.1 Begriffsdefinitionen

Für die folgenden Ausführungen sollen an dieser Stelle einige Begriffe definiert werden (siehe vgl. Abbildung 3.1 auf der nächsten Seite):

Job bestimmt einen vollständigen SCSI-Auftrag mit den Phasen Senden des Kommandos an die Platte (t_{cmd}), Positionieren auf den Beginn des Datenblockes (t_{seek}) und Übertragung der Daten an den Hostadapter ($t_{transfer}$). Die Ausführungszeiten der Jobs einer Platte differieren im Normalfall erheblich (unterschiedliche Positionierzeiten, *Zoning*, Plattencache-Einfluß)

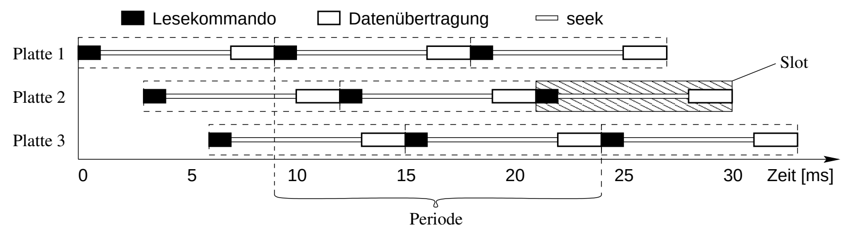

Abbildung 3.1: Begriffsdefinitionen für dieses Kapitel

Slot legt die Rahmenzeitspanne fest, in der ein Job einer Platte ausgeführt werden muß. Durch Wechselwirkung mit Jobs anderer Platten kann es zu geringfügigen (berechenbaren) Verzögerungen kommen. Ein Slot nimmt immer maximal einen Job einer Platte auf. Die Anzahl Slots n_s in einer Periode ist i. A. kleiner als die Gesamtzahl Platten n an einem SCSI-Bus (siehe Abschnitt 3.2.5 auf Seite 31).

Periode besteht aus n_s Slots. Die Anzahl n_s wird aus der *worst-case*-Zugriffszeit aller Platten errechnet. Pro Periode kann eine Platte nur maximal einen Slot belegen. Die Slots werden in der Periode quasiparallel mit verschiedenen Startzeitpunkten abgearbeitet, sodaß sich die aktiven Platten nur begrenzt beeinflussen.

Zyklus umfaßt mehrere Perioden. Ein Zusammenfassen in Zyklen ist erforderlich, um z.B. einen Strom nur während jeder zweiten Periode zu bedienen.

Die Einteilung in Slots, Perioden und Zyklen wird für jeden SCSI-Bus im System vorgenommen. Ein Slot wird durch Angabe seiner Periode und der Slotnummer gekennzeichnet, z.B. $_5 14$ für Slot 5 der Periode 14.

3.2.2 Grundmodell

Grundidee ist das periodische Senden von Kommandos nacheinander an alle beteiligten Platten so, daß eine Platte frühestens dann weiterarbeitet, wenn ihr letzter Auftrag (**Job**) erfüllt wurde. Durch eine verteilte Anordnung der Sendezeitpunkte in der Gesamt**periode** wird eine geringstmögliche Beeinflussung der Platten untereinander erreicht. Den größten zeitlichen Anteil bei der Ausführung eines Jobs nimmt die Kopf-Positionierung ein.

Die Datenblockgröße soll bei allen **Jobs** aller Platten gleich sein. Es läßt sich eine maximale Zugriffszeit (siehe Abschnitt 2.3 auf Seite 18) zur Übertragung eines dieser Datenblöcke bestimmen (*worst-case*-Fall). Der Aufbau der Festplatte in Zonen wird bei der Ansteuerung durch den Treiber nicht berücksichtigt. Bei überwiegend zufällig verteilten Zugriffen ist der Anteil der Medium-Lesezeit geringer, als beim sequentiellen Zugriff, trotzdem aber nicht zu vernachlässigen. Eine Reaktion darauf sollte im Dateisystem erfolgen. Möglich wäre z.B., in die äußeren Zonen einer Platte größere Datenblöcke zu legen.

Moderne Festplatten besitzen in der äußeren Spur bis zu 180 Sektoren, in der inneren Spur etwas mehr als die Hälfte. Bei einer Rotationsgeschwindigkeit von $5400\frac{1}{min}$ werden 64kB (128 Sektoren) aus der inneren Spur in $\frac{128}{90} \times \frac{60\frac{s}{min}}{5400\frac{1}{min}} = 16ms$ (plus die Zeit eines Spurwechsels), aus der äußeren Spur in $\frac{128}{180} \times \frac{60\frac{s}{min}}{5400\frac{1}{min}} = 8ms$ gelesen. Die *worst-case*-Zugriffszeit beträgt bei dieser Blockgröße etwa 40ms (siehe praktische Messungen in Abschnitt 5.2 auf Seite 44).

3.2.3 Periodisches Vorauslesen (*prefetching*)

Ausgehend von den Erfahrungen in Abschnitt 2.2.2 auf Seite 16 läßt sich ein Modell skizzieren, bei dem Plattenaufträge in Slots ausgeführt werden und die Platten sich untereinander nicht beeinflussen.

Dabei wird folgender Ablauf zugrunde gelegt: Ein zu lesender Block wird zuerst explizit über ein *prefetch*-Kommando in den internen Platten-Cache geladen. Das Laufwerk erbringt nach erfolgreichem Abschluß der Operation eine diesbezügliche Statusmeldung. Daraufhin kann der Datenblock ohne Unterbrechung durch ein *Read*-Kommando aus dem Plattencache gelesen werden – es erfolgt nicht das übliche *disconnect/reconnect* bei der Übertragung. Voraussetzung für das Funktionieren dieses Modell ist, daß ein Datenblock immer vollständig in den *prefetch*-Cache – meist also 64kB – paßt (Abschnitt 2.2.3 auf Seite 16).

Die Kommandos werden in vorgegebenen Abständen nacheinander an alle Platten gesendet. Bei der Ermittlung der Periodendauer ist der *worst case* der Zugriffszeit – hier die maximale Zeit, die eine Platte unter den ungünstigsten Umständen benötigt, um einen Datenblock bestimmter Größe in den Cache zu lesen – zu verwenden. Die Zeit zum Einlesen eines Datenblockes in den Cache kann stark variieren und, falls sich der Block bereits im Cache befindet, sogar Null sein. Als Konsequenz daraus wird mit diesem Verfahren niemals die maximal mögliche Datenrate einer Festplatte ausgenutzt. Dafür kann aber mit festen Zeiten gerechnet werden, wann ein Lese-Auftrag erfüllt worden ist.

Durch dieses Verfahren wird der SCSI-Bus auf einfache Art und Weise so *aufgeteilt*, daß eine Festplatte immer Daten überträgt, während die anderen Platten Zeit haben, den nächsten Plattensektor anzufahren und einzulesen. Die Belegung des Busses nimmt immer den kleinstmöglichen Zeitraum in Anspruch und erfolgt ohne Unterbrechungen.

An dieser Stelle soll aber darauf hingewiesen werden, daß die Zugriffsart „*PREFETCH-READ*“ ein Umgehen der Plattenintelligenz darstellt und nicht zu den „normalen“ Betriebsarten eines SCSI-Gerätes gezählt werden kann. Das SCSI-Kommando *prefetch* wird im Standard ausdrücklich als **optional** aufgeführt, ein solches muß also nicht vom Gerät verstanden werden. Das dargestellte Modell funktioniert aber nur dann, wenn alle Geräte am Bus dieses Kommando unterstützen.

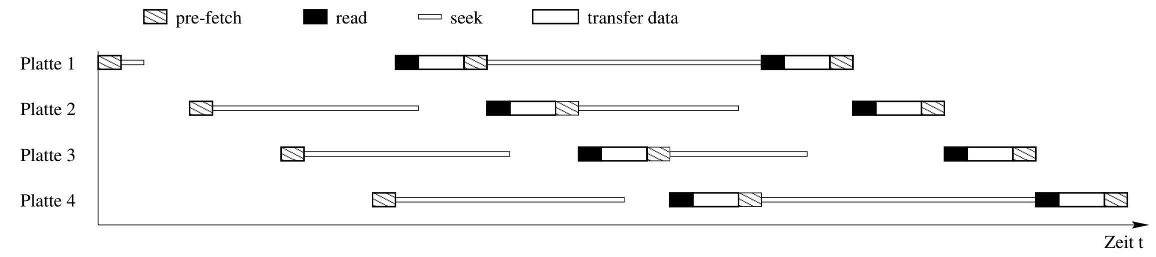

Abbildung 3.2: Scheduling durch periodisches *prefetching*

Die Zeit zwischen dem *prefetch*- und dem *read*-Kommando muß für jede Festplatte extra bestimmt werden. Als endgültiger Wert wird der größte Wert aller am Bus arbeitender Platten angenommen.

3.2.4 Periodisches Lesen ohne Vorauslesen

Die im vorangegangenen Abschnitt dargestellte Lösung hat den Nachteil, daß sie ein im SCSI-Standard als „optional“ gekennzeichnetes Kommando verwendet. Es ist daher keineswegs sichergestellt, daß eine Festplatte dieses Kommando unterstützt. Auch ergibt sich aus der Verwendung von zwei (*prefetch* und *read*) Kommandos im Gegensatz zur Verwendung von nur einem (*read*-)Kommando der Nachteil eines höheren Overheads durch Steuerbefehle auf dem SCSI-Bus.

Aufgrund dessen soll nach einer (allgemeineren) Möglichkeit gesucht werden, auch ohne ein derartiges Spezial-Kommando die Aufträge vorhersagbar ausführen zu lassen.

Grundsätzlich kann man bei obigem Modell die *prefetch*-Befehle einfach weglassen. Als Folge dessen ergeben sich zwei entscheidende Veränderungen:

- Die Übertragung des Datenblockes findet jetzt nicht mehr in **einem**, sondern in **mehreren kleinen Stücken** nicht vorhersagbarer Größe statt.
- Der Zeitpunkt, wann Daten übertragen werden, ist nicht mehr bestimmbar. Befindet sich der Block durch vorangegangene Operationen im Laufwerks-Cache, wird sofort nach Erhalt des *read*-Befehles ohne Unterbrechung mit dem Transfer begonnen. Eventuell anstehende Kommandos an andere Platten können solange nicht übertragen werden, bis sich die arbeitende Platte per *disconnect* vorübergehend oder nach vollständigem Transfer endgültig vom Bus verabschiedet.

Wichtig ist, daß die Periodenlänge immer größer ist, als die Job-Länge. dadurch wird sichergestellt, daß die Übertragung von Daten der Periode x nicht durch das Senden des neuen Befehles der Periode $x + 1$ unterbrochen bzw. verschoben wird. Weiterhin steht fest, daß der Hostadapter immer die höchste Priorität haben muß, d.h. das Absetzen von Schreib-/Lesebefehlen hat Vorrang vor dem Übertragen von Daten. Das kann dadurch erreicht werden, daß der Hostadapter die höchste ID am Bus erhält.

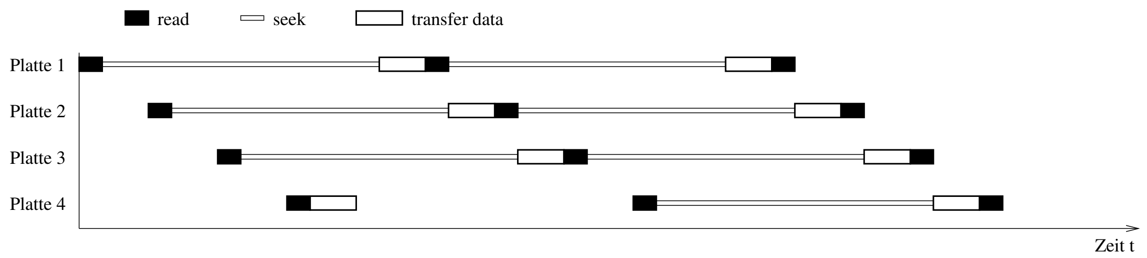

Abbildung 3.3: Periodisches Absetzen der Plattenaufträge ohne *prefetching*. Spezialfall: Wenn der angeforderte Block bereits vollständig im Plattencache vorhanden ist, wird er sofort nach Erhalt des Lesekommandos übertragen.

Es lassen sich zwei Spezialfälle konstruieren:

- Der Datenblock einer Platte befindet sich bereits im Cache und wird sofort nach Erhalt des *read*-Befehles übertragen (vgl. Bild 3.3). Das wirkt sich infolge der zeitversetzten Ausführung der Slots nicht auf die folgenden Aufträge aus.
- Eine laufende Datenübertragung verzögert das Senden des Kommandos der nächsten Periode (im Bild 3.4 auf der nächsten Seite dargestellt).

Es läßt sich nachweisen ([Ham97]), daß die dadurch entstehende Verzögerung über alle Perioden nicht größer ist als $(n-1)^2 t_{Bus}$ mit n für die Plattenanzahl und $t_{Bus} = t_{cmd} + t_{transfer}$ für die Zeit, die ein Gerät maximal den Bus (für Kommando- und Datenübertragung) belegt.

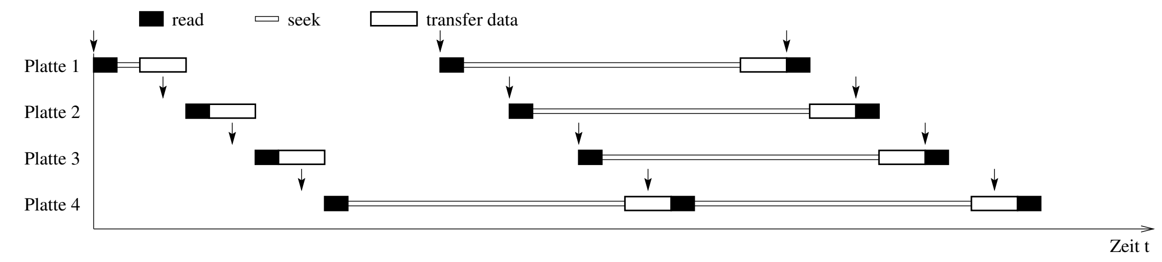

Abbildung 3.4: Periodisches Absetzen der Plattenaufträge ohne *prefetching*. Ein weiterer Spezialfall.

3.2.5 Das Slot-Modell

An einem *WIDE*-SCSI-Bus können in der Regel bis zu 15 Geräte (zzgl. Hostadapter) betrieben werden. Um eine bessere Ausnutzung der maximalen Datenrate einer einzelnen Festplatte zu erreichen, ist es sinnvoll, nur über eine Untermenge aller Platten an einem Bus zu arbeiten. Dadurch kann eine größere Auslastung der Festplatten erreicht werden. Zu beachten ist, daß der zeitliche Abstand der Aufträge für eine Platte nicht geringer als die Joblänge werden darf (siehe Abschnitt 3.4.2 auf Seite 34).

Wie kann die minimale Slotanzahl n_s bestimmt werden? Zur Bestimmung existiert keine allgemeingültige Formel. Praktisch kann folgendermaßen vorgegangen werden:

Mit einer bestimmten Blockgröße d_{block} wird die maximale Slotlänge t_s für eine Festplatte bestimmt (praktische Messung mit *worst case* der Zugriffszeit, siehe Abschnitt 5.2 auf Seite 44). Dann wird die Zeit $t_{Bus} = t_{cmd} + t_{transfer}$ abgeschätzt. Bei einer Blockgröße von 64kB sollten das in etwa $4ms + \frac{64kB}{40*10^6 \frac{Byte}{s}} \approx 6ms$ sein. Schließlich wird die Anzahl der Slots durch $n_s = \lfloor \frac{t_{Bus}}{t_s} \rfloor$ bestimmt. Bei einer angenommenen maximalen Slotlänge von $40ms$ ergäbe das eine Anzahl von 6 Slots. Durch praktische Messungen muß dieser Wert überprüft und ggf. korrigiert werden.

3.2.6 Verallgemeinerung für Datenschreiben

Wie in Abschnitt 2.3.2 auf Seite 18 angedeutet, besitzen viele SCSI-Festplatten einen schnellen Zwischenspeicher für Daten, der unter anderem ein verzögertes Schreiben zuläßt. Da die Zeit, wann ein Schreibauftrag wirklich ausgeführt wurde, unter diesem Verhalten nicht berechenbar ist, wird für dieses Echtzeitmodell der Schreib-Cache abgeschaltet. Als Ergebnis verhält sich die Platte bei Lese- und Schreibzugriffen gleich.

3.3 Injektion der SCSI-Aufträge

Aufgrund diverser Warteschlangen im Linux-Treiber (siehe Abschnitt 2.4.3 auf Seite 22) sollten neue Aufträge nicht als Blockaufträge in den Treiber eingebracht werden. Vielmehr ist es notwendig, eigene Auftragsstrukturen aufzubauen und direkt an die mittlere Schicht des Linux-SCSI-Treibers zu übergeben.

Mit Hilfe der bereits in der mittleren Schicht des Linux SCSI-Treibers enthaltenen Funktionen `allocate_device()` und `scsi_do_cmd()` kann ein vollständiger SCSI-Auftrag erstellt und dem Treiber übergeben werden.

3.4 Planungsmodelle

Der SCSI-Treiber soll als Teil eines ganzen Echtzeitsystems Aufträge in voraussagbarer Zeit ausführen. Ein Admissionmodul im Dateisystem hat die Aufgabe, vor dem eigentlichen Datentransfer zu überprüfen, ob die Voraussetzungen in Hardware (Leistungsfähigkeit des Rechnersystems) und Software (Systemauslastung) gegeben sind, um einen neuen Datenstrom bedienen zu können. Wird eine positive Entscheidung gefällt, muß das Bedienen des Stromes durchgesetzt werden. Da der SCSI-Treiber ein Bestandteil des Dateisystems ist, muß es eng mit diesem zusammenarbeiten. Hinsichtlich der Frage, wo Aufträge für den SCSI-Treiber erzeugt werden, gibt es mehrere Möglichkeiten (siehe auch [Reu98]).

3.4.1 Admission und SCSI-Planung getrennt

Ein intuitiver Ansatz besteht darin, Admission und Auftragsplanung zu trennen (Abbildung 3.5).

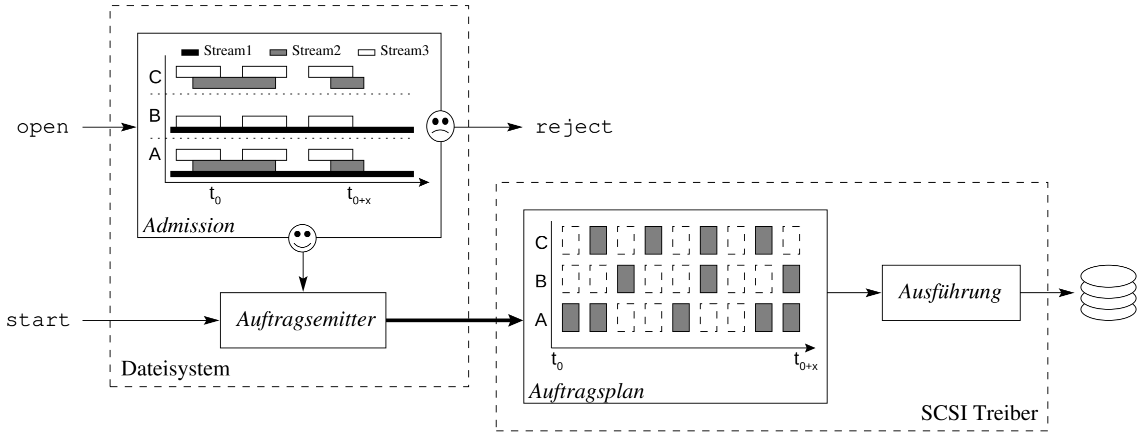

Abbildung 3.5: Erste (intuitive) Variante des Planungsmodells. Admission und Planung arbeiten getrennt.

Soll ein neuer Datenstrom abgespielt werden, überprüft die Admission anhand von Informationen über alle Ströme, die vorhandene Hardware und die Systemauslastung, ob die Forderung erfüllbar ist. Dabei kann z.B. für jeden Zeitpunkt die von den Datenströmen benötigte Bandbreite summiert und mit den Systemvorgaben verglichen werden.

Fällt bei der Admission eine positive Entscheidung, wird der neue Strom für den geforderten Zeitraum eingeplant. Ein Auftragsemitter bildet in regelmäßigen Abständen aus den zugelassenen Strömen eine Liste von Aufträgen, die wie in Abbildung 3.6 auf der nächsten Seite gezeigt aufgebaut sind.

Die Aufträge werden an den SCSI-Treiber gesendet, der sie in einen eigenen Auftragsplan unter Berücksichtigung folgender Bedingungen einsortiert:

- Ein Auftrag darf nicht eher ausgeführt werden, als Pufferspeicher verfügbar ist (*Einhaltung des Start-Zeitpunktes*).
- Ein Auftrag muß spätestens bis zur Deadline ausgeführt worden sein (*Einhaltung des Ende-Zeitpunktes*).
- Die Reihenfolge der Abarbeitung zweier Aufträge, die sich in mindestens einem Sektor überschneiden, darf nicht geändert werden (insbesondere bei verschiedenen Zugriffsarten).

```
struct request {
    unsigned long long time_valid;
      /* absolute Zeit [µs], ab wann Auftrag bearbeitet werden darf (64 Bit). */
      /* Erst ab dieser Zeit steht hinter map_address ein gültiger Puffer */
    unsigned long long time_deadline;
      /* absolute Zeit [µs], bis wann der Auftrag bearbeitet sein muß (64 Bit) */
    block_t block;
      /* zu lesende/schreibende Blocknummer (64 Bit) enthält Partition, */
      /* Blocknummer und -länge */
    unsigned int map_address;
      /* Adresse des Lese-/Schreibpuffers */
    unsigned short priority;
      /* Priorität des Auftrages (16 Bit) */
    unsigned short status;
      /* Status Bits */
};
```

Abbildung 3.6: Auftragsstruktur für den SCSI-Treiber, für dieses Planungsmodell. Sie enthält einen Zeitrahmen (*time_valid*, *time_deadline*), in dem der Auftrag ausgeführt werden muß

Pro Auftrag geben die Zeiten *time_valid* und *time_deadline* an, in welchem Zeitfenster der Auftrag ausgeführt werden muß. Die Verwendung von zwei Zeiten erscheint sinnvoll, um dem Treiber die Möglichkeit zur Optimierung zu geben. Nicht-zeitkritische Aufträge werden in „Lücken" untergebracht, die dadurch entstehen, daß die verfügbare Bandbreite des SCSI-Subsystems in der Admission immer nur zu etwa 90% verplant wird.

Bewertung

Vorteile

+ einfaches Kommunikationsmodell
+ Bedienung mehrerer Dateisysteme möglich

Nachteile

– Hoher Verbrauch an Hauptspeicher und CPU-Zeit zur Verwaltung des Auftragsplanes im SCSI-Treiber
– Fehlende Rückkopplung zur Admission. Die Auftragsausführung ist nur mit hohem Ressourcenaufwand (Pufferung eines kompletten Zyklus') zu garantieren.
– aufwendiger Algorithmus (z.B. Beachtung der Reihenfolge bei Aufträgen, die sich auf den gleichen Datenblock beziehen; Rechnung mit Zeit*bereichen* anstatt mit Zeit*punkten*)

3.4.2 Admission verbunden mit SCSI-Planung

Das Trennen von Admission und Auftragsplanung führt, wie in Abschnitt 3.4.1 auf der vorherigen Seite beschrieben, zu gravierenden Nachteilen, weshalb hier ein geeigneteres Modell genauer vorgestellt wird ([Här97]).

Das Dateisystem übernimmt hier sowohl die Admission, als auch die Auftragsplanung (siehe Abbildung 3.7). Im Unterschied zur ersten Idee arbeitet die Admission hier nicht mehr mit „Bandbreiten pro Zeiteinheit“, sondern mit realen Slots, Perioden und Zyklen (Definition siehe 3.2.1 auf Seite 27).

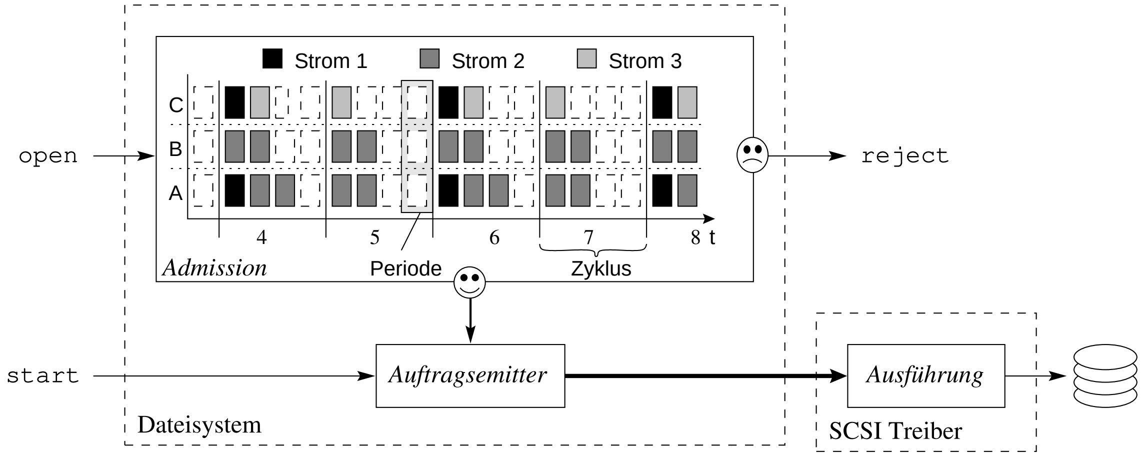

Abbildung 3.7: Zweite Variante des Planungsmodells nach [Här97]

Grundsätzlich werden die Slotlänge t_s (z.B. 40ms) und die Länge eines Datenblockes d_{block} (z.B. 128kB) bestimmt. Weiterhin muß die minimale Granularität d_{min} der Datenblockgröße der Ströme, hier mit 32kB angenommen, bekannt sein. Aus diesen Angaben läßt sich die Zyklenlänge nach der Formel $n_z = \frac{d_{block}}{d_{min}}$ bestimmen (im Beispiel also $\frac{128kB}{32kB} = 4$). Neue Ströme werden von der Admission mit Start- und Endzeitpunkt sowie der Datenrate erfaßt und eingeplant.

Der SCSI-Treiber erhält vom Dateisystem in periodischen Abständen Auftragslisten, die einen eindeutigen Zeitpunkt der Erfüllung haben. Im Gegensatz zum ersten Modell wird hier also nicht mehr mit Zeitfenstern gearbeitet.

Nicht-zeitkritische Aufträge werden auch hier in noch verbliebene Slots eingearbeitet.

Bewertung

Vorteile	*Nachteile*
+ geringer Planungsaufwand für SCSI-Treiber	– feste Anbindung des SCSI-Treibers and das Dateisystem
+ geringer Ressourcenverbrauch	

Erfassung der Aufträge, Logische Zeiten

Für jeden SCSI-Bus wird eine Periodendauer bestimmt, die festlegt, in welchen Abständen Auftäge an SCSI-Geräte gesendet werden (siehe Abschnitt 3.2.1 auf Seite 27). Die Angabe der Ausführungszeit jedes Auftrages erfolgt deshalb konsequenterweise als diskrete Zeitangabe (Abbildung 3.8 auf der nächsten Seite).

Weiterhin wird für jeden Bus bestimmt, wieviele Slots eine Periode enthält. Pro Slot kann ein Auftrag auf einer Platte ausgeführt werden (vgl. Abschnitt 3.2.5 auf Seite 31). Da sich die Ausführungszeiträume nebeneinanderliegender Slots überschneiden, muß dafür Sorge getragen werden, daß eine Platte nicht

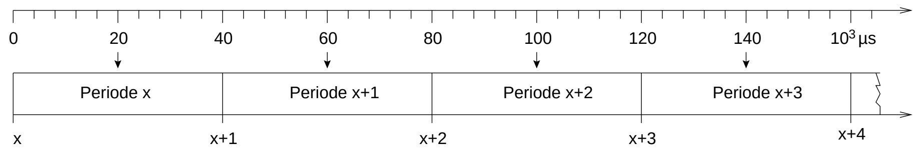

Abbildung 3.8: Übergang von der Systemzeit auf diskrete Zeitangaben (logische Zeit). Die Länge einer Periode beträgt hier 40ms.

zwei Aufträge zugleich ausführen muß (Abbildung 3.9). Diese Bedingung muß einerseits vom Dateisystem eingehalten werden, wenn es zeitkritische Aufträge erzeugt, andererseits vom SCSI-Treiber, wenn er nicht-zeitkritische Aufträge in freie Slots einordnet.

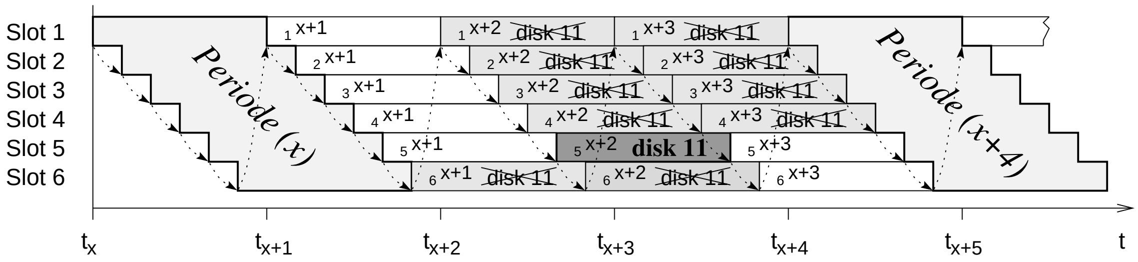

Abbildung 3.9: Da sich benachbarte Slots überlappen, müssen Geräteabhängigkeiten zwischen diesen Slots verhindert werden.

Das heißt, daß bei einer Anzahl von n Slots immer die letzten $n-1$ und die folgenden $n-1$ Slots nicht die gleiche Platte ansprechen dürfen. In Abbildung 3.9 wird in Slot $_5(x+2)$ (Slotnummer 5 der Periode $x+2$) die Platte 11 angesprochen. Daraus folgend darf Platte 11 nicht in den Slots $_1(x+2)$ bis $_4(x+2)$, $_6(x+1)$, $_6(x+2)$ und $_1(x+3)$ bis $_4(x+3)$ angesprochen werden.

Zur Untersuchung dieser Abhängigkeiten bei Einfügung eines neuen Auftrags wird ausgehend von der aktuellen logischen Zeit des SCSI-Busses ein Fenster über die umgebenden Aufträge gelegt und überprüft, ob die im Auftrag angesprochene Platte in diesen Aufträgen vorhanden ist (Abbildung 3.10).

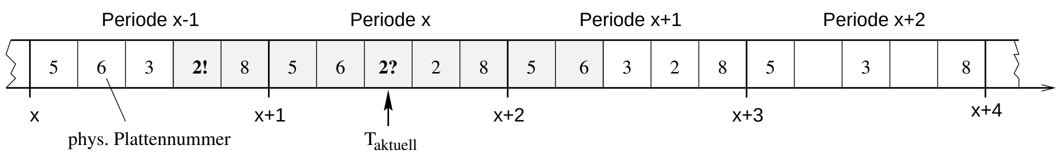

Abbildung 3.10: Ein Fenster maskiert alle zu untersuchenden Aufträge. $T_{aktuell}$ wird nach Beginn eines neuen Slots aktualisiert.

3.5 Schnittstellen-Beschreibung

Die Kommunikation mit dem eigentlichen Dateisystem unterliegt folgenden Bedingungen:

- **Schnelle Übertragung großer Auftragsmengen** für geringen Kommunikations-Overhead

- **flexible Parameterübergabe**, insbesondere die Möglichkeit, sowohl physische als auch virtuelle Adressen für die Datenpuffer vorzusehen. Direktes Ansprechen physischer Adressen spart eine Kopieroperation (vom Lesepuffer zum eigentlichen Datenpuffer), virtuelle Adressen erleichtern dagegen das Arbeiten mit Dateisystem-Hilfsstrukturen (Inodes, Freispeicherlisten).
- **Kontrollmöglichkeit** auf fehlerlose Auftragserledigung

Dabei ist eine getrennte Behandlung von zeitkritischen und nicht-zeitkritischen Aufträgen vorgesehen. Eine hohe Effizienz kann durch Nutzung der umfangreichen Möglichkeiten zur Parameterübergabe zwischen zwei Prozessen unter L4 erreicht werden. Wenn möglich ist dabei das Kopieren von Daten zu vermeiden.

3.6 Beachtung der Nebenläufigkeiten

Im Linux-Kern 2.0.x existieren grundsätzlich genau einen Hauptthread und weitere Threads, die jeweils den Hardware-Unterbrechungen zugeordnet sind. Eine Kern-Variable `intr_count` informiert darüber, ob sich der Kern in der Behandlung einer Hardwareunterbrechung aufhält. Für Echtzeitanforderungen sind genauere Zusagen für die Annahme von neuen Aufträgen erforderlich, gleichzeitig sind häufige Systemaufrufe in Hinblick auf eine gute Gesamtperformance zu vermeiden. Es muß im Treiber eine geeignete Threadstruktur gewählt werden, um diesen Anforderungen gerecht zu werden.

Weiterhin ist die Schaffung neuer Synchronisationsmittel notwendig. Es muß sichergestellt werden, daß einerseits kritische Abschnitte nicht zugleich durch zwei Threads betreten werden können, andererseits sollen aber unnötige Wartezeiten vermieden werden. Das heißt insbesondere, daß ein vor einem geschlossenen Semaphor wartender Thread genau dann aufgeweckt wird, wenn der das Semaphor blockierende Thread dieses verläßt.

Kapitel 4

Implementierung

Der in Kapitel 3 entworfene SCSI-Treiber wurde in einer ersten Version implementiert. Grundlage ist der schon bestehende Port des SCSI-Treibers auf L3 (siehe [Meh96]). Der Treiber ist auf die Präsenz des Ressourcen-Managers ([HW97]) und des Dateisystems angewiesen. Letzteres fungiert dabei als Pager und Lieferant der Datenpuffer. Weiterhin wird eine kleine C-Bibliothek benötigt, die Funktionen für die formatierte Bildschirm-Ausgabe und Speicheroperationen enthalten muß. Der Ressourcen-Manager verwaltet die Tasks und die Hardwareinterrupts und wird über eine weitere C-Bibliothek angesprochen.

Die Implementierung besteht aus etwa 3000 Zeilen C-Quellcode und 1000 Zeilen in Header-Dateien in der Linux-Emulation. Hinzu kommen etwa 5000 aus Linux 2.0.21 stammende Zeilen C-Quellcode des SCSI-Treibers, die untersucht und teilweise geringfügig geändert werden mußten. Die hardwareabhängige Schicht (Gerätetreiber für SCSI-Hostadapter der Familie ncr53c8xx) besteht aus etwa 10000 Zeilen, Änderungen wurden hier nicht vorgenommen.

Die Hauptänderungen am Linux-SCSI-Treiber betreffen das Entfernen der Unterstützung für ISA-Busmaster-Hostadapter und das Ändern der Verhaltensweise bei leeren Auftragswarteschlangen.

Viele Parameter lassen sich bei der Erstellung des Treibers konfigurieren (Datei `config`).

4.1 Emulationsumgebung für SCSI-Treiber

Die Emulation für den Linux-SCSI-Treiber auf L4 stellt eine Sammlung von Bibliotheksfunktionen dar, deren Aufgabe in der Abbildung der angebotenen L4-Funktionalität auf die Erfordernisse des Hardware-Treibers besteht. Entwickelt wurde die Bibliothek für SCSI-Hostadapter der Klasse Sym53c7,8xx der Firma Symbios Logic, daneben wurde die einwandfreie Funktion an der Hostadapterfamilie von BusLogic überprüft.

Wenngleich in Linux zwar eine definierte Schnittstelle für die Kommunikation zwischen *lowlevel*-Treiber und *midlevel*-SCSI-Treiber definiert ist, so kann doch jeder Treiber grundsätzlich jede Funktion im monolithischen Linux-Kern aufrufen. Aus diesem Grund ist es kaum möglich, eine kompromißlos vollständige Emulation für alle Linux-SCSI-Treiber zu entwickeln. Zu beachten ist weiterhin, daß die Emulationsbibliothek auf der Grundlage von Linux 2.0.21 entwickelt wurde und somit insbesondere für Treiber aus Linux 2.1.x Anpassungen vorgenommen werden müssen.

4.1.1 *Memory mapped Input/Output*

Im Gegensatz zur Ein-/Ausgabe über Ports hat die Kommunikation mit der Hardware über gemeinsamen Speicher den Vorteil einer geringeren Latenz. Unter L4 (Pentium-Kern) werden Hardware-

Speicherbereiche über virtuelle Seiten der Größe 4MB angesprochen (vgl. [Lie96]. Jeder Treiber, der auf solchen Speicher einer Hardwarekomponente zugreifen möchte, beantragt das Mappen dieser Seite in seinen Adreßraum. Aus Sicherheitsgründen sollte eine Seite nur in eine Task gemappt sein. Der Resourcen-Manager trägt dem Rechnung und verbietet das Abbilden einer solchen Seite in verschiedene Adreßräume. Liegen derartige Speicherbereiche von zwei Komponenten (z.B. SCSI-Hostadapter und Grafikspeicher) auf derselben 4MB-Seite, kann eine Komponente nicht angesteuert werden, da deren Speicher von der anderen Task gesperrt wurde.

Aus dieser Lage existieren zwei Auswege: Entweder, die Speicherbereiche lassen sich auf andere physikalische Adressen umstellen, oder der Ressourcen-Manager verwaltet diese Seiten selbst und kann sie eventuell an zwei verschiedene Tasks weitergeben.

4.1.2 Hardwareunterbrechungen

Unterbrechungen des Programmflusses seitens des SCSI-Hostadapters (IRQ) werden an eigens dafür eingerichtete Threads gesandt. Eine spezielle Einrichtung moderner Chipsätze, *shared interrupts*, bei dem sich mehrere Geräte eine Unterbrechungsleitung teilen können, wird vom Treiber nicht unterstützt. Der Grund liegt auch hier in der exklusiven Verwaltung von Ressourcen durch den Ressourcen-Manager: Ein IRQ kann immer nur einem Thread zugeordnet werden. Als praktische Konsequenz daraus muß dafür Sorge getragen werden, daß das Rechner-BIOS nicht zwei Hostadaptern den gleichen Interrupt zuordnet.

4.1.3 Systemzeit

Die Systemzeit wird unter L4 durch die *kernel time* (siehe [Lie96]) repräsentiert. Diese hat eine Auflösung von einer Mikrosekunde und wird jede Millisekunde (PC mit APIC) bzw. alle zwei Millisekunden aktualisiert. Die Linux-Kern-Variable `jiffies` wird im SCSI-Treiber aus der *kernel time* abgeleitet. Der in Linux definierte Faktor `HZ` dient der Auflösungsanpassung. Zu beachten ist der neue Wertebereich: Relative Zeitangaben dürfen eine Zeitspanne von 4294 Sekunden (etwa 1:11h) nicht überschreiten. Wenn das nicht ausreicht, kann die Systemzeit durch periodisches Aktualisieren einer eigenen `jiffies`-Variable in einem eigenen Thread nachgebildet werden. Eine hohe Genauigkeit der Linux-Zeit ist nur für Messungen interessant. Für den Echtzeitteil werden nur die unteren 32 Bit der L4-Kern-Zeit benutzt. Das ist solange statthaft, wie man nur kurze zeitliche Differenzen erfassen muß.

4.2 Threadstruktur

Der Port besteht aus mindestens vier unabhängigen Threads (siehe Abbildung 4.1 auf der nächsten Seite). Ggf. wird ein zusätzlicher Thread für die Emulation der Linux-Zeit (`jiffies`) erzeugt.

Der ***Super*-Thread** entspricht dem Linux-Kern-Thread. Er ist für die Initialisierung der Hardwarekomponenten und die Abarbeitung von Aufträgen verantwortlich. Die Aufträge werden zeitgesteuert (siehe Abschnitt 3.2.5 auf Seite 31) abgesetzt.

Pro verwendeter Hardwareunterbrechung wird ein weiterer (***IRQ*-Thread**) benötigt, der in einer Endlosschleife auf Interrupts wartet und schnellstmöglich abarbeitet. Prinzipiell ist für jeden angeschlossenen SCSI-Hostadapter eine Interruptleitung erforderlich. Das bereits vom Linux-Kern angebotene *Interrupt-Sharing*, bei dem sich mehrere Hostadapter eine Leitung teilen, wird vom L4-Port und dem Ressourcen-Manager noch nicht unterstützt.

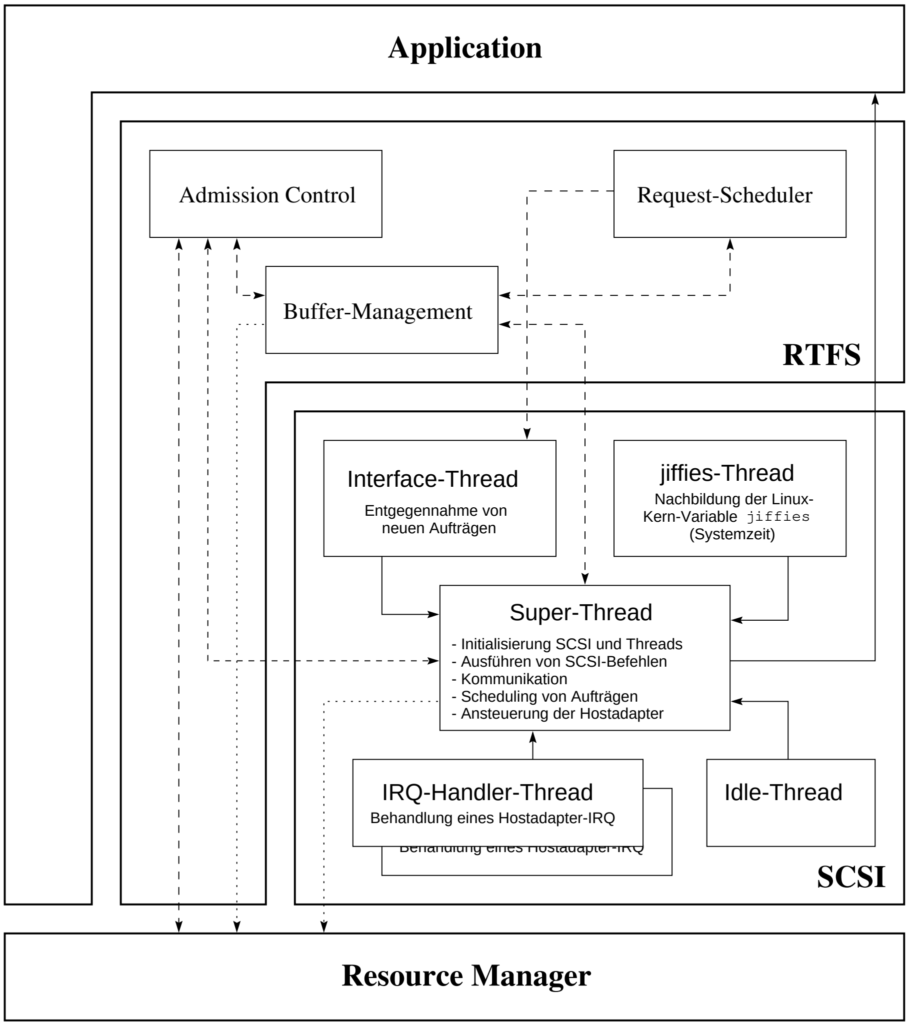

Abbildung 4.1: Der Aufbau des SCSI-Treibers und das Zusammenwirken mit den Komponenten des Dateisystems

Die Schnittstelle zu den anderen Komponenten des Dateisystems und dem Linux-Stub bildet der ***Interface*-Thread**. Er nimmt Aufträge auf verschiedene Art und Weise an und sortiert diese in die Auftragsliste ein, die dann vom *Super*-Thread abgearbeitet wird. Dieser Thread hat eine geringere Priorität als die Threads zur Interruptbehandlung und der *Super*-Thread.

Ein ***Idle*-Thread** ist immer dann aktiv, wenn keiner der anderen Threads Rechenzeit beansprucht. Seine einzige Aufgabe besteht darin, in längeren Zeitabschnitten ($\approx 100ms$) eventuelle Zeitüberschreitungen zu behandeln. Letztere treten nur dann auf, wenn eine SCSI-Operation nicht in der vorgegebenen Zeit ausgeführt wird, weil z.B. das angesprochene Gerät nicht bereit ist. Solche Zeitüberschreitungen beruhen immer auf Ausnahmesituationen und sind deshalb nicht zeitkritisch.

4.3 Synchronisationsmittel

Aufgrund der in Abschnitt 3.6 auf Seite 36 beschriebenen Anforderungen wurden auf L4 basierende Synchronisationsmittel geschaffen.

Jeder Thread des SCSI-Treibers besitzt eine eindeutige Kennung: Das oberste Doppelwort auf dem Stack erhält beim Start einen Zeiger auf thread-interne Daten. Diese Struktur enthält unter anderem auch den aktuellen Threadzustand (siehe Abbildung 4.1).

Tabelle 4.1: Mögliche Zustände der Threads des SCSI-Treibers (`scsi_thread.state`)

Zustand	Bemerkung
TH_UNBLOCK_UNLOCK	Normaler Zustand „*bereit*"
TH_UNBLOCK_LOCK	Der Thread wird innerhalb der nächsten Zeit in den Zustand „*blockiert*" übergehen, im Moment sind keine Zustandsänderungen durch andere Threads erlaubt.
TH_BLOCK_UNLOCK	Der Thread ist im Zustand „*blockiert*", andere Threads dürfen den Zustand dieses Threads verändern (insbesondere ihn durch `l4_thread_ex_regs()` in den Zustand „*bereit*" überführen
TH_BLOCK_LOCK	Der Thread soll durch einen „Weck"-Thread demnächst in den Zustand „*bereit*" überführt werden. Zustandsänderungen durch andere Threads sind nicht erlaubt. Eine *wakeup*-Funktion übernehmen die Threads *IDLE* (periodisches Aufwecken (alle 100ms) zur Überwachung von *timeouts*) und *IRQ* (Aufwecken schlafender Threads, weil ein aufgetretener Interrupt den Abschluß eines SCSI-Auftrages, worauf diese Threads wahrscheinlich gewartet haben, bedeutet.

Befindet sich ein Thread im Zustand „*blockiert*", wird er in eine entsprechende Warteschlange eingereiht und entweder automatisch nach einer vorbestimmten Zeit (*Idle*-Thread) oder explizit durch einen anderen Thread aufgeweckt.

4.3.1 Kritische Abschnitte

Programmabschnitte, die nicht von zwei Threads zugleich betreten werden dürfen, werden durch die Funktionen `enter_critical()` und `leave_critical()` gesichert. Muß ein Thread B bei `enter_critical()` auf einen anderen Thread A warten, so führt er ein *closed wait* auf diesen Thread aus. Thread A weckt Thread B durch *send*, sobald er `leave_critical()` erreicht.

4.3.2 Semaphore

Bei Verwendung von Semaphoren ist oftmals nicht bekannt, welcher Thread ein geschlossenes Semahor wieder öffnet (und zwar genau dann, wenn der Anfangszustand des Semaphores „*geschlossen*" ist). Aus diesem Grund kann hier die *closed wait*/*send*-Methode nicht angewandt werden. Ein Thread B, der an einem geschlossenen Semaphor warten muß, geht daher in den Zustand *closed wait* auf *L4_NIL_ID*. Der erweckende Thread A überführt Thread B durch direktes Setzen des Befehlszeigers vom Zustand „*blockiert*" in den Zustand „*bereit*" (`l4_thread_ex_regs()`).

4.3.3 Expliziter Aufruf des Schedulers

Linux ruft an solchen Stellen explizit den Scheduler durch `schedule()` auf, an denen z.B. aufgrund von nicht aufnahmebereiten Warteschlangen gewartet werden muß. Unter L4 existiert zwar der Funktionsaufruf `l4_thread_switch(L4_NIL_ID)` zum Umschalten auf einen vom Kern bestimmten

Nutzer-Thread. Bei Verwendung verschiedener Prioritäten ist damit aber ausgeschlossen, daß Threads mit niedrigerer Priorität als der aktuelle Thread Rechenzeit erhalten können.

Aufgrund dessen ist es sinnvoll, den Thread, der `schedule()` aufrufen möchte, durch Warten auf `L4_NIL_ID` in den Zustand „*blockiert*“ zu überführen, um dessen Rechenzeit voll anderen Threads zugute kommen zu lassen. Der *Idle*- und der *IRQ*-Thread übernehmen das Aufwecken der blockierten Threads in bestimmten Zeitabständen (siehe Abschnitt 4.3 auf Seite 39).

4.4 Schnittstelle

Aus den beiden in Kapitel 3 vorgestellten Modellen zur Auftragsplanung (Abschnitt 3.4 auf Seite 32) wurde die zweite Variante ausgewählt, insbesondere deshalb, weil damit eine bessere Abstimmung zwischen Dateisystem und SCSI-Treiber möglich ist und weniger Overhead für die Berechnung der Zeitpunkte, wann die Aufträge ausgeführt werden müssen, anfällt.

Die Aufgabe des SCSI-Treiber besteht darin, Aufträge (siehe 4.4.1) in einen Auftragsplan einzufügen und in freien Slots dieses Planes nicht-zeitkritische Aufträge unterzubringen. Das Lesen/Schreiben von **Hilfsstrukturen** (Inodes, Freispeicherlisten) wird vom Dateisystem wie zeitkritische Aufträge behandelt und muß somit nicht gesondert berücksichtigt werden.

Die Übergabe der Parameter an den SCSI-Treiber kann auf verschiedene Art und Weise geschehen:

- **Feld von Aufträgen (*request arrays*)**
  Zeitkritische Aufträge werden vorzugsweise als Feld übergeben, das in den Adreßraum des SCSI-Treibers abgebildet wird (*Flexpage*). Jeder Eintrag enthält die in Abschnitt 4.4.1 beschriebenen Einträge.

- ***short message*, *string message***
  Nicht-zeitkritische Aufträge erreichen den SCSI-Treiber entweder mit oder ohne Angabe eines Datenpuffers. Im letzteren Fall werden die Daten durch den L4-Kern zwischen den Prozessen kopiert. Nicht-zeitkritische Aufträge blockieren den aufrufenden Thread solange, bis sie ausgeführt worden sind.

Ein eigener *Interface*-Thread (Datei if.c) wartet in einer Endlosschleife auf neue Aufträge und sortiert sie anhand der eindeutigen Angabe der Slotnummer in den Auftragsplan ein. Der Eintrag im Plan besteht dabei nur aus einem Zeiger auf den Auftrag im Feld. Die Verwaltung der *map*-Adressen für Auftragsfelder erfolgt über eine Freispeicherliste. Wenn alle Aufträge eines Feldes abgearbeitet sind, wird die zugehörige Seite freigegeben. Das Dateisystem kann sich durch Überprüfen des Fehlercodes vom ordnungsgemäßen Ablauf überzeugen.

Nicht-zeitkritische Aufträge werden in einen Ringpuffer eingeordnet, aus dem sie der Reihe nach ausgelesen werden und in leere Slots der aktuell ausgeführten Periode eingeordnet werden. Dabei wird die in Anschnitt 3.4.2 auf Seite 34 angedeutete Überprüfung auf Konflikte durchgeführt. Eine Optimierung dieser Aufträge z.B. nach SCAN (Abschnitt 2.1.3) ist möglich, aber nicht notwendig, da die Ausführung eines Auftrages innerhalb eines Slots (Abschnitt 3.2.4 auf Seite 29) garantiert ist.

4.4.1 Aufbau der Aufträge

Jeder Auftrag an das SCSI-Subsystem hat den in Abbildung 4.2 auf der nächsten Seite erkennbaren Aufbau. Im Gegensatz zu Linux (siehe Abschnitt 2.4.2 auf Seite 22) sollen einmal Festplatten durch logische Partitionsnummern verwaltet werden ([Reu98]). Das Makro `disk_from_partition()` liefert die Rückabbildung von der Partitionsnummer auf die physische Platte, und ist dann entsprechend anzupassen. Eine Kenntnis dieser Zuordnung ist notwendig, um Konfliktsituationen, wie in Abschnitt 3.4.2 beschrieben, verhindern zu können. Der Eintrag `map_address` enthält eine physische Adresse, die direkt vom SCSI-Hostadapter gelesen bzw. beschrieben werden kann.

```
struct request {
    unsigned int period;
      /* diskrete Zeitangabe, in welcher Periode der Auftrag auszuführen ist */
    unsigned int slot;
      /* welcher Slot der Periode soll belegt werden */
    scsi_block_t blk;
      /* Angaben über Partitionsnummer, Blocknummer und -länge */
    byte_t *map_address;
      /* Adresse des Schreib-/Lesepuffers */
    dword_t status;
      /* Status Bits */
    dword_t reserved;
};
```

Abbildung 4.2: Auftragsstruktur, wie sie an den SCSI-Treiber gesendet wird. Sie ist genau 32 Byte lang, entsprechend der Größe einer *cache line* bei modernen Prozessoren (Intel Pentium, Intel Pentium Pro, AMD K6)

4.5 Stand der Implementierung

Der Linux-SCSI-Treiber wurde vollständig auf L4 portiert. Eine vereinfachte C-Bibliothek und ein auf einer Arbeit von Torsten Paul basierender Pager ([Pau97]) bieten die notwendigen Voraussetzungen einer Laufzeitumgebung. Der L4-Treiber ist in der Lage, mehrere SCSI-Hostadapter im System zu erkennen und anzusprechen. Eine einfache Unterstützung von CD-ROM-Geräten ist vorbereitet, aber nicht getestet.

Eine erste Version der Auftragserfassung zum Testen der erreichbaren Performance ignoriert Zeitangaben in den Aufträgen und führt sie in der Ankunftsreihenfolge, wie in Abschnitt 3.2.4 auf Seite 29 beschrieben, aus.

Zur Zeit wird an einer erweiterten Version gearbeitet, die das Einsortieren der Aufträge, wie in Abschnitt 3.4.2 auf Seite 33 beschrieben, unterstützt. Die Ausführungszeitpunkte werden durch Angabe der absoluten Slotnummer (in Notation ${}_{Slot}Periode$) festgelegt. Ein Funktionsaufruf `scsi_sync()` wurde für die Synchronisation zwischen SCSI-Treiber und Dateisystem vorgesehen. Damit wird für jeden SCSI-Bus festgelegt, wieviel Slots verwendet werden, wie lang ein Slot ist (Angabe in ms) und wann in Bezug zur Systemzeit die erste Periode beginnt.

Das Scheduling im *Super*-Thread erfolgt auf Basis von *ipc wait*. Wenn ein neuer Slot beginnt, wird die logische Zeit des Treibers um einen festen Wert inkrementiert. Sodann wird der Super-Thread solange blockiert, bis die Systemzeit (*kernel time*) den Wert der logischen Zeit erreicht hat. Dieser Algorithmus bedingt zwar eine geringe Schwankung im Beginn der Slots (maximal $\pm 2ms$), führt aber bei Betrachtung eines längeren Zeitraumes zur Synchronisation mit der Systemzeit und somit mit allen anderen Prozessen. Die Länge eines Slots kann nur in Schritten von $2ms$ verändert werden.

Die beschriebene zweite Version des Treibers ist bereits implementiert, jedoch noch nicht unter realen Bedingungen (zusammen mit dem Dateisystem) getestet worden (Stand Januar 1998).

Kapitel 5

Leistungbewertung

5.1 Last-Messungen

Für ein echtzeitfähiges System ist es interessant, wie sich die einzelnen Komponenten unter Lastbedingungen beeinflussen. Bei hohen Datenraten stellt entweder der System-Bus oder der Hauptspeicher den Flaschenhals dar (Abbildung 5.1).

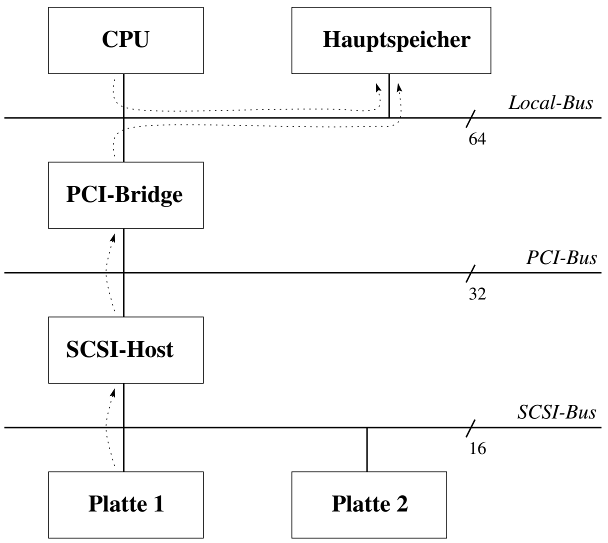

Abbildung 5.1: Anbindung des SCSI-Busses an den Systembus und Beeinflussung zwischen CPU und SCSI-Host

Es wurde versucht, zumindest ansatzweise zu klären, inwieweit sich Datenübertragungen vom SCSI-Bus auf Hauptspeicheroperationen auswirkt. Das verwendete Testprogramm erzeugt zwei Datenströme: Das SCSI-Subsystem liefert Daten von den angeschlossenen Platten per Busmaster über den PCI-Bus. Gleichzeitig werden Daten zwischen CPU und Hauptspeicher bewegt (Lesen, Schreiben oder Kopieren). Die von beiden Strömen übertragenen Daten werden erfaßt und verglichen (siehe Tabelle 5.1 auf der nächsten Seite).

In der letzten Spalte steht ein Lastfaktor, der angibt, um wieviel langsamer Operationen über dem Hauptspeicher ablaufen, wenn parallel Daten von SCSI-Platten gelesen werden. Zur Berechnung wird vom

Tabelle 5.1: Beeinflussung von SCSI- durch Hauptspeicheroperationen. Die letzte Spalte gibt an, um welchen Faktor eine SCSI-Operation die jeweilige Hauptspeicheroperation verlangsamt. Plattform: Pentium Pro 200MHz, 8kB L1-Cache, 512kB L2-Cache (*write back*, *write buffer*)

Operationen	Auslastung SCSI	Speicher-Last[a] MB/s	$\sum$ SCSI-Last[b] MB/s	Lastfaktor[c]
memory load	–	133,2	–	–
mit 2 Platten	gering	122,2	6,3	1,74
mit 2 Platten	hoch	111,7	12,5	1,72
mit 5 Platten	gering	106,2	15,6	1,73
mit 5 Platten	hoch	88,7	29,9	1,49
memory store	–	48,6	–	–
mit 2 Platten	gering	44,2	6,3	0,70
mit 5 Platten	hoch	40,1	12,5	0,68
mit 2 Platten	gering	37,8	15,6	0,69
mit 5 Platten	hoch	32,9	30,0	0,52
memory load+store	–	34,2	–	–
mit 2 Platten	gering	31,0	6,3	0,50
mit 2 Platten	hoch	29,5	12,5	0,38
mit 5 Platten	gering	26,4	15,6	0,50
mit 5 Platten	hoch	23,4	27,1	0,40

[a] Durchsatz der Speichertask bei verschiedenen Operationen
[b] Summe des Durchsatzes beim Lesen von SCSI-Festplatten
[c] Siehe Text

Speicherdurchsatz ohne SCSI der Speicherdurchsatz mit SCSI abgezogen und durch den Wert des Datendurchsatzes auf dem SCSI-Bus dividiert[1]. Der Thread zum Erzeugen der Speicherlast erhält infolge der Aktivität des SCSI-Treibers weniger Rechenzeit. Deshalb wird mittels Systemaufruf festgestellt, wieviel Rechenzeit diesem Thread während der Messung zur Verfügung stand und mit Hilfe dieses Wertes der Speicherdurchsatz errechnet.

Ein Interpretation der Meßergebnisse ist nicht trivial. Vergleicht man die erzielten Datenraten der Speicher-Last-Task mit und ohne SCSI-Transfer, stellt man fest, daß zwischen den Daten vom SCSI-Hostadapter und den zwischen CPU und Speicher ausgetauschten Daten eine Wechselwirkung besteht. Je nach der Art und Weise der Speicheroperation (nur Lesen, nur Schreiben oder Schreiben nach Lesen), ist diese Beeinflussung verschieden. Erwartungsgemäß sollte sich daraus immer ein Faktor $x \leq 1$ ergeben. Gerade die relativ großen Werte bei *memory load* sind wahrscheinlich nur mit dem Einfluß des CPU-Caches und diverser Puffer des Pentium Pro zu erklären.

5.2 Performance

Die erzielbare Datenrate beim Lesen/Schreiben von/auf SCSI-Festplatten ist nicht mit Linux vergleichbar, da hier Platten unter *worst-case*-Bedingungen betrieben werden. Die Abbildungen 5.2 und 5.3 auf der nächsten Seite zeigen Kennlinien der Festplatte IBM DORS 32160.

Je größer die Blockgröße, desto größer ist auch die erzielbare Datenrate, da sich die Anzahl der Kopfbewegungen relativ vermindert (größere Lokalität der Daten). Entscheidend ist auch die Größe des

[1] Beispiel: Der erste Lastfaktor aus der Gruppe *memory load* ergibt sich folgendermaßen: $1,74 = \frac{133,2\frac{MB}{s} - 122,2\frac{MB}{s}}{6,3\frac{MB}{s}}$

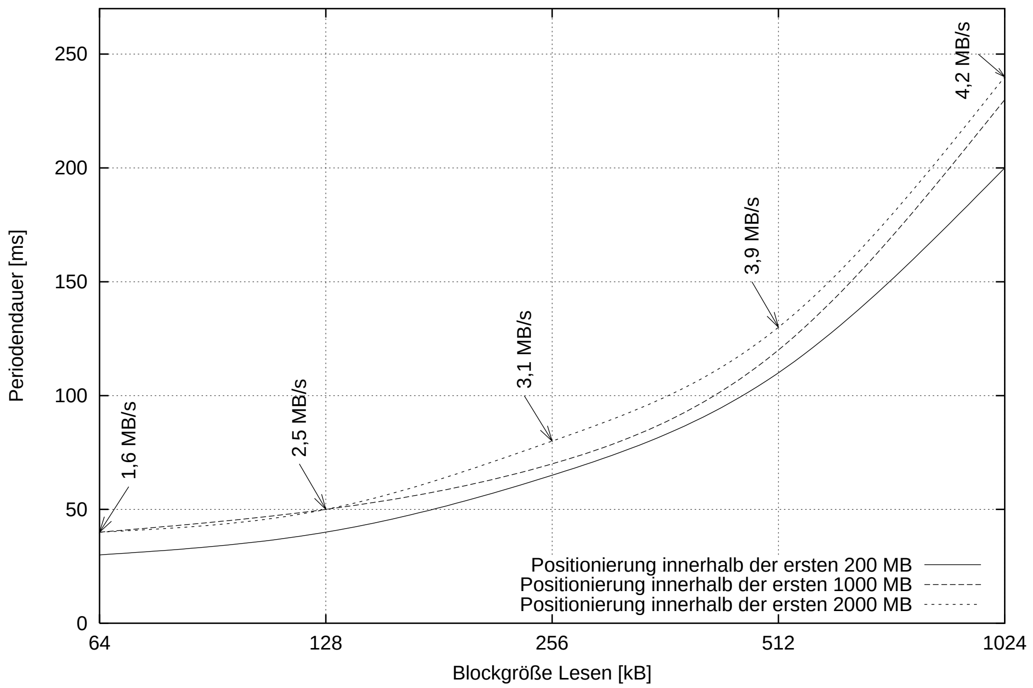

Abbildung 5.2: Slotlänge t_s Fall **LESEN** in Abhängigkeit von der Datenblockgröße und des Positionierbereiches. Festplatte IBM DORS 32160.

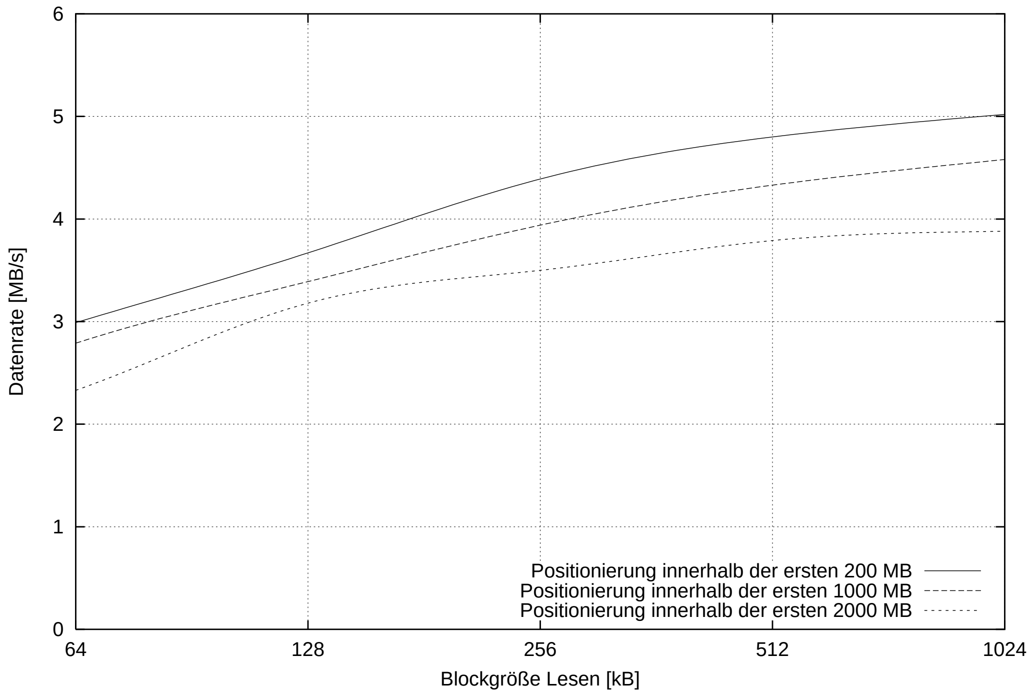

Abbildung 5.3: Erzielbare Datenraten **LESEN** in Abhängigkeit von der Datenblockgröße und des Positionierbereiches. Festplatte IBM DORS 32160.

Zugriffsbereiches, wie ebenfalls aus den Diagrammen zu entnehmen ist. Durch geschickte Einteilung der Platten in Partitionen kann man das Zugriffsverhalten verbessern.

Da, wie in Abschnitt 4.5 auf Seite 42 erläutert, das Scheduling der Aufträge auf Basis von *ipc wait* durchgeführt wird, sind die Meßwerte noch recht ungenau, insbesondere bei kurzen Periodenlängen. Insofern stellen die dargestellten Werte die untere Grenze dar. Mittels eines Scheduling mit feinerer Granularität ließen sich auf jeden Fall bessere Ergebnisse erreichen.

Der Schreib-Cache aller Festplatten wurde gemäß Abschnitt 3.2.6 auf Seite 31 abgeschaltet (*write through*. Praktische Messungen ergeben deshalb einen nicht meßbaren Unterschied zwischen den Modi **LESEN** und **SCHREIBEN**. Auf eine Wiedergabe der Kurven für die zweite Zugriffsart wurde deshalb verzichtet.

Kapitel 6

Zusammenfassung und Ausblick

Die vorliegende Arbeit beschreibt eine erste Implementation eines SCSI-Treibers auf L4. Aufträge werden zeitlich gesteuert an den SCSI-Hostadapter abgesetzt, womit zwar nicht die maximal mögliche Bandbreite der Festplatten ausgenutzt wird, dafür aber zeitliche Zusicherungen getroffen werden können. Durch geeignete Wahl der Plattenzahl kann die Bandbreite des SCSI-Busses unter den technischen Grenzen gut ausgenutzt werden.

Die Implemenatation ist noch unvollständig und bedarf einer genaueren Testung. Einige Details sind eventuell zu überarbeiten, wenn die anderen Komponenten des Dateisystems ([Reu98], [Rud98]) verfügbar sind.

Zukünftige Arbeiten an diesem Programm sollten besonders unter folgenden Gesichtspunkten ausgewählt werden:

- Aufstellen eines genaueren Platten-Modells durch Gewinnung zusätzlicher Plattenparameter (z.B. Zonen gleicher Anzahl von Sektoren pro Spur) und zeitliche Bestimmung des SCSI-Ablaufes durch Logikanalysator.
- Erstellung eines Meßprogrammes zum automatischen Ausmessen der minimalen Slotlänge unter gegebenen Voraussetzungen.
- Berücksichtigung des Schreib-Caches bei Festplatten. Bis dato wird dieser Zwischenspeicher deaktiviert, um beim Schreiben ein gleiches Verhalten wie beim Lesen zu erhalten.
- Untersuchung des Verhaltens von Platten bei Variation verschiedener Parameter der *mode pages*.
- Implementierung für die Ansteuerung mehrerer SCSI-Busse.
- Verbesserte zeitliche Auflösung beim Scheduling, um die Slotzeiten genauer einhalten zu können und damit weniger Sicherheits-Zeit einplanen zu müssen.
- Erweiterung für andere SCSI-Gerätegruppen (CD-ROM-Laufwerk, CD-ROM-Brenner)

Anhang A

Glossar

Admission Algorithmus, der eine Ja-/Nein-Entscheidung darüber fällt, ob eine neue Systemlast bedient werden kann, oder nicht (Akzeptanz-Test). Dabei werden Systemparameter (verfügbare Hardware, Auslastung) und Parameter des Datenstromes berücksichtigt.

Block-Auftrag Hier als Anforderung (engl. *request*) der Linux-Blockgeräteschicht an den SCSI-Treiber verwendet.

Command Linking Mehrere SCSI-Kommandos für ein Target werden als Kette übertragen, dabei werden einige Busphasen nicht durchlaufen und die Kommandos atomar ausgeführt.

Datenblock Die Menge von Daten, die bei einem SCSI-Kommando mit der Festplatte ausgetauscht (lesend oder schreibend) werden. Ein Datenblock liegt immer physisch zusammenhängend auf der Platte, der äquivalente Datenbereich im Hauptspeicher muß dagegen nicht zusammenhängend sein (siehe *scatter/gather*).

LUN Ein logisches Subgerät eines SCSI-Gerätes, auf einem solchen können sich mehrere LUN's befinden. Ein Beispiel sind CD-Wechsler mit mehreren LUN's, die Anzahl der LUN's entspricht der CD-Anzahl.

Initiator SCSI-Gerät, das eine Aktion auf dem SCSI-Bus anstößt (meist SCSI-Hostadapter)

Mapping Der Zugriff auf Daten von Blockgeräten geschieht üblicherweise durch Angabe einer Sektornummer. Der physische Zugriff auf eine Festplatte muß demgegenüber mittels Zylinder, Seiten- und Sektornummer erfolgen. Die Abbildungsvorschrift der logischen auf die physischen Parameter efolgt durch eine Abbildungsvorschrift, auch *Mapping* genannt.

Nutzdaten bezeichnet den Anteil Daten auf einer Festplatte, der Nutzerdaten aufnimmt. Neben den N. befinden sich noch Formatierungs- und Servo-Informationen auf der Platte. Der Anteil an N. beträgt überlicherweise 60-80% der Gesamtdaten ([Gre94]) einer Platte.

Prefetching Ein Mechanismus zum vorzeitigen Einlesen von später benötigten Daten in einen Zwischenspeicher auf der Festplatte.

Scatter/-Gather Mechanismus, der es erlaubt, mehrere physisch nicht zusammenhängende Hauptspeicherbereiche mit einem SCSI-Kommando zu bedienen. Dabei wird eine *scatter/gather*-Tabelle benutzt, die die Lage der Hauptspeicherbereiche enthält.

SCSI-Auftrag *SCSI request* innerhalb der drei Software-Schichten des Linux-SCSI-Treibers

SCSI-Kommando Bytefolge mit einer maximalen Länge von 12, die den Befehl für den Hostadapter in binärer Form enthält

Sektor Die kleinste Informationseinheit, die von einem SCSI-Gerät gelesen werden kann. Bei Festplatten umfaßt ein Sektor überlicherweise 512 Byte, theoretisch möglich sind auch 256 Byte und 1024 Byte. CD-ROM- Geräte haben Sektoren mit 2048 Byte Nutzdaten. Datengrößen werden im Kommunikationsprotokoll mit der Platte immer in Vielfachen der Sektorgröße angegeben.

Tagged Queuing Erlaubt eine Steuerung der Reihenfolge der Kommandos innerhalb des Targets (siehe Abschnitt 2.4.3 auf Seite 24)

Target SCSI-Gerät, welches eine vom *Initiator* gewünschte Aktion ausführt (z.B. SCSI-Platte: Aktion ist Lesen oder Schreiben von Daten).

Zoning Heutige Festplatten besitzen aufgrund relativ konstanter Sektorlänge eine unterschiedliche Anzahl Sektoren pro Spur auf verschiedenen Plattenbereichen. Auf den äußeren Bereichen der Platte, wo die Spurlänge größer ist, werden mehr Sektoren untergebracht, als in den inneren Bereichen der Platte. Diese Anpassung der Datendichte erfolgt in Zonen (bis etwa 20) mit gleichem Sektoren/Spur-Verhältnis. Das Verhältnis der Anzahl Sektoren/Spur äußerer Bereiche zur Anzahl Sektoren/Spur in den inneren Bereichen liegt heute bei 1,3:1 oder 2:1 (siehe [BSS95]). Auch *zone bit recording – ZBR* – genannt.

Literaturverzeichnis

[Bög97] BÖGEHOLZ, Harald: Platten-Karussell – Festplatten mit EIDE- und SCSI-Schnittstelle im Überblick. **In:** *c't 14/97*. Heise Verlag Hannover, November 1997, S. 160

[BSS95] BÖGEHOLZ, Harald ; SCHNEIDER, Ralf ; SCHNURER, Georg: Byte-Milliardäre – Moderne Festplatten mit SCSI- und IDE- Interface. **In:** *c´t 6/95*. Heise Verlag Hannover, Juni 1995, S. 120

[Ess97] ESSER, Stefan. Der NCR-Treiber und breaks. private Mail. Oktober 1997

[Gre94] GRELL, Detlef: Konstanter Trend – Festplattenentwicklung ohne Sprünge. **In:** *c´t 5/94*. Heise Verlag Hannover, Mai 1994, S. 82

[Ham97] HAMANN, Dr. rer. nat. Claude-J. Das Problem. private Mail. Oktober 1997

[Hoh96] HOHMUTH, Michael: *Linux-Emulation auf einem Mikrokern*, Institut für Betriebssysteme, Datenbanken und Rechnernetze, TU Dresden, Diplomarbeit, 1996

[Här97] HÄRTIG, Prof. Dr. Hermann. Ein alternatives Planungsmodell. mündliche Mitteilung. Dezember 1997

[HW97] HOHMUTH, Michael ; WOLTER, Jean: *rmgr – L4 System Resource Manager*. Institut für Betriebssysteme, Datenbanken und Rechnernetze, TU Dresden, 1997. – Zu finden unter http://os.inf.tu-dresden.de/L4/l4libman/rmgr.html

[IBM97] IBM Storage: *Hard disk drive technical support, quick specs*. 1997. – Zu finden unter http://www.storage.ibm.com/techsup/hddtech/fedspd.pdf

[Lie96] LIEDTKE, Jochen: *L4 Reference Manual for 486, Pentium and Pentium Pro*. zu finden auf http://os.inf.tu-dresden.de/L4/l4refx86.ps.gz: IBM Watson Technical Report, 1996

[Meh96] MEHNERT, Frank. Portierung des SCSI-Gerätetreibers von Linux nach L3. Institut für Betriebssysteme, Datenbanken und Rechnernetze, TU Dresden, Großer Beleg. November 1996

[Pau97] PAUL, Torsten. Videopräsentation mit Echtzeitsystemen. Institut für Betriebssysteme, Datenbanken und Rechnernetze, TU Dresden, Großer Beleg. 1997

[Reu98] REUTHER, Lars: *Entwicklung eines echtzeitfähigen Dateisystems*, Institut für Betriebssysteme, Datenbanken und Rechnernetze, TU Dresden, Diplomarbeit, Januar 1998

[Rud98] RUDOLPH, Sven: *Admission Control für ein echtzeitfähiges Dateisystem*, Institut für Betriebssysteme, Datenbanken und Rechnernetze, TU Dresden, Diplomarbeit, 1998

[Sch93] SCHMIDT, Friedhelm: *SCSI-Bus und IDE-Schnittstelle*. Addison-Wesley (Deutschland) GmbH, 1993

[Sch95] SCHNURER, Georg: Fire, Fibre, SSA – Die neuen seriellen Massenspeicherschnittstellen. **In:** *c´t 6/95*. Heise Verlag Hannover, Juni 1995, S. 126

[SCS93] Information technology - Small Computer System Interface - 2. Arbeitspapier. 1993. – Zu finden unter http://scitexdv.com/SCSI2/

[Sea97] Seagate: *Seagate SCSI Quick Specifications*. 1997. – Zu finden unter http://www.seagate.com/support/disc/specs/qckspec2.shtml

[SGRV97] SHENOY, Prashant J. ; GOYAL, Pawan ; RAO, Sriram S. ; VIN, Harrick M.: Symphony: An Integrated Multimedia File System / Department of Computer Sciences, University of Texas at Austin. 1997. – Forschungsbericht

[TN96] TEZUKA, Hiroshi ; NAKAJIMA, Tatsuo: Simple Continuous Media Storage Server on Real-Time Mach. **In:** *USENIX 1996 Annual Technical Conference, San Diego, California* The USENIX Association, 1996

[WGP94] WORTHINGTON, Bruce L. ; GANGER, Gregory R. ; PATT, Yale N.: Scheduling Algorithms for Modern Disk Drives. **In:** *Proceedings of the ACM Sigmetrics Conference*, 1994

[WGPW96] WORTHINGTON, Bruce L. ; GANGER, Gregory R. ; PATT, Yale N. ; WILKES, John: On-Line Extraction of SCSI Disk Drive Parameters / University of Michigan. 1996 (323-96). – Forschungsbericht

[WJB97] WILLIAM J. BOLOSKY, John R. D.: Distributed Schedule Management in the Tiger Video Fileserver. **In:** *Proceedings of the 16th Symposium of Operating Systems Principles*, 1997