Ein-/Ausgabesysteme

Proseminar *Konzepte des Betriebssystementwurfs*
am Lehrstuhl für Rechnerstrukturen
Universität Passau, Sommersemester 2000

MATTIAS SCHLENKER
4. August 2003

Inhaltsverzeichnis

1 I/O-Hardware und ihre Treiber

Computer speichern und verarbeiten Daten durch eine Vielzahl an unterschiedlichen Peripheriegeräten. Dazu gehören typischerweise Massenspeicher, wie Festplatten oder Magnetbandlaufwerke, Netzwerkkarten, serielle und parallele Schnittstellen, Drucker, serielle Terminals, Modems und viele weitere.

Die meisten der Unwägbarkeiten im Umgang mit der Hardware werden vor dem Benutzer durch das Dateisystem oder ähnliche Schnittstellen versteckt. Das I/O-Subsystem versteckt diese Unterschiede auch vor den oberen Schichten des Kernels und bietet so einen einheitlichen Zugriff z.B. auf Dateien oder Laufwerke.

Um der Komplexität grundverschiedener Geräte Herr zu werden, wird für jedes Gerät ein Modul, oder ein Gerätetreiber geschaffen, der alle Informationen enthält, die zum Zugriff auf die entsprechende Hardware benötigt werden. Der Gerätetreiber kennt die Details der verwendeten Hardware (also Adressen der Kontrollregister, Anordnung der Bits in den Registern, das Format von Kommandos oder Fehlermeldungen etc.).

Auf der anderen Seite stellt er eine einheitliche Schnittstelle zum Rest des Betriebssystems zur Verfügung. Mit dieser lässt sich jedes angeschlossene Gerät mit den prinzipiell gleichen Befehlen ansprechen (öffnen, lesen, schreiben, schließen).

1.1 Hardware

Geräte kommunizieren mit dem System in der Regel über einen *Port* (beispielsweise den seriellen Port). Falls sich mehrere Geräte einen Kabelstrang teilen, so nennt man diesen *Bus*. Formal bezeichnet ein Bus nicht nur den Kabelstrang, sondern auch das Protokoll, mit dem die Geräte angesprochen werden.

Busse sind in der Computerarchitektur weit verbreitet. Beispiele sind der *PCI-Bus* oder der *SCSI-Bus (SCSI= Small Computer Systems Interface)*.

Controller sind elektronische Bauteile, die einen Port, einen Bus oder ein Gerät steuern können.

Einfache Controller, wie der für serielle oder parallele Ports bestehen aus einem einzigen Bauteil, das die Signale des Ports kontrolliert. Im Gegensatz dazu bestehen z.B. SCSI-Controller oft aus eigenem Prozessor, Microcode, und Speicherbausteinen, um die aufwendigen Nachrichten des SCSI-Protokolls verarbeiten zu können.

Wie gibt der Prozessor dem Controller die Anweisungen und wie übergibt er die Daten? Dafür hat der Controller ein oder mehrere Register für Daten und Anweisungen. Der Prozessor kommuniziert mit dem Controller, indem er Bitmuster in diese Register schreibt und von dort liest. Eine Möglichkeit der Kommunikation besteht darin, eine spezielle I/O-Anweisung zu benutzen, die den Transfer eines Bytes an eine spezielle I/O-Port-Adresse spezifiziert. Alternativ kann der Controller Memory-Mapping unterstützen. In diesem Fall wird der Controller angesprochen, indem man direkt in den von ihm beanspruchten Speicherbereich schreibt. Mit dieser Methode geht das Schreiben und Lesen großer Datenmengen natürlich schneller als bei der ersten Methode, da der Aufwand einige Millonen Bytes zu schreiben wesentlich geringer ist, als einige Milionen I/O-Anweisungen zu geben. Allerdings ist hier das Risiko größer, in

einen falschen Speicherbereich zu schreiben, geschützter Speicher hilft natürlich dagegen.

Typischerweise besteht der I/O-Port aus vier Registern: *Status, Kontrolle, Daten-ein und Daten-aus*. Das *Statusregister* enthält Bits, die vom Host gelesen werden können. Diese Bits geben an, ob ein Byte zum Lesen im Daten-ein-Register bereitsteht oder ob ein Fehler vorliegt. Ins *Kontrollregister* kann vom Host geschrieben werden, wenn ein Befehl ausgeführt werden soll, oder um den Zustand des Gerätes zu ändern. So kann z.B. ein Bit im Kontrollregister der seriellen Schnittstelle zwischen Voll-Duplex und Halb-Duplex-Kommunikation unterscheiden. Ein weiteres ermöglicht eine Paritätsprüfung, setzt die Wortlänge auf 7 oder 8 Bit oder bestimmt die Geschwindigkeit der Schnittstelle. Das *Daten-ein*-Register wird vom Host gelesen, um Eingaben zu erhalten, während auf das *Daten-aus*-Register geschrieben wird, um Ausgaben zu erzeugen.

Das exakte Protokoll für die Kommunikation zwischen Controller und Host wäre zu aufwendig, aber die Grundzüge des Handshakes sind schnell erklärt. Man geht davon aus, dass zwei Bits benutzt werden, um das Erzeuger-Verbraucher-Verhältnis zwischen Host und Controller zu bestimmen. Der Controller zeigt seinen Zustand über das busy-Bit im Statusregister an. Der Controller setzt dieses busy-Bit, wenn er arbeitet und entfernt dieses, wenn er fertig ist. Der Host signalisiert seine Wünsche, indem er ein Bit ins Befehlsregister setzt, das anzeigt, dass ein Befehl zur Bearbeitung ansteht.

[Silb98], Kap. 12.2.

1.2 Gerätetreiber

1.2.1 Aufbau

Gerätetreiber kann man grundsätzlich in drei Teile aufteilen:

1. Routinen zum Konfigurieren und Initialisieren einer Device.
2. Routinen, die I/O-Anfragen bedienen.
3. Routinen, um Interrupts zu bedienen.

Die *Initialisierungsschicht* dient dazu, herauszufinden ob die Hardware anwesend und funktionsfähig ist, sowie sie zu testen und zu initialisieren.

Zusätzlich kann eine Routine vorhanden sein, die im Falle eines unbehebbaren Systemfehlers den *Speicherinhalt auf die Festplatte* schreibt, um ihn später zu untersuchen.

[McKu96], Kap 6.1 *Device Drivers*.

1.2.2 monolithische Kernel

Viele Unixsysteme benutzen sog. monolithische Kernel. In diesen sind alle Treiber für Hardware fest integriert und laufen im geschützten Modus (Kernelspace). Veränderte Treiber werden in der Regel durch Neukompilieren des Kernels und Rebooten eingebunden. Da keine Kommunikation zwischen verschiedenen Prozessen notwendig ist, ist dieser Aufbau momentan der leistungsfähigste.

Mittlerweile haben die meisten Unices ladbare Kernelmodule, die einzeln angepasst und kompiliert werden können. Dies erlaubt eine hohe Anpassbarkeit, ohne Geschwindigkeitsvorteile aufzugeben. Ausserdem blockieren unbenutzte Kernelmodule keinen Arbeitsspeicher, was bei Systemen mit geringem Speicherausbau sehr vorteilhaft ist Systeme mit ladbaren Modulen sind *Linux* (ab 2.0.x), die meisten *BSD-Systeme* (z.B. FreeBSD ab 2.2.x oder SunOS ab 4.x).

1.2.3 Microkernel

Seit Anfang der neunziger als der Weg in die Zukunft gepriesen, verwaltet bei Microkernelsystemen der eigentliche Kernel nur die grundlegendsten Funktionen von Speicher- und Prozessverwaltung, sowie Scheduling. Hardwaretreiber sind als eigene Programme ausgeführt, die wie normale Anwendungen außerhalb des Speicherbereiches des Kernels arbeiten.

Man unterscheidet zwischen zwei Arten von Microkernelsystemen: solchen, die für jeden Dienst einen dezidierten Server einsetzen, sowie denen, die ein einzigen Server für alle benötigten Dienste benutzen.

Systeme des ersten Typs beinhalten *MachUS, BeOS oder QNX*. Der zweite Typ kommt bei *MkLinux* (ein Mach-Kernel mit Linux-Personality) sowie *Darwin/MacOS X* (ein Mach-Kernel mit 4.4BSD-Personality) zum Einsatz.

2 Typen von Geräten

Man unterscheidet üblicherweise bei Massenspeichern zwischen Character- und Blockdevices ([Crow97], Kap. 15.1.3), die auch im Namensraum des Dateisystems vorkommen und dem Dateisystem, sowie Sockets und dafür verwendete Netzwerkinterfaces. (Anmerkung: letzteres trifft vor allem auf Unix zu, so sind Ein-/Ausgabegeräte bei anderen Betriebssystemen nicht zwingend benutzertransparent).

Blockorientierte Geräte ordnen Daten in Blöcken, während zeichenorientierten Geräten ein beliebig langer Strom von Zeichen übergeben werden kann.

Alle Gerätenamen, die hier verwendet werden, stammen aus FreeBSD 3.4. Einige andere Unices, vor allem Linux, weisen starke Unterschiede bei der Benennung von Geräten auf.

2.1 Zeichenorientierte Geräte

Beinahe jedes Gerät (außer Netzwerkinterfaces) besitzt eine Schnittstelle, mit der es als zeichenorientiertes Gerät angesprochen werden kann. Dazu gehören (unter Unix) z.B. `/dev/ttyX` (ursprünglich *Teletype*, also Fernschreiber [Step99]), `/dev/lpt0` (ursprünglich *Line Printer*), `/dev/rsa0` (erstes SCSI-Bandlaufwerk, *Raw Sequential Access*), `/dev/rwd0` (roher Zugriff auf erste Festplatte, *Raw Winchester Disk*) oder `/dev/null` (das Datengrab).

Alle Beschränkungen der Hardware werden durch die zeichenorientierte Schnittstelle weitergegeben, worauf natürlich Anwendungen Rücksicht nehmen müssen. Dadurch liegen zeichenorientierte Geräte näher am Modell des Datenstroms als andere.

Typischerweise stehen folgende Aufrufe an den Gerätetreiber zur Verfügung *(Hier 4.4BSD, aus [McKu96], andere Unix-Systeme mit leichten Änderungen)*:

open & close Öffnen oder Schließen des zeichenorientierten Gerätes. Im Gegensatz zu blockorientierten Geräten gewähren sie den direkten Zugriff auf das Medium.

read Daten von einem Gerät lesen. Beim rohen Zugriff auf blockorientierte Geräte bedeutet dies lediglich den Aufruf der *physio()*-Routine mit gerätespezifischen Parametern. Bei Terminalgeräten wird die Leseanfrage sofort an den Terminaltreiber weitergegeben. Bei anderen Geräten bedeutet der Leseaufruf, daß die benötigten Daten in den Adressraum des Kernels kopiert werden, bevor sie an das Gerät weitergegeben werden.

write Daten auf ein Gerät schreiben. Symmetrisch zum *read*-Aufruf.

ioctl Operationen ausführen, die sich vom Lesen oder Schreiben unterscheiden. Dieser Eintrag war ursprünglich dazu bestimmt, einen Mechanismus zum Auslesen und Ändern von Parametern von Terminalgeräten zur Verfügung zu stellen. 4.4BSD definiert eine Reihe von Operationen, die von jedem Bandlaufwerk unterstützt werden. Dazu gehören: Positionieren, Statusrückgabe, Schreiben von Dateiendmarken und das Sperren von Bandlaufwerken.

select Überprüfen des Gerätes, ob Daten zum Lesen vorhanden sind oder Platz zum Schreiben frei ist. Dies wird vom *select*-Systemaufruf verwendet, um die Deskriptoren von Gerätedateien zu prüfen. Für den rohen Zugriff auf Blocklaufwerke ist dieser Eintrag bedeutungslos, da der Zugriff ungepuffert erfolgt.

stop Stoppt das Schreiben auf ein Gerät. Dieser Eintrag wird nur für Geräte benutzt, die vom Terminaltreiber angesprochen werden. Bei diesen Geräten bedeutet der Aufruf ein Abbruch der Übertragung, wenn z.B. der Terminaltreiber ein Stoppzeichen, z.B. `^S` empfängt.

mmap Wandelt den Versatz einer Gerätedatei in eine Speicheradresse um. Dieser Eintrag wird vom Virtual-Memory-System aufgerufen, um eine logische in eine physikalische Adresse umzuwandeln. Wandelt z.B. einen Versatz in `/dev/mem` in eine Speicheradresse um.

reset Zurücksetzen des Gerätezustandes nach dem Reset des Gerätebusses. Die *reset*-Routine ruft den Treiber des Busadapters auf, wenn der Reset des Busses erfolgte (z.B. SCSI). Vom Gerätetreiber wird dann erwartet, dass er die Hardware in einen bekannten Zustand versetzt, üblicherweise in den Zustand, der nach dem Booten vorhanden war.

2.2 Blockorientierte Geräte

Festplatten sind ein typisches Beispiel für blockorientierte Geräte. Platten untescheiden sich in Details, bieten aber grundsätzlich die gleichen Funktionen, nämlich das Lesen und Schreiben von Blöcken fester Größe (meist zwischen 512 und 4096 Bytes) an bestimmte Adressen.

Typischerweise stehen folgende Aufrufe an den Gerätetreiber zur Verfügung *(Hier 4.4BSD, aus [McKu96], andere Unix-Systeme mit leichten Änderungen)*:

open Öffnen des Gerätetreibers um auf eine bevorstehende E/A-Operation vorzubereiten. *open* wird aufgerufen, wenn der Systemaufruf *open* erfolgt oder intern, wenn ein auf dem Gerät befindliches Dateisystem in den Verzeichnisbaum eingehängt werden soll.
Die *open()*-Routine wird zunächst die Verfügbarkeit des entsprechenden Gerätes feststellen, so z.B. ob das Gerät während der Konfiguration anwesend war und – bei Disketten- oder Bandlaufwerken – ob ein Medium anwesend ist.

strategy Startet eine Lese- oder Schreiboperation und liefert sofort einen Rückgabewert. E/A-Anfragen an ein Dateisystem auf einem Gerät oder von dem Dateisystem werden vom System in die Aufrufe an *bread()* und *bwrite()* übersetzt. Diese Block-E/A-Routinen rufen wiederum die *strategy*-Routine auf, mit der Anfrage Daten zu lesen oder zu schreiben, die sich nicht im Cache befinden.
Jeder Aufruf an die *strategy*-Routine übergibt einen Pointer an eine *buf*-Struktur, die die Parameter für eine E/A-Anfrage enthält. Treten mehrere Anfragen gleichzeitig auf, muß der Aufrufende warten, bis die andere E/A-Anfrage beendet ist.

close Schließe ein Gerät. Die *close*-Routine wird üblicherweise dann aufgerufen, wenn die letzte E/A-Anfrage an das Gerät bedient ist. Plattenlaufwerke haben nichts zu tun, wenn das Gerät geschlossen wird, während Geräte, die nur eine gleichzeitige Anfrage erlauben, das Gerät wieder freigeben müsen.
Das Schließen eines (blockorientierten) Bandlaufwerkes schreibt üblicherweise eine Dateiendmarke und veranlasst das Laufwerk dann, das Band zurückzuspulen.
Das Schließen eine CD-Laufwerkes gibt z.B. den Auswurf wieder frei.

dump Schreibt den Inhalt des Hauptspeichers auf ein Laufwerk. Der Aufruf von *dump* sichert den Inhalt des flüchtigen Speichers auf einem nichtflüchtigen Medium. Das Betriebssystem veranlasst einen *dump* automatisch, wenn es einen unbehebbaren Fehler feststellt und vor einen Crash steht.
Der Inhalt des Dumps kann für eine Postmortem-Analyse des Fehlers benutzt werden, der den Absturz verursacht hat. Die *dump*-Routine wird mit der höchstmöglichen Priorität aufgerufen; folglich muß lediglich den Status des Gerätes überprüft und nicht auf einen Interrupt gewartet werden. Üblicherweise unterstützen die Treiber aller Plattencontroller und einiger Bandlaufwerke diesen Aufruf.

psize Liefert die Größe einer Festplattenpartition. Dem Treiber wird die logische Einheit übergeben, als Rückgabe wird die Größe dieser Partition erwartet (üblicherweise als Produkt der Anzahl der Blöcke einer Partition und der Blockgröße). Dieser Eintrag wird während der Bootstrap-Prozedur benutzt, um den Platz für einen Crash-Dump zu bestimmen und um die Größe eines Swap-Bereiches zu bestimmen.

2.3 Pseudodevices

Pseudodevices sind spezielle Geräte, die nicht als Hardware existieren. In der Regel dienen sie dazu entweder nicht (mehr) vorhandene Hardware (Terminals) nachzubilden oder spezielle Datenströme zu erzeugen oder "versickern" zu lassen.

Beispiele für Pseudodevices sind `/dev/pty0` (das erste Pseudoterminal), `/dev/null` (das "Datengrab", alles, was auf dieses Device geschrieben wird versickert im Nichts), `/dev/urandom` (erzeugt einen endlosen Strom von Zufallszahlen) oder `/dev/zero` (erzeugt einen endlosen Strom von Nullen).

Gemeinsam ist all diesen Geräten, dass sie sich genauso ansprechen lassen wie normale zeichenorientierte Geräte und im Bedarfsfall einfach gegen "echte" Geräte, z.B. einen echten Zufallszahlengenerator ausgetauscht werden können.

3 Einheitlicher Geräte- und Dateizugriff

Da die Operationen für den Zugriff auf Dateien und Geräte grundsätzlich die gleichen sind (öffnen, lesen, schreiben, suchen und schließen), liegt es nahe, Dateien wie Geräte zu behandeln und Geräte wie Dateien.

3.1 Geräte über Dateien ansprechen

3.1.1 Das Dateisystem

Eine Möglichkeit des Zugriffes – die gebräuchlichste – bietet das Dateisystem. Es benutzt die Schnittstelle des blockorientierten Gerätes und des zeichenori-

entierten Gerätes und bietet auf der anderen Seite eine Schnittstelle für Dateien an. Hier sieht der Benutzer nur die Dateien.

3.1.2 Gerätedateien

Unixsysteme bieten in der Regel einen benutzertransparenten Zugriff auf Geräte an. Das bedeutet, es existieren im Dateisystem Einträge für spezielle Gerätedateien, auf die geschrieben werden kann oder von denen gelesen werden kann wie von normalen Dateien. Dies hat den Vorteil, dass für Programme, die ihre Dateien wahlweise in Dateien oder z.B. auf Bandlaufwerke schreiben, keine Rücksicht auf den Speicherort nehmen müssen.

Implementiert werden diese Einträge über Sockets, d.h. beim versuchten Zugriff auf solch eine Gerätedatei wird der Datenstrom weg vom Dateisystem über ein Socket auf das entsprechende Gerät umgeleitet.

Dies ermöglicht es auch Programmen, die eigentlich zur Operation auf Dateien bestimmt sind, auf Geräten zu agieren und umgekehrt. So kann man z.B. mit `cat /dev/rfd0 > /irgendeine/datei` eine bytegetreue Kopie einer Diskette in eine Datei schreiben.

Auf der anderen Seite ist es z.B. dem Archivierungsprogramm `tar` egal, ob es in `/irgendeine/Datei` oder auf das erste Bandlaufwerk `/dev/rsa0` schreibt.

3.2 Dateien als Geräte ansprechen

Unix (und mit zusätzlichen Treibern auch Win32) bietet die Möglichkeiten Dateien als Geräte anzusprechen. Dies kann z.B. der Fall sein, wenn man auf die Image-Datei einer CD-Rom zugreifen möchte. Unix bietet hierfür ein sog. loopback-Device an, einen Treiber, der zwischen Imagedatei und dem eigentlich zuständigen Dateisystemtreiber liegt. Mit ihm kann dann auf eine Imagedatei zugegriffen werden, als wäre sie ein physikalisches Laufwerk.

3.3 Gerätedateien zu Ende gedacht

Unix und andere gängige Betriebssysteme haben das “Everything-Is-A-File”-Paradigma nicht konsequent zu Ende gedacht. So tauchen Gerätedateien zwar im Namensraum des lokalen Dateisystems auf, können aber nicht exportiert werden (z.B. per NFS). Außerdem muß man zum Zugriff auf einen Nameserver diesen meist über seinen entsprechenden Port und das dazugehörige Protokoll ansprechen. Man arbeitet beim Programmieren von Netzwerkdiensten also immer noch sehr rechnernah.

Diesen Ansatz haben einige Netzwerkbetriebssysteme konsequent weitergeführt. So ist es bei *Inferno* [Dorw97] möglich, Gerätedateien zu exportieren, was dazu führen kann, dass sich ein Betriebssystem im Netzwerk auflöst. Weiterhin ermöglicht Inferno den Zugriff auf Netzwerkdienste über Einträge im Namensraum des Dateisystems[1]. Analog zum `/dev`-Verzeichnis bei Unix existiert bei *Inferno* ein `/net`-Verzeichnis, in dem auf die Netzwerkdienste zugegriffen werden kann. So bezeichnet `/net/dns` den ersten Nameserver. Schreibt

[1] Diese Einträge können auch bei BSD-basierten Unix-Systemen vorhanden sein. Ein sog. Portal Filesystem stellt hierfür das Skelett zur Verfügung. Die Gründe dafür, dass dieses Konzept in Unix momentan kaum weiterverfolgt wird, dürften in der Dominanz von SysVR4 und dem seit 1996 stark geschwundenen Einfluss von BSD zu suchen sein.

man in diese Gerätedatei nun den Namen eines aufzulösenden Hosts, so kann man aus ihr die IP-Adresse lesen. Damit lassen sich auch Netzwerkfunktionen mit den simplen Lese- und Schreibaufrufen ansprechen.

4 Plattenpartitionierung und Zusammenfassung

Gelegentlich steht man vor dem Problem, dass man auf einer Festplatte mehrere Äste des Dateisystems, sowie einen Platz zum Swappen unterbringen muß. Oder man möchte verschiedene Betriebssysteme, deren Dateisysteme inkompatibel sind, auf einer Festplatte unterbringen. Hier muß eine Festplatte dann in mehrere logische Festplatten unterteilt werden.

Auf der anderen Seite kann es vorkommen, dass der Platz einer Festplatte für ein Verzeichnis oder sogar für eine Datei nicht ausreicht. Hier müssen mehrere Festplatten dann zu einer logischen Einheit zusammengefasst werden.

Das Aufteilen von Platten in logische Einheiten nennt man Partitionieren, das Zusammenfassen wird – je nach Verfahren – als Raid, Vinum oder Striping bezeichnet.

4.1 Partitionierung

Beim Aufteilen von Festplatten werden der Anfangs- und der Endblock, sowie der Typ einer Partition in eine sogenannte Partitionstabelle geschrieben. Beim Zugriff auf eine logische Festplatte wird auf die Nummer des Blockes, auf den zugegriffen werden soll, die Gesamtzahl der Blöcke der vorangegangenen Partitionen addiert.

4.1.1 PCs (Microsoft)

Microsoft teilt Festplatten in max. vier Partitionen auf, von denen drei erweiterte Partitionen sein können, die weitere logische Partitionen enthalten.

Bei diesem Typ Partitionierung wird unmittelbar hinter dem Masterbootsektor eine Partitionstabelle geschrieben, die die Größe, den Typ und die Lage einer Partition enthält.

4.1.2 Unix (4.4BSD, SunOS, SCO)

Unix benutzt i.d.R. sogenannte Slices, die den Partitionen bei MS entsprechen. Allerdings können in den Slices im Gegensatz zu den erweiterten Partitionen bei MS beliebig viele Partitionen untergebracht werden.

Eine Slice kann entweder in einer MS-Partition untergebracht werden oder eine gesamte Festplatte für sich beanspruchen (Dangerously Dedicated). In diesem Fall existiert keine Partitionstabelle mehr am Anfang der Platte, stattdessen wird in den ersten Blöcken der Slice eine eigene Partitionstabelle angelegt.

4.2 Plattenzusammenfassung

Prinzipiell kann die Zusammenfassung von Platten analog zur Partitionierung erfolgen: mehrere Platten werden zu einer logischen Einheit zusammengefasst. Diese erscheint nach außen hin wie eine große Festplatte.

RAID kann auf zwei Arten implementiert sein: Als Software- oder Hardware-RAID. Eine Softwarelösung bedeutet einen Treiber, der auf dem Treiber für IDE- oder SCSI-Festplatten aufsetzt, sich um Prüfsummen und Verteilung kümmert und nach oben hin dem Dateisystem ein blockorientiertes Gerät zur Verfügung stellt.

Hardware-RAID erfordert spezielle (IDE- oder SCSI-) Controller, die dann über einen geeigneten Hardwaretreiber angesprochen werden können. In diesem Fall bekommt kein Teil des Kernels mit, dass es sich um mehrere physikalische Festplatten handelt.

Hardware-RAID ist – besonders bei großen RAID-5- und RAID-6-Volumes – wesentlich performanter als Software-RAID, da die Prüfsummenberechnung nicht vom Prozessor des Systems, sondern vom Prozessor des RAID-Controllers vorgenommen wird.

4.2.1 Vinum, LVM

Die einfachste Methode der Zusammenfassung liefert Vinum. Bei Vinum werden die Blöcke einfach über die Platten hinaus durchnummeriert:

Disk 1		Disk 2		Disk 3		Disk 4	
0	1	8	9	14	15	18	19
2	3	10	11	16	17	20	21
4	5	12	13			22	23
6	7					24	25

4.2.2 RAID-0

RAID-0 ist eigentlich eine irreführende Bezeichnung, da keine redundanten Platten vorhanden sind. Striping ist die bessere Bezeichnung.

Beim Striping werden die Blöcke in mehreren Zyklen linear über die Platten durchnummeriert:

Disk 1		Disk 2		Disk 3		Disk 4	
0	13	4	17	7	20	9	22
1	14	5	18	8	21	10	23
2	15	6	19			11	24
3	16					12	25

Im Gegensatz zu Vinum ist der Rechenaufwand beim Schreiben auf die Platten größer und damit mehr Rechenleistung notwendig. Andererseits ist die Wahrscheinlichkeit, dass bei einer Suchanfrage eine Datei auf zwei oder mehr Platten verteilt ist, sehr hoch. Eine Datei, die z.B. die Blöcke 19-22 belegt, kann bei RAID-0 von drei Platten gleichzeitig gelesen werden, während bei Vinum nur auf eine Platte zugegriffen wird.

4.2.3 RAID-1

Bei RAID-1 werden die Platten lediglich gespiegelt. Man benötigt also genau doppelt soviele identische Platten, wie man Plattenplatz benötigt. RAID-1 bietet gegenüber einzelnen Platten kaum Geschwindigkeitsgewinn, jedoch ein Höchstmaß an Sicherheit.

4.2.4 RAID-2, RAID-3 und RAID-4

Bei diesen Verfahren wird eine zusätzliche Festplatte zur Speicherung der Checksummen benötigt. Diese Checksumme wird bei RAID-2 und -3 immer auf den entsprechenden Streifen der letzten Platte geschrieben. Dadurch erhält man zwar schnellere Lesezugriffe, aber langsamere Schreibzugriffe, da zunächst die Checksummen berechnet werden müssen.

Als Verfahren zur Prüfsummenberechnung kommt bei RAID-2 ein sog. Hamming-Algorithmus zum Einsatz, bei den anderen Verfahren ein simples XOR.

Beispiel RAID-3:

Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	XOR(0,1,2)
3	4	5	XOR(3,4,5)
6	7	8	XOR(6,7,8)
9	10	11	XOR(9,10,11)

4.2.5 RAID-5 und RAID-6

Bei RAID-5 werden die Checksummen nicht ausschließlich auf der letzten Festplatte gespeichert:

Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	XOR(0,1,2)
XOR(3,4,5)	3	4	5
6	XOR(6,7,8)	7	8
9	10	XOR(9,10,11)	11

Das hat gegenüber RAID-3 wiederum den Vorteil schnellerer Schreibzugriffe. Werden z.B. die beiden Blöcke 1 und 4 neu geschrieben, würden bei RAID-3 zwei Schreibzugriffe auf Platte 4 fällig werden, da die Checksummen ja ausschließlich hier vorhanden sind. Verteilt man hingegen die Checksummen auf mehrere Platten, so führt dies auch zu einer gleichmäßigeren Verteilung der Schreibzugriffe.

RAID-6 verwendet zur Speicherung der Checksummen zwei zusätzliche Platten. Hier wird die Sicherheit von RAID-5 noch etwas gesteigert. Da die Blöcke mit den Prüfsummen wie bei RAID-5 gleichmäßig über alle Platten verteilt werden, erhält man eine sehr gute I/O-Performance.

4.2.6 Vergleich zwischen Software- und Hardware-RAID

Prinzipiell lassen sich Disk-Arrays auf zwei Arten realisieren.

Beim Software-RAID setzt ein Kernelmodul oder Treiber auf dem Blockdevice-Treiber auf, übernimmt die Verteilung der Blöcke und die Prüfsummenberechnung und stellt dem Dateisystem die normale Schnittstelle einer Blockdevice zur Verfügung. Ein Software-RAID kann meistens mit recht geringem finanziellen Aufwand realisiert werden. Es besteht freie Wahl, ob IDE- oder SCSI-Festplatten verwendet werden, auch eine Kombination beider ist denkbar.

Die Zweite Möglichkeit ist das sogenannte Hardware-RAID. Hier steht ein Controller, meist eine Steckkarte ähnlich einem SCSI-Controller zur Verfügung, der einen (recht leistungsstarken) Prozessor, sowie Microcode und Speicher enthält. Beim Hardware-RAID ist lediglich ein Treiber für den RAID-Controller erforderlich. Im Gegensatz zu Software-RAID erfährt der Kernel zu keiner Zeit von dem Vorhandensein einzelner Festplatten. Die Prüfsummenberechnung und die Verteilung der Blöcke wird hier komplett vom Controller übernommen. Da zusätzliche Hardware benötigt wird, ist Hardware-RAID oft mit einem enormen Kostenaufwand verbunden. Allerdings steht dem die höhere Leistung von Hardware-RAID, sowie die Möglichkeit defekte Festplatten im Betrieb auszutauschen, entgegen.

5 RAM-Disks

Ein Gerät, dass sich wie eine Festplatte ansprechen lässt, muß nicht immer eine Festplatte sein. Genauso, wie eine Blockdevice in einer Datei untergebracht werden kann, kann sie auch im Arbeitsspeicher untergebracht werden. Man spricht dann von einer RAM-Disk. Der RAM-Disk-Treiber alloziert einen Teil des Hauptspeichers (bubble memory), teilt diesen in Sektoren ein und schafft damit ein Gerät, das sich wie eine Festplatte ansprechen lässt.

Eine RAM-Disk vergisst üblicherweise ihren Inhalt, wenn das System heruntergefahren wird. Der Inhalt kann gesichert werden, indem man den flüchtigen Speicher mit einer Batterie sichert, oder den Inhalt der RAM-Disk beim Herunterfahren auf eine Festplatte schreibt. Da beide Möglichkeiten den Durchsatz verringern oder die Hardwarekosten erhöhen, macht es wenig Sinn, RAM-Disks auf das Überleben von Stromausfällen auszulegen.

RAM-Disks werden aus diesen Gründen hauptsächlich für Daten verwendet, die entweder häufig verändert werden oder einfach wiederhergestellt

werden können (z.B. für das `/tmp`-Verzeichnis). Ihr größter Vorteil ist der erhöhte Durchsatz, verglichen mit physikalischen Festplatten. Dieser erhöhte Durchsatz macht besonders bei Compilern oder ähnlichen Anwendungen Sinn, wenn viel Gebrauch von temporären Dateien gemacht wird.

Die Verwendung von dezidiertem Speicher ausschließlich für eine RAM-Disk ist eine äußerst teure Angelegenheit. Das System kann den gesamten Durchsatz verbessern, indem der Speicher für häufig benötigte Seiten benutzt wird. Speicher der dem Dateisystem vorbehalten ist, wird besser als Pufferspeicher verwendet. Dies benötigt nur einen Kopiervorgang innerhalb des Speichers. Die Verwendung von Speicher in einer RAM-Disk benötigt dagegen zwei Kopiervorgänge. Einen von der RAM-Disk in den Puffer und einen in den Hauptspeicher.

4.4BSD umgeht dies, indem die RAM-Disk in dem auslagerbaren Teil des Speichers angelegt wird. In diesem Fall muß die RAM-Disk mit anderen Prozessen um Speicher konkurrieren. Falls der Speicher knapp wird, werden weniger häufig benötigte Seiten ausgelagert. Dies ermöglicht es auch, die Größe der RAM-Disk wesentlich größer zu wählen, als es der physikalische Speicherausbau eigentlich zuließe. So kann das `/tmp`-Verzeichnis im virtuellen Adressbereich liegen, der über den physikalischen Speicher hinausgeht. Dies erlaubt es, kleine Dateien schnell anzusprechen, während große Dateien auch gespeichert werden können, sich allerdings typischerweise mit den Geschwindigkeiten plattenbasierter Dateisysteme ansprechen lassen.

[McKu96] Kap 8.4, *The Memory-Based Filesystem*.

Literatur

[McKu96] Marshall Kirk McKusick, Keith Bostic, Michael J. Karels und John S. Quarterman. *The Design and Implementation of the 4.4BSD Operating System*. Addison-Wesley 1996.

[Tane92] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall 1992.

[Crow97] C. Crowley. *Operating Systems*. Richard D. Irwin 1997.

[Silb98] A. Silberschatz, P.B. Galvin: *Operating System Concepts*. Addison-Wesley 1998.

[Dorw97] Sean Dorward, Rob Pike, David L. Presotto, Dennis M. Ritchie, Howard Tricky, Phil Winterbottom. *The Inferno Operating System*. Bell Labs Technical Journal, Vol. 2, No. 1, Winter 1997. http://www.vitanuova.com/.

[Step99] Neal Stephenson. *In The Beginning Was The Command Line*. Hearst Corporation 1999. http://www-classic.be.com/users/cryptonomicon/.

Index