Von der Platte zur Anwendung (Platte, Treiber, Dateisystem)

1. Einleitung

Dateien sind abstrakte Objekte, die vom Betriebssystem zur Verfügung gestellt werden. Applikationen können bequem auf diese zugreifen und so auf den Daten arbeiten. Verschiedene Abstraktionsstufen verbergen die wahre Repräsentation der Daten auf einer Festplatte und die Funktionsweise der Zugriffe auf diese.
Das Betriebssystem spielt mit Hilfe einiger seiner Komponenten eine zentrale Rolle bei der Vermittlung der Abstraktion, aber auch Hardwarekomponenten sind beteiligt.

2. Dateisysteme

Das Dateisystem ist die Schnittstelle über die der Anwender mittels Applikationen auf Dateien zugreift. Betriebssysteme können mit verschiedensten Dateisystemen arbeiten, so kann z.B. Windows 2000 mit FAT32 (*File Allocation Table*) oder NTFS (*New Technology File System*) arbeiten.
Man kann dennoch die generelle Funktionsweise eines Dateisystems mit Schichten abstrahieren.

2.1. Logisches Dateisystem

In dieser höchsten Schicht wird definiert was eine Datei ist und wie sie in einer Verzeichnisstruktur eingeordnet ist. Es muss also eine solche Verzeichnisstruktur implementieren und Funktionen zum Finden, Lesen und Schreiben bereitstellen.
Ferner verwaltet es Attribute von Dateien, die beispielsweise für Sicherheit, Zugriffsrechte und erweiterte Suchfunktionen (nach Datum oder Größe) wichtig sind.

2.2. Dateiorganisationsmodul

Sowohl das Dateisystem alsauch die Festplatte teilt Dateien in Blöcke ein. Blöcke sind die kleinste Einheit die zwischen Speicher und Platte transferiert werden. Die Blockgröße ist durch die Festplatte vorgegeben und kann durch *low-level*-Formatierung geändert werden, üblich sind 512 Byte.
Die logischen Blöcke einer Datei sind von 0 (oder 1) an fortlaufend nummeriert, die physikalischen Blöcke auf der Platte aber fortlaufend vom Beginn der Platte an. Hier sorgt das Dateiorganisationsmodul für die Übersetzung der logischen auf physikalische Blöcke.
Wie diese Übersetzung erfolgt hängt davon von ab auf welche Weise das Dateisystem die Blöcke belegt, hierfür gibt es verschieden Ansätze:

- fortlaufend (*contiguous*):
  Die Daten werden in zusammenhängenden realen Blöcken gespeichert. Dies ermöglicht schnelle Zugriffe - auch auf beliebige Teile der Datei - ohne viele Sprünge des Lesekopfes auf der Platte.
  Allerdings wird es mit fortlaufenden Betrieb aufgrund der Framentierung immer schwieriger „Löcher“ zu finden, die groß genug für eine neue Datei sind. Viele kleine Lücken können womöglich nie wieder aufgefüllt werden, was zu Verschwendung von Speicherplatz führt. Außerdem ist es problematisch Dateien zu erstellen von denen zu Beginn die Größe noch nicht bekannt ist.
  Diese Art der Belegung wurde z.B. in IBMs VM/CMS Betriebssystem verwendet.

- verkettet (*linked*):
  Die Blöcke einer Datei werden als verkettete Liste angelegt, jeder Blöck enthält also die Nummer des Folgeblocks. Damit können zwar alle Blöcke ohne Lücke belegt werden, aber es entsteht ein gewisser *Overhead* für die Blockzeiger. Ein weiterer Nachteil ist, dass es für einen Zugriff auf die Mitte der Datei nötig ist alle Blöcke vorher zu durchlaufen.

In modifizierter Form wurde diese Belegungsmethode im FAT (*File Allocation Table*) Dateisystem angewendet. In der FAT werden alle Blocknummern gespeichert und ihnen ist entweder eine Nummer (des Folgeblocks), eine „0“ (freier Block) oder ein EOF-Wert (letzter Block einer Datei) zugeordnet. Die FAT kann im Speicher gehalten werden und es ist damit schnell möglich einen beliebigen Block einer Datei direkt zu finden.

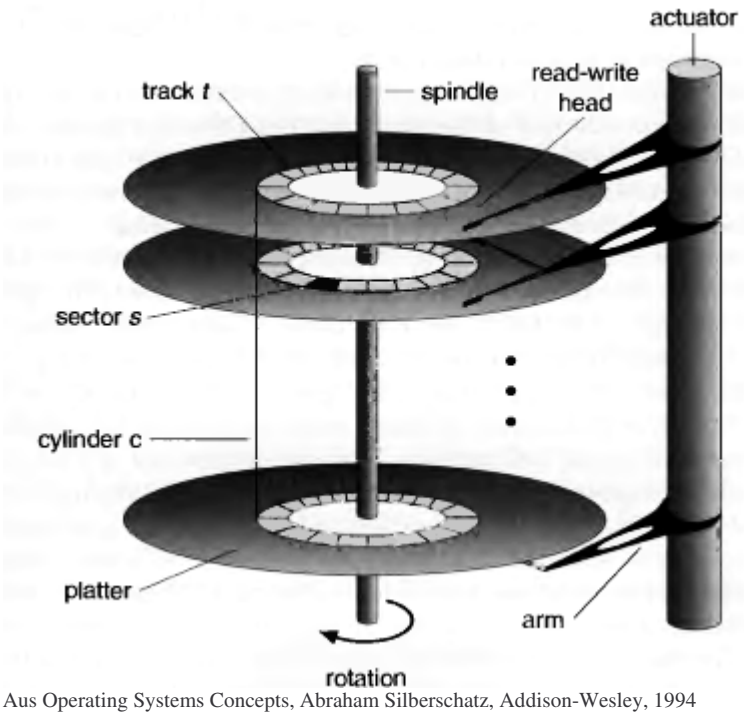
- indexiert (*indexed*):
  Der erste Block einer Datei ist Indexblock und enthält nur Zeiger auf die Datenblöcke. Somit ist schneller Zugriff und optimale Blockausnutzung gewährleistet, allerdings auf Kosten noch höheren *Overheads*. Jeder Datei, selbst wenn sie in einen einzelnen Block passen sollte, erhält ja einen eigenen Zeigerblock.
  Reale Implementationen wie z.B. im BSD UNIX Dateisystem legen sogar mehrere Indexblöcke an die grosse Dateien ermöglichen und legen zusätzlich noch weitere Informationen in die Indexblöcke (i-nodes).

2.3. Basis Dateisystem

Aufgabe des Basis Dateisystems ist es lediglich Lese- und Schreibbefehle für bestimmte Blöcke an den Plattentreiber weiterzugeben. Verschiedene Buffering- und Cachingstrategien werden hier implementiert um die Nutzung der Platten zu optimieren (z.B. verzögertes Schreiben mehrerer Blöcke auf einmal).

3. Festplattentreiber

Der Festplattentreiber ist Teil des *I/O-Subsystem* des Betriebssystems und ist ein blockorientierter Treiber. Jedem physikalischen Block entspricht eine Adresse, bestehend aus Zylinder-, Track- und Sektornummer, auf der Festplatte.



Aus Operating Systems Concepts, Abraham Silberschatz, Addison-Wesley, 1994

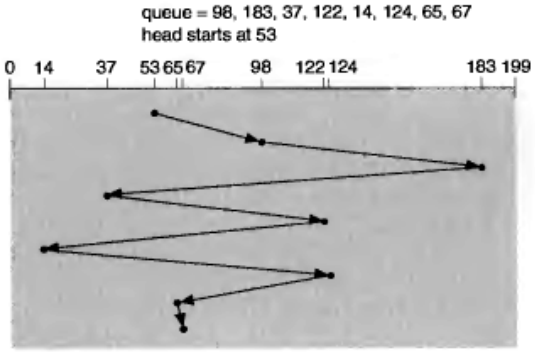
3.1. Funktionsweise

Der Treiber wird mit Lese- und Schreibanweisungen für einen physikalischen Block aufgerufen. Er überstetzt diese Blockadressen in reale Adressen auf der Festplatte. Weiterhin blockiert er zumeist den aufrufenden Prozess und schreibt dann hardwarenahe Kommandos in die Register des Festplattencontrollers.

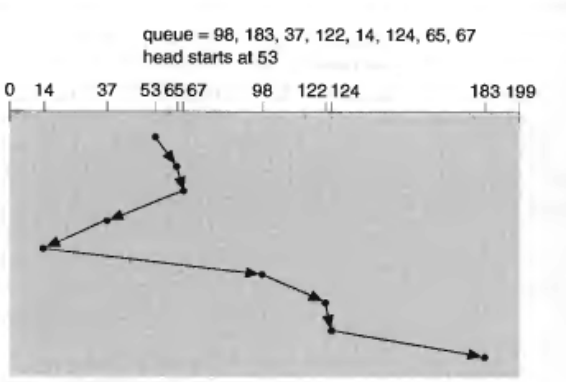
3.2. *Scheduling*-Verfahren

Werden Zugriffsanfragen für mehrere Blöcke gestellt, wird auf verschiedene *Scheduling*-Verfahren zurückgegriffen.

a) First come, first served (FCFS)



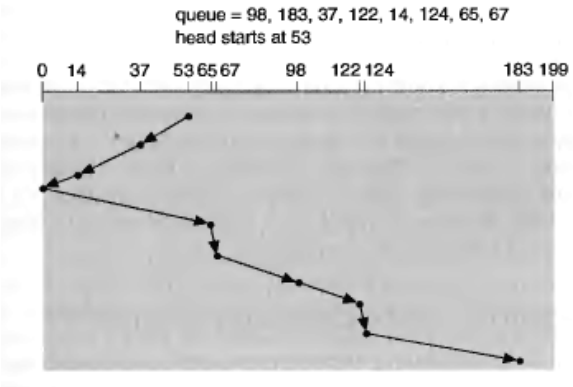
b) Shortest seek time first (SSTF)



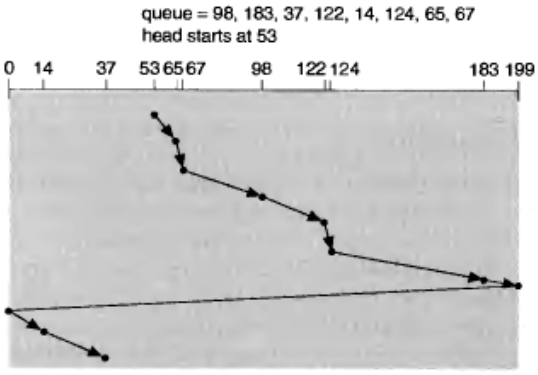
Der FCFS Algorithmus enthält keine Zugriffsoptimierung, die Anfragen werden in der Reihenfolge mit der sie gestellt werden abgearbeitet.

Der SSTF Algorithmus bearbeitet immer zunächst die Anfrage die am nächsten liegt. Dies macht Vorhersagen, wann ein Block geliefert wird allerdings schwierig. Anfragen die weit vom Lesekopf entfernt sind können „verhungern“, wenn immer neue Anfragen mit kürzeren Suchwegen kommen.

c) SCAN

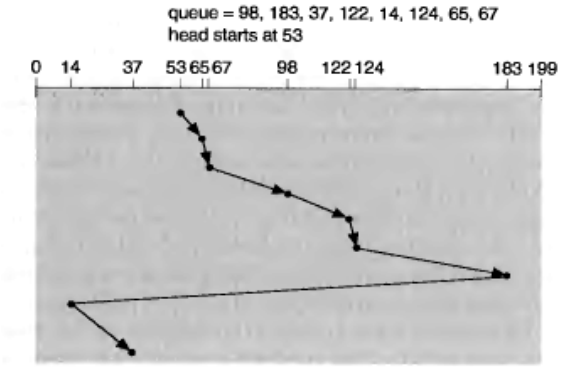


d) C-SCAN



Der SCAN Algorithmus bewegt den Lesekopf immer von Anfang bis Ende der Platte und arbeitet alle Anfragen, die auf dem Weg liegen, ab. C-SCAN tut dasselbe, ignoriert aber alle Anfragen auf dem Rückweg. Dies macht Zugriffe etwas „gerechter“, da Anfragen am Anfang meist schon länger warten wenn sich der Lesekopf am Ende befindet.

e) LOOK und C-LOOK



Alle Bilder aus Operating Systems Concepts, Abraham Silberschatz, Addison-Wesley, 1994

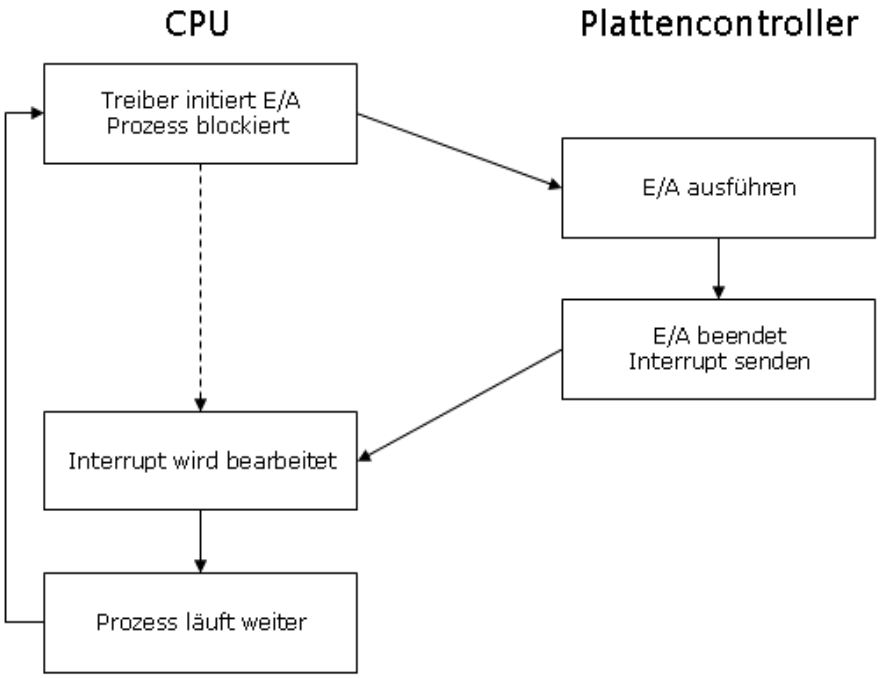
Funktioniert wie SCAN bzw. C-SCAN, bewegt den Lesekopf aber nur bis zur letzten Anfrage und auf dem Rückweg bis zur ersten Anfrage.

Da diese Algorithmen nur Suchzeiten für die Zylinder berücksichtigen, nicht aber Rotationsverzögerungen der Platte werden in den Plattencontrollern auch *Scheduling*-Verfahren angewand. Diese unterstützen das Betriebssystem um die *Performance* zu verbessern. Dennoch kann das Betriebssystem die Prioritäten setzen wenn es sinnvoll ist (z.B. das Priorisieren von Schreibzugriffen über Lesezugriffe wenn ein *Cache* zu voll wird).

4. Festplattencontroller

Der Controller ist nun die erste Hardwarekomponente auf dem Weg zu einer Datei. In seinem Register stehen eine Reihe, Kommandos die er in sinnvoller Reihenfolge ausführt.

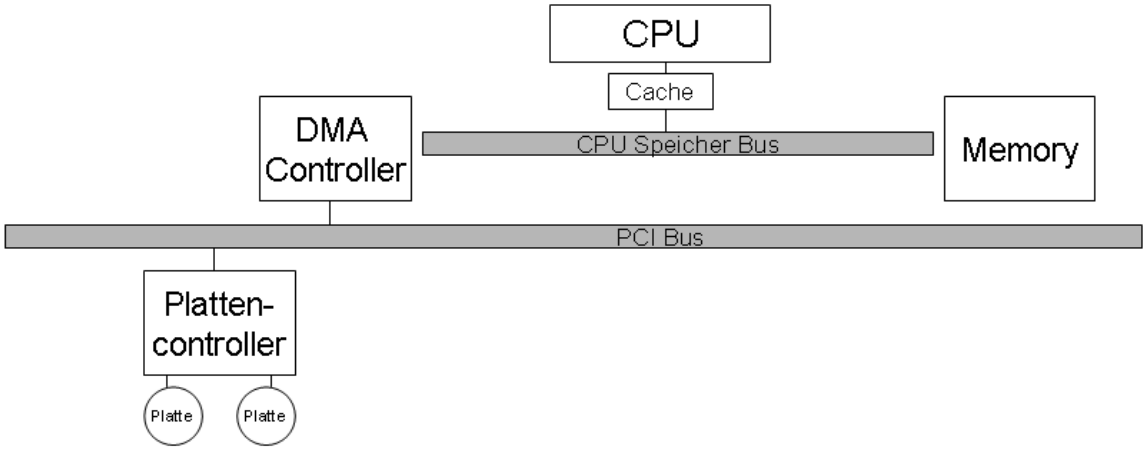
4.1. Standard E/A-Zugriff



Nach Operating Systems Concepts, Abraham Silberschatz, Addison-Wesley, 1994

Diese Art von Zugriff wird *programmed I/O* (PIO) genannt. Die CPU wird also häufig mit der eher banalen Aufgabe betraut Blöcke von der Platte in den Speicher zu kopieren oder umgekehrt und erhält viele *Interrupts*.
Daher wird normalerweise eine eigene Hardwarekomponente verwendet um der CPU diese Arbeit abzunehmen.

4.2. DMA-Zugriff



Zugriffe auf Dateien beinhalten meist viele Blockzugriffe. Der DMA (*Direct Memory Access*)-Controller übernimmt die Rolle der CPU bei der E/A-Operation auf der Platte.
DMA- und Plattencontoller kommunizieren über ein *Handshaking*-Verfahren und erst wenn die E/A-Operation ausgeführt wurde wird die CPU durch einen *Interrupt* vom DMA-Contoller dazu gebracht den aufrufenden Prozess zu entblocken.
Als einzigen Nachteil für DMA-Transfer kann man anführen, dass die Daten vom DMA-Controller über den CPU Speicherbus in den Hauptspeicher geschrieben werden und diese Speicherzyklen der CPU fehlen. Allerdings überwiegen die Vorteile und der DMA-Zugriff ist dem PIO-Zugriff überlegen.

5. Quellen

Abraham Silberschatz, Operating Systems Concepts, Addison-Wesley, 1994
Andrew S. Tanenbaum, Modern Operating Systems 2nd Edition, Prentice Hall, 2001