

Block E/A

Reinhard Bündgen
buendgen@de.ibm.com

Blockgeräte

- z.B. Platten, CDs/DVD Laufwerke, USB-Sticks, ...
- Kleinste Zugriffseinheit: HW: Sektor (SW: Block)
- Feste Anzahl von Sektoren
- Random Access Zugriff („DASD“)
 - Beliebige Reihenfolge
 - Beliebig oft
 - Lesen und Schreiben beliebig abwechseln
- I.A. langsam
 - Caching von Sektoren im Hauptspeicher: page cache

E/A Konzepte

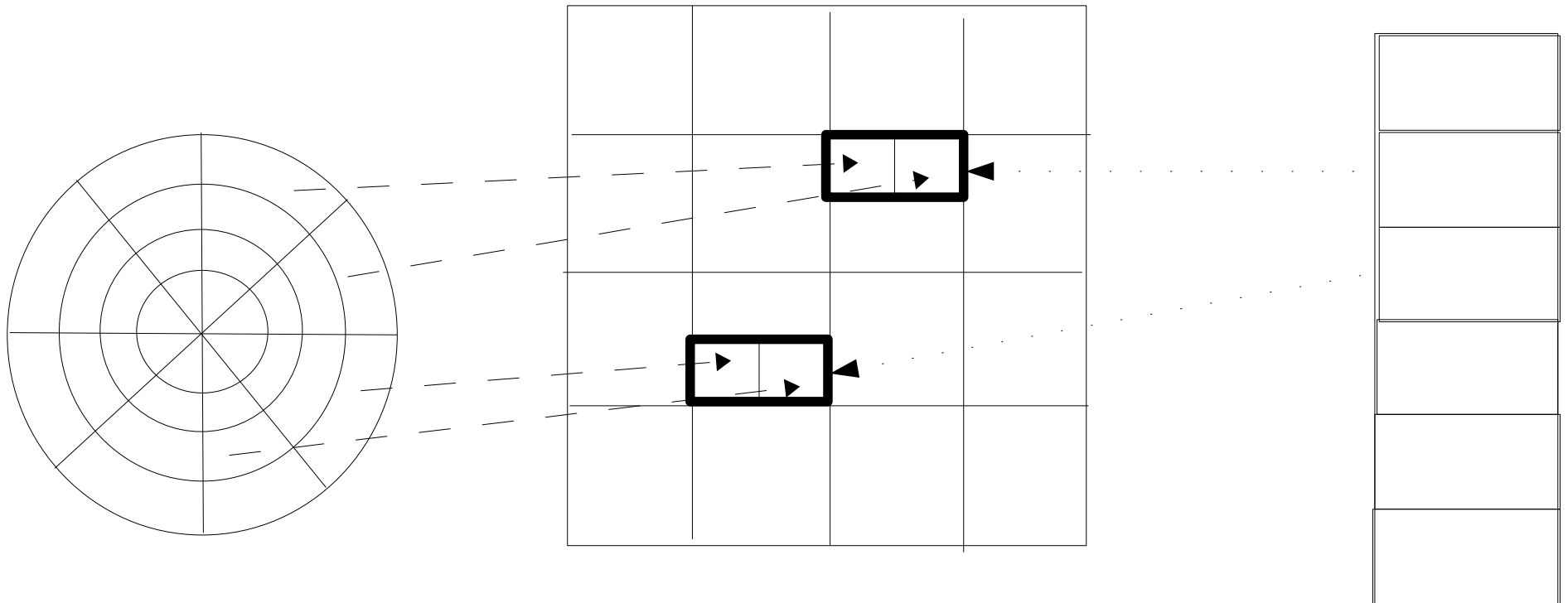
- Buffer
 - Kopie, von Daten die auf Blockgerät liegen (sollen): „Cache“
- Block E/A Operation
 - Beschreibung einer Datenbewegung zwischen Hauptspeicher (Puffern) und Gerät (Sektoren)
- E/A Anfrage (Request)
 - eine Menge von E/A Operationen deren zusammenhängender Ablauf geplant ist/wird

Block E/A: Begriffe

Platten: Sektoren,

Hauptspeicher: Seiten,

Dateien: Blöcke



$$|\text{Block}| = n * |\text{Sektor}| = 2^k \wedge |\text{Block}| \leq |\text{Seite}|$$

Puffer (buffers)

- Jeder Sektor wird durch einen buffer repräsentiert
- Buffer head
 - Daten die Buffer beschreiben
 - Typ: `struct buffer_head` in `<linux/buffer_head.h>`
 - Assoziiert Gerät/Sektor mit Seite/Position/Puffergröße
 - Zustand (`b_state` Komponente):
 - `BH_Uptodate` – enthält gültige Daten
 - `BH_Dirty` – Block aktueller als Gerät
 - `BH_Lock` – während E/A
 - `BH_Req` – gehört zu E/A Request
 - `BH_Mapped` – entspricht Daten auf Gerät
 - `BH_New` – neu, noch kein Zugriff
 - `BH_Async_Read` – an asynch read beteiligt
 - `BH_Async_Write` – an asynch write beteiligt
 - `BH_Delay` – hat noch keine entsprechende Sektoren
 - ...

Block E/A Operationen

- Lesen oder Schreiben zusammenhängender Sektoren
- wird durch `bio` Struktur repräsentiert
 - `<linux/bio.h>: struct bio`
 - Position auf Gerät
 - Operation: Lesen oder Schreiben
 - Liste von Segmenten
 - Segment (`struct bio_vec`)
 - Kontinuierlicher Datenbereich (in einer Seite)
 - Zeiger in aktuelles Segment
- Scatter/Gather Liste

Requests

- `<linux/blkdev.h>`
- Request
 - Beschreibt Zugriff auf zusammenhängende Menge von Sektoren
 - `struct request`
 - Enthält ein oder mehrere bio Strukturen
- Request Queue
 - Beschreibt eine Liste von Requests für ein Gerät
 - `struct request_queue`

E/A Planer (I/O Scheduler)

- E/A ist langsam
 - Insbesondere seek-Operationen
 - Positionierung des Plattenkopfs
 - Dauert mehrere Millisekunden
- Aufgabe:
 - Minimierung von seek-Operationen
- Mittel
 - merging & sorting
- Resultat
 - E/A Planer virtualisiert Platte für mehrere E/A Requests
 - (Leicht) unfair
 - Größerer globaler(!) Durchsatz

Aufzugs Problem

- Betrachte Aufzug im Empire State Building
- Ziel
 - fahre eine möglichst geringe Strecke
- Mittel
 - Sortiere die Anfragen
 - Mache in einer Richtung möglichst viele Stops
 - Merge
 - Nehme möglichst viele Personen pro Station mit
- „Lösung“
 - Fahre immer zur nächst gelegenen anfragenden Station
- Problem
 - Stockwerke können „verhungern“

Linus Elevator

- Veraltet, aber prototypisch
 - War Standard I/O Scheduler in Linux 2.4
- Sorting:
 - Request queue nach Sektornummer sortiert
- Merging
 - Direkt aneinander anschliessende Requests werden zu einem request vereinigt
 - Front merging & back merging
- Fairness:
 - Falls es zu alte Requests gibt werden neue (nicht merge-bare) requests ans Ende der request queue gehängt

Lese vs Schreibzugriffe

- Lesezugriffe
 - Applikation muss auf Abschluss der Leseoperation warten
 - Oft hängt eine Leseoperation von einer anderen ab
 - Latenz von Leseoperationen beeinflusst Applikations Performanz
- Schreibzugriffe
 - Können asynchron von Applikation abgearbeitet werden
 - Ausnahme sync Funktionen
 - Schreiblatenz beeinflusst Applikationsperformanz nur wenig
 - Regelmäßiges Abspeichern verringert das Risiko von Datenverlusten

Deadline I/O Scheduler (1)

- `drivers/block/deadline-iosched.c`
- 3 queues:
 - Sorted queue
 - Read FIFO
 - Write FIFO
- Jeder neue Request wird
 - Bzgl der Platten Position ge-merge-t & sortiert in die sorted queue eingefügt und
 - Je nach Zugriffstyp an die Read- oder Write FIFO gehängt
- Jeder Request wird mit einer Expirationszeit markiert
 - Read request default: 500 ms
 - Write request default 5 sec

Deadline I/O Scheduler (2)

- Abarbeiten des nächsten Requests:
 - Bearbeite den nächsten abgelaufenen Request aus den FIFO Queues
 - Wenn keine abgelaufenen Requests vorhanden, bearbeite den nächsten request aus der Sorted Queue
- Front Merging optional
- Achtung
 - Der Deadline Scheduler gibt keine Garantien darüber, dass ein Request vor Expirationszeit durchgeführt wird

Anticipatory I/O Scheduler

- `drivers/block/as-iosched.c`
- DL I/O Scheduler + antizipatorische Heuristik:
 - Nachdem ein Request bearbeitet wird
 - Wird *nicht* gleich der Nächste gesucht,
 - *Sondern* es wird einige ms gewartet, ob nicht ein Request für einen Nachbarsektor abgesetzt wird.
 - Versucht via Statistiken Applikationsverhalten zu erraten.

Weitere I/O Scheduler

- Complete-Fair-Queuing-I/O Scheduler (CFQ)
 - `drivers/block/cfq-iosched.c`
 - eine queue pro Task
 - Round-Robin: bearbeitet konfigurierbare Anzahl von Requests pro Queue
 - Anwendung: Multimedia – Fairness unter Anwendungen
- Noop-I/O Scheduler
 - `drivers/block/noop-iosched.c`
 - keine Requestsortierung, nur Merging
 - Anwendung: Geräte ohne Suchaufwand (z.B. Flash-Memory Karten)