Lernzettel

Ressourcenverwaltung und Betriebsmittelverwaltung: Dateisysteme, Blockgeräte, I/O-Planung, Pufferung

Universität:	Technische Universität Berlin
Kurs/Modul:	Systemprogrammierung
Erstellungsdatum:	September 6, 2025

Lernzettel: Ressourcenverwaltung und Betriebsmittelverwaltung (Dateisysteme, Blockgeräte, I/O-Planung, Pufferung)

(1) Überblick. Die Ressourcenverwaltung umfasst Mechanismen zur Organisation von Dateisystemen, Blockgeräten, der I/O-Planung und der Pufferung. Ziel ist eine effiziente, konsistente und sichere Nutzung der Hardware-Ressourcen durch das Betriebssystem.

(2) Dateisysteme.

- Zweck: Bereitstellung einer logischen Struktur zur Ablage von Dateien und Verzeichnissen.
- Blockbasierte Speicherung: Dateien werden in Blöcken gespeichert; typischerweise 4 KiB pro Block.
- Metadaten: Superblock, Inodes, Verzeichnisse, Journaling (Integrität der Metadaten).
- Beispiele und Konzepte: ext4, XFS, btrfs; Mountpoints; Datei- und Verzeichnishierarchie; Berechtigungen.

(3) Blockgeräte.

- Physische vs. logische Adressierung: Blöcke und Sektoren als Grundbausteine.
- Namensschema: `/dev/sdX`, `/dev/nvme0n1`, etc.
- Blockgröße: typischerweise 512 B bis 4 KiB pro Block; beeinflusst Platznutzung und Leistung.
- I/O-Requests und -Queues: Transformation von Nutzanfragen in Kernel-Strukturen; Service-Policies.
- Treiber-Stack: Blocklayer, Treiber-Schicht, Geräte-Layer.

(4) I/O-Planung.

- Ziel: faire und effiziente Ausnutzung der Speichergeräte durch Scheduling der I/O-Anfragen.
- Typische Strategien: FCFS/FIFO, Round-Robin, Elevator-Algorithmen (SCAN/LOOK), Shortest-Seek-Time-First (SSTF).
- Berücksichtigung von Latenz, Throughput, Fairness, Priorisierung wichtiger Anfragen.
- Einflussfaktoren: Geräteauslastung, Caching-Strategien, Batch-Verarbeitung.

(5) Pufferung.

- Puffer vs Cache: Puffer dienen als Zwischenablage für Metadaten/Requests; Cache speichert häufig genutzte Daten.
- Page Cache und Buffer Cache im Kernel-Kontext; dirty pages und Schreibstrategie.
- Schreibstrategien: Write-Back vs Write-Through; fsync/Barrieren zur Persistenzsicherung.

- Konsistenz und Ordering: Sicherstellung, dass Änderungen in der richtigen Reihenfolge auf das Speichersystem gelangen.

(6) Treiber und Geräteunabhängigkeit.

- I/O-Stack: Anwendung -> C-Layer (libc) -> Kernel I/O-Stack -> Blocktreiber -> Geräte-Layer.
- Geräteunabhängigkeit entsteht durch standardisierte Schnittstellen (Dateisystem-API, VFS) vs. gerätespezifische Treiber.
- Treiberarten: Blocktreiber (Blockgeräte), Charaktertreiber; zentrale Konzepte wie request_queue, bio, gendisk (hoch-niveau).

(7) Praktische Programmierung (C). Beispielfall: einfacher Lesezugriff auf eine Datei. `int`

```
fd = open("example.dat", O_RDONLY);
if (fd < 0) return 1;
char buf[1024]; ssize_t n = read(fd, buf, sizeof buf);
close(fd);
```