Inhaltsverzeichnis

1 Betriebssystem

1.1 Definition

- **Systemsicht**
  Alle Programme zur **Steuerung und Überwachung** von:
  - Ausführung v. Benutzerprogrammen
  - Verteilung der Betriebsmittel
  - Aufrechterhaltung der Betriebsart
- **Anwendersicht**
  Virtuelle Maschine, vereinfachte Ansicht des Computers

1.2 Aufgaben

- **Hardwareabstraktion**
  - einheitliche Sicht auf Geräteklassen
  - Bibliotheken und Treiber
- **Resourcenverwaltung**
  - CPU-Rechenzeit
  - Speicher
  - Gerätezugriffe
- **Sicherheitsfeatures**
  - Benutzer und Gruppen **Multi-User**
  - Parallelbetrieb **Multitasking**
  - Schutz for direkten Hardwarezugriffen

1.3 Arten

- **Mainframe** schnelles I/O, viele Prozesse, Transaktionen
- **Server** viele Anwender, Netzanbindung
- **Multiprozessor**
- **Echtzeit**

2 Prozesse

2.1 Bestandteile

- eigener Adressraum
- Programmcode
- Programmdaten
- Programm-Counter
- Stacks und Stackpointer
- Hardwareregister-Inhalte *(Prozess-Kontext)*
- Heap-Speicher
- Verwaltungsdaten
  - Identifier und VaterID
  - Resourcenliste
  - Scheduling Parameter

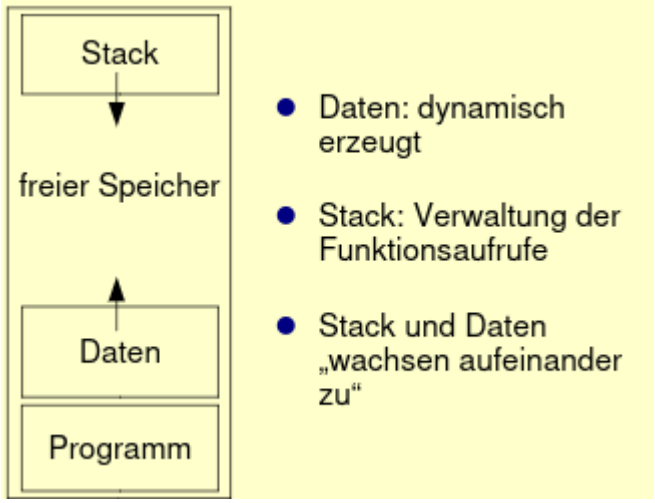

Abbildung 1: Process Control Block PCB

2.2 Hierarchie und Signale

Jeder Prozess hat **Vaterprozess** *(Prozesse erzeugen einander).*

2.2.1 Fork

```
1    int pid = fork();
2    if(pid == 0){
3        printf("Ich bin das Kind mit pid=%d\n",
             getpid());
4    }else if(pid > 0){
5        printf("Ich bin der Vater, mein Kind hat die
             pid=%d\n", pid);
6    }else{
7        printf("Error: fork() war nicht erfolgreich");
8    }
```

2.2.2 Signale

- (17) STOP *(Strg-Z oder bg)*
- (19) CONT *(fg)*
- (15) SIGTERM *(beenden)*
- (9) KILL *(abschließen)*

2.3 Prozesserzeugung

1. fork → clone → do_fork → copy_process
2. neue thread_info in task_struct
3. Kind-Status auf TASK_UNINTERRUPTABLE
4. copy_flags
5. get_pid: neue PID für Kind
6. je nach clone-Parametern: kopieren/gemeinsam nutzen
7. Scheduler

3 Threads

- Aktivitätsstrang in einem Prozess
- gemeinsamer Zugriff auf Daten

3.1 Vergleich: Prozesse/Threads

3.1.1 Gemeinsam mit Prozess:

- Adressraum
- Programmcode
- aktuelle Daten (Variablen/Konstanten)

3.1.2 Separat pro Thread

- PC
- SP
- Stack
- Register

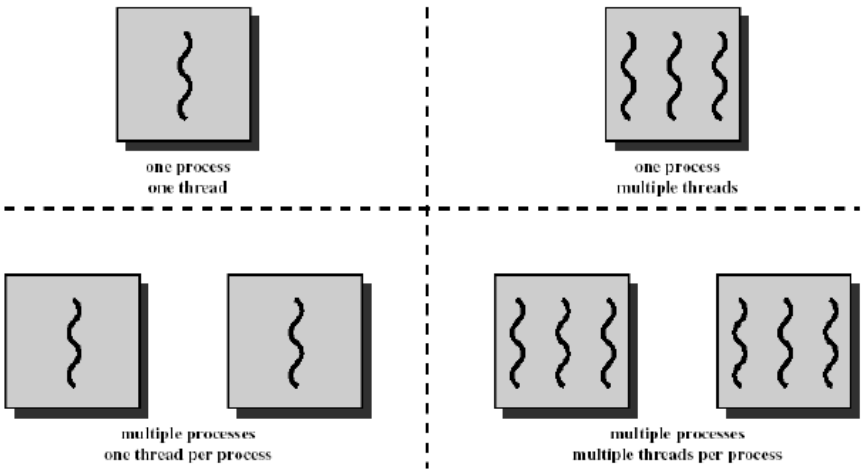

Abbildung 2: Unterschied zw. Prozessen und Threads

3.2 User - und Kernel-Level Threads

3.2.1 User-Level Threads

- Keine Systemcalls nötig
- Blockiert bei I/O
- keine Nutzung mehrerer CPUs
- Bessere Abstraktion möglich

3.2.2 Kernel-Level Threads

- BS verwalted Threads
- Zeitsteuerung nur mit Systemcalls

3.2.3 Kombinierte Threadtypen

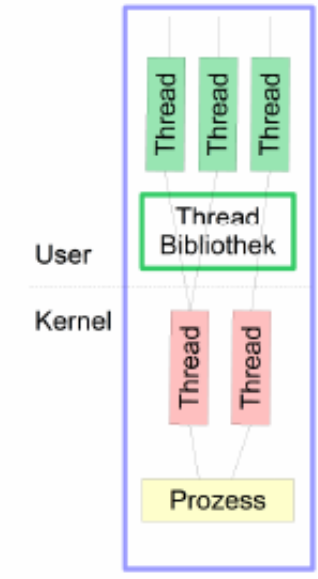

Abbildung 3: Komtiniert: ULT, KLT

3.3 Linux Threads und Prozesse

Prozesse und **Threads** werden in Linux einheitlich gehandhabt:

```
1       // Prozess
2       clone(SIGCHLD, 0);
3       // Thread
4       clone(CLONE_VM | CLONE_FS | CLONE_FILES |
            CLONE_SIGHAND, 0);
```

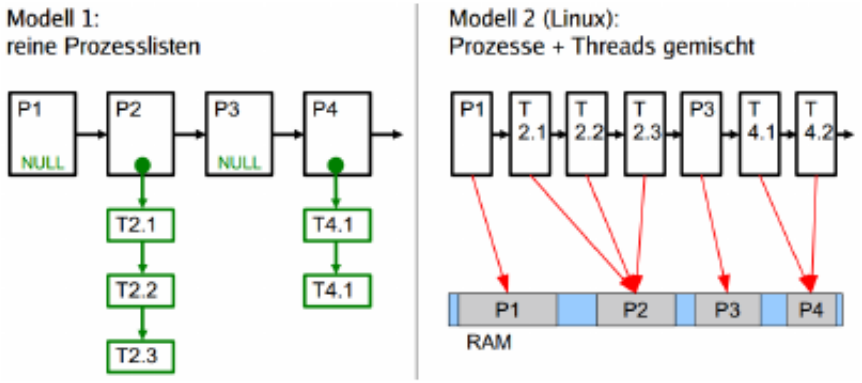

Abbildung 4: Linux Prozess- und Threadverwaltung

4 Interrupts

4.1 Interrupt-Klassen

- Hardware-Fehler
- Timer
- I/O
- Software-Interrupts
  - Arithmetik
  - Traps
  - etc.

4.2 Ablauf

1. Interrupt flag wird gesetzt
2. Nach aktuellem Befehl wird unterbrochen (BS übernimmt Kontrolle)
3. Prozess-Daten werden gespeichert (wie bei Kontextswitch)
4. Mittels Interrupt-Vector wird die entsprechende ISR aufgerufen
5. Diese ist nicht unterbrechbar und so kurz wie möglich
6. Die ISR ruft dann ein sog. Tasklet auf welches unterbrechbar ist und die eigentliche Arbeit macht

4.3 Round Robin: I/O- vs CPU-lastig

CPU-lastinge Prozesse nutzen ihre **Zeitquanten** vollständig, während **I/O** Prozesse **warten** müssen.

4.4 Interrupt Handling

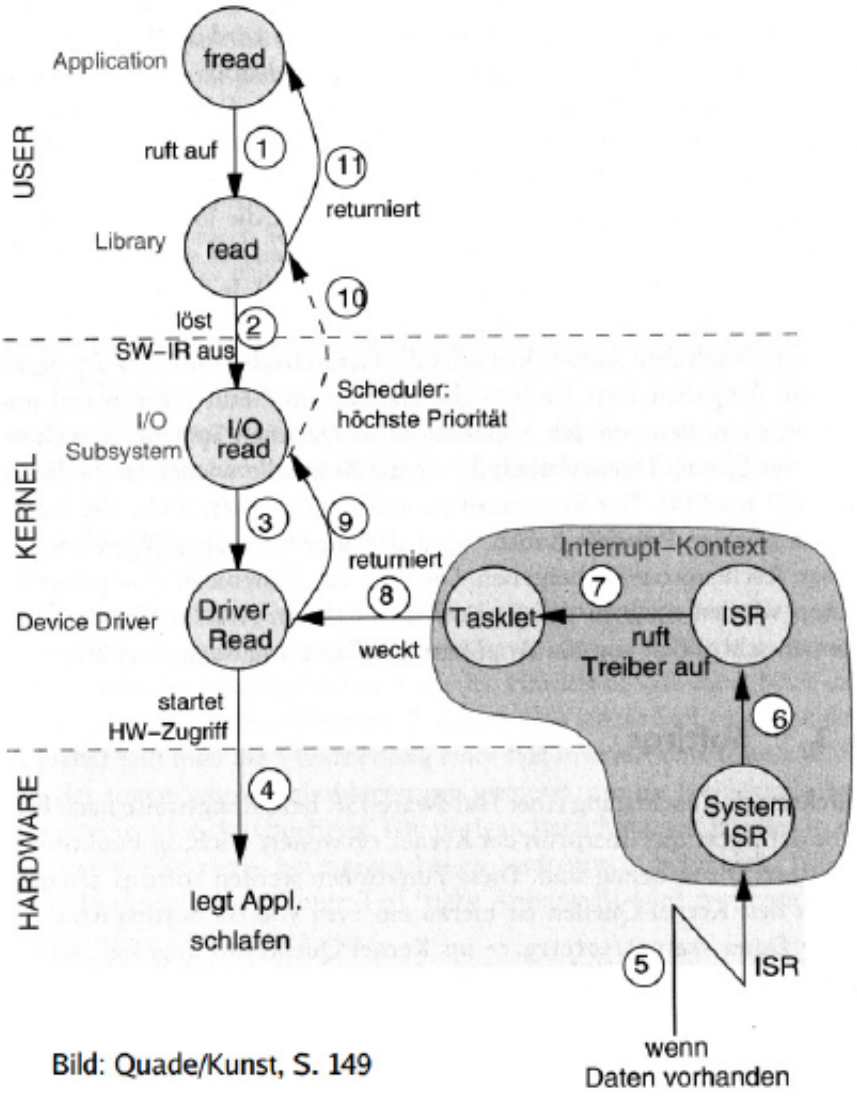

Abbildung 5: Interrupt callgraph

5 Scheduling

Scheduling: Zuteilug der CPU (Betriebsmittel) an Threads/Prozesse

5.1 Wann wird der Scheduler aktiv

- neuer Prozess entsteht
- aktiver Prozess endet
- Prozess wg. I/O blockiert
- Zeitquantum is aufgebraucht
- Interrupt tritt auf

5.2 Scheduling-Prinzipien

- Kooperativ
- Präemptiv
- Batch
  - FCFS
    - nicht Präemptiv
    - minimiert durchschnittliche Verweilzeit
    - I/O ineffizient
  - SJF und SRT
    - minimiert Wartezeit
    - starvation
  - Prioritäten

5.3 Kriterien

5.3.1 Anwendersicht

- Ausführungsdauer (Prozess-Gesamtlaufzeit)
- Reaktionszeit (Reaktionen auf Benutzerinteraktionen)
- Deadlines
- Vorhersagbarkeit (gleichartige Prozesse)
- Proportionalität

5.3.2 Systemsicht

- Durchsatz (Prozesse pro Zeit)
- Prozessauslastung (Cycles pro Zeit)
- Fairness (keine starvation)
- Prioritäten
- Resourcen Fairness

5.3.3 Wahl des Quantums

- zu groß: Latency
- zu klein: Overhead durch Kontextwechsel
- typisch: 10-100ms

5.4 Round Robin und I/O

5.4.1 Virtual Round Robin

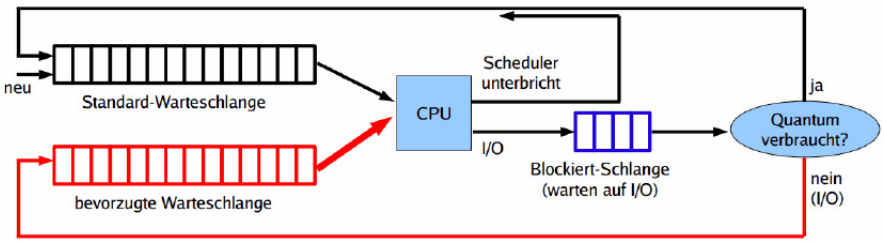

Abbildung 6: Virtual round robin prinzip

5.4.2 Prioritätsbasiert

- Dynamisch (+ variable Quantenlänge): z.B. Aging (SJF)
- Multilevel Scheduling

Priority-Inversion:

5.5 Formeln

5.5.1 Burstdauer

- $S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i = \frac{1}{n}T_n + \frac{n-1}{n}S_n$
- $S_{n+1} = \alpha T_n + (1-\alpha)S_n; \alpha \in [0,1]$

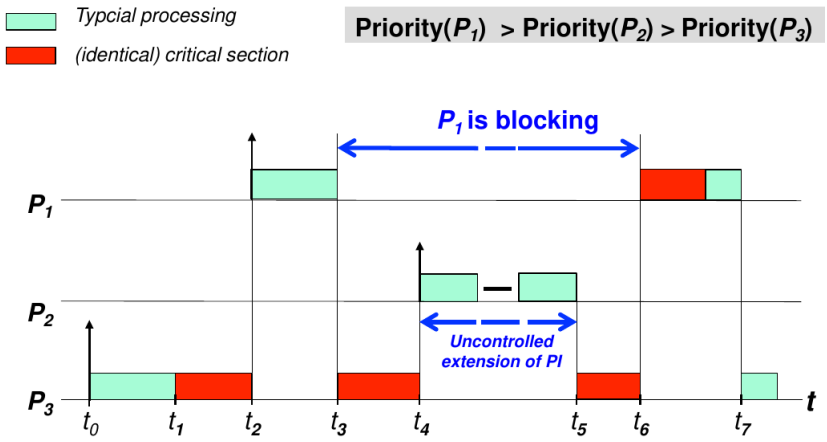

Abbildung 7: Beispiel für Priority-Inversion

5.6 Linux Prioritäten: nice

- nice, renice
- NeuePrio = Basis-Prio + CPU-Nutzung/2 + nice-value

6 Synchronisation

6.1 Race Condition

mehrere parallele Threads/Prozesse nutzen **gemeinsame Resource**.
Zustand hängt von Ausführung ab:
⇒ **nicht vorhersagbar, nicht reproduzierbar**

6.2 Synchronisationsmechanismen

- Mutex
- Semaphor
  - negativ Werte: immer um 1 erhöhen und erniedrigen
- Event (wie Condition Variable)
  - automatisch: ein Thread kehrt zurück -¿ reset
  - manuell: alle Threads kehren zurück
- Monitor (Klasse bei der Jede public Methode Synchronisiert ist)
- Locking
  - Concurrent Read
  - Concurrent Write
  - Protected Read
  - Protected Write
  - Exclusive

6.3 Spin-Locks

- Prozess/Thread geht nicht in blockiert Zustand:
  ⇒ in Interrupt Handlern Verwendbar
- nicht rekursiv
- nur bei kurzen wartezeiten

6.4 Anforderungen

- kein Deadlock (blockiert außerhalb v. kritischem Bereich)
- Starvation free (Scheduling bei mehreren Wartenden)

6.5 Deadlocks

Ein Zustand in dem Jeder Teilnehmer auf einen Anderen wartet (Alle warten gegenseitig aufeinander).

6.5.1 Vorraussetzungen

- Mutual Exclusion (mindestens eine Resource nur exklusiv verfügbar)
- Hold and Wait (hat resource, wartet auf andere resource)
- No preemption (resource kann nur von Prozess abgegeben werden)
- Circular Wait (Geschlossener kreis beim Warten auf Resourcen)

6.6 Bakner's Algorithm

- geg: Allocation (Prozess ¡-¿ Resource), Max, Available
- Algorithmus:
  1. Need = Max - Allocation
  2. While unscheduled Processes exist
     (a) Select unscheduled Process
     (b) If Need ¡= Available
         – Available + Allocation = Available

6.7 Praxis (Linux)

- Atomare:
  – Integer-Variablen
  – Bit-Operationen
- Spin-Locks
- Reader-Writer-Locks
- Semaphore/R-W-Semaphore
- Mutexe

6.8 Linux-Kernel: Semaphore

```
1    sema_init(&sem, count);
2    init_MUTEX(&sem);
3    down(&sem)
4        /* kritischer Bereich */
5    up(&sem);
```

7 Interprozesskommunikation IPC

7.1 Charakteristika

- Kommunikationsmodell
  - P2P
  - publish-subscribe
  - broadcast
- Übertragungsrichtung
  - simblex
  - duplex
- Synchronität
  - synchron/blockierend
  - asynchron/nicht-blockierend (nachrichtenbasiert)

7.2 Basismechanismen (Linux)

- Signale
- Synchronisation (prozessübergreifend)
  - shared mutex
  - shared semaphore
- Pipes (unix)
  - FIFO Bytestream
  - unidirektional
  - Synchron
- Sockets
  - Verbindungslos UDP
  - Verbindungsorientiert TCP
- Shared-Memory

7.3 Middleware-Lösungen

- Synchron
  - RPC
  - RMI (Java)
  - CORBA
  - DBus Messaging (RPC)
- Asynchron
  - SmartSockets
  - MqSeries Messaging
  - JMS

8 Speicherverwaltung

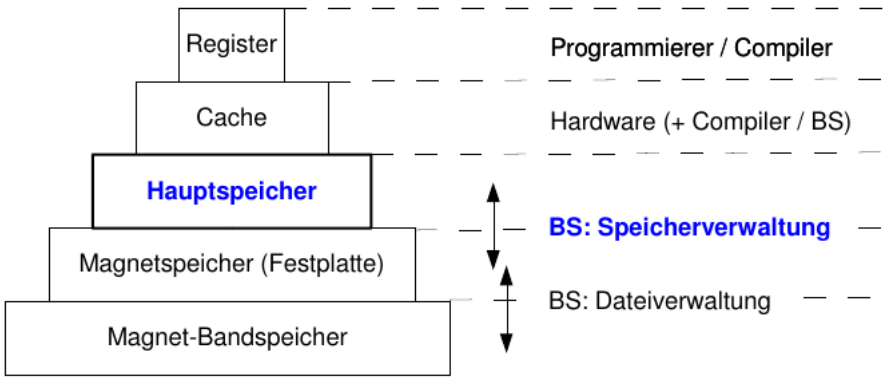

Abbildung 8: Speicherhierarchie

8.1 Aufgaben

- Finden und Zuteilung freier Speicherbereiche
- Effiziente Nutzung des Speichers
- Speicherschutz

8.2 Code-Verschiebung (Relokation)

1. Linker vermerkt absolute Adressen, werden beim Laden abgeändert
2. oder: bei jeder Adressberechnung wird ein Basisregister hinzuaddiert

8.3 Speicherschutz

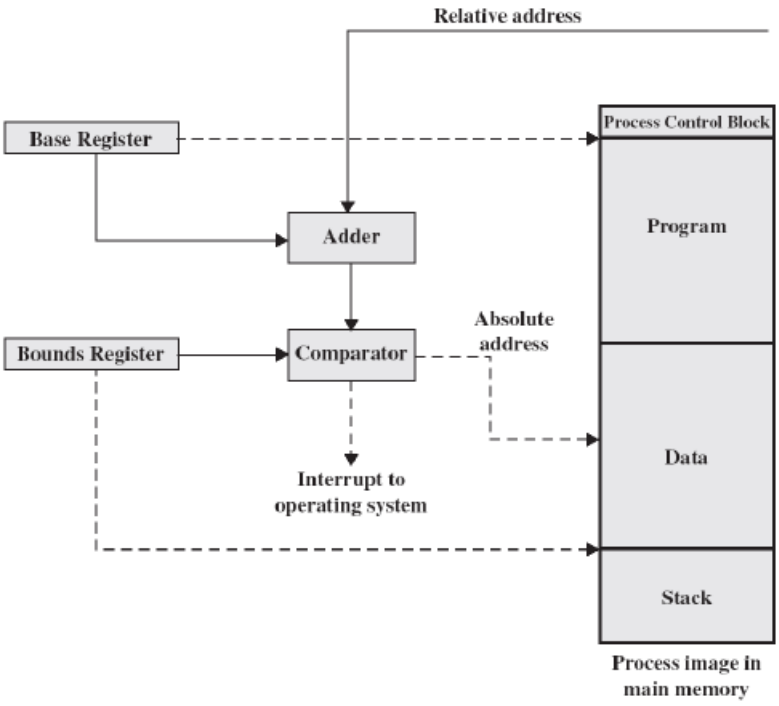

Abbildung 9: Speicherschutz

8.4 Zusammenhängende Speicheraufteilung

Aufteilung des Speichers in Partitionen fester größe

- Fragmentierung (kleine Bereiche im Speicher sind ungenutzt)
- Relokation (Speicherbereiche werden verschoben)
- Swapping (Speicherbereiche werden auf die Festplatte verschoben)

8.4.1 Suchverfahren für freien Speicher

- First Fit
- Worst Fit
- Best Fit
- Quick Fit (mehrere Listen mit verschiedenen Bereichsgrößen) (Buddy System)

8.5 Nicht Zusammenhängende Speicheraufteilung

Memory Management Unit (MMU) bildet jede logische Adresse auf eine Physische ab.

8.5.1 Paging

Aufteilung der Adressräume in Blöcke fester Größe.

- Page: Block im virtuellen Adressraum
- page frame: Block im physischen Adressraum

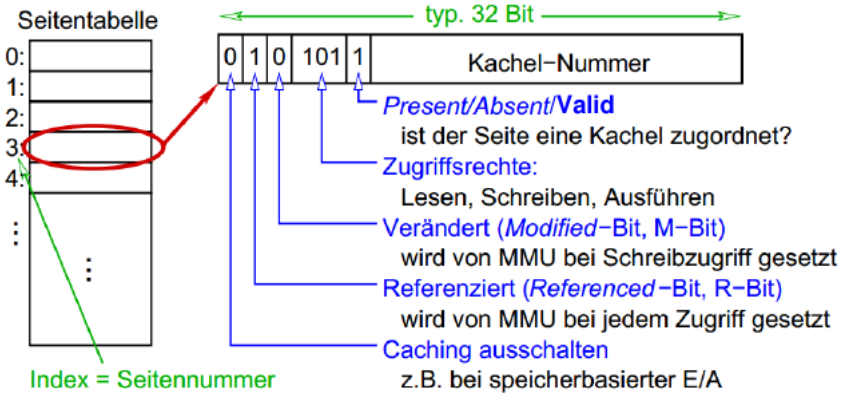

Abbildung 10: Seitentabelle

8.5.2 Translation Look-Aside Buffer

- Durch das Lokalitätsprinzip, kann der TLB hohe Trefferquoten erzielen.
- Bei Prozesswechesl
  - valid bit für alles gelöscht
  - jeder Eintrag hat PID

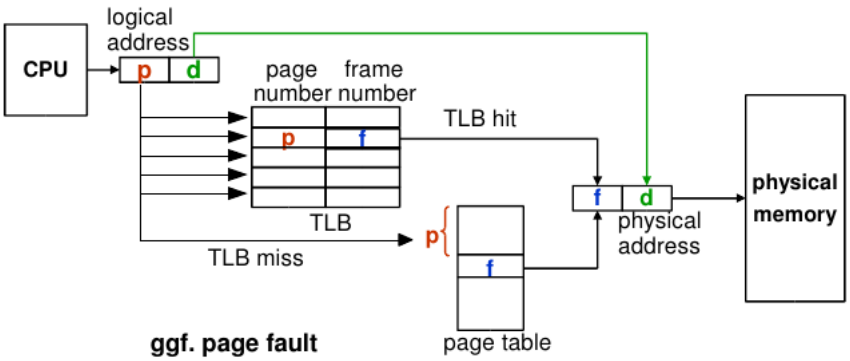

Abbildung 11: Translation Lookaside Buffer

8.5.3 Aufbaben: Betriebssystem und Hardware

Betriebssystem:

- Page-Table-Register Laden
- Page fault behandeln
- Seitenverdrängung

Hardware

- Zugriff auf TLB
- Adressumrechnung

8.5.4 Mehrstufiges Paging

z.B. 32-Bit Adressen p1(10), p2(10), p3(12)

8.5.5 Pagefaults

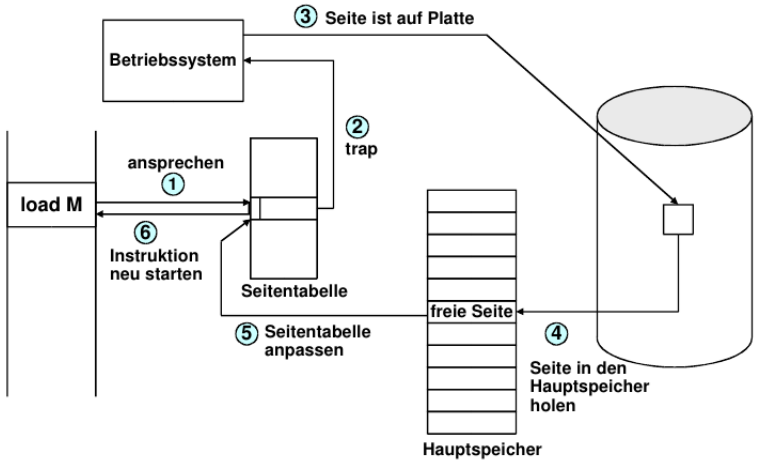

Abbildung 12: Pagefault Behandlung

8.5.6 Verdrängungsstrategie

- Not Recently Used (referenced bit *(regelmäßiger reset)* und modified bit)
  - 0: nicht referenziert, nicht modifiziert
  - 1: nicht referenziert, modifiziert
  - 2: referenziert, nicht modifiziert
  - 3: referenziert, modifiziert
- Second-Chance

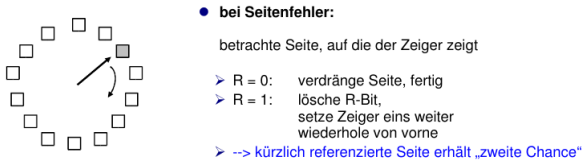

Abbildung 13: Second-Chance Algorithmus

- Least Recently Used (Umsetzung des Zeitstempels ist problematisch)

8.5.7 Trashing

Entsteht wenn mehr Seiten aktiv sind als Seitenrahmen verfügbar.

Abrufstrategien

- Demandpaging (erst bei Bedarf)
- Prepaging (z.B. bei Programmstart)
- asynchron (wenn gerade wenig last)
- Clustering (bei Fault ganzes cluster)
- Locking (Ausnahmen beim Paging)

Mittlere Speicherzugriffszeit bei Warscheinlichkeit **p** für Seitenfehler:

$$t_z = t_{HS} + p \cdot t_{PF}$$

- t_{HS}: Zugriffszeit auf Hauptspeicher
- t_{PF}: Zeit für Behandlung
- *(p sollte klein sein: sonst trashing)*