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

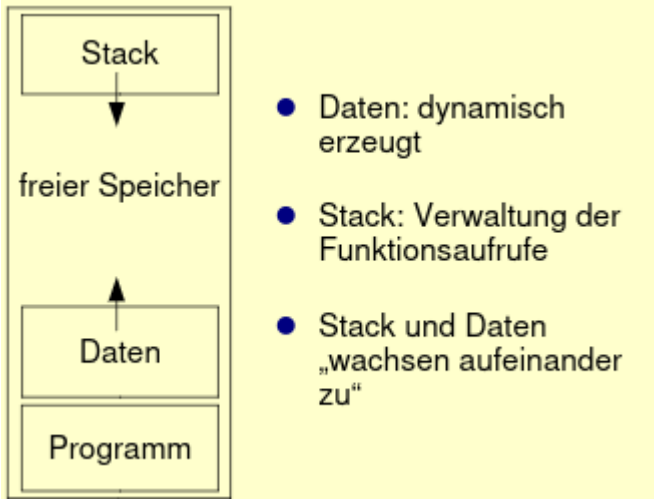

Abbildung 1: Process Control Block PCB

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)*

3 Threads

3.1 Unterschied: Prozesse/Threads

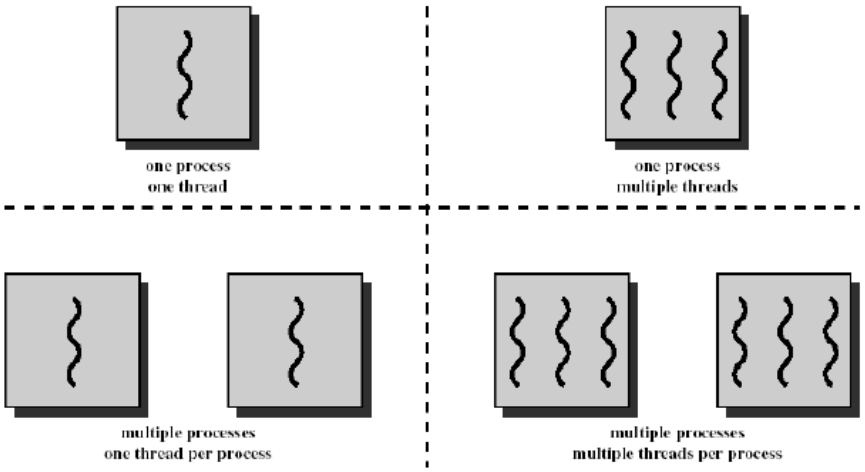

Abbildung 2: Unterschied zw. Prozessen und Threads

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

3.3 Linux Threads und Prozesse

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

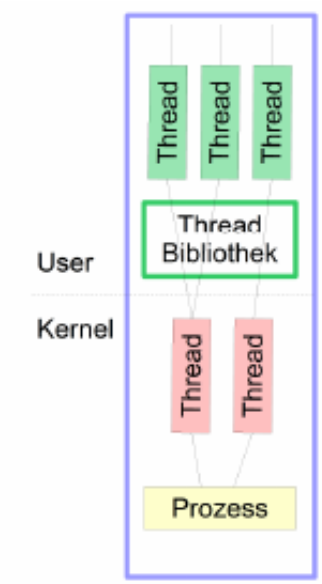

Abbildung 3: Komtiniert: ULT, KLT

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

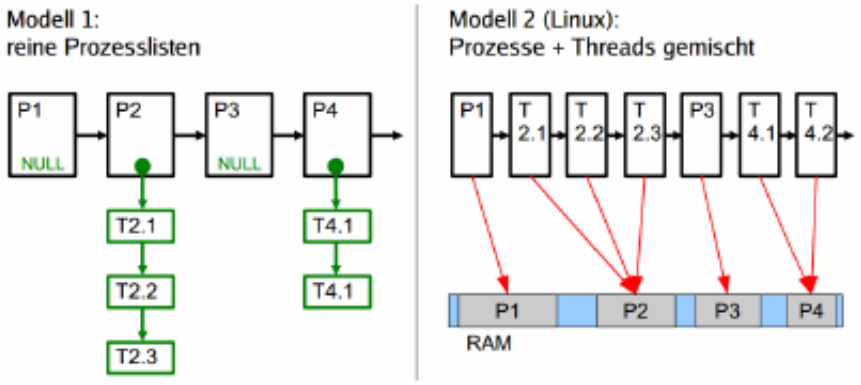

Abbildung 4: Linux Prozess- und Threadverwaltung

4 Interrupts

4.1 Interrupt-Klassen

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

4.2 Ablauf

1. Interrupt
2. Kontext-Wechsel
3. Interrupt-Vector
4. Interrupt-Handler
5. Scheduler

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

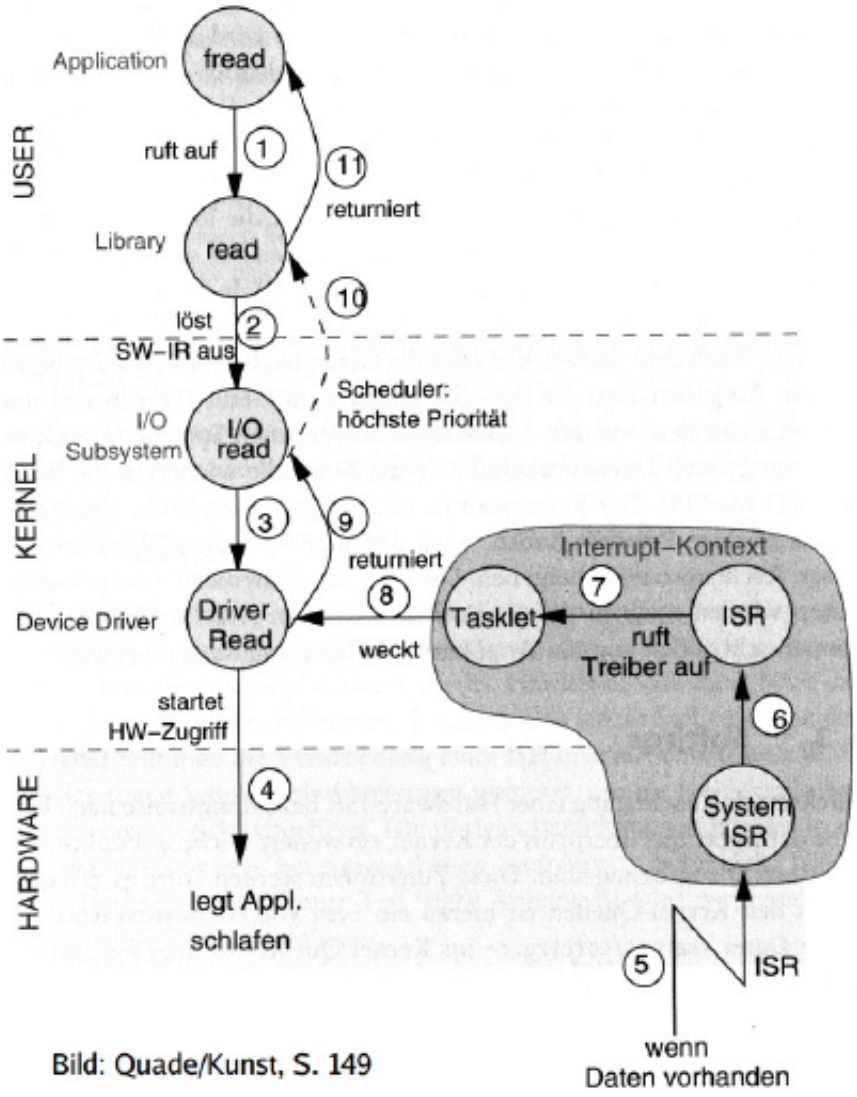

Abbildung 5: Interrupt callgraph

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
  - SJF
  - SRF
  - 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

5.4 Round Robin und I/O

5.4.1 Virtual Round Robin

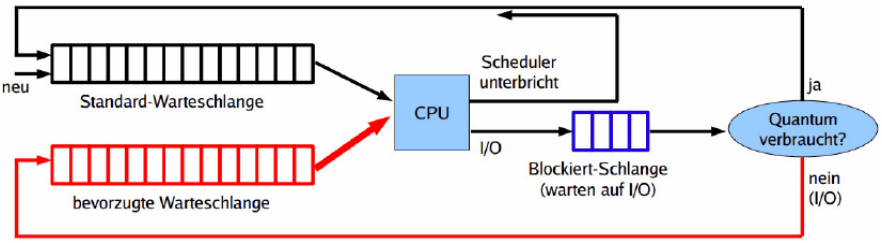

Abbildung 6: Virtual round robin prinzip

5.4.2 Prioritätsbasiert

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

Priority-Inversion:

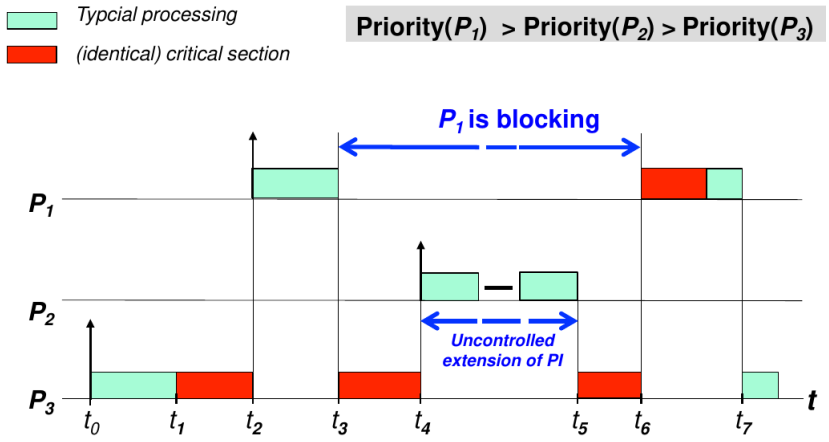

Abbildung 7: Beispiel für Priority-Inversion

5.5 Formeln

5.5.1 Burstdauer

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

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
- Event(-Queue)
- Monitor
- Locking

6.3 Anforderungen

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