# — BS //—

# Ablaufplanung

# Überblick

# Planung — *Scheduling*

**sched·ule** **1** orig., a paper with writing on it **2** a list, catalog, or inventory of details, often as an explanatory supplement to a will, bill of sale, deed, tax form, etc. ⋆**3** a list of times of recurring events, projected operations, arriving and departing trains, etc.; timetable ⋆**4** a timed plan for a procedure or project

**sched·ul·ing** **1** to place or include in a schedule **2** to make a schedule ⋆**3** to plan for a certain time

⋆ = Americanism [3]

# *Scheduling*{,-Algorithmen} [1]

The term *scheduling* is generally understood to cover the questions of when to introduce new processes into the system and the order in which processes should run.

The general objective of a *scheduling algorithm* is to arrange the pattern of work performed by the computing system so as to maximise some measure of user satisfaction.

# Ablaufplanung — *Process Scheduling*

Prozessen das Betriebsmittel { *Prozessor* / *Speicher*[†] / *Gerät*[‡] } zuteilen

[†] Das Hardware-Betriebsmittel ({Vorder,Hinter}grund-) *Speicher* steht (insbesondere im Falle von RAM) auch stellvertretend für (wiederverwendbare/konsumierbare) Software-Betriebsmittel wie z.B. Puffer, Nachrichten, Signale.

[‡] *Gerät* steht stellvertretend für Drucker, Netzwerk, Platte, . . .

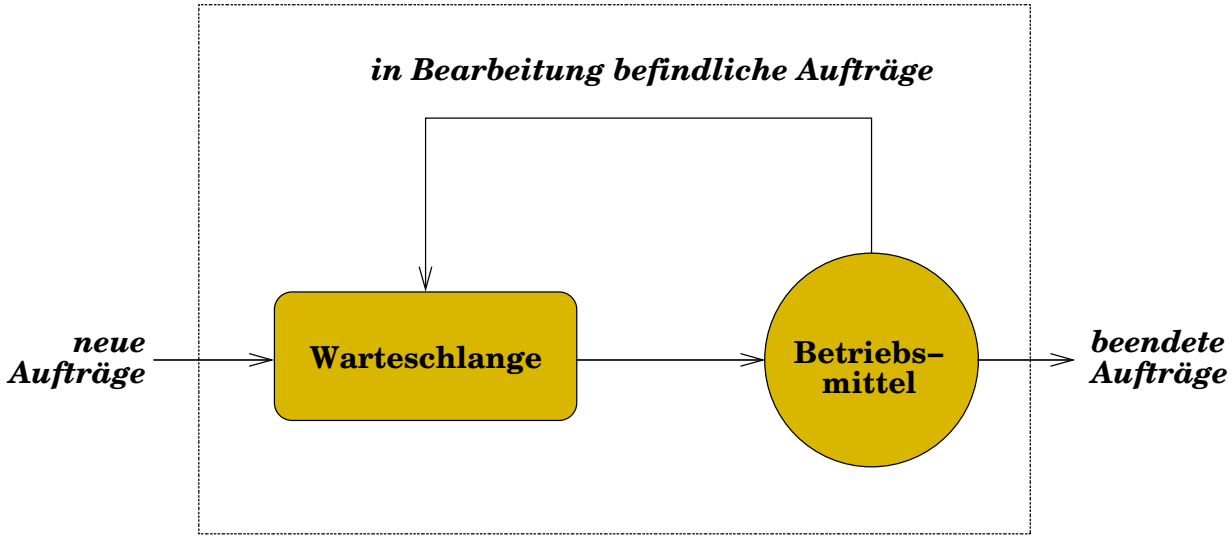
# Ablaufplan — *Process Schedule*

- Fahrplan zur Belegung der {Hard,Soft}ware-Betriebsmittel durch Prozesse
  - geordnet nach Ankunft, Zeit, Termin, Dringlichkeit, Gewicht, . . .
  - die Ordnung ist eine Funktion der Scheduling-Strategie (bzw. -Algorithmen)

- technisch (zumeist) realisiert auf Basis dynamischer Datenstrukturen
  - eine oder mehrere Queues bzw. Schlangen: Warteschlangen (→ p. 7)
  - die Elemente der Datenstruktur sind i.A. die Prozessdeskriptoren

- die gewählte Scheduling-Strategie bestimmt u.a. die Rechnerbetriebsart

# *Scheduling*-Modell




A particular *scheduling algorithm* is characterised by the order of processes in the queue and the conditions under which processes are fed back into it. [1]

# Warteschlangentheorie

- Betriebssysteme durch die „theoretische/mathematische Brille“ gesehen:
  - R. W. Conway, L. W. Maxwell, L. W. Millner: *Theory of Scheduling*, Addison-Wesley, 1967
  - E. G. Coffmann, P. J. Denning: *Operating System Theory*, Prentice Hall, 1973
  - L. Kleinrock: *Queuing Theory*, Vol. II, Computer Applications, John Wiley & Sons, 1976
- die Verfahren stehen und fallen mit den Vorgaben der jeweiligen Zieldomäne

# Einteilung . . .

- nach der **Betriebsmittelart** der zeitweilig belegten Hardware-Ressourcen
- nach der **Betriebsart** des zu bedienenden/steuernden Rechnersystems
- nach dem **Zeitpunkt** der Erstellung des Ablaufplans
- nach der **Vorhersagbarkeit** von Zeitpunkt und -dauer von Prozessabläufen
- nach dem **Laufzeitverhalten** der (Benutzer-/System-) Programme
- nach der **Rechnerarchitektur** des Hardware-/Softwaresystems
- nach der **Ebene** der Entscheidungsfindung zur Betriebsmittelvergabe

# . . . nach der Betriebsmittelart

***CPU scheduling*** des Betriebsmittels „CPU“

- die Prozessanzahl zu einem Zeitpunkt ist höher als die Prozessoranzahl
- ein Prozessor ist zwischen mehreren Prozessen zu multiplexen
- Prozesse werden dem Prozessor über eine Warteschlange zugeteilt (→ p. 6)

***I/O scheduling*** des Betriebsmittels „Gerät“, speziell: „Platte“

- gerätespezifische Einplanung der von Prozessen abgesetzten E/A-Aufträge
- *disk scheduling*, z.B., berücksichtigt typischerweise drei Faktoren:
  – (1) Positionszeit, (2) Rotationszeit, (3) Transferzeit
- Geräteparameter und Gerätezustand bestimmen die nächste E/A-Aktion
- die getroffenen Entscheidungen sind ggf. nicht konform zum *CPU scheduling*

# . . . nach der Betriebsart

***batch scheduling*** interaktionsloser bzw. -unabhängiger Programme

- nicht-verdrängende bzw. verdrängende Verfahren mit langen Zeitscheiben
- Performanzmaximierung durch Minimierung der Kontextwechselanzahl

***interactive scheduling*** interaktionsreicher bzw. -abhängiger Programme

- ereignisgesteuerte, verdrängende Verfahren mit kurzen Zeitscheiben
- Antwortzeitminimierung durch Optimierung der Systemaufrufe

***real-time scheduling*** zeitkritischer bzw. -abhängiger Programme (→ p. 11)

- ereignis- oder zeitgesteuerte deterministische Verfahren: *Vorhersagbarkeit*
- Garantie der (strikten) Einhaltung umgebungsbedingter Zeitvorgaben
- *Rechtzeitigkeit* ist entscheidend und nicht Geschwindigkeit **!**

# . . . nach dem Zeitpunkt

***online scheduling*** dynamisch, während der eigentlichen Programmausführung

- interaktive- und Stapelsysteme (→ p. 10), aber auch schwache Echtzeitsysteme

***offline scheduling*** statisch, vor der eigentlichen Programmausführung (→ p. 12)

- wenn die *Komplexität* eine Ablaufplanung im laufenden Betrieb verbietet
  - Einhaltung aller Zeitvorgaben garantieren: ein NP-vollständiges Problem
  - kritisch, wenn auf jede abfangbare katastrophale Situation zu reagieren ist
- Ergebnis der Vorberechung ist ein vollständiger Ablaufplan (in Tabellenform)
  - (semi-) automatisch erstellt per Quelltextanalyse spezieller „Übersetzer“
  - oft zeitgesteuert abgearbeitet/ausgeführt als Teil der Prozessabfertigung
- die Verfahren sind zumeist beschränkt auf *strikte Echtzeitsysteme*

# . . . nach der Vorhersagbarkeit

***deterministic scheduling*** bekannter, exakt vorberechneter Prozesse (→ p. 11)

- Prozesslaufzeiten/-termine sind bekannt, sie wurden ggf. „*offline*“ berechnet
- die genaue Vorhersage der CPU-Auslastung ist möglich
- das System garantiert die Einhaltung der Prozesslaufzeiten/-termine
- die Zeitgarantien gelten unabhängig von der jeweiligen Systemlast **!**

***probabilistic scheduling*** unbekannter Prozesse

- Prozesslaufzeiten/-termine bleiben unbestimmt
- die (wahrscheinliche) CPU-Auslastung kann lediglich abgeschätzt werden
- das System kann Zeitgarantien nicht geben und auch nicht einhalten
- Zeitgarantien sind durch Anwendungsmaßnahmen bedingt erreichbar

# Vorhersagbarkeit in Linux 2.4 — schedule()

```
asmlinkage void schedule(void) {
    ::
need_resched:
    prev = current;
    rq = this_rq();

    release_kernel_lock(prev, smp_processor_id());
    prepare_arch_schedule(prev);
    prev->sleep_timestamp = jiffies;
    spin_lock_irq(&rq->lock);
    ::
    if (unlikely(!rq->nr_running)) {
        next = rq->idle;
        rq->expired_timestamp = 0;
        goto switch_tasks;
    }
    ::
    idx = sched_find_first_bit(array->bitmap);
    queue = array->queue + idx;
    next = list_entry(queue->next, task_t, run_list);

switch_tasks:
    prefetch(next);
    clear_tsk_need_resched(prev);

    if (likely(prev != next)) {
        rq->nr_switches++;
        rq->curr = next;

        prepare_arch_switch(rq);
        prev = context_switch(prev, next);
        barrier();
        rq = this_rq();
        finish_arch_switch(rq);
    } else
        spin_unlock_irq(&rq->lock);
    finish_arch_schedule(prev);

    reacquire_kernel_lock(current);
    if (need_resched())
        goto need_resched;
}
```

# . . . nach dem Laufzeitverhalten

***cooperative scheduling*** von einander abhängiger Prozesse

- Prozesse müssen die CPU freiwillig abgeben, zugunsten anderer Prozesse
- die Programmausführung muss (direkt/indirekt) Systemaufufe bewirken
- die Systemaufufe müssen (direkt/indirekt) den Scheduler aktivieren

***preemptive scheduling*** von einander unabhängiger Prozesse

- Prozessen wird die CPU entzogen, zugunsten anderer Prozesse
- Ereignisse[1] können die Verdrängung des laufenden Prozesses bewirken
- die Ereignisverarbeitung aktiviert (direkt/indirekt) den Scheduler

[1] Ein Ereignis ist z.B. eine asynchrone Programmunterbrechung (*Interrupt*), die Freigabe eines wiederverwendbaren Betriebsmittels (`V()` bzw. `signal()`) oder die Zustellung eines konsumierbaren Betriebsmittels (`send()`).

# Kooperation — Was steckt hinter den „Systemaufrufen“?

```
main (int argc, char* argv[]) {
    if (argc) {
        puts(argv[0]);
        for (int i = 1; i < argc; i++)
            puts(argv[i]);
    }
}
```

```
#define THR       0 /* Transmitter Holding Register */
#define LSR       5 /* Line Status Register */
#define THR_EMPTY 0x20

void putc (unsigned int port, unsigned char data) {
    while (!(in(port + LSR) & THR_EMPTY));
    out(port + THR, data);
}
```

```
#define OSTREAM 0x4711

void puts (char* s) {
    if (s) {
        unsigned char c;
        while ((c = *s++)) putc(OSTREAM, c);
        putc(OSTREAM, 10);
    }
}
```

```
inline unsigned char in (unsigned int port) {
    unsigned char data;
    asm volatile ("inb  %0,%%al" : : "d" (port));
    asm volatile ("movb %%al,%0" : "=g" (data));
    return data;
}

inline void out (unsigned int port, unsigned char data) {
    asm volatile ("movb %0,%%al" : : "g" (data));
    asm volatile ("outb %%al,%0" : : "d" (port));
}
```

# . . . nach der Rechnerarchitektur

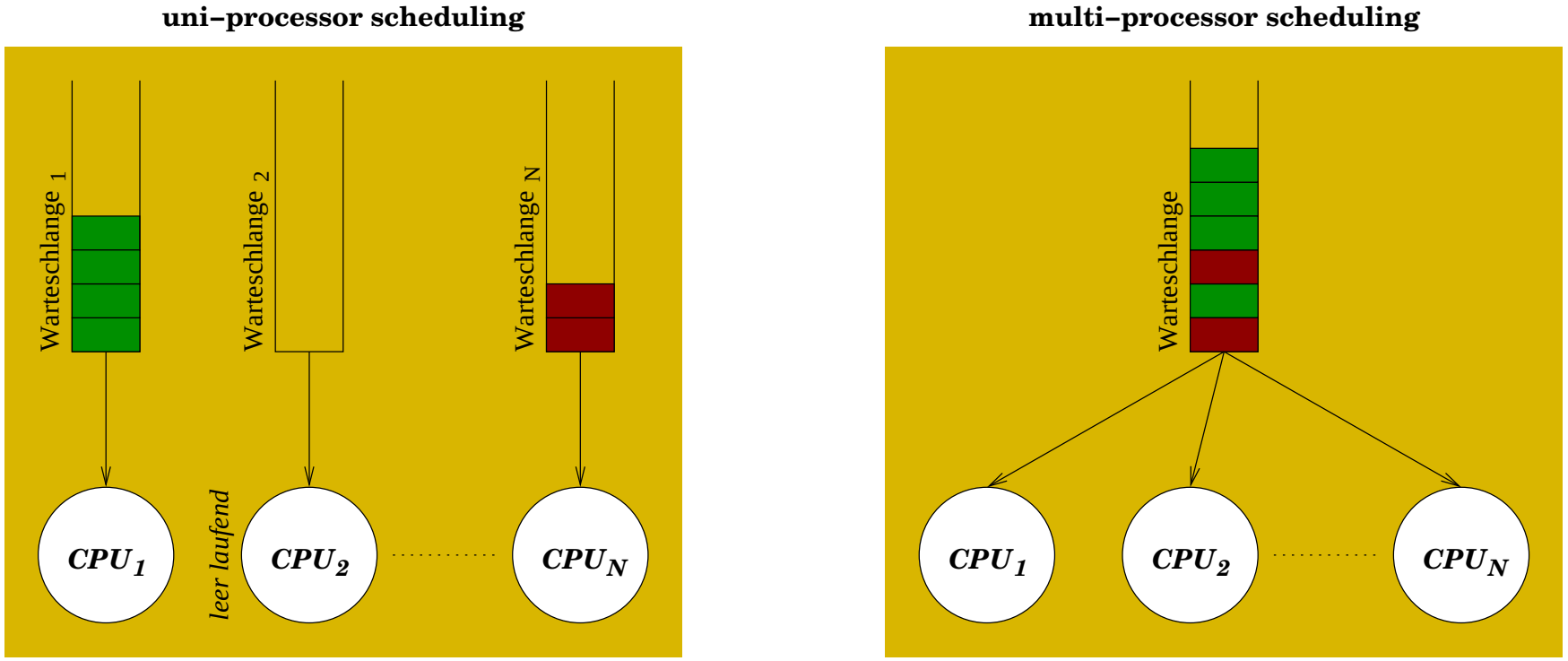
***uni-processor scheduling*** in Mehr{programm,prozess}systemen (→ p. 9)

- die Verarbeitung von Prozessen kann nur pseudo-parallel erfolgen

***multi-processor scheduling*** in Systemen mit gemeinsamen Speicher

- jeder Prozessor arbeitet seine *lobale Warteschlange* ab:
  - die Prozesse sind den Prozessoren (i.d.R.) fest zugeordnet
  - Prozessoren können leer laufen, obwohl noch Prozesse ausführbereit sind
- alle Prozessoren arbeiten eine *globale Warteschlange* ab:
  - die Prozesse sind den Prozessoren nicht fest zugeordnet
  - Prozessoren laufen erst leer, wenn keine Prozesse mehr ausführbereit sind
- die parallele Verarbeitung von Prozessen wird ermöglicht

# uni- vs. multi-processor scheduling



# . . . nach der Ebene

***long-term scheduling*** kontrolliert den Grad an Mehrprogrammbetrieb [s – min]

- Benutzer Systemzugang gewähren, Programme zur Ausführung zulassen
- Prozesse dem *medium-* bzw. *short-term scheduling* zuführen

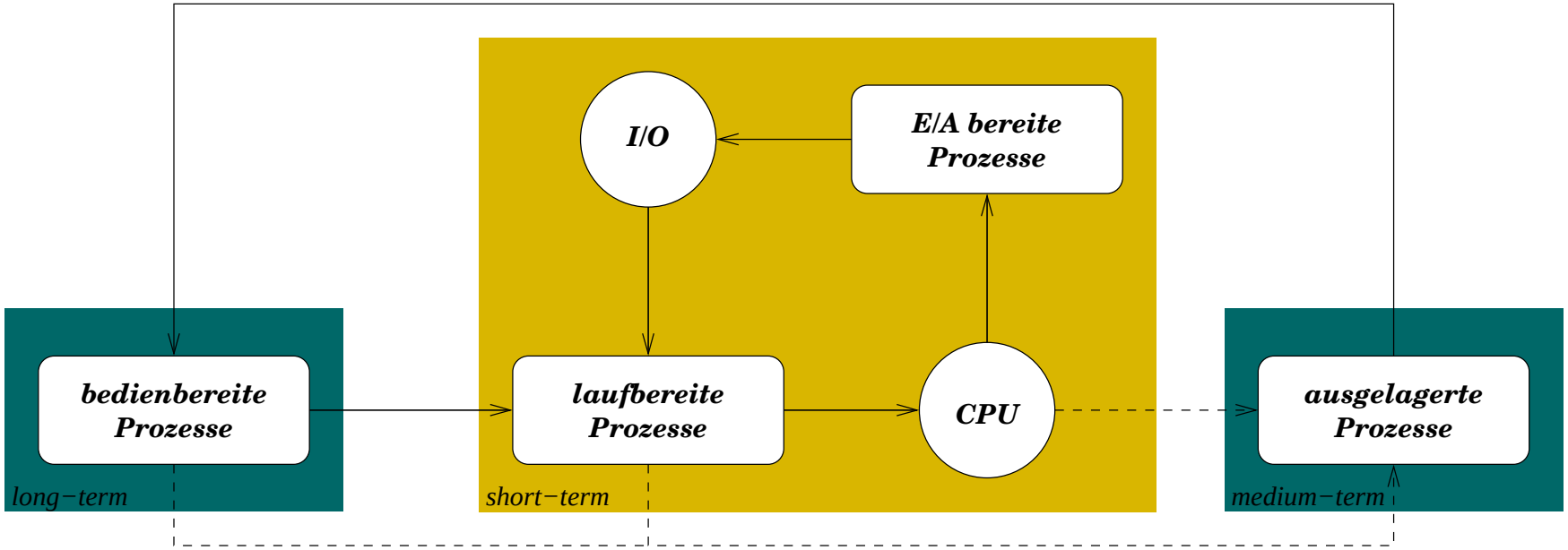
***medium-term scheduling*** als Teil der Ein-/Auslagerungsfunktion [ms – s]

- Programme zwischen Vorder- und Hintergrundspeicher hin- und herbewegen
- *swapping*: auslagern (*swap-out*), einlagern (*swap-in*)

***short-term scheduling*** regelt die Prozessorzuteilung an die Prozesse [$\mu$s – ms]

- ereignisgesteuerte Ablaufplanung: Unterbrechungen, Systemaufrufe, Signale
- Blockierung bzw. Verdrängung des laufenden Prozesses

# long- vs. short- vs. medium-term scheduling



# Dimensionen des *Scheduling*

- die Kriterien, nach der Ablaufplanung betrieben wird, sind unterschiedlich:

  **benutzerorientierte Kriterien** betrachten die *Benutzerdienlichkeit*
  - d.h. das vom jeweiligen Benutzer wahrgenommene Systemverhalten
  - bestimmen im großen Maße die Akzeptanz des Systems beim Benutzer

  **systemorientierte Kriterien** betrachten die *Systemperformanz*
  - d.h. die effektive und effiziente Auslastung der Betriebsmittel
  - sind von Bedeutung bei kommerziellen Dienstleistungsanbietern

- die Benutzerdienlicheit ist auch stark durch die Systemperformanz bedingt

# Benutzerorientierte Kriterien

**Antwortzeit** Minimierung der Zeitdauer von der Auslösung einer Systemanforderung bis zur Entgegennahme der Rückantwort, bei gleichzeitiger Maximierung der Anzahl interaktiver Prozesse.

**Durchlaufzeit** Minimierung der Zeitdauer vom Starten eines Prozesses bis zu seiner Beendigung, d.h., der effektiven Prozesslaufzeit und aller Prozesswartezeiten.

**Termineinhaltung** Starten und/oder Beendigung eines Prozesses zu einem fest vorgegebenen Zeitpunkt.

**Vorhersagbarkeit** Deterministische Ausführung des Prozesses unabhängig von der jeweils vorliegenden Systemlast.

## Systemorientierte Kriterien

**Durchsatz** Maximierung der Anzahl vollendeter Prozesse pro vorgegebener Zeiteinheit. Liefert ein Maß für die geleistete Arbeit im System.

**Prozessorauslastung** Maximierung des Prozentanteils der Zeit, während der die CPU Prozesse ausführt, d.h., „sinnvolle“ Arbeit leistet.

**Gerechtigkeit** Gleichbehandlung der auszuführenden Prozesse und Zusicherung, den Prozessen innerhalb gewisser Zeiträume die CPU zuzuteilen.

**Dringlichkeiten** Bevorzugte Verarbeitung des Prozesses mit der höchsten (statisch/dynamisch zugeordneten) Priorität.

**Lastausgleich** Gleichmäßige Betriebsmittelauslastung bzw. bevorzugte Verarbeitung der Prozesse, die stark belastete Betriebsmittel eher selten belegen.

# Betriebsart vs. Kriterien

***allgemein*:** Durchsetzung (der Strategie), Gerechtigkeit, Lastausgleich[2]

**Stapelsysteme** Durchsatz, Durchlaufzeit, CPU-Ausnutzung[3]

**interaktive Systeme** Antwortzeit, Proportionalität[4]

**Echtzeitsysteme** Dringlichkeit, Termineinhaltung, Vorhersagbarkeit

---

[2] Gerechtigkeit und Lastausgleich sind zwar wünschenswert auch in Echtzeitsystemen, jedoch darf die Umsetzung dieser Kriterien nicht zu Lasten der Echtzeitfähigkeit gehen. Diese Kriterien sind daher nicht immer erfüllbar.

[3] Die CPU die gesamte Zeit über sinnvolle Arbeit verrichten und Jobs ausführen lassen.

[4] Benutzer haben meist eine inhärente Vorstellung darüber, wie lange bestimmte Aktionen dauern müssten. Dieser (oft auch falschen) Vorstellung sollte das System aus Gründen der Benutzerakzeptanz möglichst entsprechen.

# Zusammenfassung

- Scheduling hat großen Einfluss auf die Performanz des Gesamtsystems
  - es legt fest, { welche Prozesse warten und welche voranschreiten / welche Betriebsmittel wie ausgelastet sind }
- die Ziele des Scheduling variieren mit dem Einsatzfeld des Rechnersystems
  - einerseits optimierte Verfahren riskieren { Leistungsverluste / Systemblockaden / Katastrophen } andererseits
- zum Scheduling kann es die „Eiermilch legende Wollmilchsau“ nicht geben

Ein Gerücht besagt, dass bei der Abschaltung eines Rechners am MIT im Jahre 1973 ein Job in der CPU-Warteschlange gefunden worden ist, der dort sechs Jahre zuvor hineinkam, jedoch aufgrund seiner niedrigen Priorität nicht genügend oft die CPU zugeteilt bekam und dadurch seine Aufgabe noch nicht vollständig hat erfüllen können.

# Referenzen


[1] A. M. Lister and R. D. Eager. *Fundamentals of Operating Systems*. The Macmillan Press Ltd., fifth edition, 1993. ISBN 0-333-59848-2.

[2] J. Nehmer and P. Sturm. *Systemsoftware: Grundlagen moderner Betriebssysteme*. dpunkt.Verlag GmbH, zweite edition, 2001. ISBN 3-89864-115-5.

[3] V. E. Neufeld, editor. *Webster's New World Dictionary*. Simon & Schuster, Inc., third college edition, 1988. ISBN 0-13-947169-3.

[4] A. Silberschatz and P. B. Galvin. *Operating System Concepts*. Addison-Wesley, 1994. ISBN 0-201-59292-4.

[5] W. Stallings. *Operating Systems: Internals and Design Principles*. Prentice-Hall, fourth edition, 2001. ISBN 0-13-031999-6.

[6] A. S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, second edition, 2001. ISBN 0-13-031358-0.