Systemprogrammierung

Betriebssystemkonzepte: Prozesse

Wolfgang Schröder-Preikschat

Lehrstuhl Informatik 4

18. Juni 2014

Gliederung

Gerichteter Ablauf eines Geschehens [11]

Betriebssysteme bringen Programme zur Ausführung, in dem dazu Prozesse erzeugt, bereitgestellt und begleitet werden

- im Informatikkontext ist ein Prozess ohne Programm nicht möglich
  - die als Programm kodierte Berechnungsvorschrift definiert den Prozess
  - das Programm legt damit den Prozess fest, gibt ihn vor
  - gegebenenfalls bewirkt, steuert, terminiert es gar andere Prozesse[1]
- ein Programm beschreibt (auch) die Art des Ablaufs eines Prozesses

  sequentiell
  - eine Folge von zeitlich nicht überlappenden Aktionen
  - verläuft deterministisch, das Ergebnis ist determiniert

  parallel
  - nicht sequentiell
- in beiden Arten besteht ein Programmablauf aus **Aktionen** (S. 6 *ff.*)

Beachte: Programmablauf und Abstraktionsebene

Ein und derselbe Programmablauf kann auf einer Abstraktionsebene sequentiell, auf einer anderen parallel sein. [10]

[1] Wenn das Betriebssystem die dazu nötigen Befehle anbietet.

Ursprünglich als Rechtsbegriff

Prozess bedeutet „streitiges Verfahren vor Gericht, mit dem Ziel, den Streit durch eine verbindliche Entscheidung zu klären“ [11, Recht]

- Analogie in der Informatik bzw. zu Betriebssystemkonzepten:
  - **Streit**
    - Rivalität[2] bei Inanspruchnahme von Betriebsmitteln
    - Konkurrenz (lat. *concurrere* zusammenlaufen)
  - **Verfahren**
    - Vorgehensweise zur planmäßigen Problemlösung
    - Strategie (*policy*) oder Methode der Problemlösung
  - **Gericht**
    - Funktion zur Einplanung (*scheduling*), Koordinierung
    - Synchronisationspunkt in einem Programm
  - **Verbindlichkeit**
    - Konsequenz, mit der die Einplanungszusagen gelten
    - Einhaltung zugesagter Eigenschaften, Verlässlichkeit
- in der Regel folgen die Verfahren einer hierarchischen Gerichtsbarkeit
  - dabei wird ein Verfahrensabschnitt als Instanz (*instance*) bezeichnet
    - Übernahme von „Instanz“ in die Informatik war jedoch ungeschickt **!!!**
  - Betriebssysteme verfügen oft über eine mehrstufige Prozessverarbeitung

[2] lat. *rivalis* „an der Nutzung eines Wasserlaufs mitberechtigter Nachbar“

Gliederung

Programm: Programmier-/Assembliersprachenebene

Definition

Die für eine Maschine konkretisierte Form eines Algorithmus.

- virtuelle Maschine C
  - nach der Editierung und
  - vor der Kompilierung

```
1 #include <stdint.h>
2
3 void inc64(int64_t *i) {
4   (*i)++;
5 }
```

- eine Aktion (Zeile 4)

- virtuelle Maschine ASM (x86)
  - nach der Kompilierung[3] und
  - vor der Assemblierung

```
1 inc64:
2   movl 4(%esp), %eax
3   addl $1, (%eax)
4   adcl $0, 4(%eax)
5   ret
```

- drei Aktionen (Zeilen 2–4)

Definition (Aktion)

Die Ausführung einer Anweisung einer (virtuellen/realen) Maschine.

[3] `gcc -O6 -m32 -static -fomit-frame-pointer -S`, auch i.F.

Programm: Maschinenprogrammebene

- Adressbereich und virtuelle Maschine SMC[4]
  - Textsegment
  - Linux
- nach dem Binden und
- vor dem Laden
- reale Maschine
  - nach dem Laden
  - ablauffähig

```
1  0x080482f0:    mov  0x4(%esp),%eax    8b 44 24 04
2  0x080482f4:    add  $0x1,(%eax)       83 00 01
3  0x080482f7:    adc  $0x0,0x4(%eax)    83 50 04 00
4  0x080482fb:    ret                    c3
```

- gleiche Anzahl von Aktionen (Zeilen 1–3, jew.), aber verschiedene Darstellungsformen

Hinweis (`ret` bzw. `c3`)

Die Aktion zum Unterprogrammrücksprung korrespondiert zur Aktion des Unterprogrammaufrufs (`gdb`, `disas /rm main`):

```
0x080481c9: c7 04 24 b0 37 0d 08  movl $0x80d37b0,(%esp)
0x080481d0: e8 1b 01 00 00        call 0x80482f0 <inc64>
```

[4] symbolischer Maschinenkode (*symbolic machine code*): x86 + Linux.

Nichtsequentielles Programm I

Definition

Ein Programm P, das Aktionen spezifiziert, die parallele Abläufe in P selbst zulassen.

- ein Ausschnitt von P am Beispiel von *POSIX Threads* [6]:

```
1  pthread_t tid;
2
3  if (!pthread_create(&tid, NULL, thread, NULL)) {
4    /* ... */
5    pthread_join(tid, NULL);
6  }
```

- der in P selbst zugelassene parallele Ablauf:

```
1  void *thread(void *null) {
2    /* ... */
3    pthread_exit(NULL);
4  }
```

Nichtsequentielles Programm II

- trotz Aktionen für Parallelität, **sequentielle Programmabläufe**:

```
1  pid_t pid;
2
3  if (!(pid = fork())) {
4    /* ... */
5    exit(0);
6  }
7
8  wait(NULL);
```

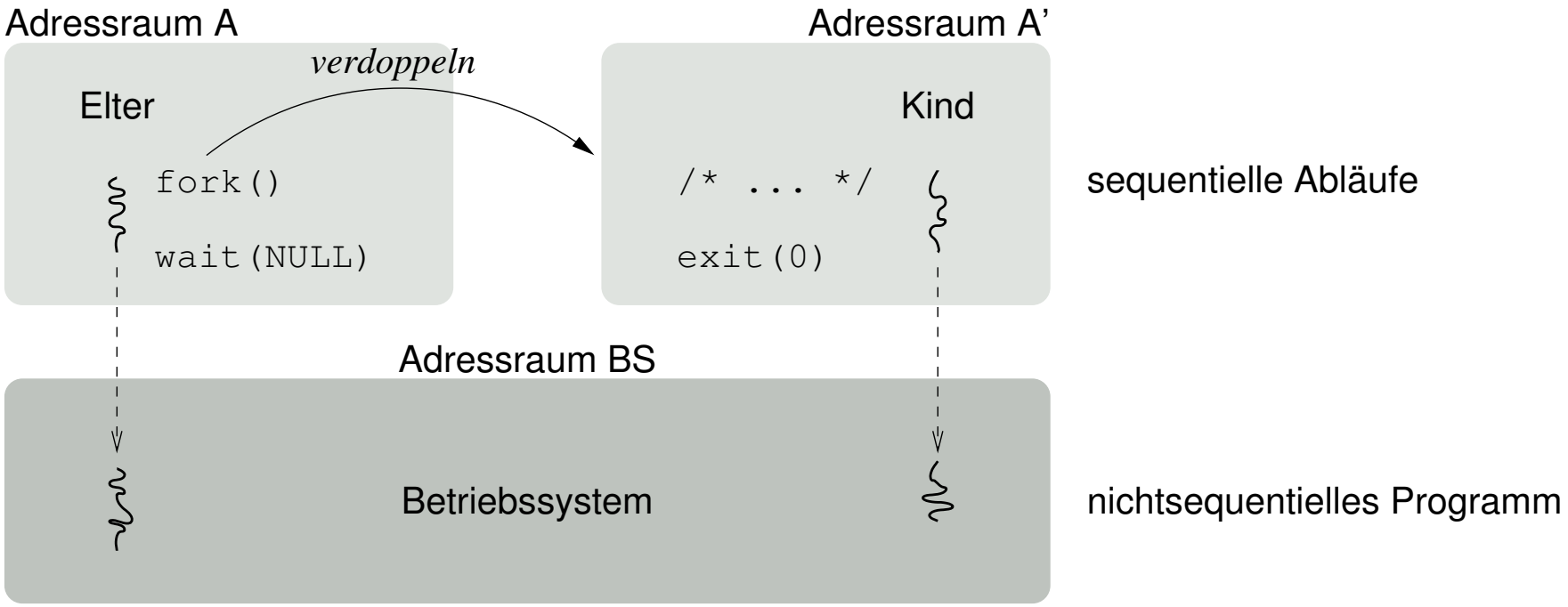
- `fork` dupliziert den Adressraum A von P, erzeugt A' als Duplikat von A
- in A als Ursprungsadressraum entsteht damit jedoch kein paralleler Ablauf
- unabhängig vom Parallelitätsgrad in P, setzt `fork` diesen für A' immer auf 1

  - sequentielle Abläufe können allerdings parallele Abläufe in einer Domäne bilden, die die sequentiellen Abläufe (logisch) umfasst

- die Aktionen bedingen parallele Abläufe innerhalb des Betriebssystems
  - Simultanbetrieb (*multiprocessing*) sequentieller Abläufe benötigt das Betriebssystem in Form eines nichtsequentiellen Programms
  - hilfreiches Merkmal: Mehrfädigkeit (*multithreading*) im Betriebssystem
- „Betriebssystem“ ist **Inbegriff** für „nichtsequentielles Programm“[5]

[5] Ausnahmen (strikt kooperative Systeme) bestätigen die Regel.

Simultanverarbeitung sequentieller Abläufe



- Prozessor(kern)ausprägung vs. Betriebssystemarchitektur:
  - Uni
    - prozessbasiertes Betriebssystem, nämlich:
    - Pseudoparallelität durch **Multiplexen** des Prozessor(kern)s
  - Multi
    - dito; aber auch ereignisbasiertes Betriebssystem, nämlich:
    - echte Parallelität durch **Vervielfachung** des Prozessor(kern)s
- beide Fälle bewirken **parallele Prozesse** (S. 15) im Betriebssystem

Prozess

Definition (Programmablauf)

Ein Programm in Ausführung durch einen Prozessor.

- das Programm spezifiziert eine Folge von Aktionen des Prozessors
  - die Art einer Aktion hängt von der betrachteten Abstraktionsebene ab

    Ebene $_5$ $\mapsto$ Programmanweisung ≥ 1 Assembliermnemoniks
    Ebene $_4$ $\mapsto$ Assembliermnemonik ≥ 1 Maschinenbefehle
    Ebene $_3$ $\mapsto$ Maschinenbefehl ≥ 1 Mikroprogramminstruktionen
    Ebene $_2$ $\mapsto$ Mikroprogramminstruktion
  - die Aktion eines Prozessors ist damit nicht zwingend unteilbar (atomar)
    - sowohl für den abstrakten (virtuellen) als auch den realen Prozessor
- das Programm ist statisch (passiv), ein Prozess ist dynamisch (aktiv)

Hinweis (Prozess $\neq$ Prozessinkarnation)

Eine Prozessinkarnation ist **Exemplar** *eines Prozesses.*[a]

[a] Genauso wie ein Objekt Exemplar einer Klasse ist.

Unteilbarkeit I

Definition (in Anlehnung an den Duden)

Das Unteilbarsein, um etwas als Einheit oder Ganzheit in Erscheinung treten zu lassen.

- eine Frage der „Distanz“ des Betrachters (Subjekts) auf ein Objekt
  - **Aktion** auf höherer, **Aktionsfolge** auf tieferer Abstraktionsebene

Ebene	Aktion	Aktionsfolge
5	`i++`	

4–3	`incl i`*	`movl i,%r`
		`addl $1,%r`*
		`movl %r,i`

	`addl $1,i`*	
2–1		* **read** *from memory into accumulator* **modify** *contents of accumulator* **write** *from accumulator into memory*

  - typisch für den Komplexbefehl eines „abstrakten Prozessors“ (C, CISC)

Unteilbarkeit II

Ganzheit oder Einheit einer Aktionsfolge, deren Einzelaktionen alle scheinbar gleichzeitig stattfinden (d.h., synchronisiert sind)

- wesentliche nichtfunktionale Eigenschaft für eine atomare Operation[6]
  - die logische Zusammengehörigkeit von Aktionen in zeitlicher Hinsicht
  - wodurch die Aktionsfolge als **Elementaroperation** (ELOP) erscheint

Beispiele von (kritischen) Aktionen zum Inkrementieren eines Zählers:

- Ebene $5 \mapsto 3$

C/C++

```
1  i++;
```

ASM

```
1  movl  i, %eax
2  addl  $1, %eax
3  movl  %eax, i
```

- Ebene $3 \mapsto 2$

ASM

```
1  incl  i
```

ISA

```
1  read   A from <i>
2  modify A by 1
3  write  A to <i>
```

- die Inkrementierungsaktionen (`i++`, `incl`) sind nur bedingt unteilbar
  - unterbrechungsfreier Betrieb (Ebene $5 \mapsto 3$), Uniprozessor (Ebene $3 \mapsto 2$)
  - Problem: zeitliche Überlappung von Aktionsfolgen hier gezeigter Art

[6] von (gr.) *átomo* „unteilbar“.

Sequentieller Prozess

Definition

Ein Prozess, der sich aus einer Folge von zeitlich nicht überlappenden Aktionen zusammensetzt.

- die Aktionsfolge bildet einen Ausführungsfaden (*execution thread*)
  - von dem es im sequentiellen Prozess nur einen einzigen gibt
  - der sich mit jedem Wiederanlauf des Prozesses anders entwickeln kann
    - andere Eingabedaten, Programmänderung, transiente Hardwarefehler
- die Folge ist durch eine **totale Ordnung** ihrer Aktionen definiert
  - sie ist bei unveränderten Ausgangsbedingungen reproduzierbar

Hinweis (Ausführungsfaden $\neq$ *Thread*)

Annahmen zur technischen Umsetzung der Aktionsfolge werden nicht getroffen und sind hier auch ohne Belang.
Ein Thread ist nur eine technische Option, um die Inkarnation eines sequentiellen Prozesses zu realisieren.

Nichtsequentieller Prozess

Definition

Ein auch als „parallel“ bezeichneter Prozess, der sich aus einer Folge von Aktionen zusammensetzt, die sich zeitlich überlappen können.

- Voraussetzung ist ein **nichtsequentielles Programm** (vgl. S. 8)
  - das wenigstens eine weitere Prozessinkarnation (Kindprozess) zulässt
  - das die Behandlung von Ereignissen externer Prozesse zulässt[7]
- wodurch die Überlappung von Aktionsfolgen erst ermöglicht wird:
  - i simultane Mehrfädigkeit (*multithreading*), und zwar:
    - pseudo-gleichzeitig • Multiplexbetrieb eines einzelnen Prozessor(kern)s
    - echt gleichzeitig • Parallelbetrieb → (mehrkerniger) Multiprozessor
  - ii asynchrone Programmunterbrechung (*interrupt*)
- die Folge aller Aktionen ist durch eine **partielle Ordnung** definiert
  - da externe Prozesse zeitlich/kausal unabhängige Aktionen ermöglichen

[7] Unterbrechungsanforderung von einem Gerät (IRQ) oder Prozess (Signal).

Gleichzeitige Prozesse (*concurrent processes* [5])

Definition

Wenigstens zwei Aktionsfolgen eines nichtsequentiellen Prozesses, die sich zeitlich überlappen.

- sind sie unabhängig, heißen diese Prozesse **nebenläufig** (*concurrent*)
  - keiner der gleichzeitigen Prozesse ist Ursache/Wirkung des anderen
  - keine Aktionsfolge dieser Prozesse benötigt das Resultat einer anderen
- gleichzeitige Prozesse stehen miteinander in **Wettstreit** (*contention*)
  - sie teilen sich den Prozessor(kern), (Zwischen-)Speicher, Bus/Geräte
  - dadurch entsteht **Interferenz**[8] (*interference*) im Prozessverhalten
- der effektive Überlappungsgrad ist irrelevant für die „Gleichzeitigkeit“
  - nicht jedoch für zeitabhängige Prozesse, die Termine einzuhalten haben
  - beachte: je größer die Überlappung, desto größer die Verzögerung
    - desto wahrscheinlicher verpasst der verzögerte Prozess seinen Termin
  - ebenso Interferenz, die zur Verletzung von Zeitbedingungen führen kann

[8] Abgeleitet von (frz.) *s'entreferir* „sich gegenseitig schlagen“.

Gekoppelte Prozesse (*interacting processes* [5, S. 77])

Definition (auch: „abhängige Prozesse“)

Gleichzeitige Prozesse, die interagierend auf gemeinsame Variablen zugreifen oder Betriebsmittel gemeinsam benutzen.

- die Aktionen geraten in **Konflikt**, wenn mind. einer der Prozesse. . .
  - eine der gemeinsamen Variablen verändert (Zugriffsart) oder
  - ein gemeinsames unteilbares Betriebsmittel[9] belegt (Betriebsmittelart)
- daraus kann sich eine **Wettlaufsituation** (*race condition*) entwickeln
  - um die gemeinsame Variable oder das gemeinsame Betriebsmittel
  - um die beabsichtigte Aktionsfolge starten oder beenden zu können
- Konflikte werden durch **Synchronisationsmaßnahmen** unterbunden:

blockierend	• die beabsichtigte Aktionsfolge nicht starten lassen
nicht-blockierend	• die gestartete Aktionsfolge stoppen & wiederholen
reduzierend	• die teilbare Aktionsfolge „elementarisieren“, d.h., auf eine ELOP der realen Maschine abbilden

- **Koordination** der Kooperation und Konkurrenz zwischen Prozessen

[9] Drucker, Maus, Plotter, Tastatur.

Gekoppelte Prozesse: Wettlaufsituationen

```
1  int64_t cycle = 0;
2
3  void *thread_worker(void *null) {
4    for (;;) {
5      /* ... */
6      inc64(&cycle);
7    }
8  }
9
10 void *thread_minder(void *null) {
11   for (;;) {
12     printf("worker cycle %lld\n", cycle);
13     pthread_yield();
14   }
15 }
```

- `inc64`: siehe S. 6 (bzw. Kap. V-4, S. 28)

- welche `cycle`-Werte gibt der Aufpasserfaden (*minder*) aus?
- welche Werte entstehen bei mehreren Arbeiterfäden (*worker*)?
  - wenn `thread_worker` in mehreren identischen Inkarnationen existiert

Gekoppelte Prozesse: 1. Wettlaufsituation

- angenommen, das nichtsequentielle Programm läuft im 32-Bit Betrieb
  - Exemplare von `int64_t` bilden ein Paar von 32-Bit Worten
  - Operationen auf diesen Paaren sind keine Einzelaktionen mehr

- Arbeiterfaden

```
1  .L6:
2    movl $cycle, (%esp)
3    call inc64
4    jmp  .L6

1  inc64:
2    movl 4(%esp), %eax
3    addl $1, (%eax)
4    adcl $0, 4(%eax)
5    ret
```

- Aufpasserfaden

```
1  movl cycle+4, %edx  ; high &
2  movl cycle, %eax    ; low word
3  movl $.LC0, (%esp)
4  movl %edx, 8(%esp)
5  movl %eax, 4(%esp)
6  call printf
```

- angenommen $cycle = 2^{32} - 1$
  - `inc64` überlappt Aktionen 1–2 (o.)
  - dann gilt $edx = 0$ und $eax = 0$
  - `printf` gibt 0 aus, nicht 2^{32} !!!

Gekoppelte Prozesse: 2. Wettlaufsituation

- angenommen, die Entwicklungs- oder Laufzeitplattform variiert
  - verschiedene Kompilierer, Assemblierer, Binder und Lader
  - verschiedene Betriebssysteme – jedoch der gleiche reale Prozessor (x86)
- GCC 4.7.2, Linux

```
1 inc64:
2   movl 4(%esp), %eax
3   addl $1, (%eax)
4   adcl $0, 4(%eax)
5   ret
```

- GCC 4.2.1, MacOSX

```
1 _inc64:
2   movl 4(%esp), %eax
3   movl (%eax), %ecx
4   movl 4(%eax), %edx
5   addl $1, %ecx
6   adcl $0, %edx
7   movl %edx, 4(%eax)
8   movl %ecx, (%eax)
9   ret
```

- **pseudo-gleichzeitige Aktionen**
  - (UNIX-) Signal oder *Interrupt*
  - asynchr. Programmunterbrechung
- **echt gleichzeitige Aktionen**
  - insbesondere sind `addl` und `adcl` kritisch: teilbares *read-modify-write*
- ein klassischer Fehler: ggf. wirkungslose Zählaktionen [7, S. 26–29]

Gliederung

Konsistenz: Koordination gekoppelter Prozesse

Beschaffenheit in Bezug auf sich überlappende Aktionsfolgen

- Wettlaufsituationen vorbeugen, kritische Abschnitte schützen
  - explizit eine Aktionsfolge (physisch oder logisch) unteilbar machen
  - grundsätzlich: je kürzer diese Zeitspanne, desto besser die Lösung **!!!**

Überlappungen ausschließen durch **Synchronisation** von Prozessen:

- Blockierung wettstreitiger Prozesse – vergleichsweise lange Zeit

```
1 void mutex_inc64(int64_t *i, pthread_mutex_t *lock) {
2   pthread_mutex_lock(lock);     /* indivisible, now */
3   inc64(i);                     /* reuse code @ p.6 */
4   pthread_mutex_unlock(lock);   /* divisible, again */
5 }
```

- Reduzierung auf unteilbare 64-Bit-Operation des realen Prozessors

```
1 void inc64(int64_t *i) {        /* renew code @ p.6 */
2   asm ("lock incq %0" : : "m" (*i) : "memory");
3 }
```

  - allg., überall einsetzbare, um Größenordnungen effizientere Lösung

Koordinationsmittel: Austausch von Zeitsignalen

Semaphor (von gr. *sema* „Zeichen“ und *pherein* „tragen“)

- „nichtnegative ganze Zahl“ ($\mathbb{N}_0$), für die – nach der ursprünglichen Definition [3] – zwei **unteilbare Operationen** definiert sind:[10]
  $P(s)$ (von hol. *prolaag* „erniedrige“)
  - koppelt den Prozess (logisch, ggf. auch physisch) an s
  - blockiert den Prozess, falls s den Wert 0 hat
  - dekrementiert den Wert von s um 1, sonst

  $V(s)$ (von hol. *verhoog* „erhöhe“)
  - inkrementiert den Wert von s um 1, entkoppelt den Prozess
  - bewirkt die Bereitstellung der/des an s blockierten Prozesse/s
    - Fortgang aller beteiligten Prozesse regelt der **Planer** (*scheduler*)
- binär ($s = [0, 1]$) oder allgemein ($s = [0, n], n > 0$) ausgelegt
  - funktional entspricht ein binärer Semaphor einer Sperrvariablen (*mutex*)

Beachte: Verortung von Prozessen

Prozesse und die zugehörigen Koordinationsmittel müssen demselben Abstraktionsniveau entsprechen, funktional und nichtfunktional.

[10] vgl. auch Erweiterung auf „ganze Zahlen“, $\mathbb{Z}$ [4, S. 345 – 346].

Verortung: Betriebssystem- vs. Anwendungsebene

Prozesse sind in einem Rechensystem verschiedenartig verankert

- und zwar inner- oder oberhalb der Maschinenprogrammebene

  innerhalb
  - ursprünglich, im Betriebssystem bzw. Kern (*kernel*)
  - Prozessinkarnation als Wurzel
  - partielle Virtualisierung des Prozessor(kern)s

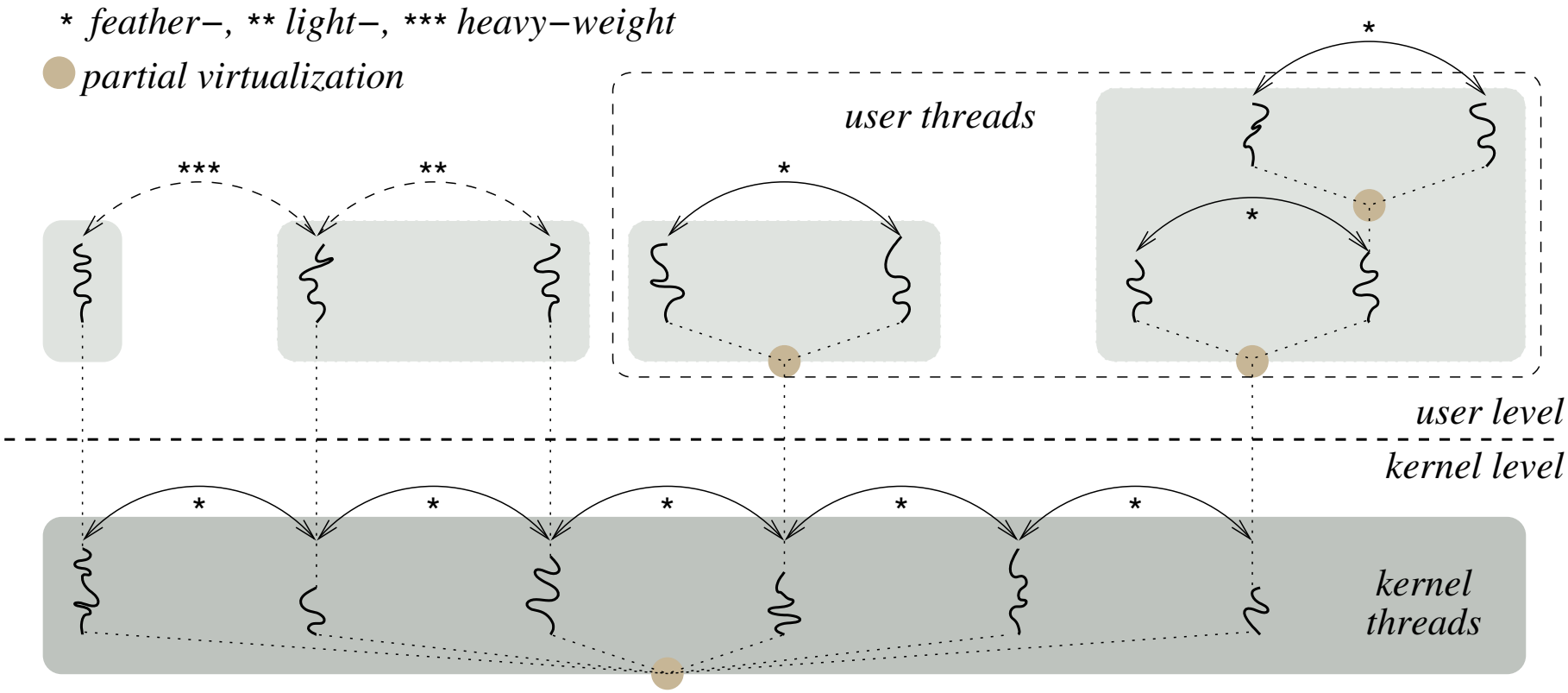
  ↪ „*kernel thread*“ in der Informatikfolklore

  oberhalb
  - optional, im Laufzeit- oder gar Anwendungssystem
  - Prozessinkarnation als Blatt oder innerer Knoten
  - partielle Virtualisierung der Wurzelprozessinkarnation

  ↪ „*user thread*“ in der Informatikfolklore

- die jew. Entität weiß nicht, dass sie ggf. (partiell) virtualisiert wird
  - einem „*kernel thread*“ sind seine „*user threads*“ gänzlich unbewusst
  - ein „*user thread*“ entsteht durch Zeitmultiplexen eines „*kernel threads*“
- Betriebssysteme kennen nur ihre eigenen Prozessinkarnationen
  - schaltet ein „*kernel thread*“ weg, setzen alle seine „*user threads*“ aus
  - ein „*kernel thread*“ entsteht durch Raum-/Zeitmultiplexen der CPU

Gewichtsklasse: Unterbrechungs-/Wiederaufnahmeaufwand



Arten von **Prozesswechsel** zur partiellen **Prozessorvirtualisierung**:

- * im selben (Benutzer-/Kern-) Adressraum, ebenda fortsetzend
- ** im Kernadressraum, denselben Benutzeradressraum teilend
- *** im Kernadressraum, im anderen Benutzeradressraum landend

Planung der Einlastung

„*scheduling*“ von „*dispatching*“

Prozesseinplanung (engl. *process scheduling*) stellt sich allgemein zwei grundsätzlichen Fragestellungen:

1. Zu welchem **Zeitpunkt** sollen Prozesse in das Rechensystem eingespeist werden?
2. In welcher **Reihenfolge** sollen eingespeiste Prozesse ablaufen?

Zuteilung von Betriebsmitteln an konkurrierende Prozesse kontrollieren

Einplanungsalgorithmus (engl. *scheduling algorithm*)

Implementiert die **Strategie**, *nach der ein von einem Rechnersystem zu leistender Ablaufplan zur Erfüllung der jeweiligen Anwendungsanforderungen entsprechend aufzustellen und zu aktualisieren ist.*

Reihenfolgebestimmung

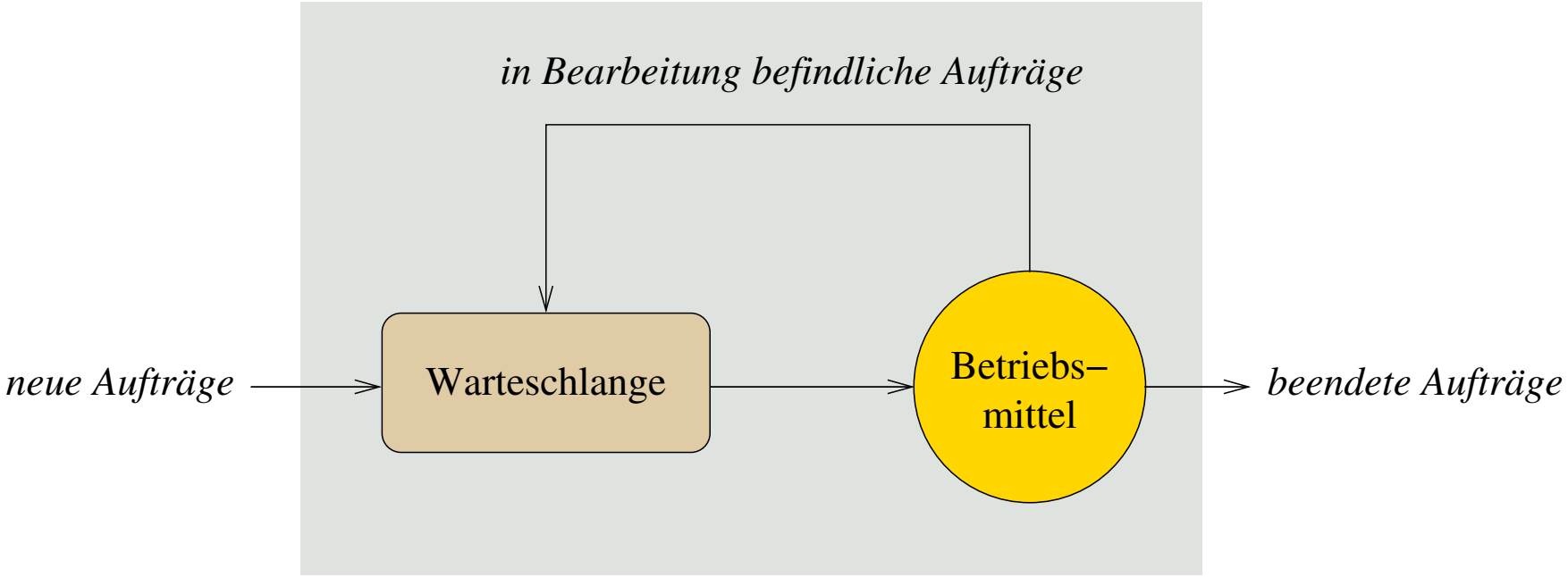
Ablaufplan (engl. *schedule*) zur Betriebsmittelzuteilung erstellen

- geordnet nach Ankunft, Zeit, Termin, Dringlichkeit, Gewicht, ...
- entsprechend der jeweiligen Einplanungsstrategie
- zur Unterstützung einer bestimmten Rechnerbetriebsart

Prozessen das Betriebsmittel { Prozessor, Speicher, Gerät, ⋮, Datei, Nachricht, Puffer } koordiniert zuteilen

Einplanungsalgorithmen

Verwaltung von (betriebsmittelgebundenen) Warteschlangen



Ein einzelner Einplanungsalgorithmus ist charakterisiert durch die Reihenfolge von Prozessen in der Warteschlange und die Bedingungen, unter denen die Prozesse in die Warteschlange eingereiht werden. [9]

Warteschlangentheorie

Theoretische Grundlagen des Scheduling

Betriebssysteme durch die „theoretische/mathematische Brille“ gesehen:

- R. W. Conway, L. W. Maxwell, L. W. Millner. *Theory of Scheduling*.
- E. G. Coffman, P. J. Denning. *Operating System Theory*.
- L. Kleinrock. *Queuing Systems*, *Volume I: Theory*.

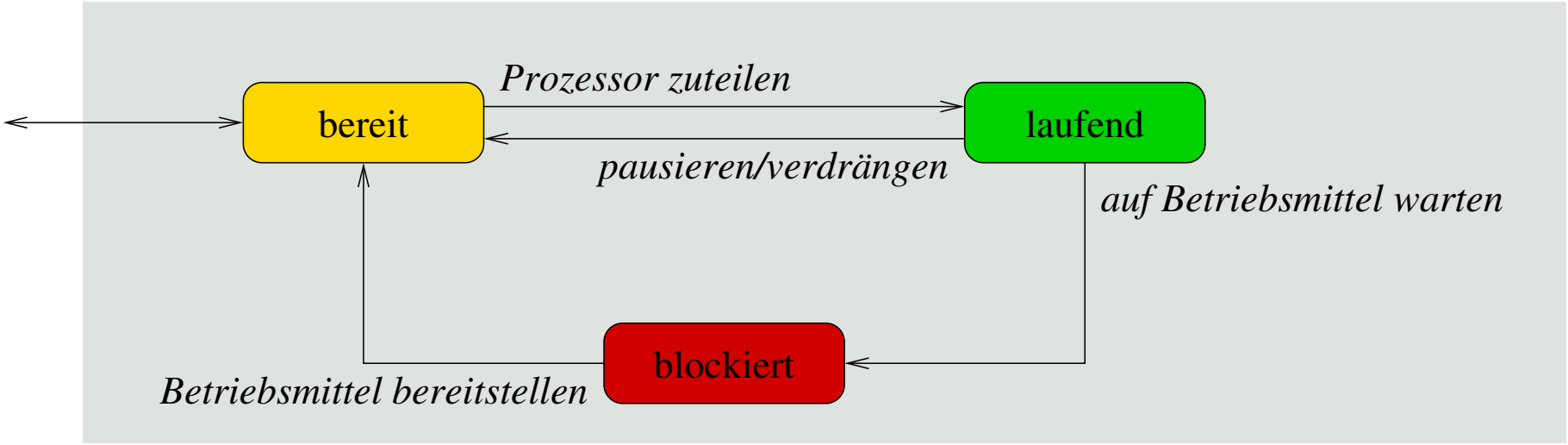
Einplanungsverfahren stehen und fallen mit Vorgaben der Zieldomäne

- die „Eier-legende Wollmilchsau“ kann es nicht geben
- Kompromisslösungen sind geläufig
  - aber nicht in allen Fällen tragfähig

- Scheduling ist ein **Querschnittsbelang** (engl. *cross-cutting concern*)

Verarbeitungszustände

Zustandsübergänge implementiert ein Planer (engl. *scheduler*)



Prozessverarbeitung impliziert die Verwaltung mehrerer Warteschlangen:

- häufig sind Betriebsmitteln eigene Warteschlangen zugeordnet
  - in denen Prozesse auf Zuteilung des jew. Betriebsmittels warten
- im Regelfall sind in Warteschlangen stehende Prozesse blockiert. . .
  - mit Ausnahme der **Bereitliste** (engl. *ready list*)
  - die auf Zuteilung der CPU wartenden Prozesse sind laufbereit

Gliederung

Resümee

- Programm: für eine Maschine konkretisierte Form eines Algorithmus
  - sequentiell: Aktionen, die nur serielle Abläufe in P zulassen
  - nichtsequentiell: Aktionen, die parallele Abläufe in P selbst zulassen
  - Aktion: Ausführung einer Maschinenanweisung
- Prozess: ein Programm in Ausführung durch einen Prozessor
  - sequentiell: eine Folge von zeitlich nicht überlappenden Aktionen
  - nichtsequentiell: eine Folge von zeitlich überlappenden Aktionen
  - gleichzeitig: unabhängige zeitlich überlappende Aktionsfolgen
  - gekoppelt: abhängige zeitlich überlappende Aktionsfolgen
- Planer: legt Zeitpunkt und Reihenfolge von Prozessen fest
  - Prozessinkarnationen werden Verarbeitungszustände zugeschrieben
    - Zustandsübergänge (in EBNF): $\{\{\text{bereit, laufend}\}-, [\text{blockiert}]\}$
  - UNIX: verdrängend, nicht-deterministisch, gekoppelt, Zeitmultiplex
- Betriebssystem: Inbegriff für ein nichtsequentielles Programm
  - asynchrone Programmunterbrechung (*interrupt*)
  - Simultanverarbeitung (*multiprocessing*) sequentieller Programme

Literaturverzeichnis

[1] Coffman, E. G. ; Denning, P. J.:
Operating System Theory.
Prentice Hall, Inc., 1973

[2] Conway, R. W. ; Maxwell, L. W. ; Millner, L. W.:
Theory of Scheduling.
Addison-Wesley, 1967

[3] Dijkstra, E. W.:
Over seinpalen / Technische Universiteit Eindhoven.
Eindhoven, The Netherlands, 1964 ca. (EWD-74). –
Manuskript. –
(dt.) Über Signalmasten

[4] Dijkstra, E. W.:
The Structure of the "THE"-Multiprogramming System.
In: *Communications of the ACM* 11 (1968), Mai, Nr. 5, S. 341–346

[5] Hansen, P. B.:
Concurrent Processes.
In: *Operating System Principles*.
Englewood Cliffs, N.J., USA : Prentice-Hall, Inc., 1973. –
ISBN 0–13–637843–9, Kapitel 3, S. 55–131

Literaturverzeichnis (Forts.)

[6] IEEE:
POSIX.1c Threads Extensions / Institute of Electrical and Electronics Engineers.
New York, NY, USA, 1995 (IEEE Std 1003.1c-1995). –
Standarddokument

[7] Kleinöder, J. ; Schröder-Preikschat, W. :
Rechnerorganisation.
In: Lehrstuhl Informatik 4 (Hrsg.): *Systemprogrammierung*.
FAU Erlangen-Nürnberg, 2014 (Vorlesungsfolien), Kapitel Betriebssystemebene

[8] Kleinrock, L. :
Queuing Systems. Bd. I: Theory.
John Wiley & Sons, 1975

[9] Lister, A. M. ; Eager, R. D.:
Fundamentals of Operating Systems.
The Macmillan Press Ltd., 1993. –
ISBN 0–333–59848–2

[10] Löhr, K.-P. :
Nichtsequentielle Programmierung.
In: Institut für Informatik (Hrsg.): *Algorithmen und Programmierung IV*.
Freie Universität Berlin, 2006 (Vorlesungsfolien)

Literaturverzeichnis (Forts.)

[11] Wikipedia:
Prozess.
http://de.wikipedia.org/wiki/Prozess, Nov. 2013