Systemprogrammierung

Grundlagen von Betriebssystemen

Teil B – V.1 Betriebssystemkonzepte: Prozesse

22. Mai 2025

Rüdiger Kapitza

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Technische Fakultät

Agenda

Gliederung

Lehrstoff

- **Prozess** als das zentrale Konzept von Betriebssys. kennenlernen
  - wobei von der **Verkörperung** (Inkarnation) dieses Konzepts getrennt wird
    - ob also ein Prozess z.B. als *Thread* oder *Task* implementiert ist bzw.
    - ob er allein oder mit anderen zusammen im selben Adressraum verweilt und
    - ob sein Adressraum eine physisch durchzusetzende Schutzdomäne darstellt
  - auf das Wesentliche konzentrieren: Prozess als „*program in execution*“ [5]
- auf (die Art der) **Betriebsmittel** eingehen, die ein Prozess benötigt
  - wiederverwendbare und konsumierbare Betriebsmittel unterscheiden
  - implizite und explizite Koordinierung von Prozessen verdeutlichen, d.h.,
  - geplante und programmierte **Synchronisation** von Prozessen erklären
  - mehr- und einseitige Synchronisation beispielhaft zeigen: *bounded buffer*
- **Prozessausprägungen** und zugehörige Funktionen betrachten
  - typische (logische) Verarbeitungszustände von Prozessen einführen
  - Einplanung (*scheduling*) und Einlastung (*dispatching*) differenzieren
  - Verortung von Prozessen auf Benutzer- und Systemebene skizzieren
  - Prozesskontrollblock, -zeiger und -identifikation begrifflich erfassen

Gliederung

Grundlagen

Virtualität

CPU, Arbeitsspeicher und Festplatte aus Sicht eines Programms

Sicht des Status quo (vereinfacht):

- CPUs führen einen endlosen Strom von Befehlen (im Speicher) aus
- Arbeitsspeicher bildet einen lückenlosen physischen Adressraum
- Persistenter Speicher setzt sich aus Blöcken zusammen
- Alle Instruktionen können direkt ausgeführt werden
  (wie bspw. im privilegierten Modus)

Damit das Betriebssystem gleichzeitig mehrere Programme unterstützen kann, muss es die Ausführung der verschiedenen Programme trennen und jedem Programm die Illusion vermitteln, es sei das einzige laufende Programm.

$\Rightarrow$ Die Prozessabstraktion sorgt genau für diese Illusion

Begrifflichkeit

Prozessabstraktion

- Ein Programm besteht aus Code und Daten
  (bspw. abgelegt auf einer Festplatte)
- Ein Prozess ist eine *Instanz* eines Programms
  (zu jedem Zeitpunkt können ein oder mehr Instanzen eines Programms ausgeführt werden)

Prozessdefinition

- Ein Prozess ist ein Ausführungsstrom im Kontext eines Prozesszustands
- Der Ausführungsstrom ist die Abfolge der ausgeführten Befehle
- Der Prozesszustand umfasst alles, was von den ausgeführten Befehlen beeinflusst werden kann oder von ihnen beeinflusst wird
  (z. B. Register, Adressraum und persistenter Zustand)

Prozess- und Threaderzeugung

Prozesserzeugung

- Programme werden zu Prozessen, wenn sie in eine konkrete Ausführungsumgebung (Prozessinstanz) geladen werden

Threaderzeugung (bzw. Fadenerz.)

- Ein Prozess beginnt mit *einem* Thread und einem Adressraum
- Ein Prozess kann *mehrere* Threads im selben Adressraum starten
  - Jeder Thread erhält seinen eigenen Stack, aber sie teilen sich globale Daten, Code und Heap.
- Prozesse können weitere Prozesse erzeugen (siehe `fork()`)

Vorausblende: Teilen von Ressourcen

Zwei Arten des Teilens

- ***Zeitlich*** geteilt versus ***räumlich*** geteilt
  - (Bsp.: Ich miete das Restaurant. Oder ich reserviere einen Tisch.)

Virtualisierung der CPU

- Ziel: Jedem Prozess die Illusion eines exklusiven CPU-Zugriffs geben
- Realität: CPU ist eine gemeinsam genutzte Ressource der Prozesse

Unterschiedliche Strategien für CPU, Arbeitsspeicher und Festplatte

- CPU: Zeitaufteilung, Wechsel zwischen Programmabläufe
- Arbeitsspeicher: Speicherplatzaufteilung (mehr dazu später)
- Festplatte: Speicherplatzaufteilung (dito.)

Partielle Virtualisierung

- Die Prozessabstraktion ermöglicht es, simultan mehrere Programmabläufe im **Multiplexverfahren** auf einem Prozessor stattfinden zu lassen
  - dabei sind die Abläufe Teil eines einzelnen oder mehrerer Programme
    - Mehrfädigkeit (*multithreading*)/Mehrprogrammbetrieb (*multiprogramming*)
  - für den Ablauf lastet das Betriebssystem einen Prozess ein (*dispatching*)
    - Laufzeitkontext umschalten aktiviert dann einen anderen Programmablauf
  - hierzu plant das Betriebssystem Prozesse entsprechend ein (*scheduling*)
- geläufig ist das **Zeitteilverfahren** (*time-sharing*; CTSS [3]), von dem es verschiedene Ausführungen gibt
  - je nachdem, wie viel und wie oft den Prozessen Rechenzeit innerhalb einer bestimmten Zeitspanne zugeordnet werden kann, soll oder muss
  - pro **Zeitschlitz** laufen im Prozess meist mehrere Aktionen ab

Simultanverarbeitung

multiprocessing

- Prozesse sind das Mittel zum Zweck, (pseudo/quasi) **gleichzeitige Programmabläufe** stattfinden zu lassen $\rightsquigarrow$ **Parallelität**

  multiprogramming — mehrere Programme

  multitasking — mehrere Aufgaben mehrerer Programme

  multithreading — mehrere Fäden eines oder mehrerer Progra.

  - pseudo/quasi gleichzeitig, wenn weniger reale Prozessoren zur Verfügung stehen als zu einem Zeitpunkt Programmabläufe möglich sind
- ein Programmablauf ist möglich, wenn:

  i er dem Betriebssystem explizit gemacht worden ist und

  ii alle von ihm benötigten **Betriebsmittel** (real/virtuell) verfügbar sind
- ist eine gemeinsame Benutzung oder logische Abhängigkeit von Betriebsmitteln gegeben, wird **Synchronisation** erforderlich
  - die Fäden/Aufgaben/Programme teilen sich dieselben (realen) Daten
  - formuliert in dem Programm, das damit als **nichtsequentielles Programm** in Erscheinung tritt

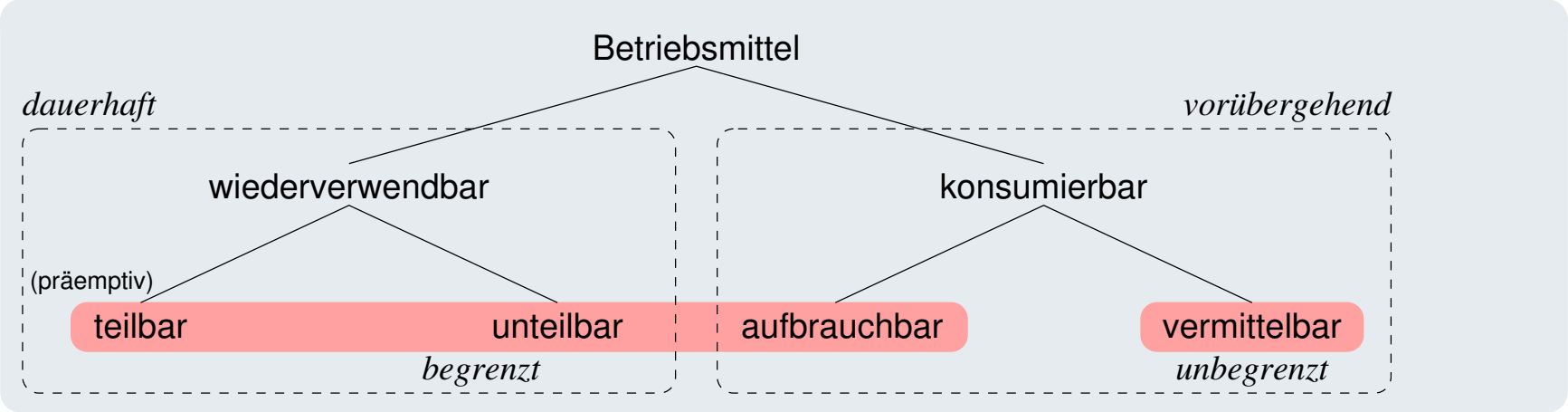
  $\hookrightarrow$ die Maßnahmen dazu gestalten sich recht unterschiedlich, je nach Art des Betriebsmittels und Zweck des Prozesszugriffs

Grundlagen

Betriebsmittel

Betriebsmittelklassifikation

nach [12, 13]



- **dauerhafte**[1] Betriebsmittel sind von Prozessen **wiederverwendbar**
  - sie werden angefordert, belegt, benutzt und danach wieder freigegeben
  - in Benutzung befindliche Betriebsmittel sind ggf. **zeitlich teilbar**
    - je nachdem, ob der **Betriebsmitteltyp** eine gleichzeitige Benutzung zulässt
  - falls unteilbar, sind sie einem Prozess **zeitweise exklusiv** zugeordnet
- **vorübergehende** Betriebsmittel sind von Prozesse **konsumierbar**
  - sie werden produziert, zugeteilt/vermittelt, benutzt und aufgebraucht
- wiederverwendbare (gegenständliche) und aufbrauchbare (messbare) Betriebsmittel stehen nur begrenzt zur Verfügung

[1] auch: persistente

Eigentümlichkeiten von Betriebsmitteln

- vom Betriebssystem zu verwaltende Betriebsmittel:

wiederverwendbar (Hardware)	**konsumierbar**
Prozessor (CPU, FPU, GPU; MMU)	**Signal** (IRQ, NMI, *trap*)
Speicher (RAM, *scratch pad*, *flash*)	**Größe** (Zeit, Energie)
Peripherie (Ein-/Ausgabe, *storage*)	

- von jedem Programm verwaltete Softwarebetriebsmittel:

wiederverwendbar	**konsumierbar**
Text (kritischer Abschnitt)	**Signal** Meldung
Daten (Variable, Platzhalter)	**Nachricht** Datenstrom

  - wiederverwendbare Betriebsmittel sind Behälter für vermittelbare
    - zur Verarbeitung müssen letztere in Variablen/Platzhaltern vorliegen
  - Verfügbarkeit ersterer beschränkt Erzeugung/Verbrauch letzterer
- gleichzeitige Zugriffe auf unteilbare und Übernahme vermittelbarer Betriebsmittel erfordern die **Synchronisation** involvierter Prozesse

Grundlagen

Programme

Gerichteter Ablauf eines Geschehens [17]

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
    - wenn das Betriebssystem die dazu nötigen Befehle anbietet!

- ein Programm beschreibt 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**

Definition (Programm)

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[2] und
  - vor der Assemblierung

```
11 inc64:
12   movl 4(%esp), %eax
13   addl $1, (%eax)
14   adcl $0, 4(%eax)
15   ret
```

- drei Aktionen (Zeilen 12–14)

Definition (Aktion)

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

[2] Übersetzung des Unterprogramms (Z. 1–5) mit -S.

Nichtsequentielles Maschinenprogramm

Definition

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

- ein Ausschnitt von *P* am Beispiel von *POSIX Threads* [14]:

```
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:

```
7  void *thread(void *null) {
8    /* ... */
9    pthread_exit(NULL);
10 }
```

~~Nicht~~sequentielles Maschinenprogramm

- Aktionen für Parallelität, aber weiterhin **sequentielle Abläufe in** P

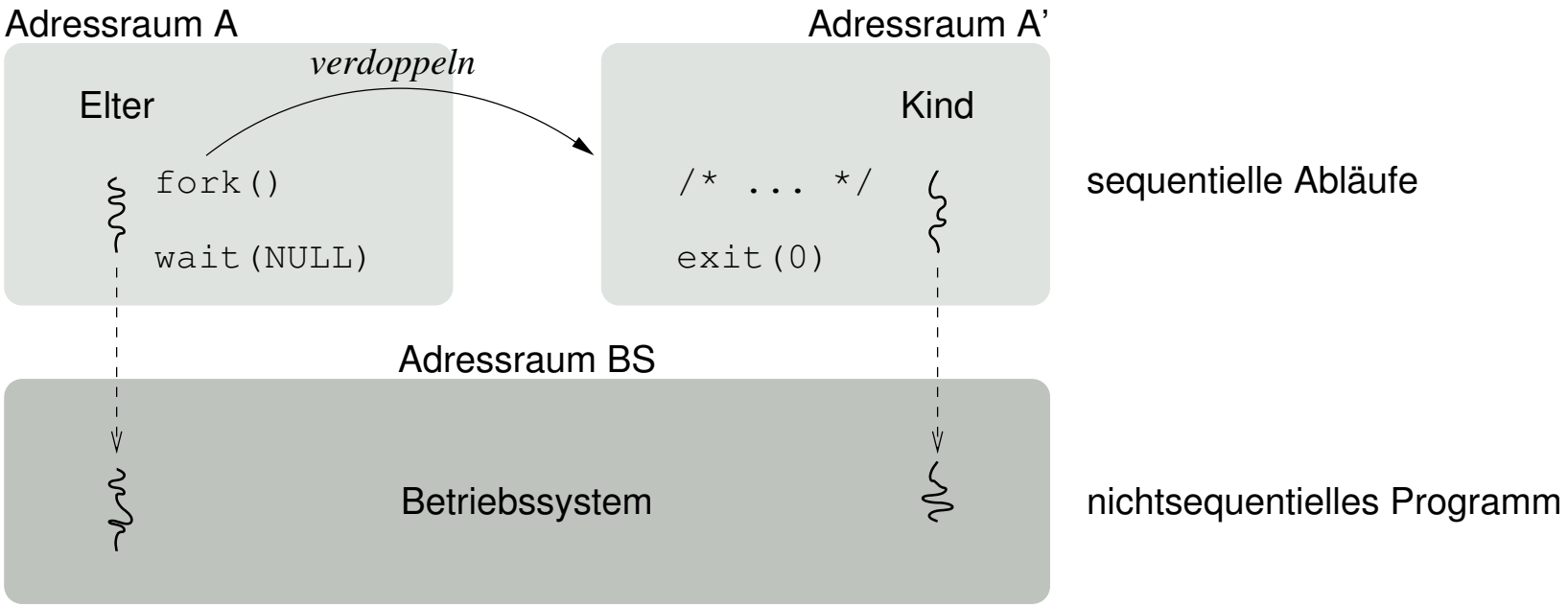
```
1  pid_t pid;
2
3  if (!(pid = fork())) {
4    /* ... */
5    exit(0);
6  }
7
8  wait(NULL);
```

- `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

  - Programm P spezifiziert zwar Aktionen, die Parallelität zulassen, diese kommt jedoch nur allein durch `fork` nicht in P selbst zur Wirkung
- die Aktionen bedingen parallele Abläufe innerhalb des Betriebssys.
  - Simultanbetrieb (*multiprocessing*) sequentieller Abläufe benötigt das Betriebssystem in Form eines nichtsequentiellen Programms
  - hilfreiches Merkmal: Mehrfädigkeit (*multithreading*) im Betriebssystem
- ein Betriebssystem ist **Inbegriff** des nichtsequen. Programms[3]

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

Simultanverarbeitung sequentieller Abläufe



- dabei ist die **Parallelität** im System unterschiedlich ausgeprägt:
  - **pseudo**
    - durch *Multiplexen* eines realen/virtuellen Prozessors (S. 11)[4]
  - **echte**
    - durch *Vervielfachung* eines realen Prozessors
- Folge der Operationen sind **parallele Prozesse** im Betriebssystem
  - auch als **nichtsequentielle Prozesse** bezeichnet
  - nämlich Prozesse, deren Aktionen sich zeitlich überlappen können

[4](gr.) *pseúdein* belügen, täuschen

Gliederung

Verwaltung

Planung

Planung der Einlastung

„scheduling“ von *„dispatching“*

Prozesse werden gestartet, unterbrochen, fortgesetzt und beendet

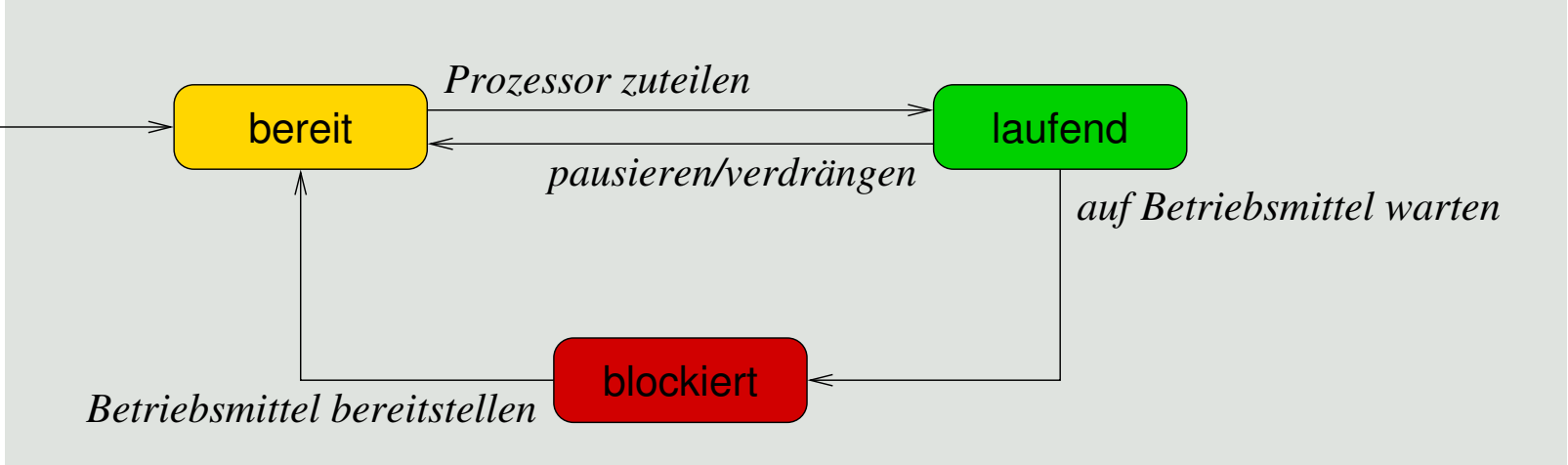
- zentrale Funktion dabei die **Prozesseinplanung** (*proce. scheduling*), die allgemein zwei grundsätzliche Fragestellungen zu lösen hat:
  - i Zu welchem (logischen/physikalischen) Zeitpunkt sollen Prozesse in den Kreislauf der Programmverarbeitung eingespeist werden?
  - ii In welcher Reihenfolge sollen die eingespeisten Prozesse stattfinden?

- Zweck aller hierzu erforderlichen Verfahren ist es, die **Zuteilung von Betriebsmitteln** an konkurrierende Prozesse zu kontrollieren

Einplanungsalgorithmus (*scheduling algorithm*)

Beschreibt und formuliert die **Strategie**, nach der ein von einem Rechensystem zu leistender Ablaufplan zur Erfüllung der jeweiligen **Anwendungsanforderungen** entsprechend der gewählten **Rechnerbetriebsart** aufzustellen, abzuarbeiten und fortzuschreiben ist.

- ein Prozess kann angeordnet werden und stattfinden, wenn alle dazu benötigten Betriebsmittel verfügbar sind



- die Zustandsübergänge bewirkt der **Planer** (*scheduler*), sie definieren verschiedene Phasen der Prozessverarbeitung

scheduling
- beim Übergang in die Zustände „bereit“ oder „blockiert“

dispatching
- beim Übergang in den Zustand „laufend“

- je **Rechenkern** kann es zu einem Zeitpunkt stets nur einen laufenden, jedoch mehr als einen blockierten oder bereiten Prozess geben

Reihenfolgebestimmung

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

- Betriebsmittel, die in **Hardware** oder **Software** ausgeprägt vorliegen

- fehlen Prozessen nur noch ein Prozessor als Betriebsmit. definiert die **Bereitliste** (*ready list*) den **Ablaufplan** zur Prozessorzuteilung
  - Listenelemente sind **Prozesskontrollblöcke** (siehe S. 38), geordnet[5] nach Ankunft, Zeit, Termin, Dringlichkeit, Gewicht, …
  - die Liste repräsentiert sich als statische oder dynamische Datenstruktur

[5] Gemäß Einplanungsstrategie für eine bestimmte Rechnerbetriebsart (Stapel-, Mehrzugangs-, Echtzeitbetrieb).

Verwaltung

Synchronisation

Koordinierung

explizit — die Ausnahme bestätigt die Regel

- Prozesseinplanung profitiert von **Vorwissen** zu Kontrollfluss- und Datenabhängigkeiten, die vorhersehbar sind
  - dann ist ein Ablaufplan möglich, der die Prozesse impliziert koordiniert
- ohne Abhängigkeitsvorwissen sind Prozesse explizit zu koordinieren, per **Programmanweisung**
  - der Ablaufplan reiht zwar Prozesse, koordiniert diese jedoch nicht

Definition (Synchronisation [11])

Koordination der Kooperation und Konkurrenz zwischen Prozessen.

- verläuft unterschiedlich, je nach Betriebsmittel- und Prozesszugriffsart

Beachte: Auch vorhergesagte Prozesse finden unvorhersehbar statt, wenn nämlich der Ablaufplan sich als nicht durchsetzbar erweist.

- weil das Vorwissen unvollständig, durch **Ungewissheit** geprägt ist
- weil die **Berechnungskomplexität** den engen Zeitrahmen sprengt
- weil plötzlichem **Hintergrundrauschen** nicht vorgebeugt werden kann
  - Unterbrechungen, Zugriffsfehler auf Zwischen- oder Arbeitsspeicher
  - Befehlsverknüpfung (*pipelining*), Arbitrationslogik (Bus)

Synchronisationsarten

- bei unteilbaren Betriebsmitteln greift Synchronisation **multilateral**
  - vorausgesetzt die folgenden beiden Bedingungen treffen zu:
    - i Betriebsmittelzugriffe durch Prozesse geschehen (quasi) **gleichzeitig** und
    - ii bewirken **widerstreitende Zustandsänderungen** des Betriebsmittels
  - Zugriffe auf gemeinsam benutzte Betriebsmittel sind zu koordinieren
    - was sich blockierend oder nichtblockierend auf die Prozesse auswirken kann
    - im blockierenden Fall wird das Betriebsmittel von einem Prozess exklusiv belegt, im nichtblockierenden Fall kann die Zustandsänderung scheitern
- bei konsumierbaren Betriebsmit. wirkt Synchronistaion **unilateral**
  - allgemein auch als logische oder bedingte Synchronisation bezeichnet:

    logisch – wie durch das Rollenspiel der involvierten Prozesse vorgegeben

    bedingt – wie durch eine Fallunterscheidung für eine Berechnung bestimmt
  - Benutzung eines vorübergehenden Betriebsmittels folgt einer Kausalität
    - nichtblockierend für Produzenten und blockierend für Konsumenten

- Prozesse, die gleichzeitig auftreten, überlappen einander zeitweilig
  - sie interagieren zwingend, wenn sie sich dann auch räumlich überlappen
  - dies bedeutet **Interferenz** (*interference*: Störung, Behinderung)…

Koordinationsmittel: Semaphor

- Primitiven [7] für Erwerb/Abgabe von Betriebsmitteln, wobei die Operationen folgende **intrinsische Eigenschaften** haben:
  - **P** Abk. für (Hol.) ***prolaag***; alias *down*, *wait* oder *acquire*
    - verringert[6] den Wert des Semaphors s um 1:
      - i genau dann, wenn der resultierende Wert nichtnegativ wäre [8, p. 29]
      - ii logisch uneingeschränkt [9, p. 345]
    - ist oder war der Wert vor dieser Aktion 0, blockiert der Prozess
      - er kommt auf eine mit dem Semaphor assoziierte Warteliste
  - **V** Abk. für (Hol.) ***verhoog***; alias *up*, *signal* oder *release*
    - erhöht[6] den Wert des Semaphors s um 1
    - ein ggf. am Semaphor blockierter Prozess wird wieder bereitgestellt
      - welcher Prozess von der Warteliste genommen wird, ist nicht spezifiziert
- beide Primitiven sind logisch oder physisch **unteilbare Operationen**, je nachdem, wie dies technisch sichergestellt ist [16]
- ursprünglich definiert als **binärer Semaphor** ($s = [0, 1]$), generalisiert als **allgemeiner Semaphor** ($s = [n, m], m > 0$ und $n \leq m$)

[6] Nicht zwingend durch Subtraktion oder Addition im arithmetischen Sinn.

Fallbeispiel

- als Tabelle implementierte **Universalzeigerliste** begrenzter Länge:

```
1  typedef struct table {
2      size_t get, put;
3      void *bay[TABLE_SIZE];
4  } table_t;
5
6  #define PUT(list,item) list.bay[list.put++ % TABLE_SIZE] = item
7  #define GET(list,item) item = list.bay[list.get++ % TABLE_SIZE]
```

- angenommen, mehrere Prozesse agieren mit `GET` oder `PUT` gleichzeitig auf derselben Datenstruktur `list` $\rightsquigarrow$ **kritischer Wettlauf**
  - `++` läuft Gefahr, falsch zu zählen (vgl. [15, S. 28])
  - `PUT` läuft Gefahr, Listeneinträge zu überschreiben[7]
  - `GET` läuft Gefahr, denselben Listeneintrag mehrfach zu liefern[7]
- **Simultanverarbeitung** lässt die beliebige zeitliche Überlappung von Prozessen zu, so dass **explizite Koordinierung** erforderlich wird

[7] Mehrere sich zeitlich überlappende Prozesse könnten denselben Wert aus der Indexvariablen (`put` bzw. `get`) lesen, bevor diese verändert wird.

Multilaterale Synchronisation

- **wechselseitiger Ausschluss** (*mutual exclusion*) sich sonst womöglich überlappender Ausführungen von `PUT` & `GET`: **binärer Semaphor**

Vorbelegung des Semaphors

```
/* critical section is free */
buffer_t buffer = {{1}};
```

```
1  typedef struct buffer {
2      semaphore_t lock;
3      table_t data;
4  } buffer_t;
5
6  inline void store(buffer_t *pool, void *item) {
7      P(&pool->lock);          /* enter critical section */
8      PUT(pool->data, item);   /* only one process at a time */
9      V(&pool->lock);          /* leave critical section */
10 }
11
12 inline void *fetch(buffer_t *pool) {
13     void *item;
14     P(&pool->lock);          /* enter critical section */
15     GET(pool->data, item);   /* only one process at a time */
16     V(&pool->lock);          /* leave critical section */
17     return item;
18 }
```

- ein **Unter-/Überlauf** der Universalzeigerliste bzw. des Puffers kann nicht ausgeschlossen werden $\rightsquigarrow$ **Programmierfehler**

Unilaterale Synchronisation

- **Reihenfolgenbildung** von Prozessen, die als Produzent (`stuff`) oder Konsument (`drain`) agieren: **allgemeiner Semaphor**

```
1  typedef struct stream {
2      semaphore_t free, full;
3      buffer_t data;
4  } stream_t;
5
6  void stuff(stream_t *pipe, void *item) {
7      P(&pipe->free);          /* prevent overflow */
8      store(&pipe->data, item);
9      V(&pipe->full);          /* signal consumable */
10 }
11
12 void *drain(stream_t *pipe) {
13     void *item;
14     P(&pipe->full);          /* prevent underflow */
15     item = fetch(&pipe->data);
16     V(&pipe->free);          /* signal space */
17     return item;
18 }
```

Vorbelegung der Semaphore

```
/* all table items available, no consumable
 * critical section is free */
stream_t stream = {{TABLE_SIZE}, {0}, {{1}}};
```

  - typisches Muster der Implementierung eines Klassikers — nicht nur in der Systemprogrammierung: **begrenzter Puffer** (*bounded buffer*)

Verwaltung

Implementierungsaspekte

Prozess ↔ Faden

process ↔ thread

- Gemeinsamkeit besteht darin, einen **Vorgang** auszudrücken, Unterschiede ergeben sich in der technischen Auslegung

Prozess
- Ausführung eines Programms (*locus of control* [6])
- seit Multics eng verknüpft mit „eigenem Adressraum“ [4]
- seit Thoth, als ***Team*** im selben Adressraum teilend [2]

Faden
- konkreter Strang, roter Faden (*thread*) in einem Programm
- **sequentieller Prozess** [1, S. 78] — ein „Thoth-Prozess“

andere
- Aufgabe (*task*), Arbeit (*job*) ∼ Handlung (wie Prozess)
- Faser (*fiber*), Fäserchen (*fibril*) ∼ Gewichtsklasse (wie Faden)

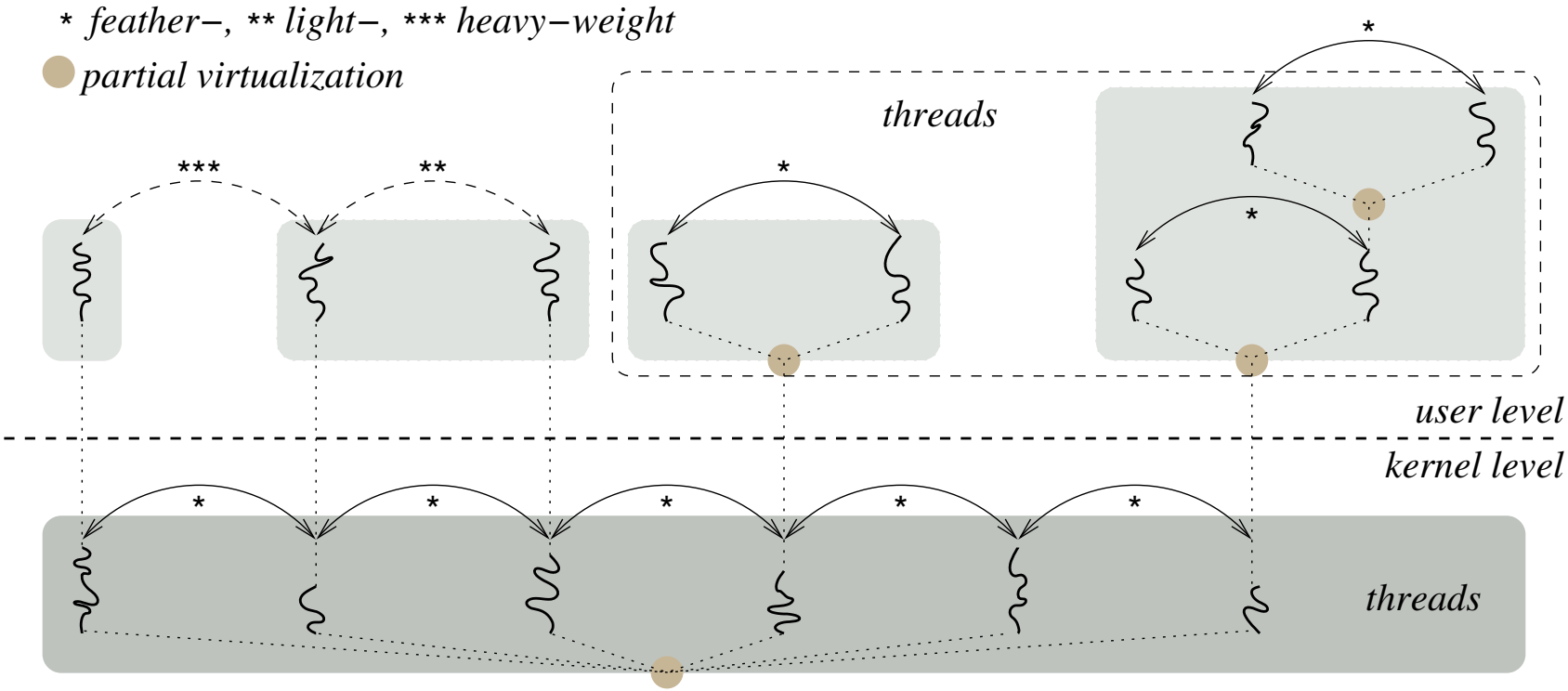
separation of concerns [10, S. 1]

Steht das „Was“ (ein ohne Zweifel bestehender Programmablauf) oder das „Wie“ (Art und Grad der Isolation) im Vordergrund der Diskussion?

- Informatikfolklore vermischt Programmablauf und Adressraumschutz
  - ein Prozess bewegt sich in dem Adressraum, den ein Programm definiert
  - das tut er aber unabhängig davon, ob dieser Adressraum geschützt ist
    - wenn überhaupt, dann ist daher sein Programmspeicher zu schützen…

Prozesse sind in einem Rechensystem verschiedenartig verankert

- unter oder auf der Maschinenprogrammebene
  - **unter**
    - ursprünglich, im Betriebssystem bzw. Kern (*kernel*)
    - Prozessinkarnation als Wurzel
    - partielle Virtualisierung des realen Prozessor(kern)s
    - ↪ „*kernel-level thread*“ in der Informatikfolklore
  - **auf**
    - optional, im Laufzeit- oder gar Anwendungssystem
    - Prozessinkarnation als Blatt oder innerer Knoten
    - partielle Virtualisierung eines abstrakten Prozessor(kern)s
    - ↪ „*user-level thread*“ in der Informatikfolklore
- der jew. Prozessor weiß nicht, dass er ggf. (part.) virtualisiert wird
  - ein „*user-level thread*“ ist ein in Zeit gemultiplexter „*kernel-level thread*“
  - einem „*kernel-level thread*“ sind seine „*user-level threads*“ unbewusst
- Betriebssysteme kennen nur ihre eigenen Prozessinkarnationen
  - ein „*kernel-level thread*“ entsteht durch Raum-/Zeitmultiplexen der CPU
  - hält ein „*kernel-level thread*“ inne, setzen seine „*user-level threads*“ aus



- Arten von **Prozesswechsel** zur partiellen Prozessorvirtualisierung:
  - `*` im selben (Anwendungs-/Kern-) Adressraum, ebenda fortsetzend
  - `**` im Kernadressraum, denselben Anwendungsadressraum teilend
  - `***` im Kernadressraum, im anderen Anwendungsadressraum landend

Verkörperung

- der **Prozesskontrollblock**[8] (*process control block*, PCB) bündelt alle zur partiellen Virtualisierung relevanten Attribute eines Prozesses
  - in dem (pro Prozess) typischerweise folgende Daten verbucht sind:
    - Adressraum, Speicherbelegung, Laufzeitkontext, …, Ressourcen allgemein
    - Verarbeitungszustand, Blockierungsgrund, Dringlichkeit, Termin
    - Name, Domäne, Zugehörigkeit, Befähigung, Zugriffsrechte, Identifikationen
  - als die zentrale **Informations- und Kontrollstruktur** im Betriebssystem
- pro Prozessor verwaltet das Betriebssystem einen **Prozesszeiger**, der die jeweils laufende Prozessinkarnation identifiziert
  - so, wie der Befehlszähler der CPU den laufenden Befehl adressiert, zeigt der Prozesszeiger des Betriebssystems auf den gegenwärtigen Prozess
  - beim Prozesswechsel (*dispatch*) wird der Prozesszeiger weitergeschaltet
- eine so beschriebene Prozessinkarnation wird systemweit eindeutig durch eine **Prozessidentifikation** (PID) repräsentiert
  - wobei „systemweit“ recht dehnbar ist und sich je nach Auslegung auf ein Betriebssystem, ein vernetztes System oder verteiltes System bezieht

[8] auch: **Prozessdeskriptor** (*process descriptor*, PD).

Gliederung

Resümee

…Trennung von Belangen trägt zum Verständnis bei

- in der Einführung zunächst prinzipielle **Begrifflichkeiten** erklärt
  - einen **Prozess** als „Programm in Ausführung“ definiert und damit die originale (klassische) Definition [5] übernommen
  - den Unterschied zur **Prozessinkarnation**/-verkörperung hervorgehoben
- darauf aufbauend wichtige **Grundlagen** zum Thema behandelt
  - partielle Virtualisierung und **Simultanverarbeitung**
  - **Betriebsmittel**, deren Klassifikation und Eigentümlichkeiten
  - Programm als Verarbeitungsvorschrift für eine Folge von **Aktionen**
  - **nichtsequentielles Programm**, das Aktionen für Parallelität spezifiziert
- verschiedene Aspekte der **Ausprägung** von Prozessen beleuchtet:

Planung
- implizite Koordinierung, **Einplanung** von Prozessen
- logische Verarbeitungszustände, **Einlastung**

Synchronisation
- explizite **Koordinierung** durch Programmanweisung
- binärer/allgemeiner **Semaphor**, Abgrenzung *Mutex*

Repräsentation
- **Verortung** der Prozesse im Rechensystem
- **Fäden** inner-/oberhalb der Maschinenprogrammebene

↪ Ressource: Prozesskontrollblock, -zeiger, -identifikation

Zusammenfassung

Bibliographie

Literaturverzeichnis (1)

[1] Bauer, F. L. ; Goos, G. :
Betriebssysteme.
In: *Informatik: Eine einführende Übersicht* Bd. 90.
Springer-Verlag, 1971, Kapitel 6.3, S. 76–92

[2] Cheriton, D. R. ; Malcolm, M. A. ; Melen, L. S.:
Thoth, a Portable Real-Time Operating System.
In: *Communications of the ACM* 22 (1979), Febr., Nr. 2, S. 105–115

[3] Corbató, F. J. ; Merwin-Daggett, M. ; Dalex, R. C.:
An Experimental Time-Sharing System.
In: *Proceedings of the AIEE-IRE '62 Spring Joint Computer Conference*, ACM, 1962, S. 335–344

[4] Daley, R. C. ; Dennis, J. B.:
Virtual Memory, Processes, and Sharing in MULTICS.
In: *Communications of the ACM* 11 (1968), Mai, Nr. 5, S. 306–312

Literaturverzeichnis (2)

[5] Denning, P. J.:
Third Generation Computer Systems.
In: *Computing Surveys* 3 (1971), Dez., Nr. 4, S. 175–216

[6] Dennis, J. B. ; Horn, E. C. V.:
Programming Semantics for Multiprogrammed Computations.
In: *Communications of the ACM* 9 (1966), März, Nr. 3, S. 143–155

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

Literaturverzeichnis (3)

[8] Dijkstra, E. W.:
Cooperating Sequential Processes / Technische Universiteit Eindhoven.
Eindhoven, The Netherlands, 1965 (EWD-123). – Forschungsbericht. –
(Reprinted in *Great Papers in Computer Science*, P. Laplante, ed., IEEE Press, New York, NY, 1996)

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

[10] Dijkstra, E. W.:
On the Role of Scientific Thought.
`http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD447.PDF`, Aug. 1974

Literaturverzeichnis (4)

[11] Herrtwich, R. G. ; Hommel, G. :
Kooperation und Konkurrenz — Nebenläufige, verteilte und echtzeitabhängige Programmsysteme.
Springer-Verlag, 1989. –
ISBN 3–540–51701–4

[12] Holt, R. C.:
On Deadlock in Computer Systems.
Ithaca, NY, USA, Cornell University, Diss., 1971

[13] Holt, R. C.:
Some Deadlock Properties of Computer Systems.
In: *ACM Computing Surveys* 4 (1972), Sept., Nr. 3, S. 179–196

Literaturverzeichnis (5)

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

[15] Kleinöder, J. ; Schröder-Preikschat, W. :
Betriebssystemmaschine.
In: Lehrstuhl Informatik 4 (Hrsg.): *Systemprogrammierung*.
FAU Erlangen-Nürnberg, 2015 (Vorlesungsfolien), Kapitel 5.3

[16] Schröder-Preikschat, W. :
Semaphore.
In: Lehrstuhl Informatik 4 (Hrsg.): *Concurrent Systems — Nebenläufige Systeme*.
FAU Erlangen-Nürnberg, 2014 (Vorlesungsfolien), Kapitel 7

Literaturverzeichnis (6)

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