Überblick

Prozesseinlastung
- Koroutine
- Programmfaden
- Prozessdeskriptor
- Zusammenfassung

Routinenartige Komponente eines Programms

Gleichberechtigtes Unterprogramm

Ko{existierende, operierende}-Routine

> *An autonomous program which communicates with adjacent modules as if they were input or output subroutines.*
>
> *[. . .]*
>
> *Coroutines are subroutines all at the same level, each acting as if it were the master program.* [48]

Koroutinen wurden erstmalig um 1963 in der von Conway entwickelten Architektur eines Fließbandübersetzers (engl. *pipeline compiler*) eingesetzt. Darin wurden Parser konzeptionell als Datenflussfließbänder zwischen Koroutinen aufgefasst. Die Koroutinen repräsentierten *first-class* Prozessoren wie z.B. Lexer, Parser und Codegenerator.

Autonomer Kontrollfluss eines Programms

Kontrollfaden (engl. *thread of control*, TOC)

Koroutinen konkretisieren Prozesse (implementieren Prozessinstanzen), sie repräsentieren die Aktivitätsträger von Programmen

1. ihre Ausführung beginnt immer an der letzten „Unterbrechungsstelle“
   - d.h., an der zuletzt die Kontrolle über den Prozessor abgegeben wurde
   - die Kontrollabgabe geschieht dabei grundsätzlich kooperativ (freiwillig)
2. zw. aufeinanderfolgenden Ausführungen ist ihr Zustand invariant
   - lokale Varbiablen (ggf. auch aktuelle Parameter) behalten ihre Werte
   - bei Abgabe der Prozessorkontrolle terminiert die Koroutine nicht

Koroutinen repräsentieren „zustandsbehaftete Prozeduren“, deren Aktivierungskontext während Phasen der Inaktivität erhalten bleibt

Koroutine deaktivieren $\mapsto$ Kontext „einfrieren“ (sichern)

Koroutine aktivieren $\mapsto$ Kontext „auftauen“ (wieder herstellen)

Programmiersprachliches Mittel zur Prozessorweitergabe

Multiplexen des Prozessors zwischen Prozessinstanzen

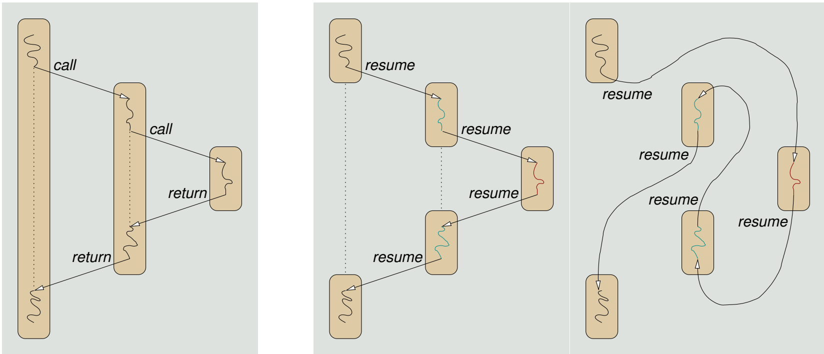
Koroutinen sind Prozeduren ähnlich, es fehlt jedoch die Aufrufhierarchie

Beim Verlassen einer Koroutine geht anders als beim Verlassen einer Prozedur die Kontrolle nicht automatisch an die aufrufende Routine zurück. Stattdessen wird mit einer *resume*-Anweisung beim Verlassen einer Koroutine explizit bestimmt, welche andere Koroutine als nächste ausgeführt wird. [55, S. 49]

Routine kein Kontrollflusswechsel bei Aktivierung/Deaktivierung
- asymmetrische Aktivierung, ungleichberechtigte Rollen

Koroutine Kontrollflusswechsel bei Aktivierung/Deaktivierung
- symmetrische Aktivierung, gleichberechtigte Rollen

Routine vs. Koroutine
Aufrufhierarchie vs. Nebenläufigkeit



Routinen | Koroutinen

- Aufrufhierarchie
- bestenfalls Aktivierungshierarchie
- Nebenläufigkeit inhärent im Konzept

Routine vs. Koroutine (Forts.)
Spezialfall eines generischen Konzeptes

Routine spezifischer als Koroutine
- ein einziger Einstiegspunkt
  - immer am Anfang
- ein einziger Ausstiegspunkt
  - kehrt nur einmal zurück
- Lebensdauer nach LIFO
  - *last in, first out*

Koroutine generischer als Routine
- ggf. mehrere Einstiegspunkte
  - dem letzten Ausstieg folgend
- ggf. mehrere Ausstiegspunkte
  - kehrt ggf. mehrmals zurück
- Lebensdauer nach LIAO
  - *last in, any out*

Routinen können durch Koroutinen implementiert werden [60]:

call $\mapsto$ *resume* der aufgerufenen Routine an ihrer Einsprungadresse
- Rücksprungkontext einfrieren, Aktivierungskontext erzeugen

return $\mapsto$ *resume* der aufrufenden Routine an ihrer Rücksprungadresse
- Aktivierungskontext zerstören, Rücksprungkontext auftauen

Buchführung über Fortsetzungspunkte
Fortsetzung (engl. *continuation*) einer Programmausführung

Fortsetzungspunkt ist die Stelle in einem Programm, an der die Wiederaufnahme (engl. *resumption*) der Programmausführung möglich ist
- eine Adresse im Textsegment, an der ein Kontrollfluss (freiwillig, erzwungenermaßen) unterbrochen wurde
- die Stelle, an der der CPU-Stoß der einen Koroutine endet und der CPU-Stoß einer anderen Koroutine beginnt

Koroutinen zu implementieren bedeutet, Programmfortsetzungen zu verbuchen und Aktivierungskontexte zu wechseln:
- Fortsetzungsadressen sind dynamisch festzulegen und zu speichern
  - z.B. wie im Falle der Rücksprungadresse einer Prozedur
- Laufzeitzustände sind zu sichern und wieder herzustellen
  - z.B. die Inhalte der von einer Koroutine benutzten Arbeitsregister

Kontrollflussverfolgung
Programmfortsetzung und Aktivierungskontext

```
void *foo, *bar;

void emuser (void *foo, void **bar) {
    *bar = foo;
}

int main () {
    emuser(foo, &bar);
}
```

```
emuser:
    movl  8(%esp),%edx
    movl  4(%esp),%eax
    movl  %eax,(%edx)
    ret

main:
    ...
    pushl $bar
    pushl foo
    call  emuser
    ...
```

Beispiel Stapelmaschine (x86, m68k):
- Rücksprungadresse und aktuelle Parameter liegen auf dem Laufzeitstapel (engl. *runtime stack*) des Prozessors
- Aktivierungskontexte im Stapel sichern bzw. daraus wieder herstellen

Kontrollflussverfolgung (Forts.)

Programmfortsetzung verbuchen und Aktivierungskontext wechseln

```
void *foo, *bar;

extern void resume (void *, void **);

int main () {
    resume(foo, &bar);
}
```

```
resume:
    movl  8(%esp),%edx
    movl  %esp,(%edx)
    movl  4(%esp),%esp
    ret
```

Trick ist es, jeder Koroutine einen eigenen Laufzeitstapel bereitzustellen und *resume* als Prozedur zu implementieren:

- wechseln des Aktivierungskontextes (von Koroutinen) bedeutet dann:
  - *resume* aufrufen, um die Rücksprungadresse gesichert zu bekommen
  - in *resume*: den aktuellen Wert des Stapelzeigers der CPU sichern und dem Stapelzeiger einen neuen Wert geben ⇝ Stapel umschalten
  - *resume* verlassen, um an einer anderen Rücksprungadresse fortzufahren
- ggf. muss *resume* den kompletten Prozessorstatus austauschen

Instanzenbildung

Erzeugung des initialen Aktivierungskontextes

Koroutinen sind *first-class* Objekte, die im Regelfall dynamisch zur Laufzeit angelegt werden

- Objektzustand ist der Aktivierungskontext einer Koroutine
  - ihre Fortsetzungsadresse und ggf. ihr kompletter Prozessorzustand
- Startadresse ist die Adresse einer Routine (*second-class* Objekt)

```
extern void replay ();

char* enable (char *stkp, void (*funp)()) {
    void (**sp)() = (void (**)())stkp;

    *--sp = funp;
    *--sp = replay;

    return (char *)sp;
}
```

```
replay:
    movl 0(%esp),%eax
    call *%eax
    jmp  replay
```

`enable` macht eine Routine zur Koroutine: legt (1) die Startadresse der Koroutine und (2) die Adresse der aufrufenden Routine auf den Stapel der Koroutine ab

`replay` bildet die Aufrufumgebung einer Koroutine: liest (1) die Startadresse der Koroutine vom Stapel und ruft (2) die Koroutine initial als Routine auf

Termination der Koroutine von selbst ist nicht möglich, da Wissen über die alternativ zu aktivierende Koroutine fehlt ⇝ Fäden, Fadenverwaltung

Koroutinen „mechanisieren" Programmfäden

Technisches Detail zum Multiplexen der CPU zwischen Prozessen

Mehrprogrammbetrieb basiert auf Koroutinen des Betriebssystems

- für jedes auszuführende Programm wird eine Koroutine bereitgestellt
  - ggf. für jeden Programmfaden ⇝ leichtgewichtiger Prozess
- ist eine Koroutine aktiv, so ist das ihr zugeordnete Programm aktiv
  - der durch die Koroutine implementierte Programmfaden ist aktiv
- um ein anderes Programm auszuführen, ist die Koroutine zu wechseln

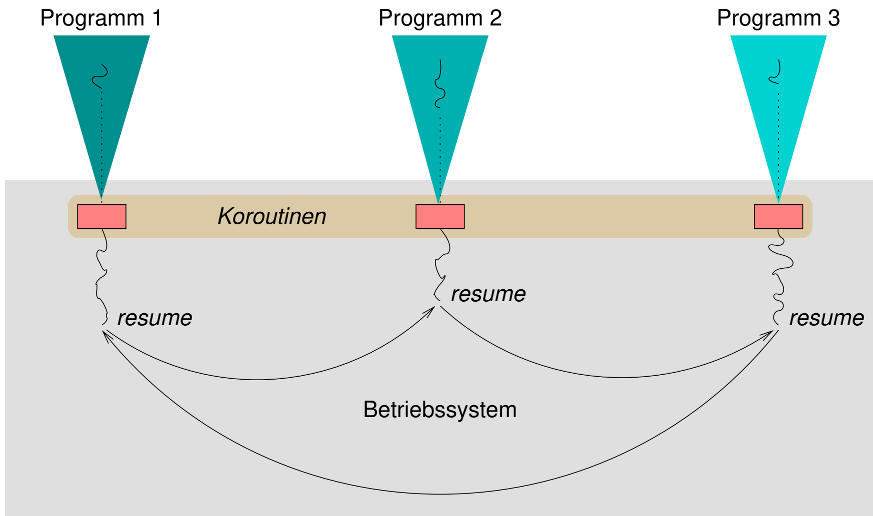
Koroutinen sind (in dem Modell) die Aktivitätsträger des Betriebssystems

- ihr Aktivierungskontext ist globale Variable des Betriebssystems
- für jede Prozessinstanz gibt es eine solche Betriebssystemvariable

☞ ein Betriebssystem ist Inbegriff für das nicht-sequentielle Programm

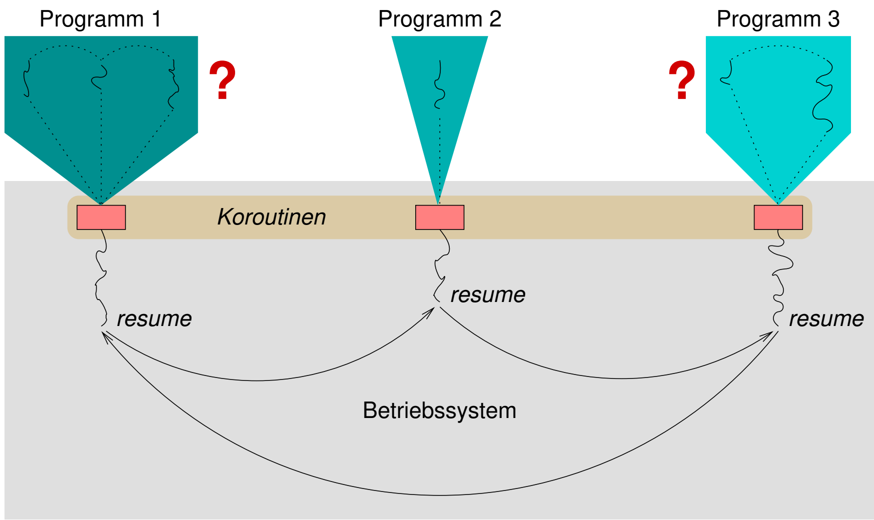
Verarbeitung sequentieller Programme

Koroutine als abstrakter Prozessor — Bestandteil des Betriebssystems



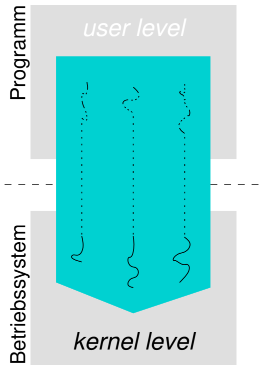
Verarbeitung nicht-sequentieller Programme

Multiplexen eines abstrakten Prozessors



Fäden der Kernebene

Klassische Variante von Mehrprozessbetrieb



Anwendungsprozesse sind durch Kernfäden (engl. *kernel-level threads*) implementiert

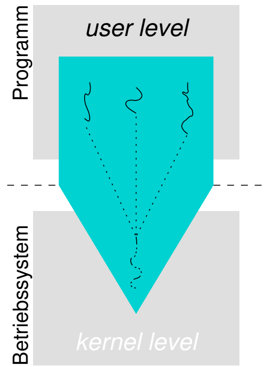
- egal, ob die Anwendungsprogramme ein- oder mehrfädig ausgelegt sind
  - jeder Anwendungsfaden ist Kernfaden
  - nicht jeder Kernfaden ist Anwendungsfaden
- Fäden sind keine Prozessinstanzen der Ebene $_3$
  - Maschinenprogramme verwenden Fäden, implementieren sie jedoch nicht selbst
- Ebene $_2$-Programme implementieren die Fäden

Einplanung und Einlastung der (leichtgewichtigen) Anwendungsprozesse sind Funktionen des Betriebssystem(kern)s

- Erzeugung, Koordination, Zerstörung von Fäden sind Systemaufrufe

Fäden der Benutzerebene

Ergänzung oder Alternative zu Kernfäden



Anwendungsprozesse sind durch Benutzerfäden (engl. *user-level threads*) implementiert

- virtuelle Prozessoren bewirken die Ausführung der (mehrfädigen) Anwendungsprogramme
  - 1 Prozessor trägt ggf. *N* Benutzerfäden
- der Kern stellt ggf. Planeransteuerungen (engl. *scheduler activations* [61]) bereit
  - zur Propagation von Einplanungsereignissen
- Fäden sind Prozessinstanzen der Ebene $_3$
  - implementiert durch Maschinenprogramme

Einplanung und Einlastung der (federgewichtigen) Anwendungsprozesse sind keine Funktionen des Betriebssystem(kern)s

- Erzeugung, Koordination, Zerstörung von Fäden als Unterprogramme

Prozesskontrollblock (engl. *process control block*, PCB)

Datenstruktur zur Verwaltung von Prozessinstanzen

Kopf eines Datenstrukturgeflechts zur Beschreibung einer Prozessinstanz und Steuerung eines Prozesses

- oft auch als Prozessdeskriptor (PD) bezeichnet
  - UNIX Jargon: *proc structure* (von „`struct proc`“)
- ein abstrakter Datentyp (ADT) des Betriebssystem(kern)s

Softwarebetriebsmittel zur Verwaltung von Programmausführungen

- jeder Faden wird durch eine Instanz vom Typ „PD“ repräsentiert

  Kernfaden Instanzenvariable des Betriebssystems
  Benutzerfaden Instanzenvariable des Anwendungsprogramms
- die Instanzenanzahl ist statisch (Systemkonstante) oder dynamisch

Objekt, das mit einer Prozessidentifikation (PID) assoziiert und für die gesamte Lebensdauer des betreffenden Prozesses gültig ist

- auch dann, wenn der Adressraum des Prozesses ausgelagert wurde

Aspekte der Prozessauslegung
Verwaltungseinheit einer Prozessinstanz

Dreh- und Angelpunkt, der alle prozessbezogenen Betriebsmittel bündelt

- Speicher- und, ggf., Adressraumbelegung †
  - Text-, Daten-, Stapelsegmente (*code, data, stack*)
- Dateideskriptoren und -köpfe (*inode*) †
  - {Zwischenspeicher,Puffer}deskriptoren, Datenblöcke
- Datei, die das vom Prozess ausgeführte Programm repräsentiert †

Datenstruktur, die Prozess- und Prozessorzustände beschreibt

- Laufzeitkontext des zugeordneten Programmfadens/Aktivitätsträgers
- gegenwärtiger Abfertigungszustand (*Scheduling*-Informationen) †
- anstehende Ereignisse bzw. erwartete Ereignisse †
- Benutzerzuordnung und -rechte †

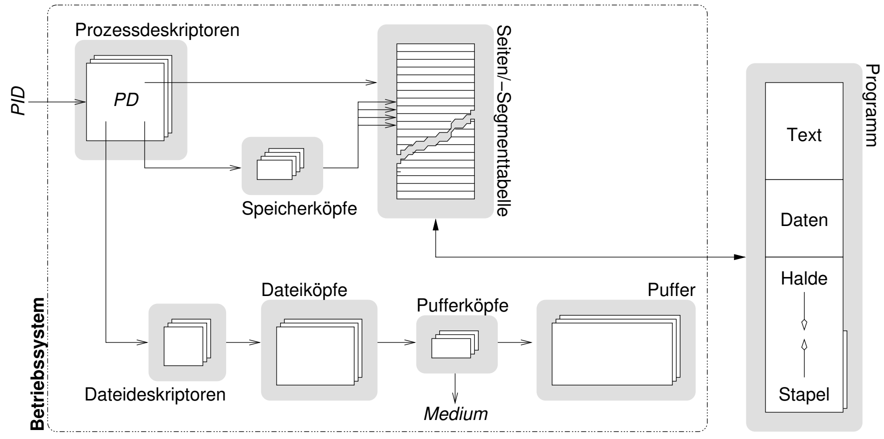
Aspekte der Prozessauslegung (Forts.)
Prozessinstanz vs. Betriebsart

† Auslegung des PD ist höchst abhängig von Betriebsart und -zweck:

1. Adressraumdeskriptoren sind nur notwendig in Anwendungsfällen, die eine Adressraumisolation erfordern
2. für ein Sensor-/Aktorsystem haben Dateideskriptoren/-köpfe wenig Bedeutung
3. in ROM-basierten Systemen durchlaufen die Prozesse oft immer nur ein und dasselbe Programm
4. in Einbenutzersystemen ist es wenig sinnvoll, prozessbezogene Benutzerrechte verwalten zu wollen
5. bei statischer Prozesseinplanung ist die Buchführung von Abfertigungszuständen verzichtbar
6. Ereignisverwaltung fällt nur an bei ereignisgesteuerten und/oder verdrängend arbeitenden Systemen

☞ Festlegung auf eine Ausprägung grenzt Einsatzgebiete unnötig aus

Generische Datenstruktur
Logische Sicht eines Geflechts abstrakter Datentypen



Einlastung $\rightsquigarrow$ Umsetzung von Einplanungsentscheidungen
Koroutinenwechsel $\models$ Fadenwechsel $\models$ Prozesswechsel

- Koroutinen konkretisieren Prozesse, implementieren Prozessinstanzen
  - die Gewichtsklasse von Prozessen spielt eine untergeordnete Rolle
    - feder-, leicht-, schwergewichtige Prozesse basieren auf Koroutinen
  - ihr Aktivierungskontext überdauert Phasen der Inaktivität
    - gesichert („eingefroren“) im jeder Koroutine eigenen Stapelspeicher
- Programmfäden (engl. *threads*) sind durch Koroutinen repräsentiert
  - unterschieden werden zwei Fadenarten, je nach Ebene der Abstraktion:
    - Kernfaden implementiert durch Programme der Befehlssatzebene
    - Benutzerfaden implementiert durch Programme der Maschinenebene
  - Einlastung eines Fadens führt einen Koroutinenwechsel nach sich
- der Prozessdeskriptor ist Objekt der Buchführung über Prozesse
  - Datenstruktur zur Verwaltung von Prozess- und Prozessorzuständen
    - insbesondere des Aktivierungskontextes der Koroutine eines Prozesses
  - Softwarebetriebsmittel zur Beschreibung einer Programmausführung