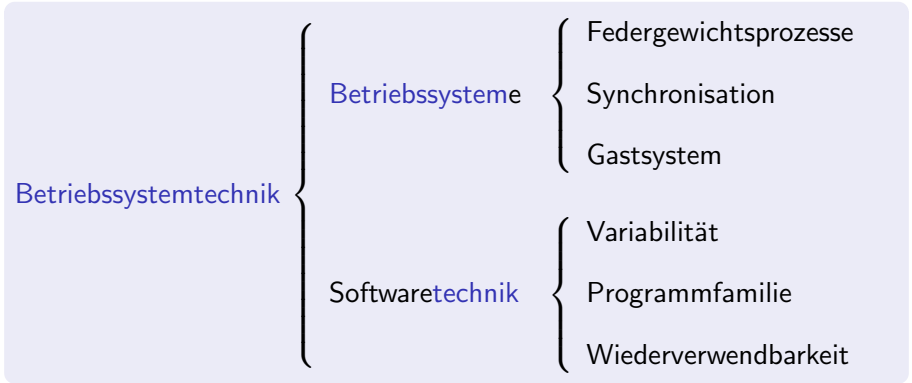
Betriebssystemtechnik

Einleitung

26. April 2010

Überblick

Hinter der Kulisse: BST in aller Kürze. . .



Softwaretechnik ↔ Betriebsysteme

Schichtenstruktur	1968	Dijkstra [1]	↪ THE
?	1969	Ritchie *et al.* [2]	↪ Unix
Botschaft	1970	Hansen [3]	↪ RC 4000
Abstraktion	1971	Liskov [4]	↪ Venus
C	1971	Ritchie *et al.* [5]	↩ Unix
Monitor	1972	Hansen [6]	↩ RC 4000
Geheimnisprinzip	1972	Parnas [7]	
Betriebssystemfamilie	1973	Parnas *et al.* [8]	
Objekt	1974	Wulf *et al.* [9]	↪ HYDRA
abstrakter Datentyp	1974	Liskov *et al.* [10]	↩ Venus
Benutztbeziehung	1975	Parnas [11]	
Parallelprogrammierung	1975	Hansen [12]	↩ RC 4000
Transparenz	1975	Parnas *et al.* [13]	
funktionale Hierarchie	1976	Habermann *et al.* [14]	↪ FAMOS
Programmfamilie	1976	Parnas [15]	

Fallstudie: Fadenbaugruppe (engl. *threads package*)

Operationsprinzip [16, 17]

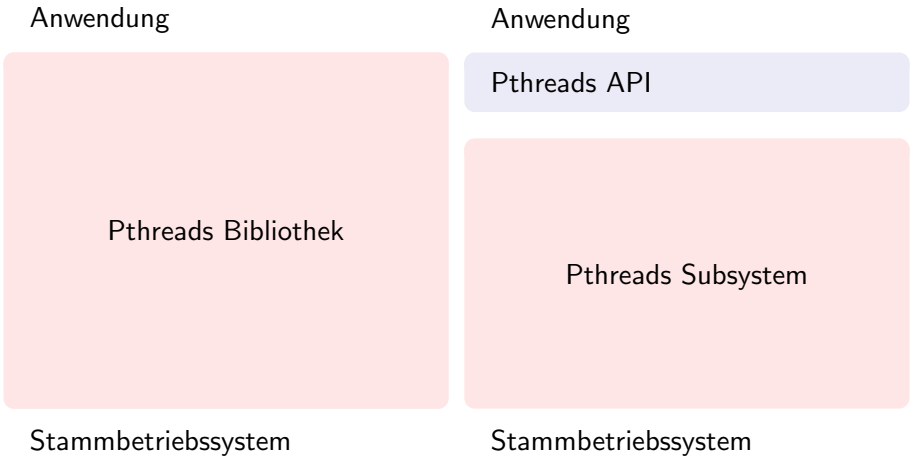
- statisch konfigurierbar und ggf. dynamisch änderbar in Abhängigkeit vom jeweiligen Anwendungsfall auslegen

Informationsstruktur $\mapsto$ Fäden, Fortsetzung
- invariant zu haltender Prozessorstatus bei Ausführungspausen
  - kontextabhängig $\rightsquigarrow$ mehrere Gewichtsklassen
  - min. Befehlszähler (PC), max. kompletter Registersatz
- problemspezifisch ausgeprägte Aktivitätsträger(spezialisierungen)
  - Makro, Prozedur, Methode, Koroutine, Faden
  - gemeinsam bzw. allein benutzter Laufzeitstapel

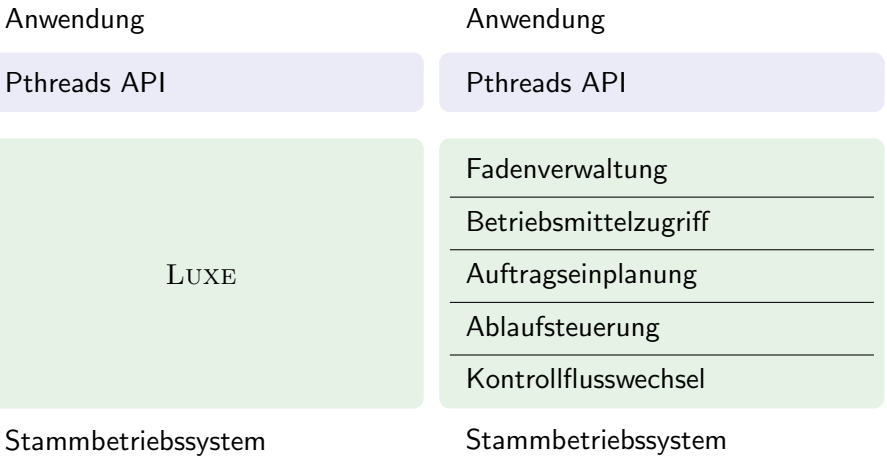
Kontrollstruktur $\mapsto$ Einplanung, Koordinierung
- nicht verdrängend, ggf. aber unterbrechend arbeitend
- verdrängend arbeitend: verzögert, unverzögert

☞ Fadenfamilie eigentümlicher (nicht-) funktionaler Eigenschaften

Pthreads [18]



Pthreads *de* LUXE



Nomen est omen — Der Name ist ein Zeichen...

Bausatzausstattung folgerichtiger[1] Betriebssystemanbauteile $\models$ LUCSE

logical (dt. folgerichtig)
unit (dt. Anbauteil)
construction-set (dt. Bausatz)
environment (dt. Ausstattung)

(CS $\mapsto$ X) $\rightsquigarrow$ LUXE

- in bester Tradition mit Multics und Unix: (Multi $\mapsto$ Uni) $\cup$ (cs $\mapsto$ x)

[1] Als Synonym für „durchdacht“.

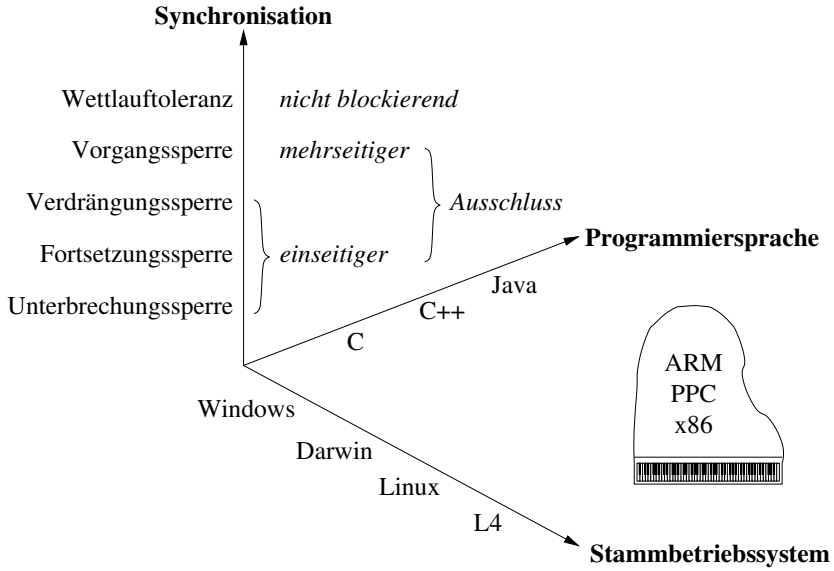
Konkurrenz — als Triebfeder

Wettbewerb zwischen den Arbeitsgruppen in Hinblick auf das Ziel, das beste Fädenangebot (engl. *threads package*) zu liefern:

- Betriebsmittelbedarf
  - Laufzeit
  - Speicher
  - ggf. Energie
- Skalierbarkeit
  - n-fädiger Betrieb
  - n-kerniger Betrieb
  - $n = 1, 2, \ldots, N$

☞ anwendungsfallspezifische, spezialisierte Subsystemvarianten

Dimensionen untersuchter Variabilität



Fadenfamilie ↔ Betriebssystem

Betriebssystem

Parallelarbeit { nebeneinander / übereinander } verschränkter gleichzeitiger Prozesse

nebeneinander verschränkt ↦ synchroner Kontrollflusswechsel
- „freiwillige“ Prozessorübergabe (Pausierung/Blockierung)
  - Verdrängung ist der Zwang zur Pausierung !!!
- direkt/indirekt durch den laufenden Programmfaden

übereinander verschränkt ↦ asynchrone Programmunterbrechung
- „unfreiwillige“ Prozessorübergabe (Unterbrechungsbehandlung)
  - kann Verdrängung zur Folge haben !!!
- durch den vom *Interrupt* betroffenen Programmfaden

Fadenfamilie

☞ bestimmte Schichten eines Betriebssystems umfassender Komplex

Programmfäden durchziehen Betriebssysteme

Simultanbetrieb [17] geht (schon immer) einher mit der Fähigkeit, *im* Betriebssystem gleichzeitige Prozesse [19, 20] stattfinden zu lassen
- jeder Prozess ist als eigenständige Kontrollflussinstanz realisiert[2]
- so können im Betriebssystem viele Programmfäden zugleich agieren
  - insb. sich gegenseitig verdrängen, auf Betriebsmittelzuteilung warten

Mehrfädigkeit eines Betriebssystems muss deshalb aber noch längst nicht an der Systemaufrufschnittstelle sichtbar sein
- erst Thoth [25] hat Fäden in die Anwendungsprogramme gebracht
- Standardbetriebssysteme haben (zu) lange darauf warten lassen

☞ Fadenkonzepte gibt es heute inner-/außerhalb von Betriebssystemen

[2] Ausnahmen wie z.B. TinyOS [21], Contiki [22], BCC-Varianten von OSEK [23] oder Betriebssysteme mit stapelbasierter Einplanung [24] bestätigen die Regel.

Programmfäden und Betriebssysteme: Orthogonalität

Stammbetriebssystem $\mapsto$ bodenständig (engl. *native*)

- Programm der Ebene $_2$, auf Befehlssatzebene aufsetzend
  - wobei Ebene $_2$ eine reale oder virtuelle Maschine darstellt
- implementiert die Maschinenprogrammebene, Ebene $_3$
  - virtuelle Maschine, interpretiert Maschinenprogramme partiell [17]
- kann Fäden auf Kernebene (engl. *kernel-level threads*) bieten

Gastbetriebssystem $\mapsto$ nicht bodenständig

- Programm der Ebene $_3$, auf Maschinenprogrammebene aufsetzend
  - stellt damit selbst ein (gewöhnliches) Maschinenprogramm dar
- tritt als Bibliotheksbetriebssystem in Erscheinung
  - virtuelle Maschine, privilegierte Operationen mittels Systemaufrufe
- bietet ggf. Fäden auf Benutzerebene (engl. *user-level threads*)

Programmfaden $\mapsto$ Konzept einer virtuellen Maschine

- existiert auf Benutzerebene (eines BS) ebenso wie auf Kernebene

Fallbeispiel Stammbetriebssystem [17]

Schicht	Funktion	Konzepte
12	Programmverwaltung	Text, Daten, Überlagerung
11	Dateiverwaltung	Dateisystem; Verzeichnis, Verknüpfung
10	Prozessverwaltung	Aktivitätsträger, Kontext, Stapel
9	Adressraumverwaltung	Arbeitsspeicher, Segment, Seite
8	Informationsaustausch	Paket, Nachricht, Kanal, Portal
7	Geräteprogrammierung	Kern; Signal, Zeichen, Block, Datenstrom
6	Platzanweisung	Hauptspeicher, Fragment, Seitenrahmen
5	Zugriffskontrolle	Subjekt, Objekt, Domäne, Befähigung
4	Betriebsmittelzugriff	Verdrängungs-/Vorgangssperre
3	Auftragseinplanung	Ereignis, Priorität, Zeitscheibe, Energie
2	Ablaufsteuerung	Unterbrechungs-/Fortsetzungssperre, Wettlauftoleranz
1	Kontrollflusswechsel	Koroutine, Unterbrechung, Fortsetzung
0	Stammprozessorabstraktion	Stammsystem
−1	Peripherie	MMU, (A)PIC, DMA, UART, ATA, SCSI, USB, ...
−2	Zentraleinheit	ARM, AVR, PowerPC, SPARC, x86, ...

Fallbeispiel Gastbetriebssystem [17]

Schicht	Funktion	Konzepte
12	Programmverwaltung	Text, Daten, Überlagerung
11	Dateiverwaltung	Dateisystem; Verzeichnis, Verknüpfung
10	Prozessverwaltung	Aktivitätsträger, Kontext, Stapel
9	Adressraumverwaltung	Arbeitsspeicher, Segment, Seite
8	Informationsaustausch	Paket, Nachricht, Kanal, Portal
7	Geräteprogrammierung	Kern; Signal, Zeichen, Block, Datenstrom
6	Platzanweisung	Hauptspeicher, Fragment, Seitenrahmen
5	Zugriffskontrolle	Subjekt, Objekt, Domäne, Befähigung
4	Betriebsmittelzugriff	Verdrängungs-/Vorgangssperre
3	Auftragseinplanung	Ereignis, Priorität, Zeitscheibe, Energie
2	Ablaufsteuerung	Unterbrechungs-/Fortsetzungssperre, Wettlauftoleranz
1	Kontrollflusswechsel	Koroutine, Unterbrechung, Fortsetzung
0	Stammprozessorabstraktion	Gastsystem
−1	Betriebssystem	Darwin, L4, Linux, Solaris, Windows (Cygwin), ...
−2	Zentraleinheit	ARM, AVR, PowerPC, SPARC, x86, ...

Fallbeispiel Gastbetriebssystem *de* LUXE

Schicht	Funktion	Konzepte
12	*leer*	*keine*
11	*leer*	*keine*
10	Prozessverwaltung	Aktivitätsträger, Kontext, Stapel
9	*leer*	*keine*
8	Informationsaustausch	Paket, Nachricht; `socket(2)`
7	Geräteprogrammierung	Kern, `sched_set_affinity(2)`; `signal(3)`
6	Platzanweisung	`sbrk(2)`
5	*leer*	*keine*
4	Betriebsmittelzugriff	Verdrängungs-/Vorgangssperre
3	Auftragseinplanung	Ereignis, Priorität, Zeitscheibe
2	Ablaufsteuerung	Unterbrechungs-/Fortsetzungssperre, Wettlauftoleranz
1	Kontrollflusswechsel	Koroutine, Unterbrechung, Fortsetzung
0	Stammprozessorabstraktion	Gastsystem
−1	Betriebssystem	Darwin, L4, Linux, Windows (Cygwin)
−2	Zentraleinheit	PowerPC, x86

Schicht (engl. *layer*, *tier*) als logisches Gebilde

Anwendungen legen die in einem Betriebssystem wesentlich/fallweise realisierten funktionalen und nichtfunktionalen Eigenschaften fest

- Instanzenbildung (Existenz) einer Schicht ist niemals Dogma
- Schichtenstrukturen definieren sich über eine Benutztbeziehung [11]
- nicht benutzte (gebrauchte) Funktionen sind nicht real vorhanden

Universalbetriebssysteme sind typisch für eine komplett umgesetzte Schichtenstruktur — so sie überhaupt schichtenstrukturiert sind

- wenngleich auch mit Optionen in den Diensten einzelner Schichten

Spezialbetriebssysteme sind typisch für eine nur teilweise umgesetzte Schichtenstruktur — sofern eine solche definiert ist

- ausgewählte Schichten sind komplett „leer“ d.h. nicht realisiert

☞ der Systementwurf ist komplett, die Umsetzung nicht unbedingt

Fadenfamilie ↔ Betriebssystem *de*LUXE

Schicht. . . Ausführung von Programmfäden — *wesentlich*

- 10 Prozessverwaltung
- 3 Auftragseinplanung
- 1 Kontrollflusswechsel

Schicht. . . Koordinierung gleichzeitiger Prozesse — *fallweise*

- 4 Betriebsmittelzugriff
- 2 Ablaufsteuerung

Schicht. . . Anschluss an die Außenwelt — *wesentlich*

- 7 Geräteprogrammierung
- 0 Stammprozessorabstraktion

Fadenfamilie ↔ Betriebssystem *de*LUXE (Forts.)

Schicht. . . Stammsystemspezifisch — *bodenständig*

- 10 Prozessverwaltung
- 4 Betriebsmittelzugriff
- 3 Auftragseinplanung
- 2 Ablaufsteuerung
  - Unterbrechungssperre/Wettlauftoleranz: CPU abhängig
- 1 Kontrollflusswechsel
  - Koroutine/Unterbrechung: CPU abhängig

Schicht. . . Gastsystemspezifisch — *nicht bodenständig*

- 7 Geräteprogrammierung
- 0 Stammprozessorabstraktion

Fadenfamilie ↔ Variabilität

Funktional *gleiche*, nicht-funktional *ungleiche* Implementierungen

- Koroutine: Varianten zur Stapelanbindung
  - von einer Koroutine allein oder mehreren gemeinsam benutzter Stapel
- Programmfaden: Varianten der Zustandssicherung
  - alle, nur die nicht-flüchtigen oder gar keine Arbeitsregister sichern
- Sperre: Varianten in Bezug auf die Zielbereiche
  - Unterbrechungen, Fortsetzungen, Verdrängungen, Vorgänge sperren
- kritischer Abschnitt: Varianten von Schutzverfahren
  - Ereignis sperrende/zulassende Synchronisation gleichzeitiger Prozesse

Querschneidende Belange

- gemeinsam/allein von Koroutinen bzw. Fäden benutzter Stapel
- verdrängende und verzögerte/unverzögerte Fadeneinplanung
- Wettstreitigkeiten gegenüber tolerante/intolerate Koordinierung

Resümee

Einordnung $\mapsto$ *logical unit construction-set environment* $\models$ LUXE

- Untersuchung von Variabiliät in der Systemsoftware
- Synchronisation, Stammbetriebssystem, ggf. Programmiersprache

Schichtenstruktur $\mapsto$ Fadenfamilie im Betriebssystemkontext

- Stammbetriebssystem $\models$ Gastbetriebssystem $\supset$ LUXE
- Schichten 1–4, 10: Fädenangebot im Stamm-/Gastbetriebssystem

Variantenvielfalt $\mapsto$ Betriebsart bzw. Architektur eines Rechensystems

- fallweise, Reaktion darauf ist von querschneidendem Belang

  Verdrängung $\Rightarrow$ Koordinierung durch Schicht $_4$ $\supset$ Schicht $_2$

  Unterbrechung $\Rightarrow$ Koordinierung durch Schicht $_2$
- wesentlich, wenngleich anwendungsfallabhängige Funktionen
  - Prozessverwaltung, Auftragseinplanung, Kontrollflusswechsel
  - Geräteprogrammierung, Stammprozessorabstraktion

Literaturverzeichnis

[1] Edsger Wybe Dijkstra.
The structure of the THE-multiprogramming system.
Communications of the ACM, 11(5):341–346, May 1968.

[2] Dennis MacAlistair Ritchie and Ken Thompson.
The Unix time-sharing system.
Communications of the ACM, 17(7):365–370, July 1974.

[3] Peer Brinch Hansen.
The nucleus of a multiprogramming system.
Communications of the ACM, 13(4):238–241/250, April 1970.

Literaturverzeichnis (Forts.)

[4] Barbara H. Liskov.
The design of the Venus operating system.
In *Proceedings of the 3rd ACM Symposium on Operating Systems Principles (SOSP '71)*, volume 6 of *ACM SIGOPS Operating Systems Review*, pages 11–16, New York, NY, USA, June 1972. ACM Press.

[5] Dennis MacAlistair Ritchie, Ken Thompson, Steven C. Johnson, and Michael E. Lesk.
The C programming language.
Bell System Technical Journal, 57(6), July/August 1978.

[6] Peer Brinch Hansen.
Structured multiprogramming.
Communications of the ACM, 15(7):574–578, July 1972.

Literaturverzeichnis (Forts.)

[7] David Lorge Parnas.
On the criteria to be used in decomposing systems into modules.
Communications of the ACM, pages 1053–1058, December 1972.

[8] David Lorge Parnas and William Robert Price.
The design of the virtual memory aspects of a virtual machine.
In Ugo O. Gagliardi, Louise Bolliet, and Robert P. Goldberg, editors, *Proceedings of the Workshop on Virtual Computer Systems, March 26–27, Cambridge, MA, USA*, pages 184–190, 1973.

[9] William Allan Wulf, Ellis S. Cohen, William M. Corwin, Anita K. Jones, Roy Levin, Charles Pierson, and Fred J. Pollack.
HYDRA: The kernel of a multiprocessor operating system.
Communications of the ACM, 17(6):337–345, June 1974.

Literaturverzeichnis (Forts.)

[10] Barbara H. Liskov and Stephen N. Zilles.
Programming with abstract data types.
ACM SIGPLAN Notices, 9(4):50–59, April 1974.

[11] David Lorge Parnas.
Some hypothesis about the "uses" hierarchy for operating systems.
Technical report, TH Darmstadt, Fachbereich Informatik, 1976.

[12] Per Brinch Hansen.
The programming language Concurrent Pascal.
IEEE Transactions on Software Engineering, 1(2):199–207, June 1975.

[13] David Lorge Parnas and Daniel P. Siewiorek.
Use of the concept of transparency in the design of hierarchically structured systems.
Communications of the ACM, 18(7):401–408, July 1975.

Literaturverzeichnis (Forts.)

[14] Arie Nicolaas Habermann, Lawrence Flon, and Lee W. Cooprider.
Modularization and hierarchy in a family of operating systems.
Communications of the ACM, 19(5):266–272, 1976.

[15] David Lorge Parnas.
On the design and development of program families.
IEEE Transactions on Software Engineering, SE-2(1):1–9, March 1976.

[16] Wolfgang K. Giloi.
Rechnerarchitektur.
Springer-Verlag, Berlin Heidelberg u.a., 1993.

[17] Wolfgang Schröder-Preikschat.
Betriebssysteme — Grundlagen, Entwurf, Implementierung.
Springer, 2009.

Literaturverzeichnis (Forts.)

[18] Institute of Electrical and Electronics Engineers, Inc.
IEEE Std 1003.1c-1995 thread extensions.
New York, NY, USA, 1995.

[19] Per Brinch Hansen.
The Solo operating system: A Concurrent Pascal program.
Software—Practice and Experience, 6(2):141–149, April–June 1976.

[20] Per Brinch Hansen.
The Architecture of Concurrent Programs.
Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

Literaturverzeichnis (Forts.)

[21] Jason L. Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister.
System architecture directions for networked sensors.
In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX), Cambridge, MA, USA, November 12–15, 2000*, pages 93–104, New York, NY, USA, 2000. ACM Press.

[22] Adam Dunkels, Björn Grönvall, and Thiemo Voigt.
Contiki — a lightweight and flexible operating system for tiny networked sensors.
In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, November 2004.

[23] OSEK/VDX group homepage.
`http://www.osek-vdx.org/`.

Literaturverzeichnis (Forts.)

[24] Theodore P. Baker.
Stack-based scheduling of realtime processes.
Real-Time Systems, 3(1):67–99, 1991.

[25] David Ross Cheriton.
Multi-Process Structuring and the Thoth Operating System.
PhD thesis, University of Waterloo, Ontario, Canada, 1978.