Betriebssysteme (BS)

VL 8.4 – Koroutinen und Fäden – Multitasking

Volkmar Sieh / **Daniel Lohmann**

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen Nürnberg

WS 20 – 14. Dezember 2020

https://www4.cs.fau.de/Lehre/WS20/V_BS

Agenda

- Motivation
- Grundbegriffe
- Implementierung
- Ausblick
  - Koroutinen als Hilfsmittel für das BS
  - Mehrfädrigkeit
- Zusammenfassung
- Referenzen

Ausblick: Betriebssystemfäden

- Koroutinen sind (eigentlich) ein Sprachkonzept
  - Multitasking auf Sprachebene
  - wir haben es hier für C/C++ (bzw. ein ABI) „nachgerüstet”
  - Kontextwechsel erfordert keine Systemprivilegien!

    $\rightsquigarrow$ muss also nicht zwingend im BS-Kern erfolgen
- Vorraussetzung für echtes Multitasking: Kooperation
  - Anwendungen müssen als Koroutinen implementiert sein
  - Anwendungen müssen sich gegenseitig kennen
  - Anwendungen müssen sich gegenseitig aktivieren
  - …

Ausblick: Betriebssystemfäden

- Koroutinen sind (eigentlich) ein Sprachkonzept
  - Multitasking auf Sprachebene
  - wir haben es hier für C/C++ (bzw. ein ABI) „nachgerüstet”
  - Kontextwechsel erfordert keine Systemprivilegien!
    $\leadsto$ muss also nicht zwingend im BS-Kern erfolgen
- Vorraussetzung für echtes Multitasking: Kooperation
  - Anwendungen müssen als Koroutinen implementiert sein
  - Anwendungen müssen sich gegenseitig kennen
  - Anwendungen müssen sich gegenseitig aktivieren
  - …

Problem

Für uneingeschränkten Mehrprogramm-Betrieb ist das unrealistisch.

Ausblick: Betriebssystemfäden

Alternative: „Kooperationsfähigkeit” als Aufgabe des Betriebssystems auffassen

Ansatz: Anwendungen „unbemerkt” als eigenständige Fäden ausführen

- BS sorgt für die Erzeugung der Koroutinen-Kontrollflüsse
  - jede Anwendung wird als Routine aus einer BS-Koroutine aufgerufen
  - $\leadsto$ indirekt läuft jede Anwendung als Koroutine
- BS sorgt für die Suspendierung laufender Koroutinen-Kontrollflüsse
  - so dass Anwendungen nicht kooperieren müssen
  - erfordert einen Verdrängungsmechanismus
- BS sorgt für die Auswahl des nächsten Koroutinen-Kontrollflusses
  - so dass Anwendungen sich nicht gegenseitig kennen müssen
  - erfordert einen Scheduler

Ausblick: Betriebssystemfäden

Alternative: „Kooperationsfähigkeit” als Aufgabe des Betriebssystems auffassen

Ansatz: Anwendungen „unbemerkt” als eigenständige Fäden ausführen

- BS sorgt für die Erzeugung der Koroutinen-Kontrollflüsse
  - jede Anwendung wird als Routine aus einer BS-Koroutine aufgerufen
  - $\rightsquigarrow$ indirekt läuft jede Anwendung als Koroutine
- BS sorgt für die Suspendierung laufender Koroutinen-Kontrollflüsse
  - so dass Anwendungen nicht kooperieren müssen
  - erfordert einen Verdrängungsmechanismus
- BS sorgt für die Auswahl des nächsten Koroutinen-Kontrollflusses
  - so dass Anwendungen sich nicht gegenseitig kennen müssen
  - erfordert einen Scheduler

Mehr dazu in der nächsten Vorlesung!

Agenda

Motivation
Grundbegriffe
Implementierung
Ausblick
Zusammenfassung
Referenzen

Zusammenfassung: Quasi-Parallelität

- Ziel war die Ermöglichung von „Quasi-Parallelität”
  - Verschränkte Ausführung von Funktionen
    - Suspendierung und Reaktivierung von Funktions-Ausführungen
    - Begriff der Fortsetzung
- Routinen $\mapsto$ asymmetrisches Fortsetzungsmodell
  - Ausführung nach LIFO (und damit nicht „quasi-parallel”)
  - CPU und Übersetzer stellen Elementaroperationen bereit
- Koroutinen $\mapsto$ symmetrisches Fortsetzungsmodell
  - Ausführung in beliebiger Reihenfolge
  - erfordert eigenen Kontext: minimal PC, i. a. auch Register und Stapel
  - CPU und Übersetzer stellen i. a. keine Elementaroperationen bereit
- Fäden $\mapsto$ vom BS verwaltete Koroutinen