# Betriebssysteme Test 2

Jan Fässler

2. Semester (FS 2012)

# 1 Parallelität - Nebenläufigkeit

## 1.1 Einleitung

Falls ein Rechner mehrere Prozesse parallel verarbeiten kann (Multi-Tasking), so führt er jeden Prozess als separate Aktivität aus. Diese Prozesse konnen unabhängig voneinander ablaufen (Nebenläufigkeit), müssen jedoch synchronisiert werden, da bestimmte Ressourcen (insbesondere die CPU) gemeinsam genutzt werden. Stehen mehrere CPUs zur Verfügung, konnen Prozesse parallel ablaufen, jedoch weiterhin mit Synchronisation und Wartezuständen bei gemeinsam genutzten Ressourcen. Werden Threads innerhalb eines Prozess-Adressraums eingesetzt, gilt diese Aussage sinngemäss auch für jeden Thread innerhalb eines Prozesskontexts.

## 1.2 Der Sheduler

Der Scheduler teilt Prozessen Rechenzeit zu und sequentialisiert damit die nebenläufige Ausführung von Prozessen oder Threads auf einer oder mehreren CPUs. Bei genügender Performance des Systems geschieht dies so rasch, dass für den Benutzer der Eindruck der Parallelität entsteht. Neben der Ressourcenauslastung beeinflusst die Scheduling-Strategie den Grad der Quasi-Parallelität.

## 1.3 Pre-emption

Prozesse konnen in Unix zu fast jeder Zeit unterbrochen werden:

- freiwillig durch Systemaufruf mit Wartezustand oder Abgabe der CPU (sleep, wait)
- ungeplant durch den Scheduler auf Basis von Priorisierung und Ressourcenverbrauch oder nach Ablauf der Zeitscheibe
- ungeplant durch asynchronen Events (z.B. Interrupts, die durch den gerade laufenden Prozess behandelt werden müssen)

Ein Prozess muss seinen Ausführungskontext unter- brechen, abspeichern, zum neuen Kontext wechseln, im neuen Kontext ablaufen, den alten Kontext wiederherstellen und neu starten konnen (Aufgabe des Kernels). Behandelt der Prozess im Programmcode Ausnahmen selbst (z.B. Signale), muss der Programmierer selbst für die Konsistenz von Variablen und Zuständen sorgen.

## 1.4 Synchronisation

### 1.4.1 Das Problem der Synchronisierung

5 Personen sitzen am Tisch, zwischen den Personen liegen 5 Stäbchen. Zum Essen benotigt jede Person 2 Stäbchen.

**Warten** Griff ins Leere

**Synchronisation** Gleichzeitiger Zugriff

**Deadlock** jede Person nimmt das rechte Stäbchen und wartet auf das linke

**Starvation** alle Personen sollen in sinnvolle Frist essen

### 1.4.2 Synchronisation

Zwischen Kernel und Prozessen:

- Signalisierung
- Schlaf-/Wartezustand des Prozesses
- Aufwecken & Scheduling

Zwischen Prozessen:

- Einseitige Synchronisation
- Mehrseitige Synchronisation
- Gegenseitiger Ausschluss aus kritischen Abschnitten

## 1.5 Synchronisationsmittel

### 1.5.1 Benachrichtigung & Warten

- Prozess wartet aktiv auf das Eintreffen einer Nachricht (busy waiting)
- Prozess wartet(schläft) auf das Eintreffen eines Wecksignals (typisiert) der Signal-Mechanismus von Unix weckt dann (via den Kernel / Scheduler) alle auf dieses Ereignis wartenden Prozesse

### 1.5.2 Locks/Schlossvariablen

- Im Dateisystem (Lockfile, Lock-Bits im Superblock)
- Im Speicher (Lock Bits, gemeinsame Variablen)
- Modell: der Prozess prüft zyklisch auf Zustandsänderung des Bits / der Variablen und modifiziert das Bit bzw. die Variable, falls erlaubt.
- Problem: meist keine atomare Operation wegen jederzeitiger Unterbrechbarkeit, es kann daher zu Problemen kommen, wenn die Implementation nicht durch unteilbare CPU-Instruktionen unterstützt wird.

### 1.5.3 Ereigniszähler

- Variante eines Locks, welches durch eine Zählervariable (z.B. in einem Shared Memory Segment) implementiert wird.
- Sinnvoll, wenn mehrere Prozesse zu synchronisieren sind, z.B. maximale Anzahl quasiparalleler Leseprozesse auf einer Datenbank wegen Performance-Garantien.
- Gleiches Problem der unterbrechbaren, nicht atomaren Operation wie bei Locks, benotigt daher ähnliche Schutzmechanismen.

### 1.5.4 Petri-Netze

Modellierung von nebenläufigen Systemen und ihrer Synchronisation

- Stellen = Kreis
- Transitionen = Rechteck
- Marken = Punkt
- Schaltregeln = alle vorausgehenden Stellen enthalten mind. 1 Marke, alle nachfolgenden Stellen erhalten eine Marke

### 1.5.5 Dijkstra Semaphore

- Modul/Kapsel mit geschützter Statusvariablen
- Bereitstellung von Operationen für:
  - Initialisierung
  - Eintritt/Signalisierung (P)
  - Austritt/Freigabe (V)
  - Deallokation

- Typen:
  - Binär
  - Zähler
  - Set / Array
- Implementierung: atomare CPU-Instruktion oder kurzzeitige Erhohung des Interrupt-Levels

#### 1.5.6 Barrier

Die fehlerhafte Programmierung eines wechselseitigen Ausschlusses mit Semaphoren kann zu signifikanten Fehlern führen - neues Sprachkonstrukt MONITOR ohne explizite Programmierung von P und V Operationen  stattdessen Generierung durch den Compiler .

- Modul, welches Daten und Methoden/Prozeduren enthält.
- Aufruf des Monitor Entry durch beliebig viele Prozesse; Garantie des wechselseitigen Ausschlusses
- Prozeduren/Methoden eines Monitors konnen auf globale Daten zugreifen, die lokalen Daten des Monitors sind aber von aussen nicht zugänglich
- Im Innern eines Monitors ist es ggf. notig, dass eine Aktivität wartet. Eine solche Aktivität wird durch den Monitor aus dem Monitor ausgelagert. Auf diese Weise bleibt der Monitor nicht blockiert und eine andere Aktivität kann in den Monitor eintreten. Falls eine andere Aktivität den Wartenden befreit, so kann diese, sobald der Monitor frei ist, diesen wieder betreten.

#### 1.5.7 Rendezvous

- Synchronisation von entfernten Prozedurauf- rufen (remote procedure calls)
- Der Prozedur-Aufrufer wird blockiert, bis die ent- fernte Prozedur ausgeführt wurde
- Der Prozedur-Anbieter bleibt blockiert, bis eine seiner Prozeduren aufgerufen wird.
- Operationen:
  - Rendenzvous anbieten (Anbieter)
  - Rendezvous beantragen (Aufrufer) - Warteschlange
  - Rendezvous annehmen (Anbieter) - erster Aufrufer
  - Rendezvous ausführen & Resultat melden (Anbieter)

### 1.6 Deadlock Erkennung und Vermeidung/Behebung

Deadlocks treten auf, wenn:

- die umstrittenen Ressourcen nur exklusiv nutzbar sind,
- die umstrittenen Ressourcen nicht entzogen werden konnen,
- die Belegung von Ressourcen schon moglich ist, auch wenn auf die Zuweisung weiterer Ressourcen gewartet werden muss,
- eine zyklische Kette von Prozessen auftritt, in der jeder Prozess mindestens eine Ressource besitzt, die der nächste Prozess in der Kette benotigt.

Gegenmassnahmen:

- Sicherstellen, dass immer eine der Deadlock-Bedingungen nicht erfüllt ist (Regeln für die Nutzung / erzwungene Freigabe etc).
- Zukünftigen Ressourcenbedarf der Prozesse analysieren und Zustände erkennen/verbieten, die zu Deadlocks führen.
- Bereits eingetretenden Deadlock erkennen und auflosen.

# 2 Verteilte vs. monolithische Betriebssysteme

## 2.1 Strategien / Varianten

- Hardware / Software / Benutzung
- Client-Server-System: Viele Clients greifen auf einen oder mehrere Server bzw. Services zu.
- Verteiltes Dateisystem: Ein über mehrere Server verteiltes virtuelles Dateisystem steht Clients zur transparenten Benutzung zur Verfügung.
- Netzwerkfähiges Betriebssystem: das Betriebssystem stellt systemübergreifende Funktionalität zur Verfügung.
- Verteiltes Betriebssystem: Das Betriebssystem selbst ist verteilt, für Benutzer und Anwendungen ist dies nicht sichtbar.
- Verteilte Anwendung: Durch die Programmierung der Anwendung wird das verteilte System erstellt das Programm muss in der Regel die Verteillogik kennen.

## 2.2 Vorteile & Risiken

### 2.2.1 Vorteile

- Verteilung versus Parallelität
- Lastverteilung
- Leistungssteigerung
- Skalierbarkeit/Flexibilität
- Sicherheit/Zuverlässigkeit/Verfügbarkeit
- Preis/Leistungbzw.Kostenreduktion
- Gemeinsame Datennutzung
- Gemeinsame Nutzung teurer Peripherie
- Applikatorische Verteilung

### 2.2.2 Risiken

- Erhohte Komplexität
- Erschwerte Fehlersuche
- Verlängerte Abhängigkeitsketten
- Sicherheitsdispositiv wird aufwendiger
- Nicht alle Subsysteme eignen sich für die Verteilung
- Sicherstellung der Konsistenz und Synchronisation

## 2.3 Anforderungen an verteilte Betriebssysteme

- Gemeinsamer Kernel Code und System Calls
- Gemeinsam genutzter Hauptspeicher
- Gemeinsames Prozess-Steuersystem
- Gemeinsames Dateisystem
- Gemeinsame Interprozess-Kommunikation
- Gemeinsame Ausnahmebehandlung
- Gemeinsame Synchronisationsmechanismen
- Gemeinsames System Management
- Spezialisierte Funktionen für das Management der Verteilung
- Transparente Benutzerschnittstelle

### 2.3.1 Transparenz

**Ortstransparenz:**
Der Ort der genutzten Ressourcen / erbrachten Dienste ist für den Anwender nicht sichtbar.

**Migrationstransparenz:**
Ressourcen konnen verlagert werden, ohne dass sich ihr Name bzw. ihre Nutzung verändert.

**Replikationstransparenz:**
Anwender konnen nicht erkennen, wie viele Instanzen es gibt.

**Nebenläufigkeitstransparenz:**
mehrere Anwender konnen die Ressourcen automatisch gemeinsam und unabhängig voneinander nutzen.

**Parallelitätstransparenz:**
Aktivitäten konnen parallel bzw. Nebenläufig ausgeführt werden, ohne dass der Anwender es bemerkt.

### 2.3.2 Flexibilität

Services sollen dynamisch an den Bedarf anpassbar sein, entweder durch administrative Eingriffe oder durch Eigenkonfiguration zur Laufzeit. änderungen sollen keinen kompletten Neustart des verteilten Systems erfordern.

### 2.3.3 Zuverlässigkeit

- Erhohte Zuverlässigkeit gegenüber Einzel- systemen trotz additiver Ausfallwahrschein- lichkeiten
- End-zu-End Verfügbarkeit statt Komponentenverfügbarkeit.
- Sicherheit  gleiche Richtlinien & Umsetzung im gesamten verteilten System.
- Fehlertolerenz
- Automatisches Recovery von Komponenten.

### 2.3.4 Performance

- Verteilungs- und Kommunikations- Mehraufwand muss den Aufwand wert sein
- Wiederholbarkeit / Determinismus von Leistungsindikatoren.
- Abhängigkeit von nicht direkt kontrollier- baren Komponenten
- End-zu-End Performance statt Komponenten-Performance.

### 2.3.5 Skalierbarkeit

Angebotsseite:

- Statische oder dynamische Zufügung / Weg- nahme von Servern.
- Verrechnung: Durchschnitt oder peaks?
- Vermeiden von Flaschenhälsen durch zu starke Serialisierung.

Dienstnehmerseite:

- Nicht vorhersagbare Anzahl Dienstnehmer.
- Einhaltung von Dienstgütegarantien

## 2.4 Synchronisation in verteilten Systemen

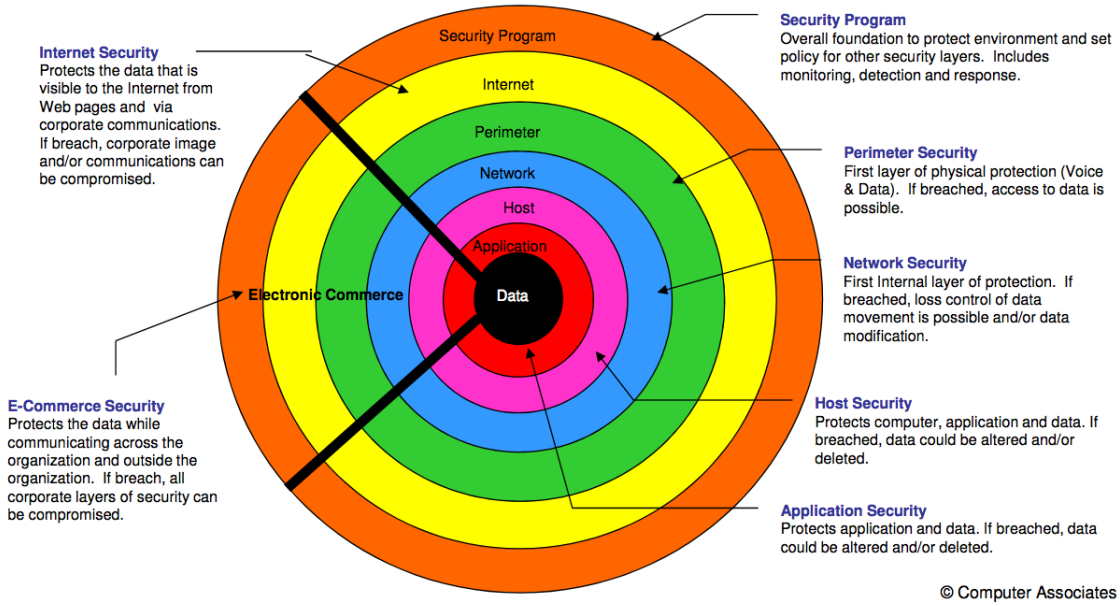
**Zeitsynchronisation**

- Absolut
- Relativ
- Vorstellen ist einfacher als Rückstellen

**Gegenseitiger Ausschluss über Systemgrenzen**

- Zentraler Algorithmus (mit Redundanz)
  - Zuverlässig, berechenbar
  - Starke Serialisierung
- Verteilter Algorithmus
  - Zeitliche Ordnung im Gesamtsystem
  - Getimte Nachricht an alle Prozesse und Warten auf Bestätigung
- Token Ring Algorithmus
  - Explizite Erlaubnis zum Zugriff in vordefinierter Reihenfolge
  - Suboptimale Ressourcennutzung / Wartezeiten

# 3 Sicherheitsaspekte im Betriebssystem



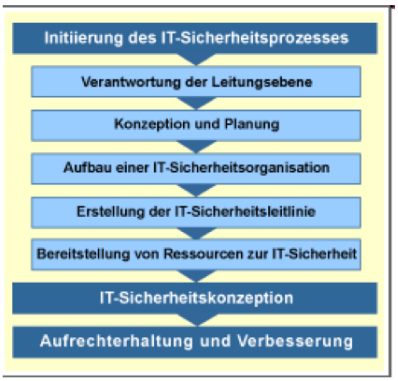

## 3.1 Typische Schwachstellen

- Lieferwege (physisch, aber vor allem elektronisch)
- Software-Voreinstellungen(Defaults)
- Funktionale Fehler der Software / Ausnutzbarkeit von Nebeneffekten
- Nicht getestete oder nicht autorisierte änderungen
- Fremd-/Fernzugriffe oder Auslagerung aus dem Sicherheits-Kontext
- Der Benutzer
- Der Administrator

## 3.2 Ursachen

- Komplexes Zusammenwirken verschiedener Effekte
- Ungerechtfertigtes Vertrauen
- Gutwilligkeit der Beteiligten
- Fehlerhafte Ausführung und/oder Kontrolle
- Boswilligkeit / Vorsatz

## 3.3 Sicherheits-Management


## 3.4 Risiko-Management

- Die wissentliche oder unwissentliche Akzeptanz einer Verlustwahrscheinlichkeit und moglichen Schadenhohe.
- Risiko-Management durch:
  - Analyse
  - Vermeidung (nur bedingt moglich, ...)
  - Übertragung (Versicherung, Werkschutz,...)
  - Begrenzung (präventive Massnahmen,...)
  - Akzeptanz (formaler, dokumentierter Willensakt)
  - Ignoranz (wegschauen,...)

Meist gibt es eine Mischung der Massnahmen, abhängig vom Risikotyp, der Risikokultur, den Kostenfolgen usw.