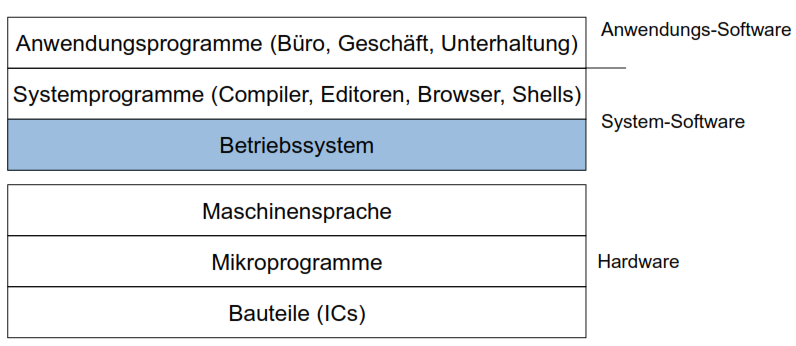
Zusammenfassung Betriebssysteme FS 17

# Entwicklung und Typen

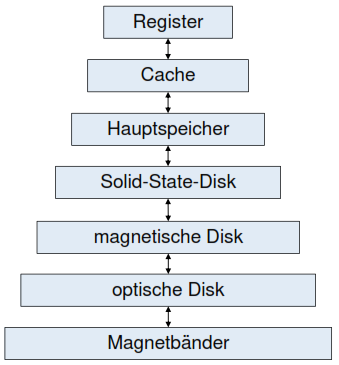
### Definition Betriebssystem

Nach dem internationalen Standard ist ein **Betriebssystem** die Programme eines digitalen Rechensystemes, die zusammen mit der Rechenanlage die Basis der Betriebsart legen und insbesondere die Abwicklung von Programmen steuern und überwachen.

Das Betriebssystem ist das Bindeglied zwischen Computer-Benutzer und Computer-Hardware. Es dient der bequemen Bedienung eines Computers und sorgt durch Vermeidung von Überlastsituationen oder Untätigkeitszeiten für eine effiziente Ausnutzung der Computer-Hardware.



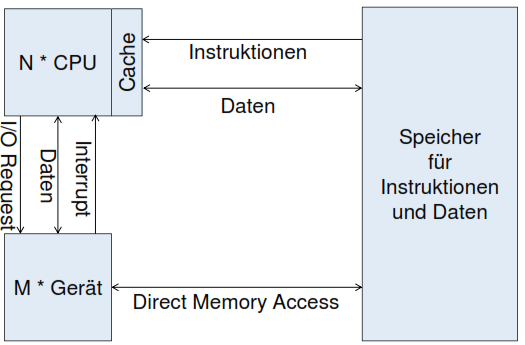
### Speicherhierarchie



Dabei nimmt der Speicherplatz nach unten zu.

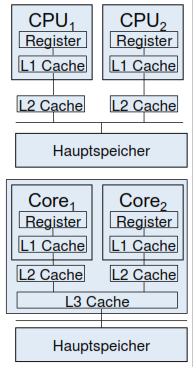
## Architektur von Rechnersystemen

### Einprozessorsysteme



Hier können Geräte autonom ohne Einfluss der CPU Daten lesen/schreiben und dann erst die CPU benachrichtigen, die derweil andere Sachen gemacht hat.

### Mehrprozessorsysteme



Ein Mehrprozessorsystem hat mehr als einen Prozessor. Durch einen gemeinsamen Speicher und einen gemeinsamen Takt geschieht eine enge Systemkopplung, beziehungsweise Synchronisation.

Dabei wird zwischen zwei Typen unterschieden. Beim **symmetrischen Multiprocessing** führen die Prozessoren eine identische Kopie des Betriebssystems aus. Ein Problem hierbei ist gleichmässige Auslastung durch Benutzung gemeinsamer Datenstrukturen.   
Beim **asymmetrischen Multiprocessing** werden gewisse Aufgaben (Tasks) den Prozessoren zugeteilt. Dabei plant der Master die Tasks für die Slave-Prozessoren (Scheduling).

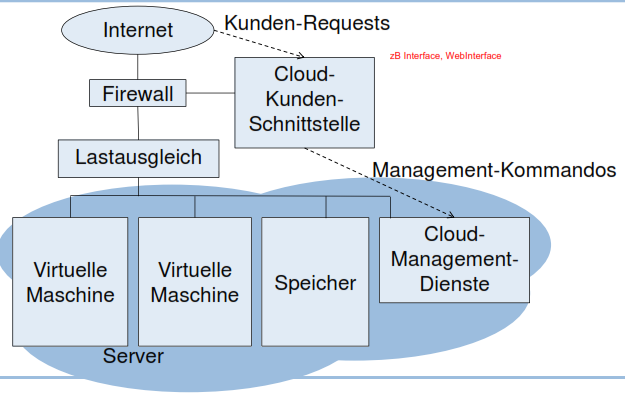
## Cluster-Systeme

Ein **Cluster-System** ist die Sammlung von eigenständingen Rechnern zur Durchführung von Berechnungen. Eine typische Definition ist das Teilen von gemeinsamem Speicher und die Verbindung über ein lokales Netz. Durch die hohe Überwachung und Übernahme von Anwendungen (Cluster-Software) entsteht eine hohe Verfügbarkeit. Dabei kann zwischen **asymmetrischem Clustering** (Ein Rechner ist im Hot-Standby-Modus, in dem er den anderen Rechner überwacht und allenfalls anspringt, und überwacht so den anderen, verarbeitenden Rechner) und **symmetrischen Clustering** (zwei oder mehr sich gegenseitig überwachende Rechner verarbeiten Anwendungen).

## Verteilte Systeme

Hierbei werden die Berechnungen auf mehrere (heterogene) Rechnersysteme verteilt, die loose über Kommunikationsnetze gekoppelt sind. Vorteile davon sind Lastausgleich und Vermeidung von Redundanz.

## Cloud Computing

Hier sind die Dienstleistungen auf einen Server ausgelagert, worüber der Kunde über eine Schnittstelle zugreifen kann.   


Dabei können diese Dienstleistungen nur im Internet verfügbar sein, nur für interne Benutzer oder eine Mischung aus diesen Varianten. Dabei kann auch zwischen Software as Service (Verfügbarkeit von Anwendungen), Platform as a Service (Verfügbarkeit von Middleware, zB Datenbank-Server) und Infrastructure as a Service (Bereitstellung von virtuellen Servern und Speicher) unterschieden werden.

## Sichtweisen auf ein Betriebssystem

Bei den Sichtweisen wird für üblich zwischen der Anwendersicht (top-down), möglichst einfaches Benutzen des Computers und Maximieren der Leistung und der Systemsicht (bottom-up), Robustes Ausführen und Sicherheit unterschieden.

### Abstrakte Maschine

Ein Betriebssystem stellt eine virtuelle Ablaufumgebung bereit. So wird die Hardware vor den Anwendungsprogrammen verdeckt, diese haben also keinen direkten Zugriff auf die Festplatte. Dabei wird eine schönere, einfachere, abstrahierte Schnittstelle angeboten.

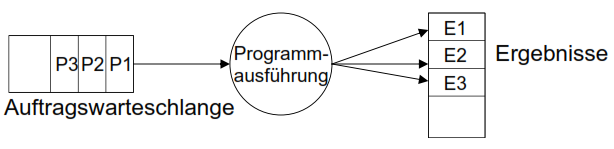
### Ressourcenverwaltung

Ein Computersystem besteht aus verschiedenen Komponenten, die vom Betriebssystem verwaltet werden.

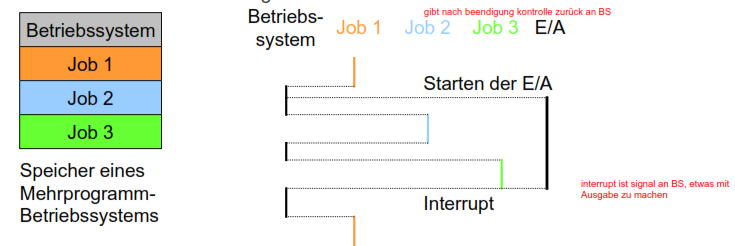
## Betriebssystemtypen

### Mainframe-Systeme

#### Stapelverarbeitung (Batch-Systeme)

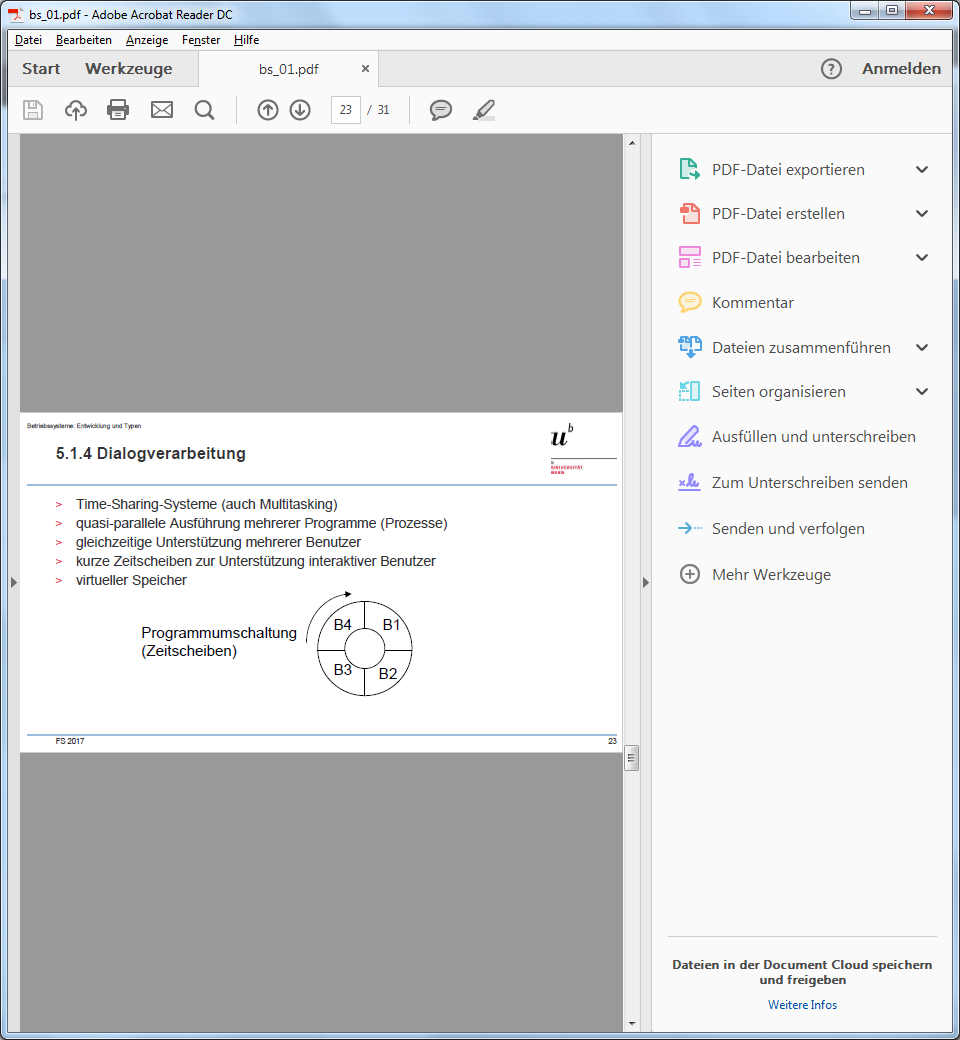
Die auszuführenden Programme werden auf ein Band geschrieben. Der Operatpr lädt dann ein spezielles Programm, das die Programme einliest und sequenziell ausführt. Die Ereignisse werden dann protokolliert.   


#### Mehrprogrammbetrieb



#### Dialogverarbeitung

Das ist ein Time-Sharing-System. Das heisst, nach einer gewissen Zeit wird die Kontrolle über die CPU abgeben. So entsteht der Eindruck von Gleichzeitigkeit.



### Server-Betriebssysteme

Das sind grössere PC, Arbeitsplatzrechner oder Mainframes, die mehrere Nutzer über ein Netz bedienen und dabei Soft- und Hardwareressourcen wie Drucker, Datei-, oder Webdienste bereitstellen.

### Multiprozessor-Betriebssysteme

Das ist eine Variante der Server-Betriebssysteme, die spezielle Funktionen für Kommunikation, Konnektivität und Konsistenz bereitstellt.

### PC-Betriebssysteme

Ein PC wird meist nur von einem einzigen Benutzer zu einem Zeitpunkt genutzt. Dabei werden viele Ein-/Ausgabe-Geräte unterstützt. Für diese Art Betriebssysteme sind einfache Benutzeroberflächen notwendig.

### Handheld-Computer-Betriebssysteme

Das sind zB Mobiltelefone. Dabei gelten spezielle Anforderungen wegen den begrenzten Ressourcen. Die Verarbeitung ist dabei ereignisgesteuert.

### Eingebettete Betriebssysteme

Das sind Betriebssysteme für kleine Geräte wie Mikrocontroller. Dabei gelten geringere Sicherheitsanforderungen, da kein Herunterladen von unzulässiger Software möglich ist.

### Sensor-Betriebssysteme

Sensoren sind kleine, batteriebetriebene Rechner.

### Echtzeit-Betriebssysteme

Die Verarbeitung geschieht innerhalb fester Zeitschranken. Durch diese Zeitschranken geschieht die Umschaltung zwischen Programmen, beziehungsweise durch das Betriebssystem. Dabei kann zwischen **strikten Echtzeitsystemen (hard real-time systems)** bei sehr harten Zeitforderungen und **weichen Echtzeitsystemen (soft real-time systems)** bei zwar Priorisierung für Realzeitaufgaben, aber seltenes Verfehlen der Zeitanforderungen akzeptabel unterschieden werden.

# Aufgaben und Strukturen

## Systemkomponenten

Die **Prozessverwaltung** ist zuständig für das Erzeugen, Löschen, Ausführen, Anhalten und Weiterführen von Prozessen. Zusätzlich kümmert es sich um die Synchronistation von diesen und behandelt Verklemmungen.

Die **Hauptspeicherverwaltung** ist zuständig für das Belegen und Freigeben von Hauptspeicher. Ausserdem ordnet es den Prozessen Hauptspeicher zu.

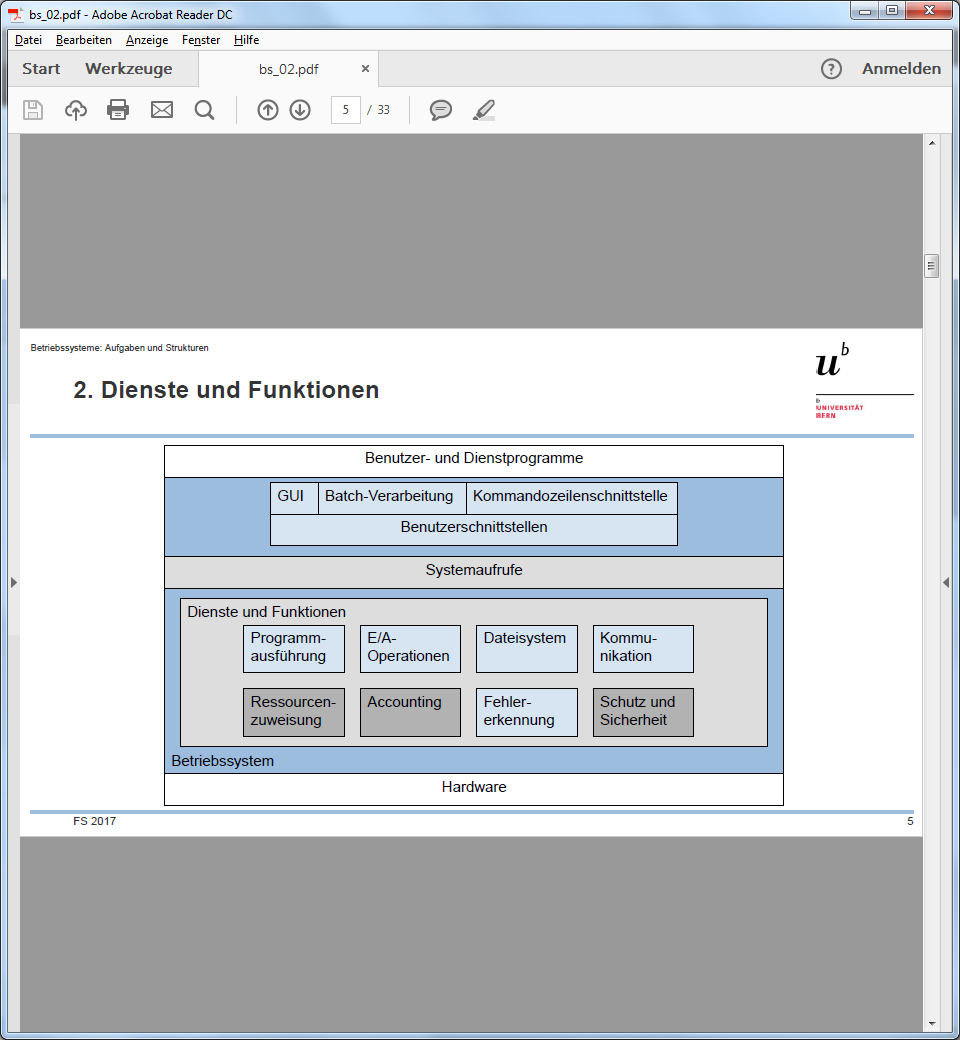
Das **Ein-/Ausgabe-System** ist zuständig für Speicherverwaltung. Ausserdem stellt es eine allgemeine Gerätetreiberschnittstelle dar und stellt für spezielle Hardware-Geräte die benötigten Treiber zur Verfügung.

Die **Sekundärspeicherverwaltung** ist eine Dateisystem-Verwaltung auf dem Sekundärspeicher und eine Massenspeicherverwaltung (Auslagern auf Sekundärspeicher, Zuweisen von Speicherplatz, Scheduling bei Festplatten). Ausserdem gehört das Caching dazu.

Die **Schutz- und Sicherheitssysteme** regeln die Zugriffskontrolle auf System- oder Benutzerressourcen.

Ebenfalls Systemkomponenten sind der **Kommando-Interpreter, grafische Benutzerschnittstellen** und **Kommunikationssysteme.**

## Dienste und Funktionen



### Dienste

Dienste dienen für Programme und Benutzer und zur Vereinfachung der Programmierung. Das sind zB die Benutzerschnittstellen, Programmausführung, Dateisystem-Manipulationen oder Fehlererkennung.

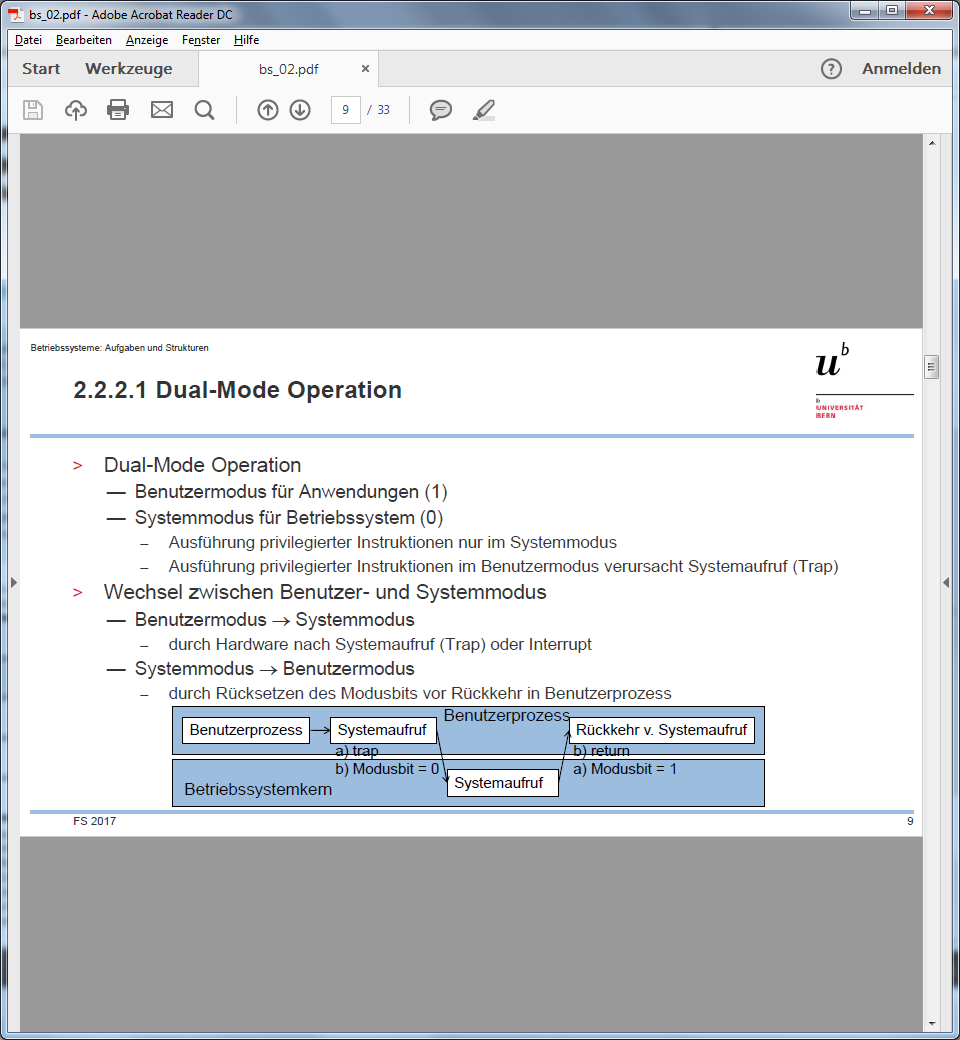
### Funktionen

Funktionen dienen zur effizienten Ausführung des Systems. Das sind zB Zuweisung von Ressourcen, Accounting, Schutzmechanismen oder Sicherheitsmechanismen.

#### Schutzmechanismen

Die Fehererkennung geschieht häufig durch die Hardware, die Fehlerbehandlung durch das Betriebssystem.

Bei **Dual-Mode Operationen** gibt es den Benutzermodus für Anwendungen und den Systemmodus für das Betriebssystem. Dabei können privilegierte Instruktionen nur im Systemmodus ausgeführt werden. Wird versucht, eine solche privilegierte Instruktion im Benutzermodus auszuführen wird ein Trap (Systemaufruf) ausgelöst. Dann wird durch das Setzen des Modusbits auf 0 in den Systemmodus gewechselt. Wenn die Instruktion ausgeführt ist, wird durch das Setzen des Modusbits auf 1 wieder zurückgewechselt. Das Gleiche gilt bei Interrupts.



Beim **E/A-Schutz** sind die E(ingabe)/A(usgabe)-Instruktionen privilegiert und können nur vom Betriebssystem aufgerufen werden. Wollen also Anwendungen E/A-Operationen ausführen müssen sie Systemaufrufe tätigen.

Der **Speicherschutz** schützt die Prozesse untereinander vor inkorrektem Speicherzugriff. Dabei wird der zulässige Speicherbereich durch ein Basis- (kleinste Adresse) und ein Limit-Register (Grösse des Speicherbereichs) beschrieben. Der Vergleich der Speicheradressen mit den Registern geschieht durch die CPU Hardware.

Der **CPU-Schutz** verhindert die Monopolisierung der CPU durch eine Anwendung. Dabei löst das Betriebssystem nach einer bestimmten Zeit ein Interrupt aus und übergibt danach die Kontrolle der CPU an eine andere Anwendung.

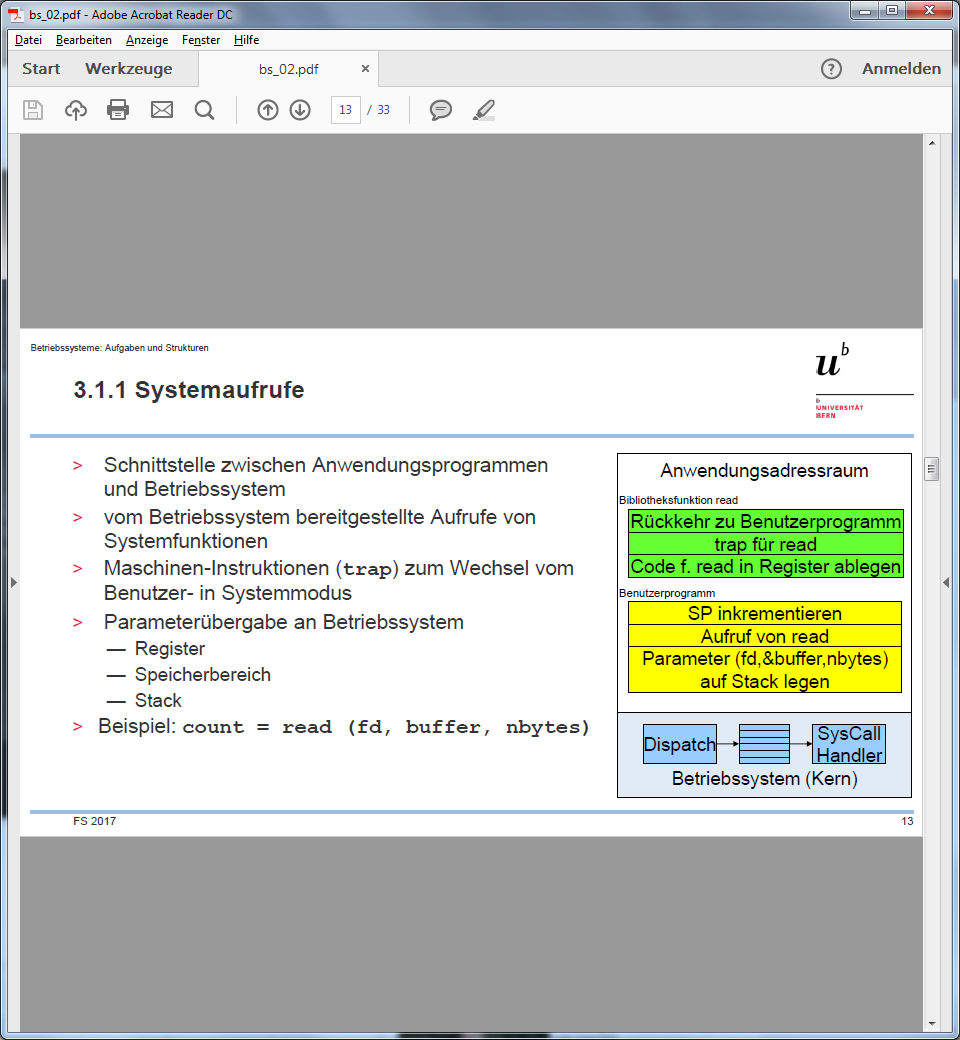
## Schnittstelle zwischen Anwendungen und Betriebssystemen

### Systemaufrufe

**Systemaufrufe** bilden die Schnittstelle zwischen Anwendungsprogrammen und Betriebssystem und werden vom Betriebssystem bereitgestellt um Systemfunktionen aufrufen zu können.

Für den Wechsel vom Benutzer- in den Systemmodus gibt es die Maschinen-Instruktion **trap**.

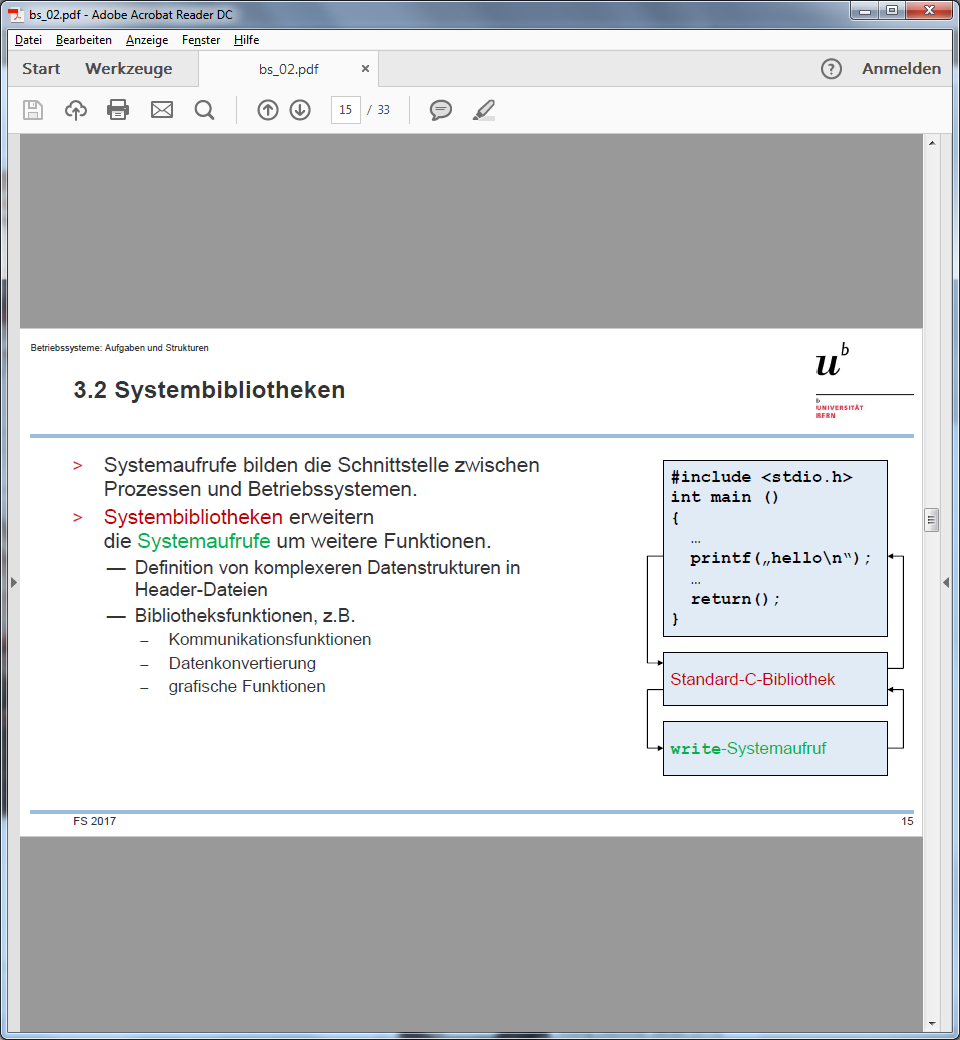
Bei einem Systemaufruf können die Parameter Register, Speicherbereich und Stack mit übergeben werden. Ein Beispiel dafür wäre *count = read (fd, buffer, nbytes)*.



Andere Beispiele für Systemaufrufe sind u.a. *fork()* (Erzeugen eines Kindprozesses), *open()* (öffnen einer Datei) oder *gettimeofday(timeval)* (Abrufen der Systemzeit).

### Systembibliotheken

Die **Systembibilotheken** erweitern die Systemaufrufe um weitere Funktionen. Dabei werden komplexere Befehle oder Instruktionen durch einfachere Aufrufe ummantelt. Beispiel *printf()* der C-Bibliothek stdio.h :



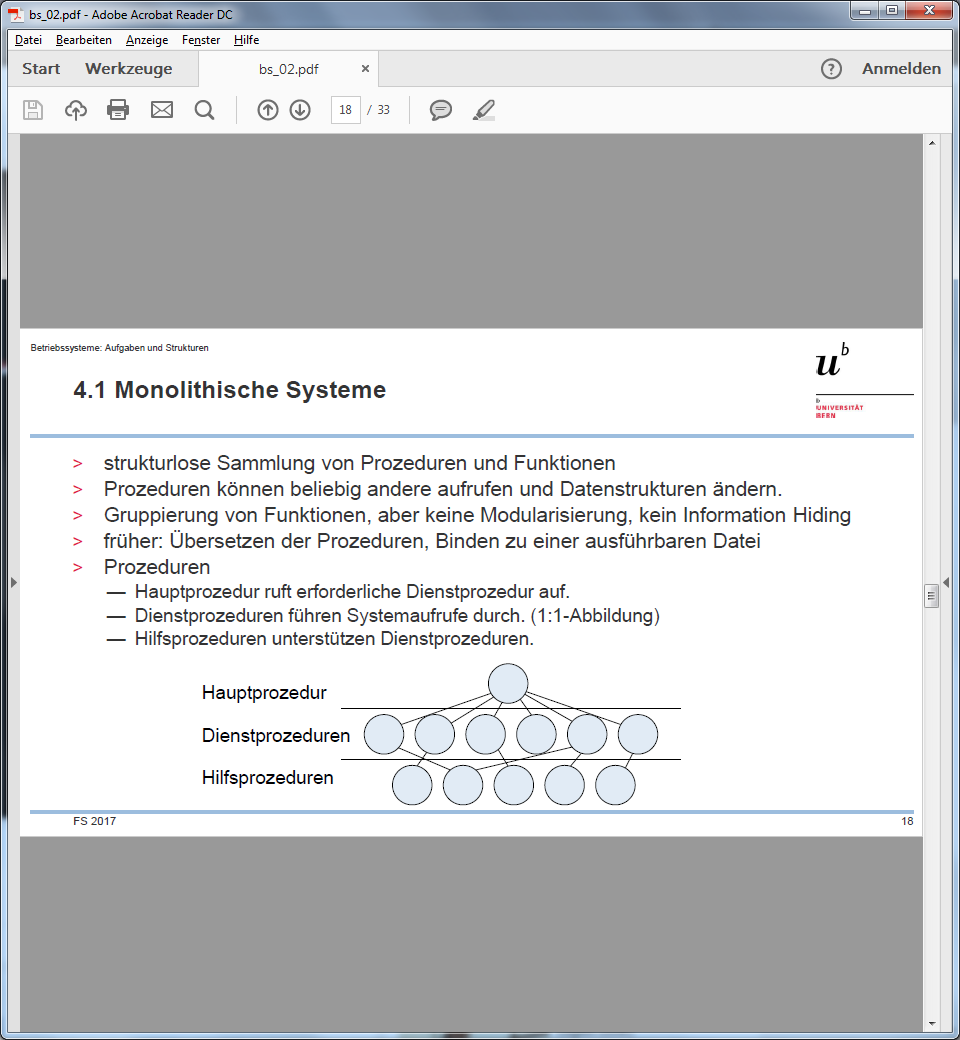
### Systemprogramme

**Systemprogramme** bieten die Grundlagen für Programmentwicklung und -ausführung. Beispiele dafür sind Programme wie der Standard-Editor vi, ftp oder ls.

## Betriebssystemarchitekturen

### Monolithische Systeme

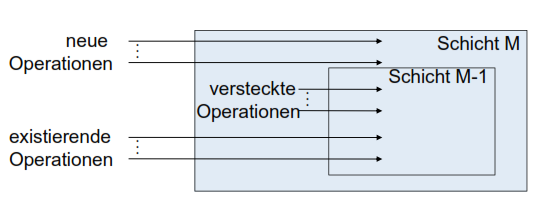
**Monolithische Systeme** sind strukturlose Sammlungen von Prozeduren und Funktionen., Die Prozeduren können dabei beliebige, andere Prozeduren aufrufen und auch Datenstrukturen ändern. Dabei ruft die Hauptprozedur die erforderlichen Dienstprozeduren auf, die jeweils einen Systemaufruf tätigen und dabei von Hilfsprozeduren unterstützt werden können.



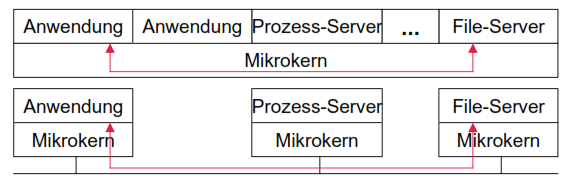
Beispiele dafür sind u.a. MS-Dos oder UNIX.

### Geschichtete Systeme

Hier besteht das Betriebssystem aus mehreren Schichten, welche aufeinander aufbauen und Funktionen der darunterliegenden Schicht brauchen. Vorteile dabei sind die Modularität (Schichten für sich funktionieren) und die Testbarkeit/das Debugging.

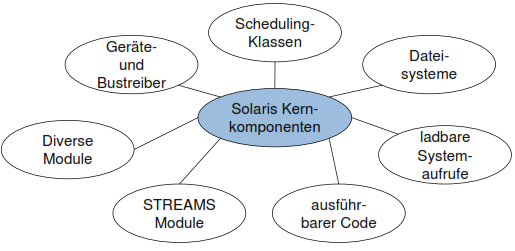


### Mikrokerne

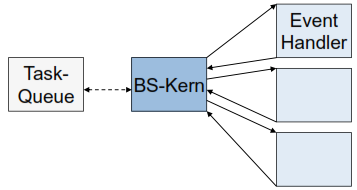
Diese Struktur sorgt für kleine Betriebssysteme, da Teilfunktionen in Server ausgelagert werden. Die Systemaufrufe werden dann an die zuständingen Server weitergeleitet.   


### Module

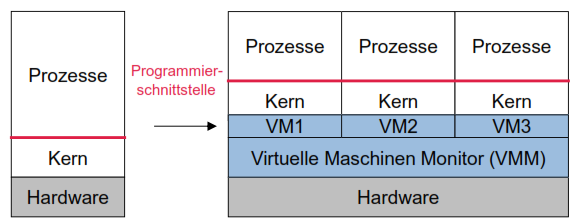
Mit der objektorientierten Programmierung werden Betriebssystemkerne modular erzeugt. Der Betriebssystemkern läuft dann mit den Kernkomponenten und bindet die zusätzlichen Dienste beim Booten oder zur Laufzeit ein. Beispiel Solaris:



### Ereignisgesteuerte Betriebssysteme

Hierbei reagiert das Betriebssystem nur auf bestimmte Ereignisse und arbeitet diese ab. Für länger andauernde Berechnungen werden unterbrechbare Tasks benutzt. Diese Art Betriebssystem wird häufig in Realzeitsystemen eingesetzt.   


### Virtuelle Maschinen

Virtuelle Maschinen abstrahieren die Hardware eines Computers und erzeugen so die Illusion, dass der Umgebung ein eigener Computer zur Verfügung hat. Durch virtuellen Speicher, CPU-Scheduling, etc können so auch verschiedene virtuelle Maschinen emuliert.   


Vorteile virtueller Maschinen sind die Sicherheit und Robustheit durch die Isolation, die Entwicklung oder das bequeme Wechseln von Betriebssystemen. Die Nachteile sind, dass der Kontextwechsel und die Emulationen Overhead erzeugen und Hardwareunterstützung erforderlich ist.

### Hybride Systeme

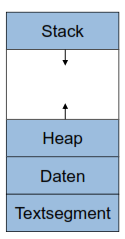
Häfuig werden in der Praxis solche Architekturkonzepte kombiniert. Linux z.B. ist monolithisch, kann aber auch (modular) dynamische Funktionen nachladen.

# Prozesse und Threads

## Prozesse

### Prozesskonzept

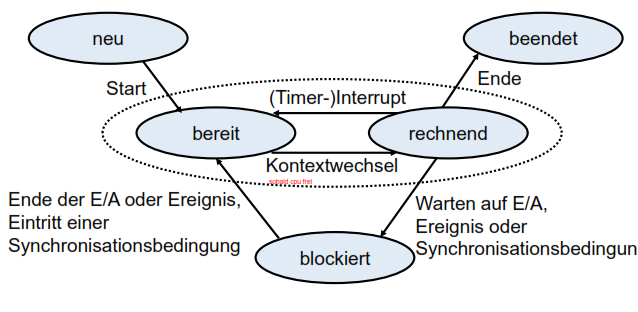
Ein **Prozess** kann als Programm in Ausführung verstanden werden. Er besteht aus Programmcode (Textsegment), einem Befehlszähler (zeigt, wo im Code wir uns befinden) + Prozessorregisterinhalt, einem Stack mit temporären Daten, gloablen Daten und einem Heap für dynamisch zugeteilten Speicher.



Dabei besitzt jedre Prozess eine virtuelle CPU.

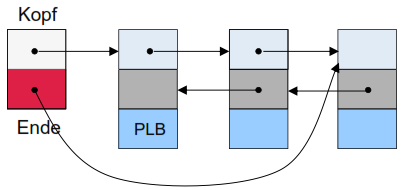
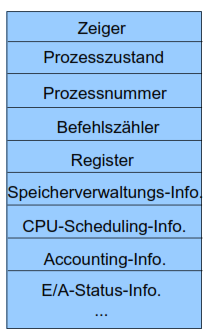
#### Prozesszustände

Im Zustand **neu** befindet sich der Prozess, wenn er neu erzeugt wurde.   
Ein Prozess ist im Zustand **rechnend**, wenn seine Instruktionen auf der CPU ausgeführt werden.   
Wenn er auf ein Ereignis warten muss (zB Ende E/A) ist er im Zustand **blockiert**.   
Wartet er auf CPU-Zuteilung ist er im Zustand **bereit**.   
Hat er die Ausführung abgeschlossen ist er im Zustand **beendet**.



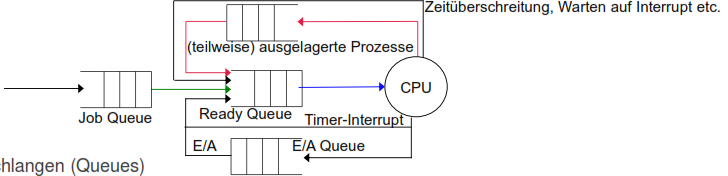
#### Prozessleitblock

Der Prozessleitblock beschreibt den Prozesszustand. Dabei repräsentiert er einen Prozess im Betriebssystem. Diese Prozessleitblöcke werden dann in den Warteschlangen verkettet.

### Prozesswechsel

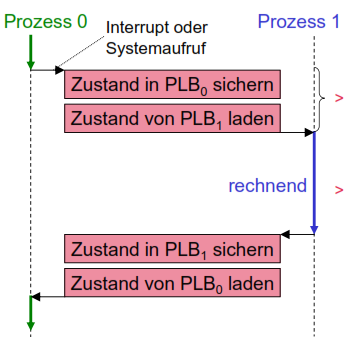
#### Scheduling

Es gibt verschiedene Warteschlangen (Queues). Die **Job Queue** sind die abgeschickten, auf Massenspeicher abgelegten Prozesse. In der **Ready Queue** sind die Prozesse, die im Zustand bereit sind. In der **E/A-Queue** sind die Prozesse, die auf ein E/A-Gerät warten. Dann gibt es noch eine Queue von ausgelagerten Prozessen.   


#### Dispatcher

Der **Dispatcher** übergibt dem beim Scheduling ausgewählten Prozess die Kontrolle über die CPU, nimmt dabei den Kontextwechsel vor, macht den Wechsel in den Benutzermodus und vollführt den Sprung an die korrekte Stelle im Anwendungsprogramm und die Fortsetzung der Ausführung. Der Dispatcher sichert also auch den aktuellen Zustand des Prozesses auf der CPU.

#### Prozessumschaltung



Der Zustand von Prozess 0 wird in den Prozessleitblock 0 gesichert. Dann wird der Zustand von Prozess 1 aus dem Prozessleitblock 1 geladen. Dann arbeitet dieser, ehe sein Zustand wieder in den Prozessleitblock 1 gesichert wird und der Zustand von Prozess 0 aus dem PLB 0 geladen wird.

### Prozesssteuerung

#### Prozesserzeugung

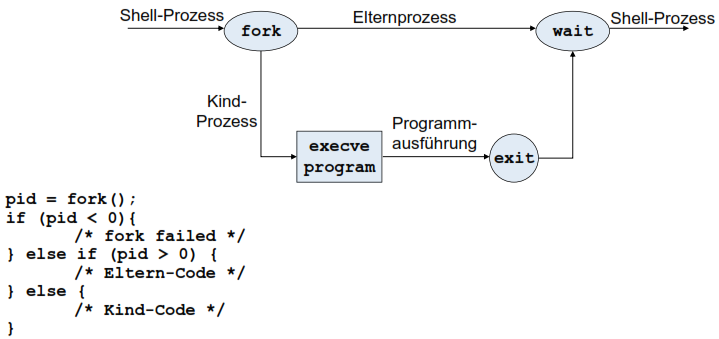
Prozesse werden bei Systeminitialisierung durch Benutzer oder laufende Prozesse erzeugt. Erzeugt dabei ein Prozess einen anderen Prozess hat er als Elternprozess einen Kindprozess erzeugt. Dabei gibt es mehrere Optionen für das Teilen von Ressourcen (Eltern und Kinder teilen alle Ressourcen, Kinder teilen nur Teilmenge der Elternressourcen, Eltern und Kinder teilen keine Ressourcen), die Ausführung (Nebenläufiges Ausführen von Eltern/Kindern, Eltern warten auf Beendigung der Kinder) und den Adressraum (Kindprozess als Duplikat des Elternprozesses, Kindprozess lädt eigenes Programm).

Beispiel: *fork()* erzeugt eine Prozesskopie mit neuer Prozessnummer

#### Beenden von Prozessen

Wenn der Prozess die letzt Anweisung ausgeführt hat beauftragt er das Betriebssystem mit der Löschung (*exit)*. Der Elternprozess kann aber auch auf die Beendigung eines Kindprozesses inklusive der Datenrückgabe warten (*wait*). Schliesslich kann der Elternprozess den Kindprozess auch beenden (*abort*), zB wenn die zugeteilten Ressourcen verbraucht wurden, der Kindprozess nicht länger benötigt wird oder der Elternprozess terminiert.

#### Beispiel UNIX

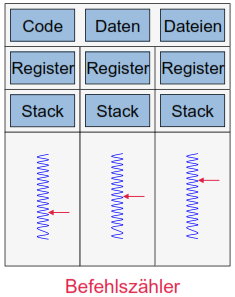


## Threads

### Threads-Konzept

Ein **Thread** kann als leichtgewichtiger Prozess angesehen werden. Er besteht aus einer ID, einem Befehlszähler, einem Register und einem Stack. Die Threads teilen untereinander den Programmcode, globale Daten und Betriebssystemressourcen (Signale, offene Dateien). Ein **Task** ist dann die Ansammlung von Threads. Ein traditioneller, schwergewichtiger Prozess besteht aus einem Task und einem Thread.

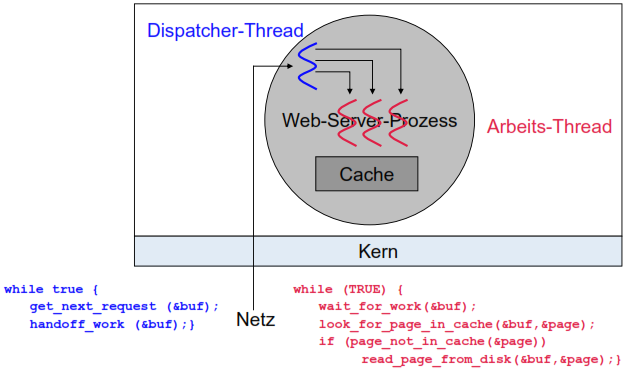
Ein Thread eines Tasks kann auch dann arbeiten, während andere Threads blockiert sind. Dazu können sie Kind-Threads erzeugen. Das erfordert Koordination und Steuerung. Durch den gemeinsamen Speicher wird die Kooperation vereinfacht und durch die gemeinsamen Ressourcen der Kontextwechsel effizient gestaltet.

Multi-Threading bezeichnet dann das abhandeln mehrerer Threads in einem Prozess. Die Zustände sind dabei gleich wie bei den Prozessen.   


#### Nutzung und Vorteile von Threads

Wenn bei Applikationen mehrere Aktivitäten gleichzeitig ablaufen, können manche davon blockieren. Solche quasiparallelen Aktivitäten können mit Threads einfacher programmiert werden. Für Threads werden auch keine **Betriebsmittel** (Für Prozesse benötigte Systemressourcen, zB Drucker, Arbeitsspeicher) benötigt, so können sie einfacher und schneller erzeugt werden. Ausserdem können sie Ressourcen teilen, sind reaktionsfreudig und können auf verschiedene Prozesse abgebildet werden.

Ein Beispiel für Threads ist ein Browser mit mehreren Fenstern oder ein Web-Server mit einem Thread pro Request:



### Thread-Typen

#### User Threads

User Threads laufen unabhängig vom Betriebssystem. Es gibt auch keinen Wechsel des Adressraums bei einem Thread-Wechsel. Dazu hat jeder Prozess eine eigene Thread-Tabelle.