

ITEO – IT Engineering & Operations

Systemsoftware

Bruno Joho

Ziele

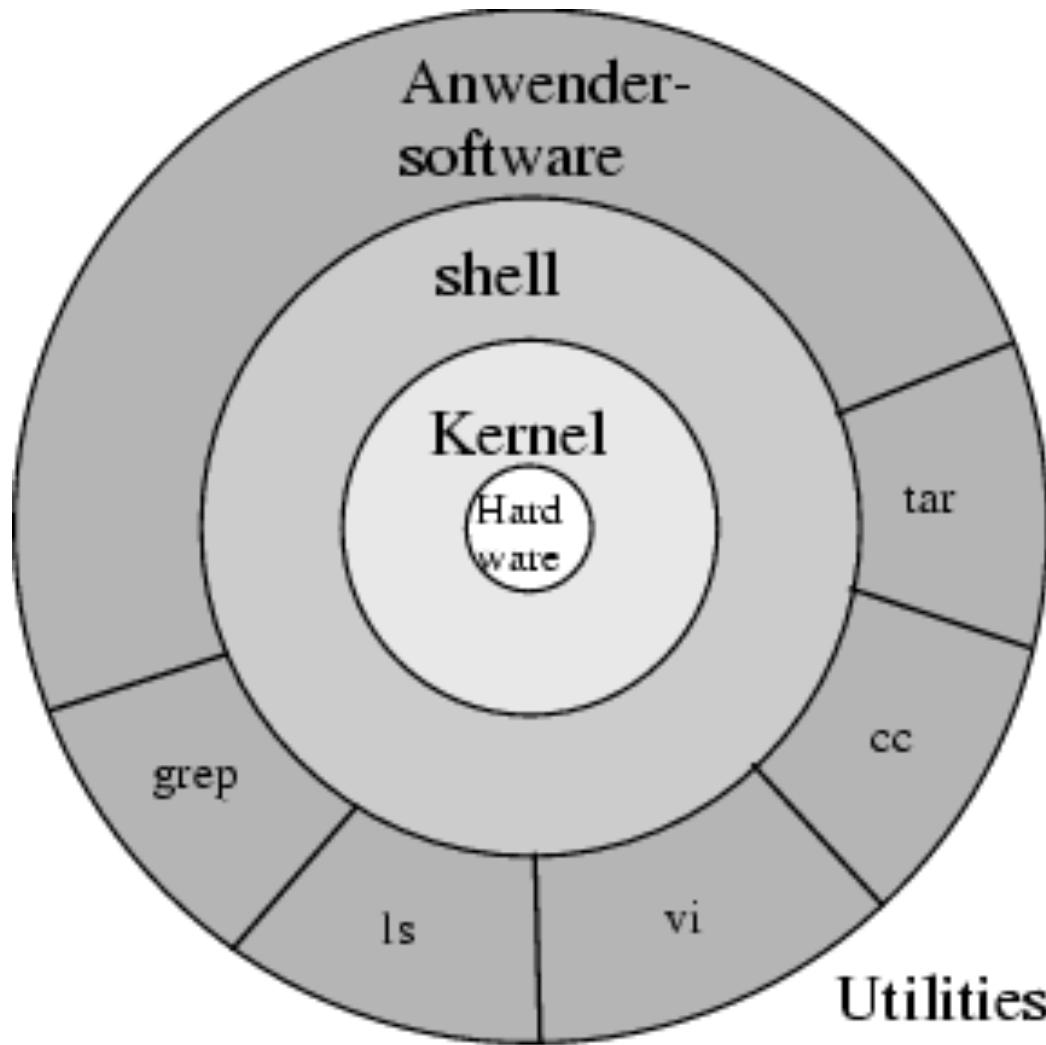
- Sie sind mit der Architektur eines typischen Serverbetriebssystems vertraut.
- Sie wissen wie Ressourcen durch das Betriebssystems verwaltet werden.
- Sie kennen die Vor und Nachteile eines Server Betriebssystems.
- Sie kennen die Relationen zwischen Prozessen und Threads im Betriebssystemkontext.

System Software

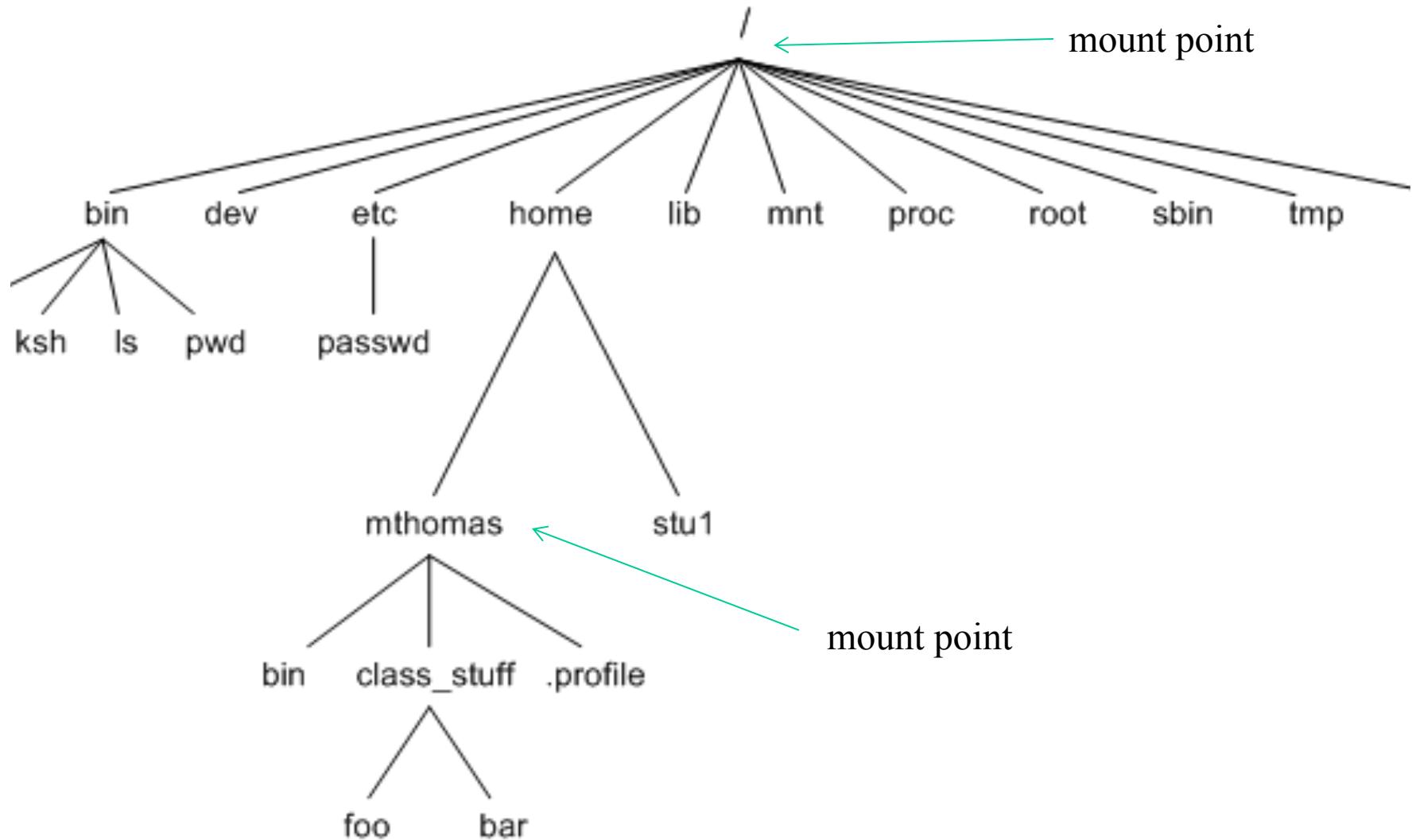


- System Struktur
- Prozess Elemente
- Prozess erzeugen
- Traps (Prozesse unterbrechen)
- Prozess Kontext
- Threads

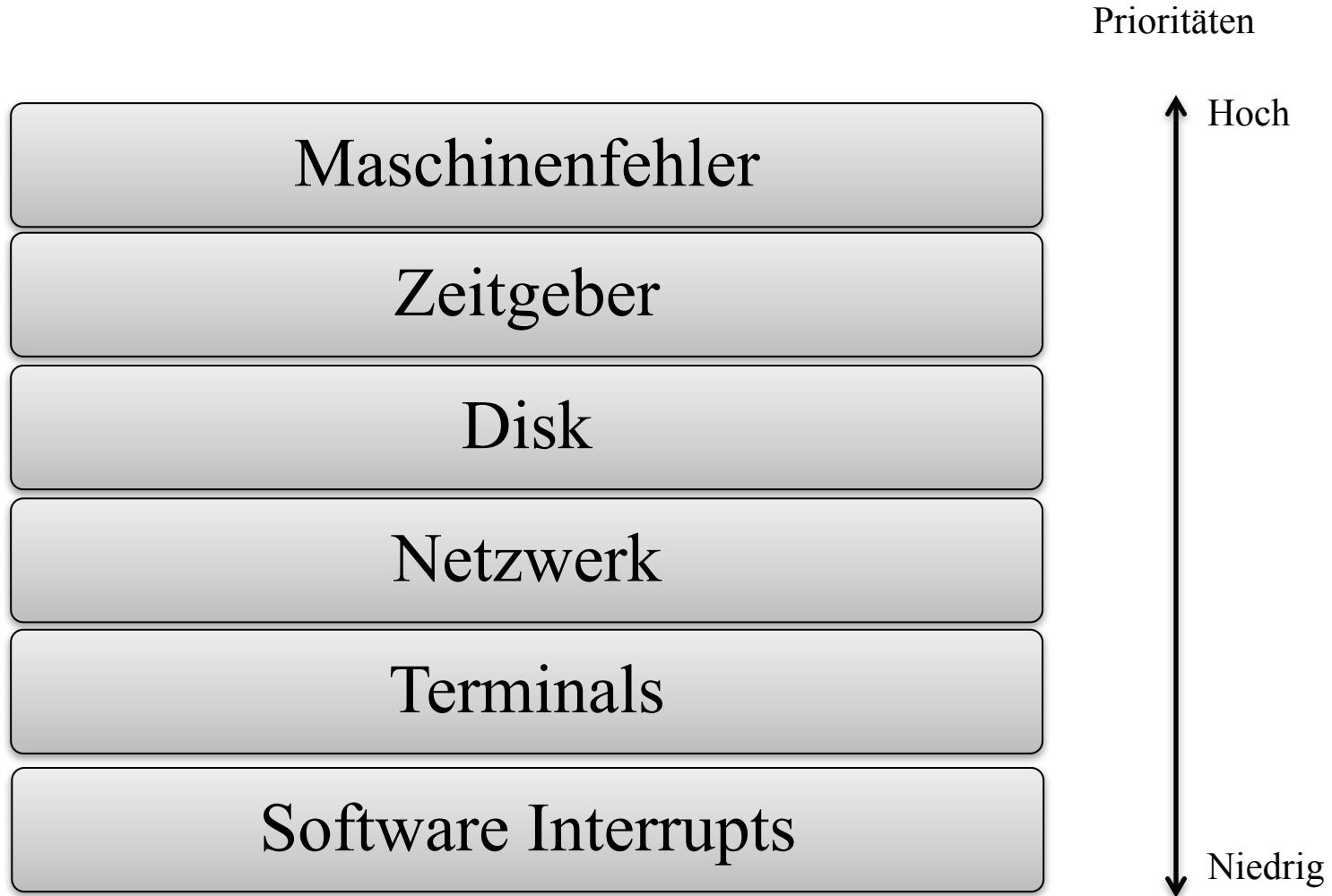
System Structure



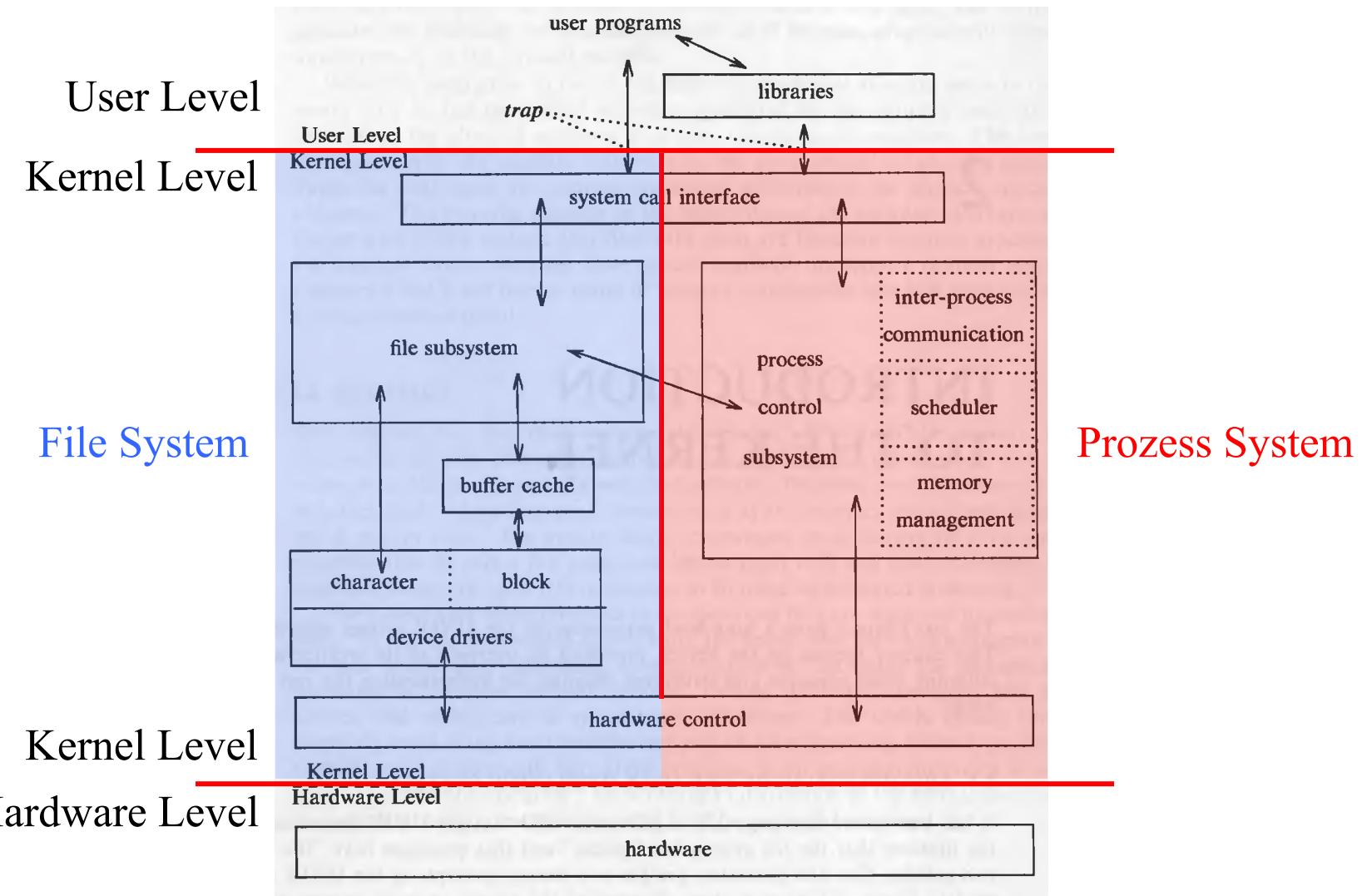
Sample System Tree



Interrupt Prioritäten



Blockdiagramm Systemkern

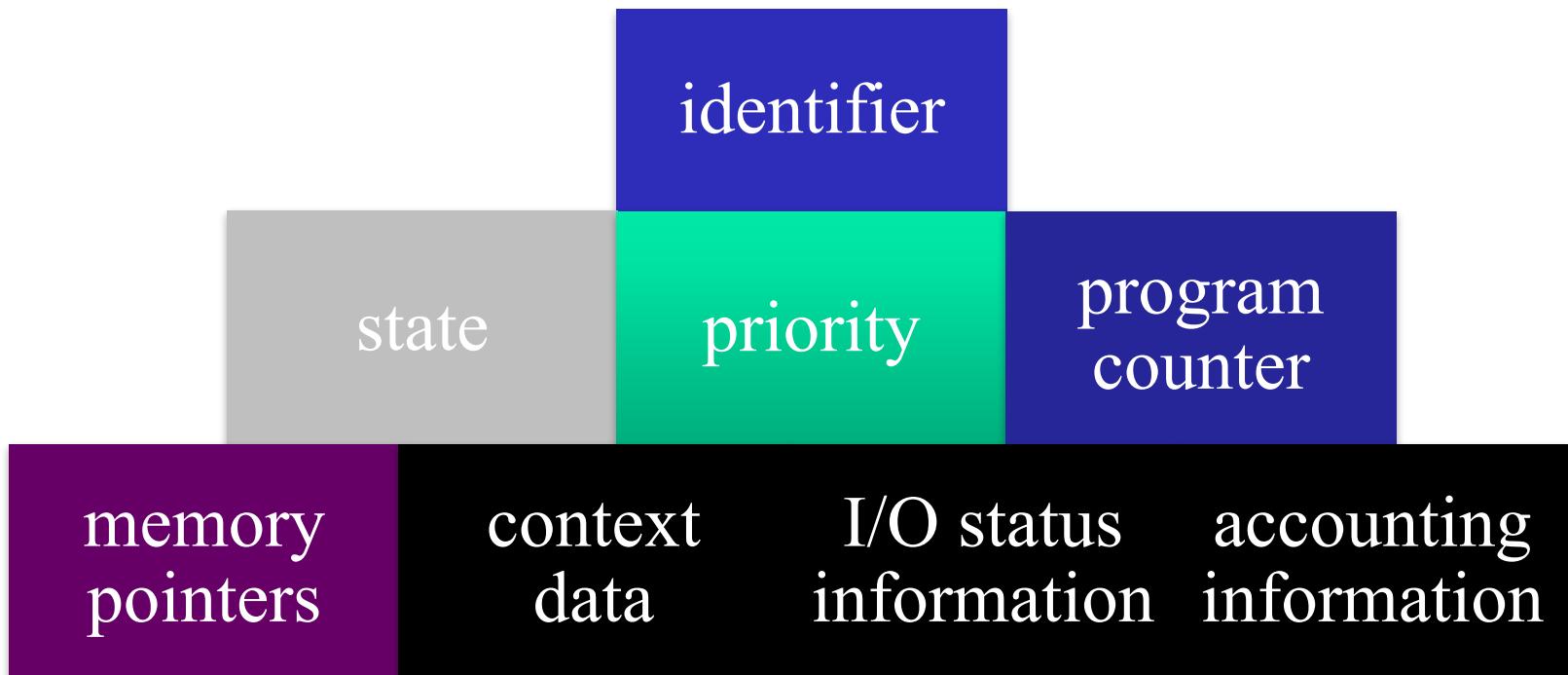


System Software

- System Struktur
- Prozess Elemente
- Prozess erzeugen
- Traps (Prozesse unterbrechen)
- Prozess Kontext
- Threads

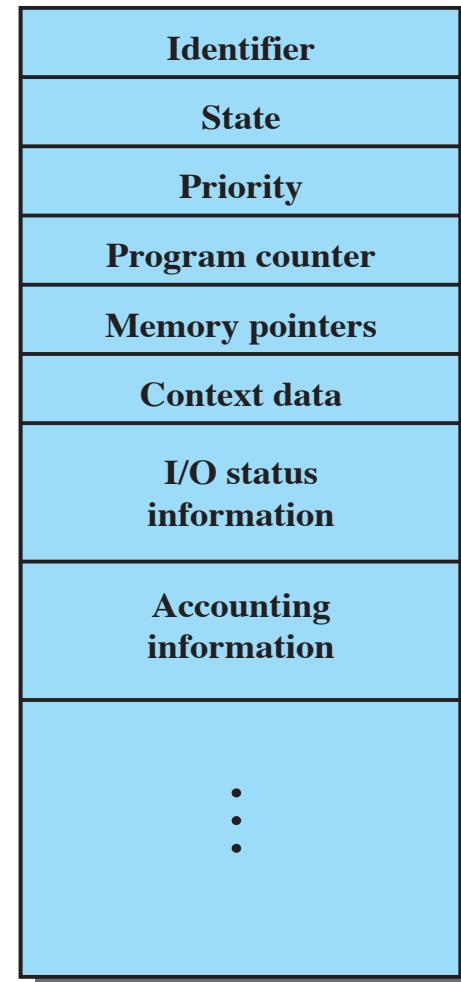
Prozess Elemente

Wenn ein Programm ausgeführt wird kann der dazugehörige Prozess einzigartig charakterisiert werden durch ein paar Elemente wie:



Prozess Kontroll Block

- Enthält die Prozess Elemente.
- Es ist somit möglich einen laufenden Prozess zu unterbrechen um ihn später wieder laufen zu lassen so als wäre er nie unterbrochen worden.
- Erzeugt und verwaltet durch das OS.
- Schlüsselbaustein der mehrere Prozesse unterstützt.



Prozess Status

*Trace
(verfolgt)*

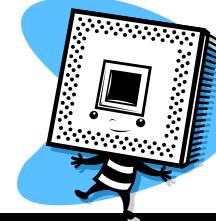
Verfolgt das Verhalten der individuellen Prozesse indem er die Sequenzen der Instruktionen auflistet die er für diesen Prozess ausführt.

Das Verhalten des Prozessors kann charakterisiert werden indem gezeigt wird wie die traces der Prozesse verschachtelt sind.



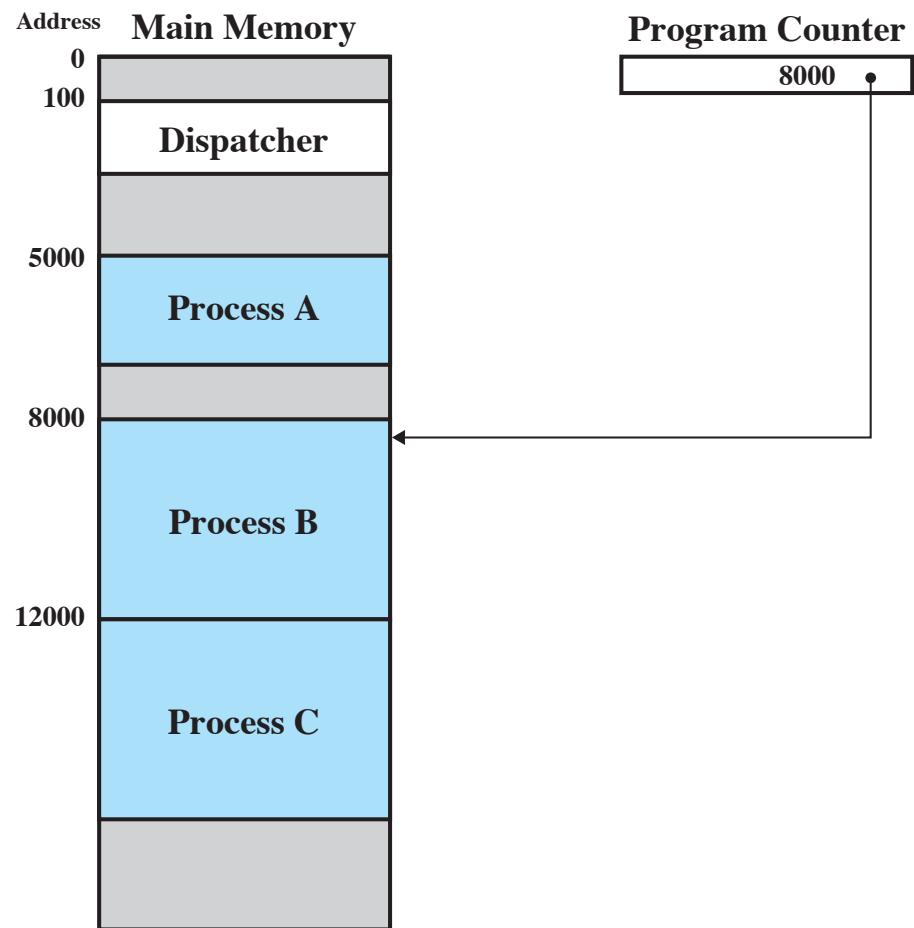
*Dispatcher
(verteilt)*

Programm das von einem Prozess auf den anderen wechselt welche dann auf dem Prozessor ausgeführt werden.



Prozess Ausführung

- Memory Layout von 3 Prozessen (vereinfacht: kein virtuelles Memory)
- Der Dispatcher wechselt den Prozessor von einem Prozess zum Anderen.

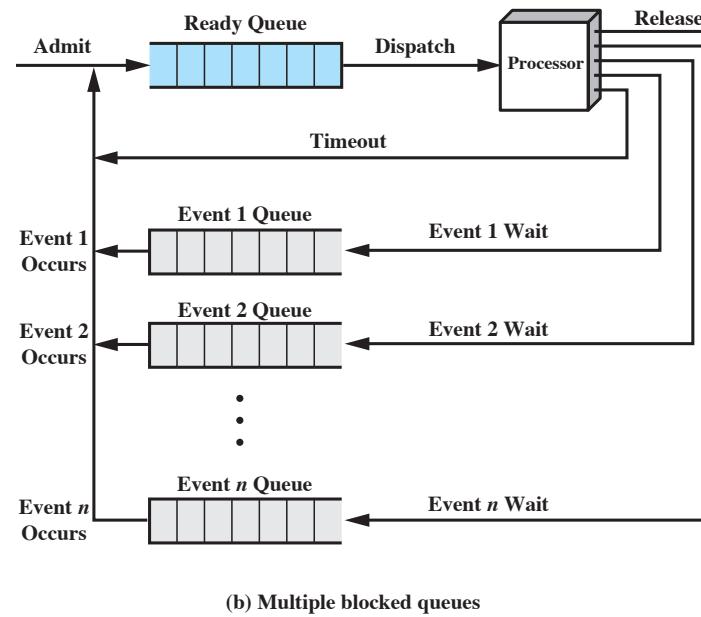
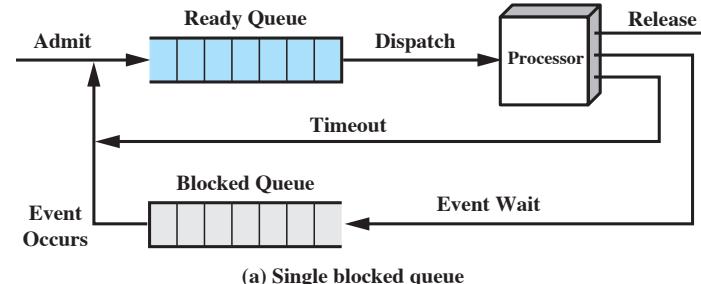


Queuing Modell

- a. Das OS muss immer die ganze Queue durchsuchen um den geeigneten Kandidaten (Prozess) zu finden.
- b. Normalerweise können das hunderte oder tausende Prozesse sein. Daher werden mehrere Queues verwendet.

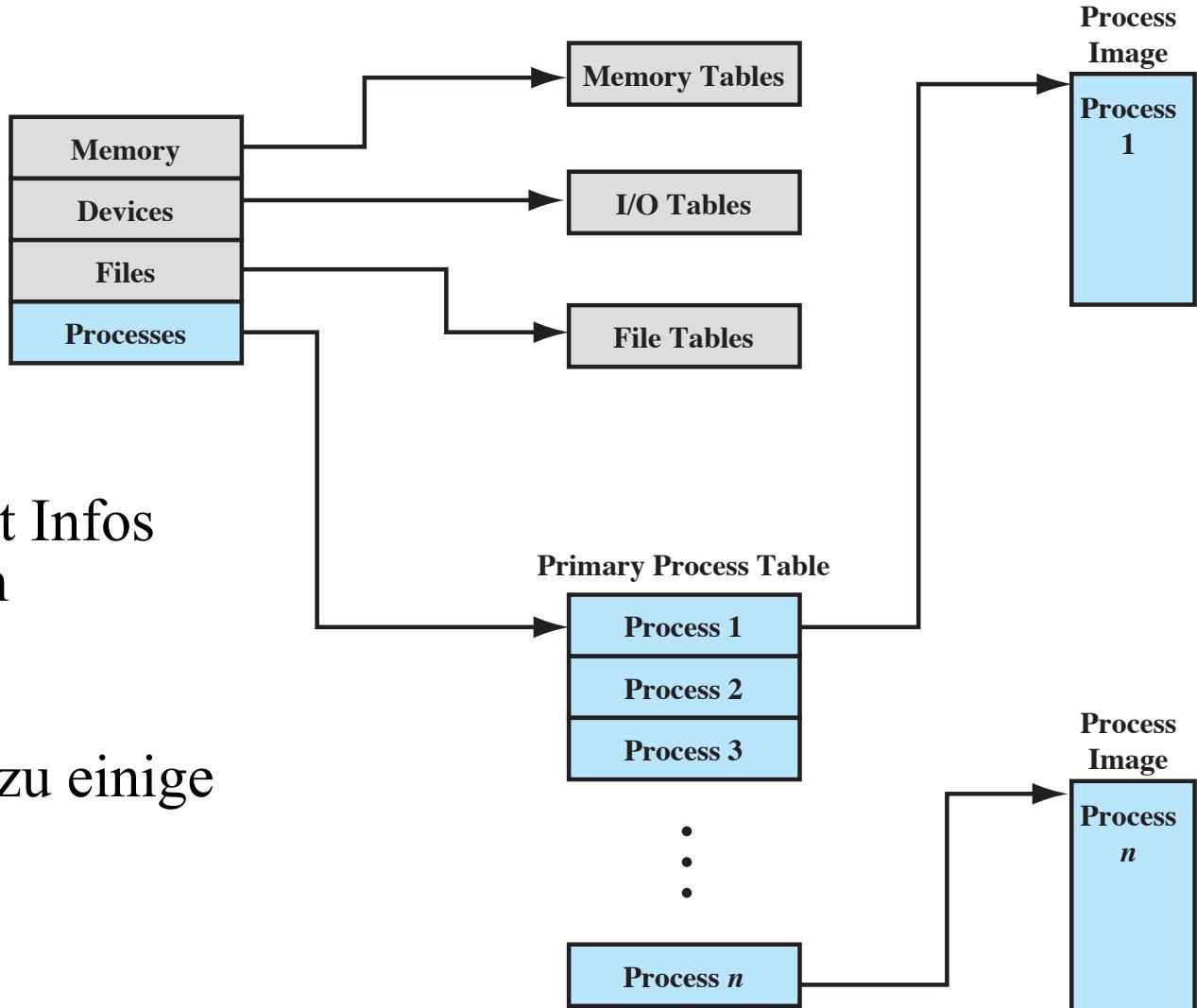
Trifft ein Event ein so werden alle Prozesse in dieser Queue in die „ready“ Queue verschoben.

Es werden eher verkettet Listen verwendet als Queues (FIFO).



OS Control Tables

- Das OS braucht Infos vom Status von Prozessen und Ressourcen.
- Es unterhält dazu einige Listen



Process Control Structures (1)

Um einen Prozess zu verwalten muss das OS wissen:

- Wo der Prozess zu finden ist.
- Die Attribute vom Prozess die für dessen Verwaltung essentiell sind.

Process Control Structures (2)

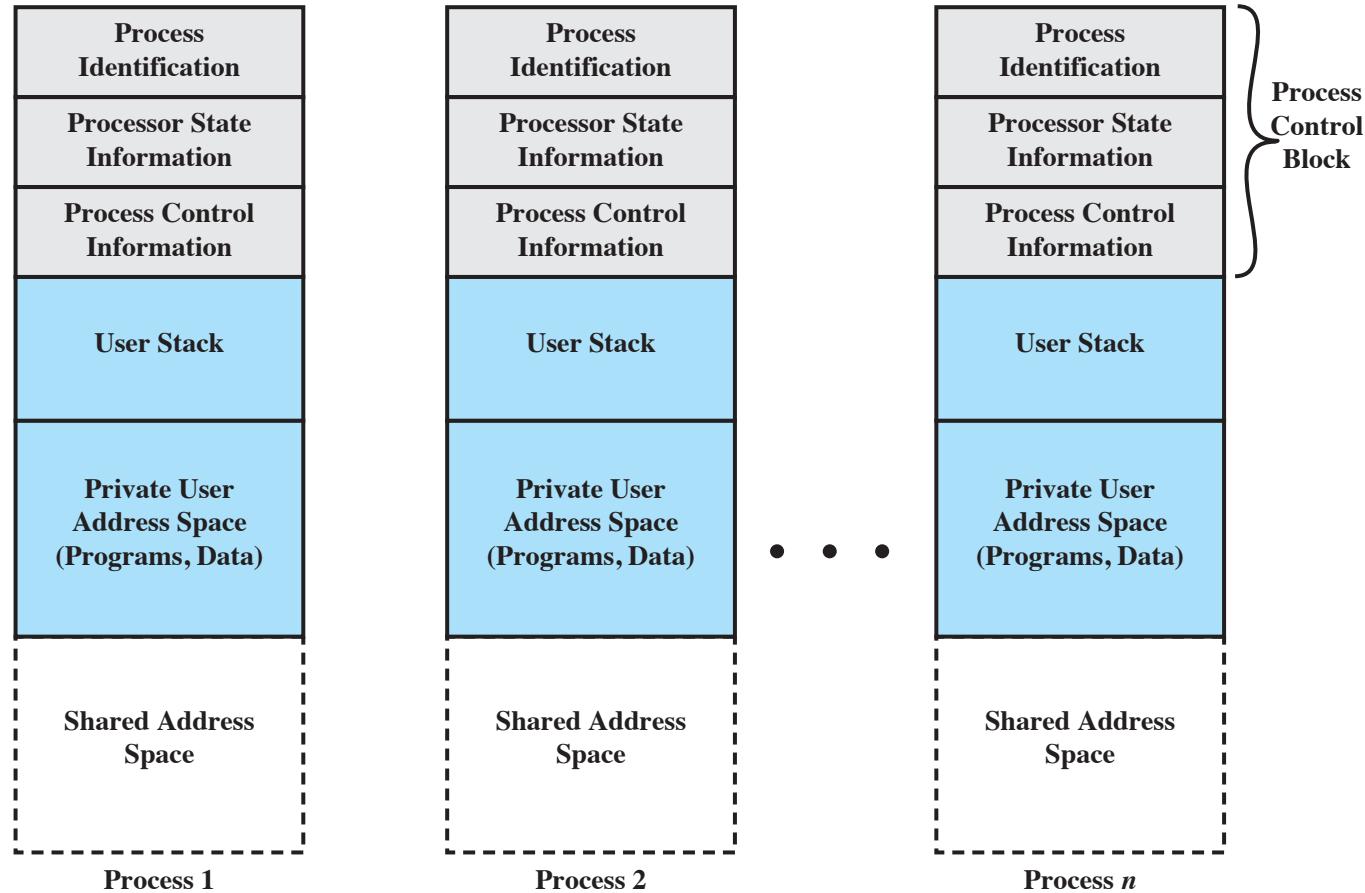
Prozess Location

- Ein Prozess enthält ein oder mehrere Programme zum ausführen.
- Ein Prozess muss mindestens genug Memory aufweisen um das Programm und dessen Daten zu halten.
- Die Ausführung des Programmes involviert typischerweise einen Stack der verwendet wird um die Ausführung von Prozeduraufrufen und das Mitgeben von Parametern zu diesen ermöglicht.

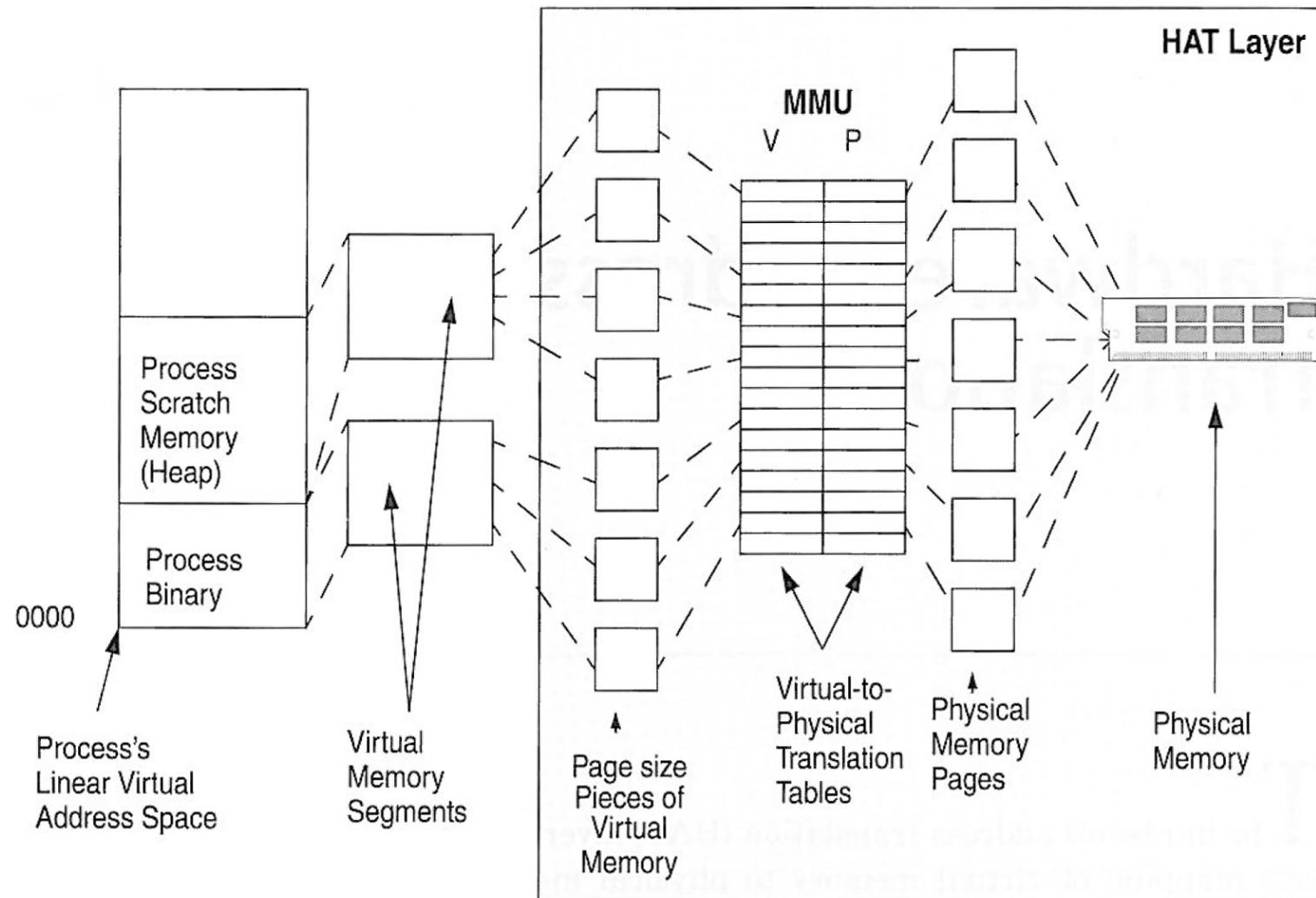
Prozess Attributes

- Jeder Prozess verfügt über eine Anzahl von Attributen die verwendet werden um dem OS die Kontrolle über diesen Prozess zu ermöglichen.
- Die Sammlung des Programms, Daten, Stack und Attributen wird als Prozess Image bezeichnet.
- Der Standort vom Prozess Image hängt vom Memory Management Schema in Verwendung ab.

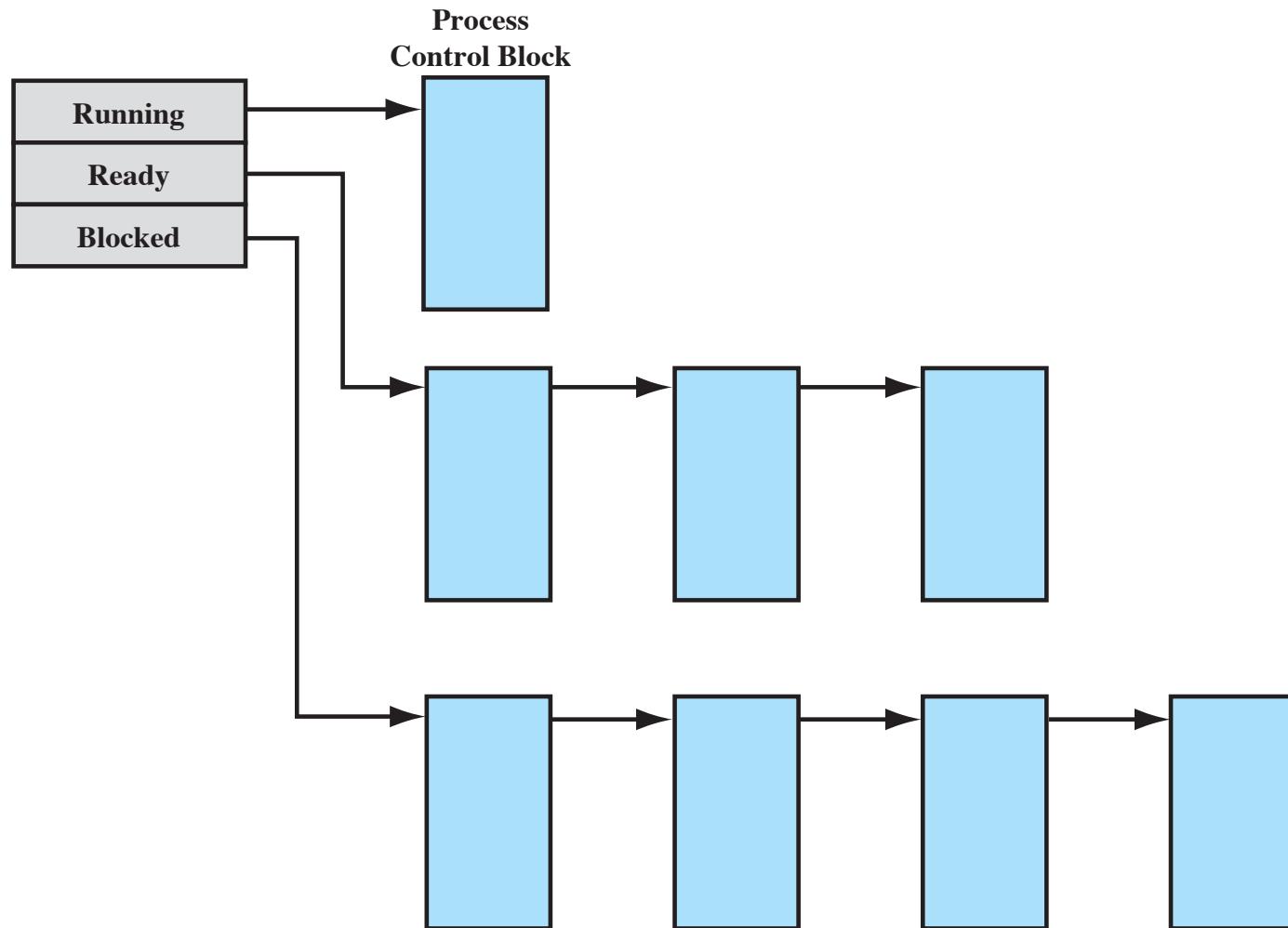
Prozess Images im virtuellen Memory



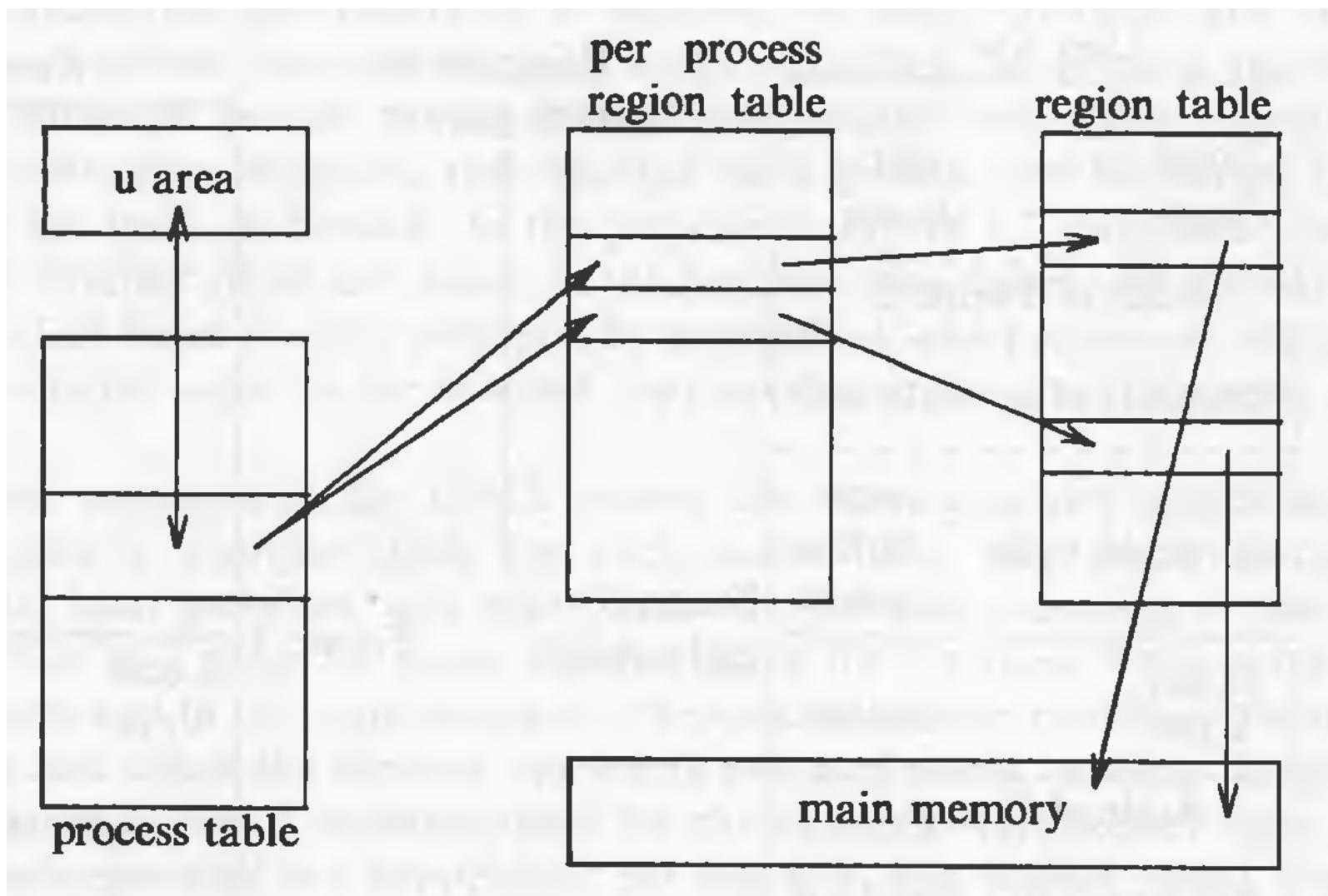
Virtuelle zu Physikalische Adress-Übersetzung



Prozess Control Block



Prozess Datenstrukturen



Modus Wechsel

Keine Interrupts liegen am Prozessor an:



Fahre weiter mit der “fetch stage” und hole die nächste Instruktion vom aktuellen Programm im aktuellen Prozess

Wenn ein Interrupt am Prozessor anliegt:



Setzt den Programm-Counter auf die Startadresse vom Interrupt Handler Programm



Wechselt vom User Modus in den Kernel Modus, so dass der Interrupt Handler Code damit auch privilegierte Instruktionen ausgeführt werden können.

Prozess Status Wechsel

Die Schritte eines ganzen Prozess Wechsels sind:

Speichere den Kontext vom Prozessor

Update vom Prozess Kontrol Block vom Prozess momentan im "run" Status.

Verschiebe den Prozess Kontrol Block zur richtigen Queue.

Falls der momentan laufende Prozess in einen anderen Status verschoben wird (Ready, Blocked, etc.), dann muss das OS substantielle Wechsel in seinem Environment machen.

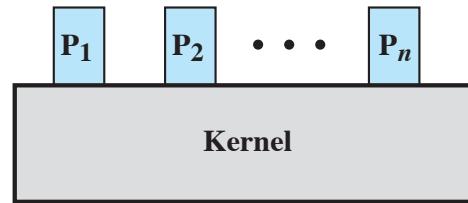
Restore vom Prozessor Kontext, so dass er genau da weiterfährt als der Prozess weggenommen wurde.

Update vom Prozess Control Block vom ausgewählten Prozess. Update Memory Management Data Strukturen.

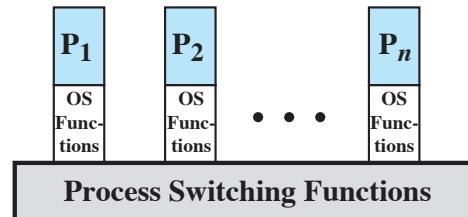
Wähle einen anderen Prozess fuer die Ausführung aus.

Ausführung im OS

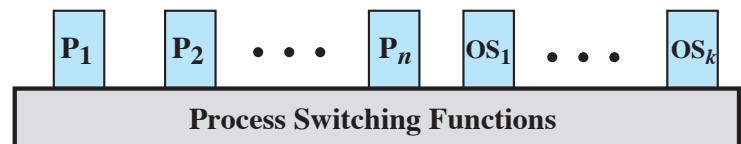
- a. **Traditionell:** Der Kernel wird ausserhalb jedes Prozesses ausgeführt. Der OS Code wird als besondere Entität im privilegierten Modus ausgeführt.
- b. **Execution innerhalb vom User Prozess:** Das OS ist wie eine Sammlung von Programmen die vom User im User Environment aufgerufen werden. Jedes Prozess Image beinhaltet: Programm, Data, Stack Bereiche fuer Kernel Programme
- c. Prozess basierende OS: Das OS wird als Sammlung von System-Prozessen implementiert. Wie bei den anderen Beispielen, wenn die SW ein Teil des Kernels ist, wird sie im Kernel Modus ausgeführt. Wobei hier die wichtigsten Kernel Funktionen als separate Prozesse organisiert sind.



(a) Separate kernel



(b) OS functions execute within user processes

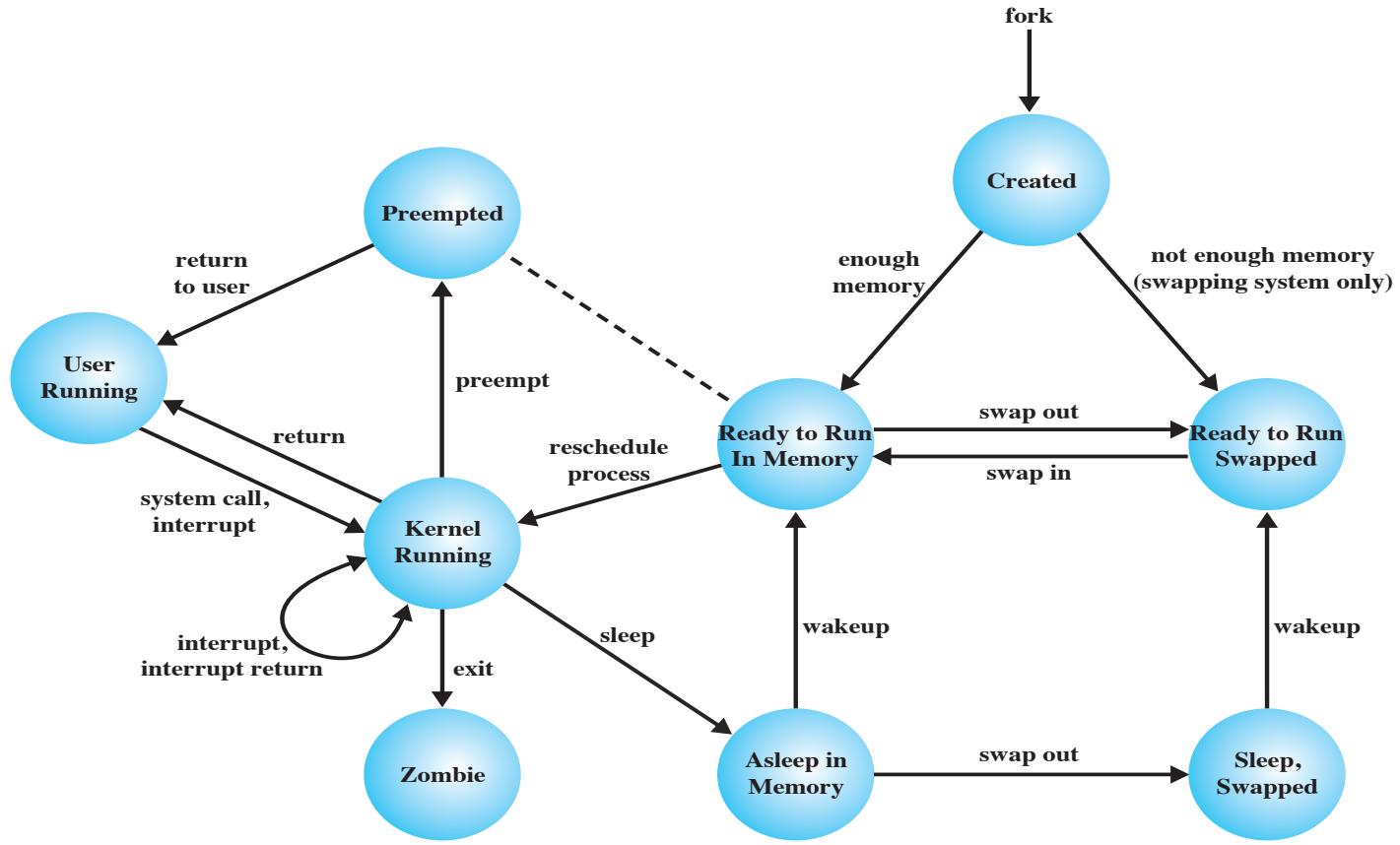


(c) OS functions execute as separate processes

UNIX Process Status

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

Prozesszustände und Übergänge



Auszug aus Stallings Modern Operating Systems

System Software

- System Struktur
- Prozess Elemente
- Prozess erzeugen
- Traps (Prozesse unterbrechen)
- Prozess Kontext
- Threads

Prozess erzeugen

- Prozess wird erzeugt durch den Kernel system call, `fork()`
- im Kernel Mode, OS macht:
 - 1 • Alloziere einen freien Platz in der Prozesstabelle fuer den neuen Prozess.
 - 2 • Vergib eine einzigartige Prozess ID fuer den Child Prozess.
 - 3 • Mach eine Kopie vom Parent-Prozess Image mit der Ausnahme vom “gesharteten” Memory.
 - 4 • Inkrementiere die Zähler für jedes File das vom Parent besitzt um anzugeben, dass ein zusätzlicher Prozess diese Files auch besitzt.
 - 5 • Füge den Child Prozess der “ready to run” Queue zu.
 - 6 • Retourniere die ID Nummer vom Child Prozess zum Vater Prozess und den Wert 0 zum Child Prozess.

Nach dem Erzeugen eines Prozesses

Nach der Prozess-Erzeugung kann der Kernel mir seiner Dispatcher Routine folgendes tun:

- Bleib im Parent Prozess.
- Transferiere die Kontrolle zum Child Prozess.
- Transferiere die Kontrolle zu einem anderen prozess.



System Software

- System Struktur
- Prozess Elemente
- Prozess erzeugen
- Traps (Prozesse unterbrechen)
- Prozess Kontext
- Threads

Unterbrechung der Ausführung eines Prozesses

Mechanism	Cause	Use
Interrupt	External to the execution of the current instruction	Reaction to an asynchronous external event
Trap	Associated with the execution of the current instruction	Handling of an error or an exception condition
Supervisor call	Explicit request	Call to an operating system function

Selbststudium

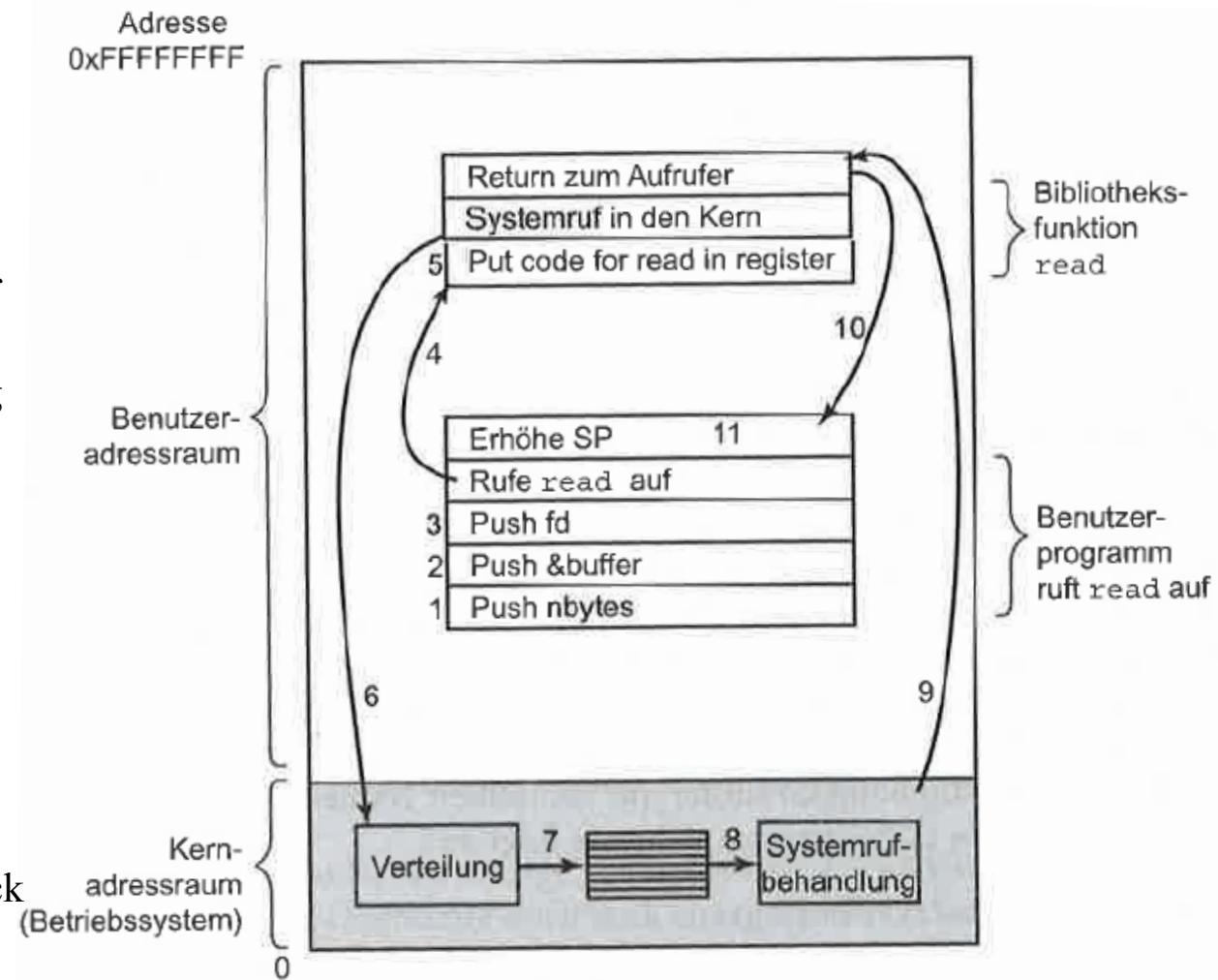
Bitte lesen sie das Papier: **Systemaufrufe.pdf**

Diese Lektüre ist essentiell um die Traps richtig zu verstehen. Traps sind Voraussetzung in der Vorlesung „Server Virtualisierung“ (HW Virtualisierung).

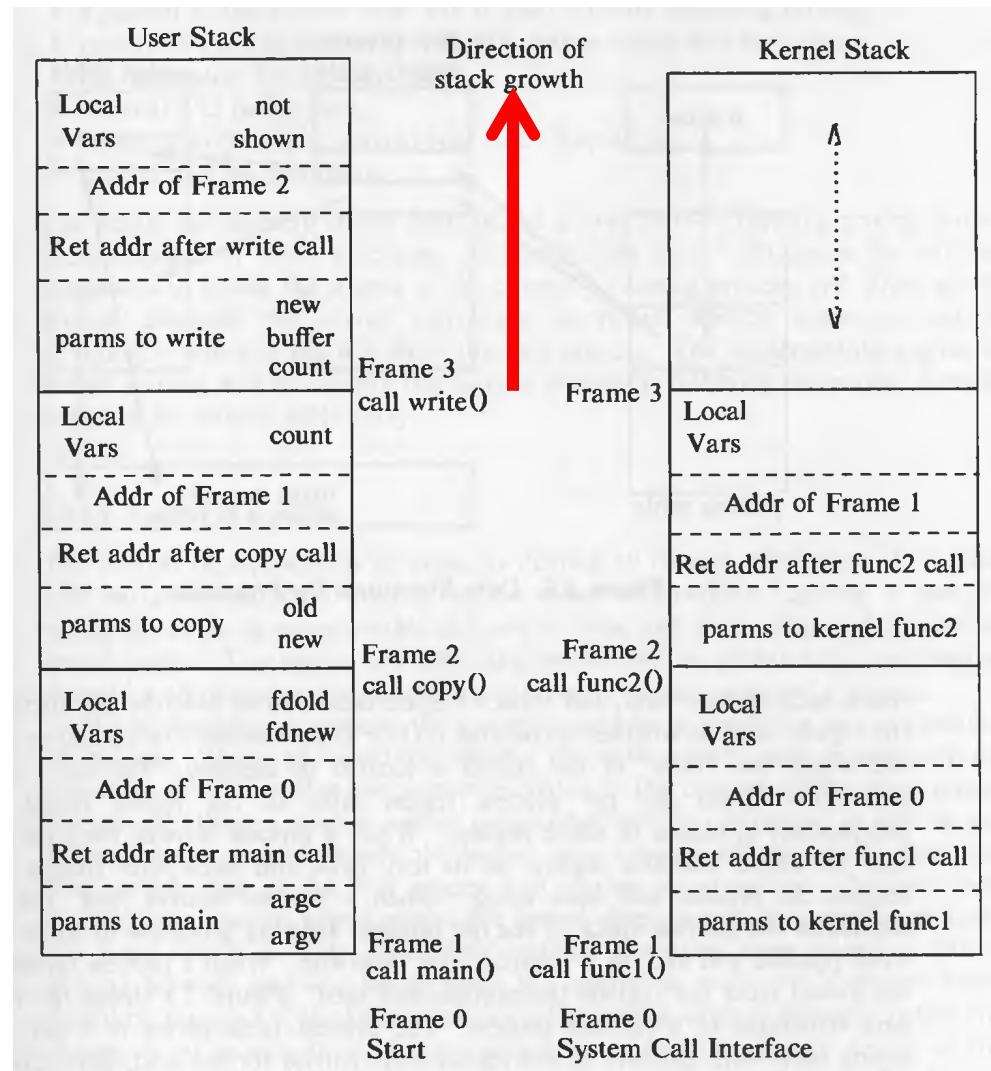
Weiterführende Lektüre finden sie im Paper:
„Interrupts, Traps.pdf“

System Call (Trap)

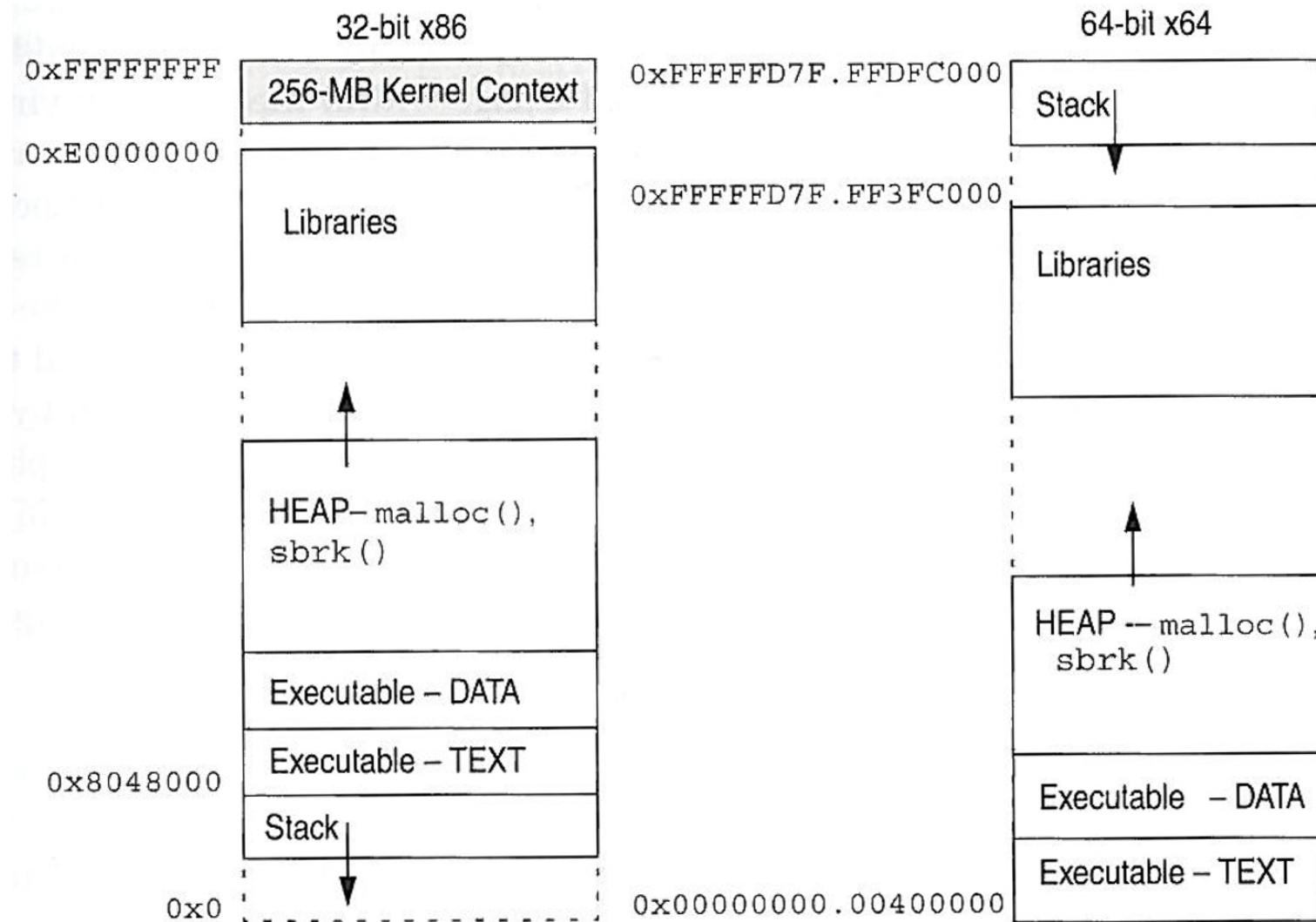
1. Aufrufendes Programm legt Parameter auf Stack
4. Sprung in Bibliotheksfunktion
5. Nummer v. Systemaufruf in Register
6. Trap Ausführung: Sprung an feste Adresse im Kern
7. Map: Nummer – Funktionszeiger
8. Start vom Systemaufruf
9. Beenden Systemaufruf: Kontrolle geht an Bibliotheksfunktion zurück
10. Kontrolle geht an Benutzerprogramm zurück
11. SP wird erhöht, User Programm läuft weiter



Benutzer- und System Stack



Prozess Adressraum



System Software

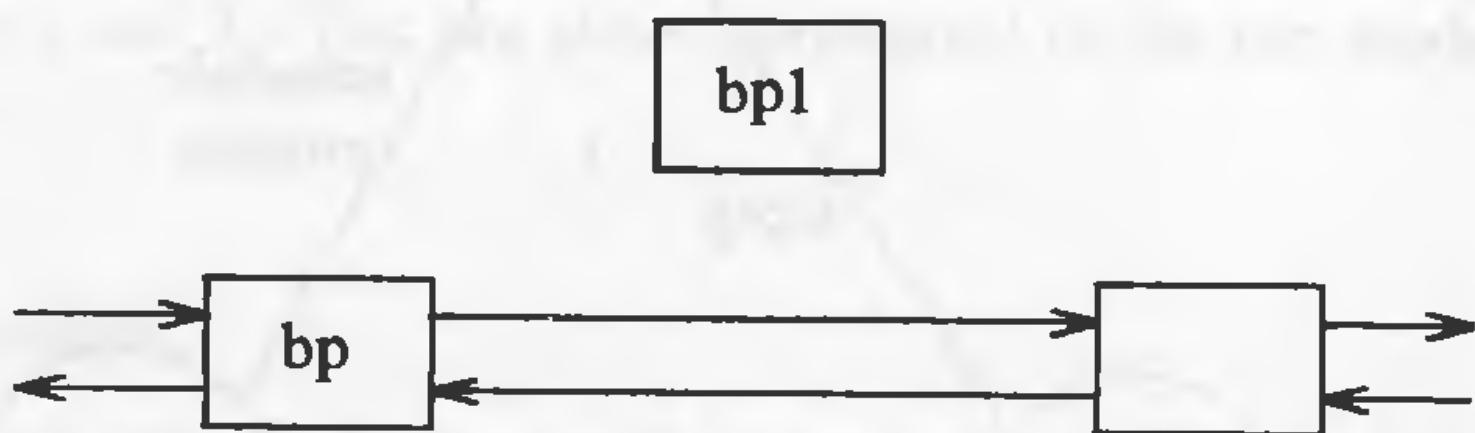
- System Struktur
- Prozess Elemente
- Prozess erzeugen
- Traps (Prozesse unterbrechen)
- Prozess Kontext
- Threads



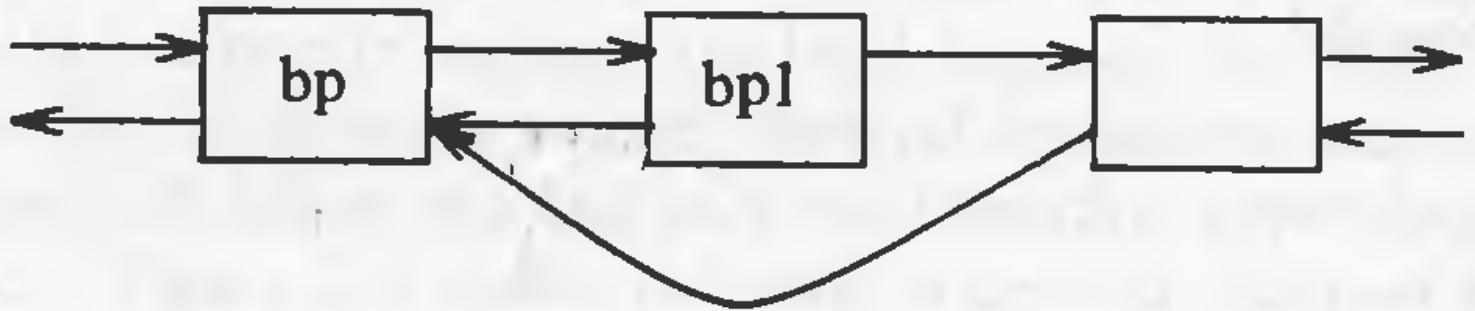
Prozess Kontext

- Benutzer Kontext
 - zugewiesener Adressraum und Daten
- Hardware Kontext (multitasking relevant)
 - Inhalte CPU Register
 - weitere wichtige Infos wie z.B: Seitentabelle
- System Kontext (Sicht OS)
 - Prozessnummer, geöffnete Dateien, Info Eltern und Kind Prozess, Prioritäten, etc.

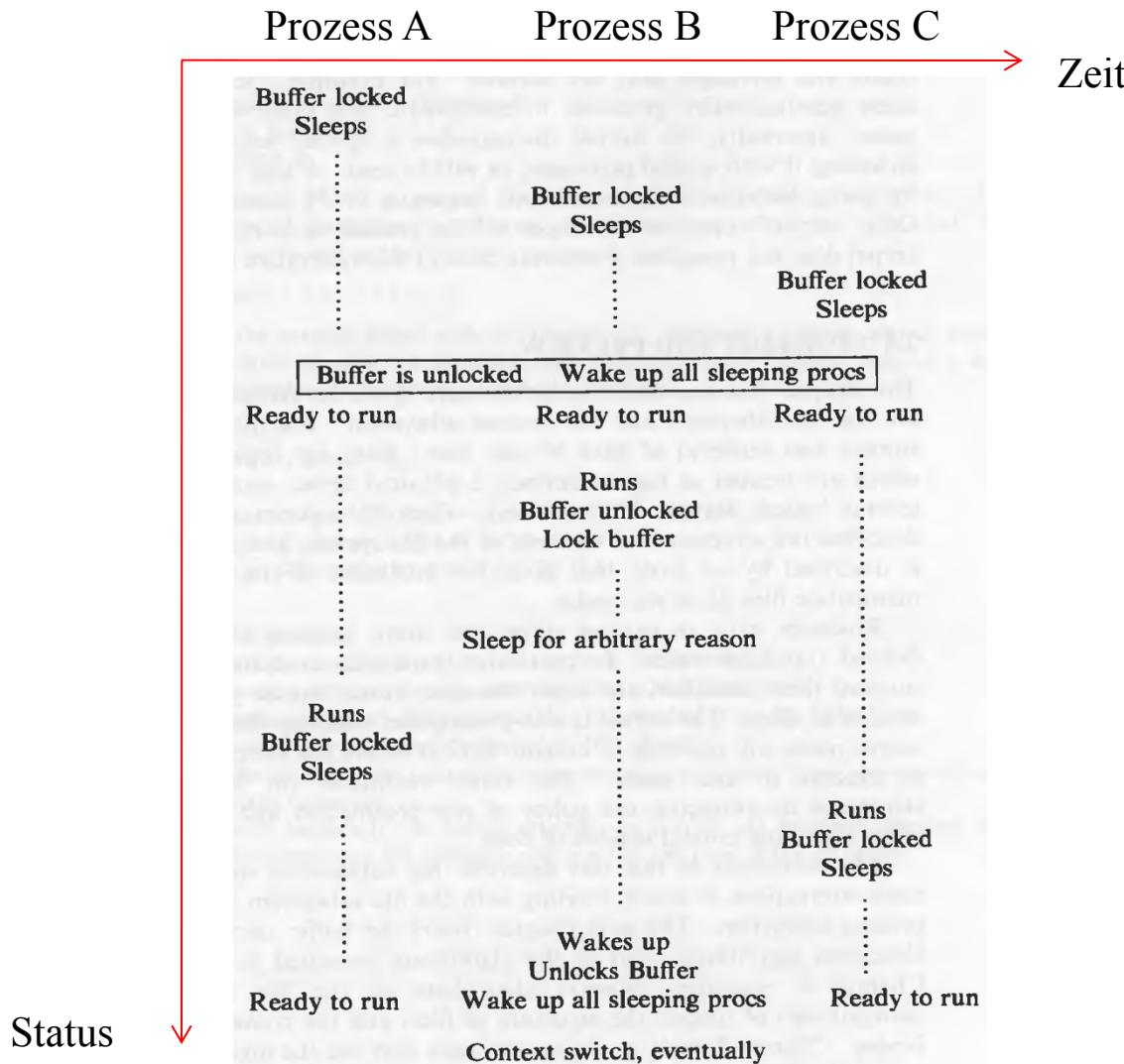
Auswirkungen Kontextswitch



Placing bpl on doubly linked list



Prozesse im Schlaf vor einer Sperre



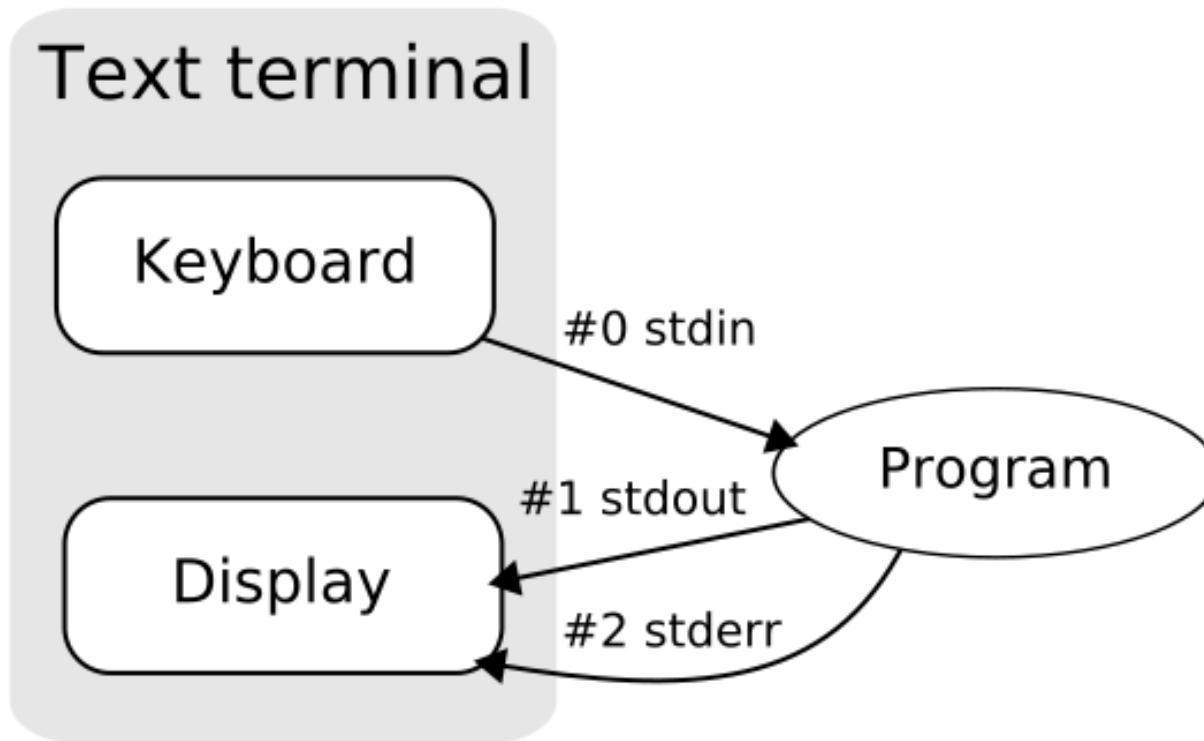
Kurze Übung fuer's Selbststudium

File Deskriptoren:

Jedes geöffnete File bekommt eine ID

<http://www.catonmat.net/blog/bash-one-liners-explained-part-three/>

Standard File Descriptors



Jeder Prozess hat seine eigene File Deskriptoren Tabelle. welch Pointer enthält zu allen I/O Streams

Quelle: Wiki.org

File Descriptor ausprobieren

Machen sie ein paar der im Papier: “file descriptors.pdf” aufgezeigten Beispiele für File Deskriptor Manipulationen (“umbiegen”)

Sie können das auf folgenden hosts machen:

1. testbed01.el.eee.intern
2. testbed02.el.eee.intern
3. testbed03.el.eee.intern

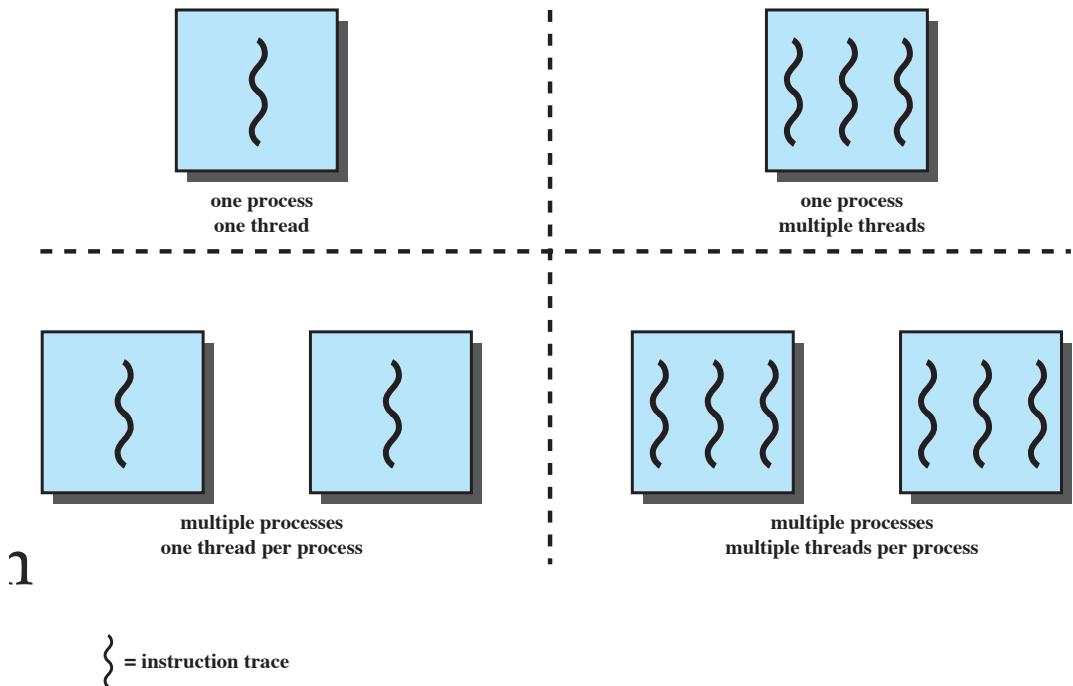
System Software

- System Struktur
- Prozess Elemente
- Prozess erzeugen
- Traps (Prozesse unterbrechen)
- Prozess Kontext
- Threads

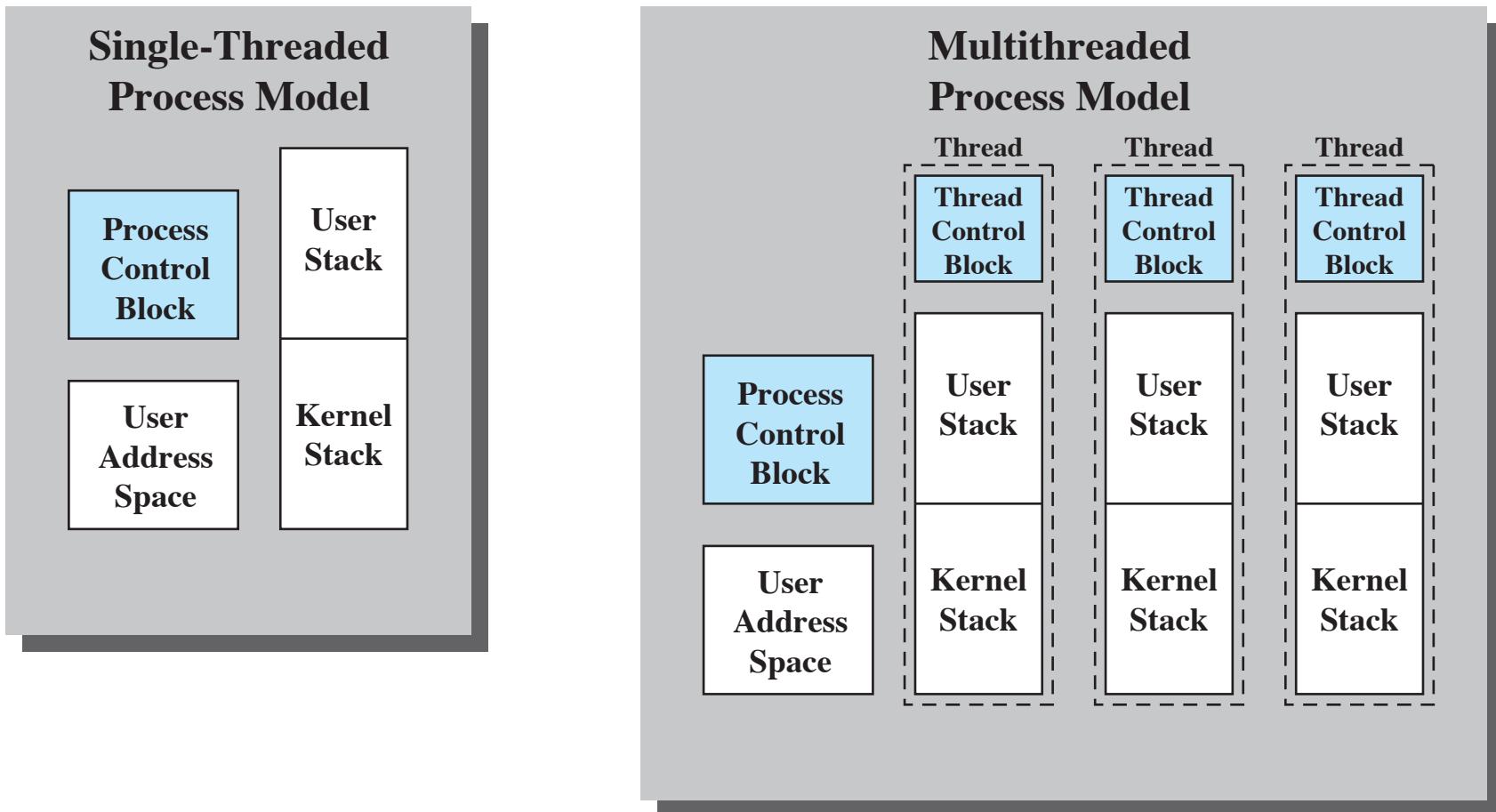


Multithreaded Ansatz

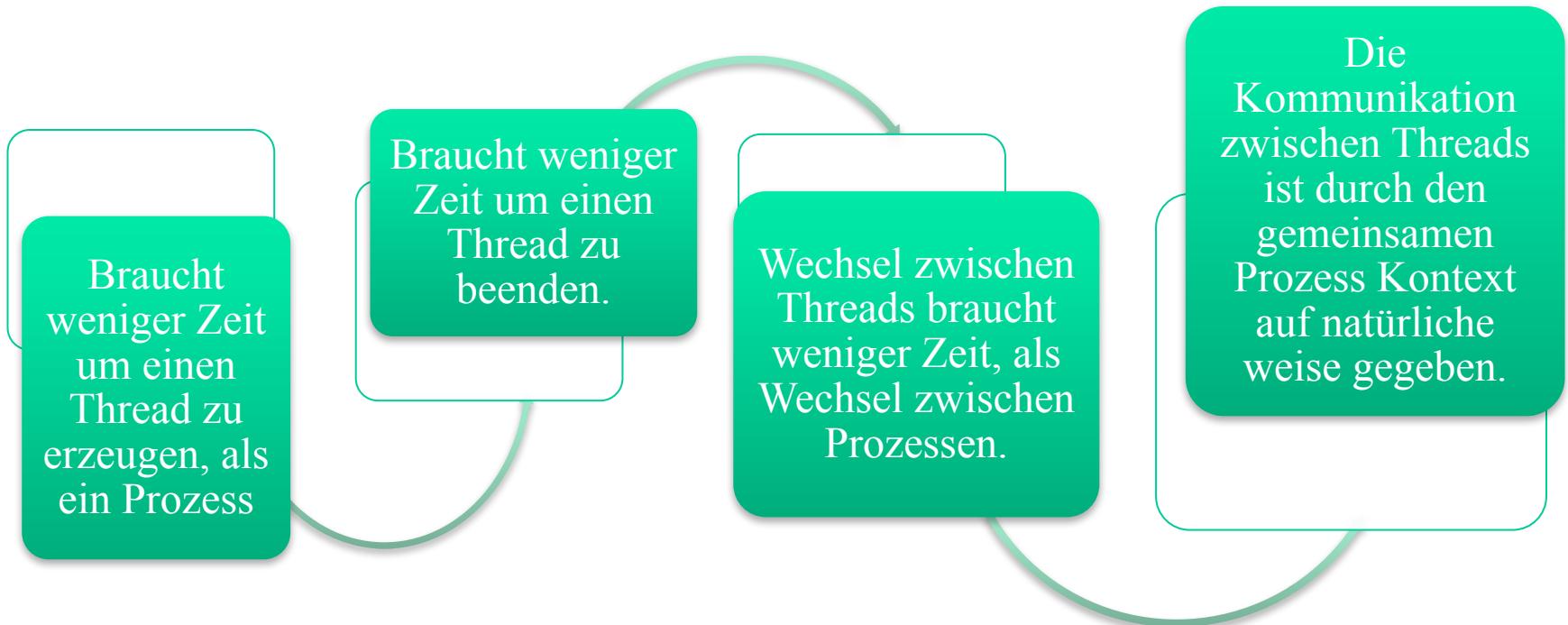
- Die rechte Hälfte zeigt den multi Threaded Ansatz
- Die Java Run Time Umgebung ist ein Vertreter von einem Prozess mit mehreren Threads



Single und multi Thread Prozess Modell



Vorteile von Threads



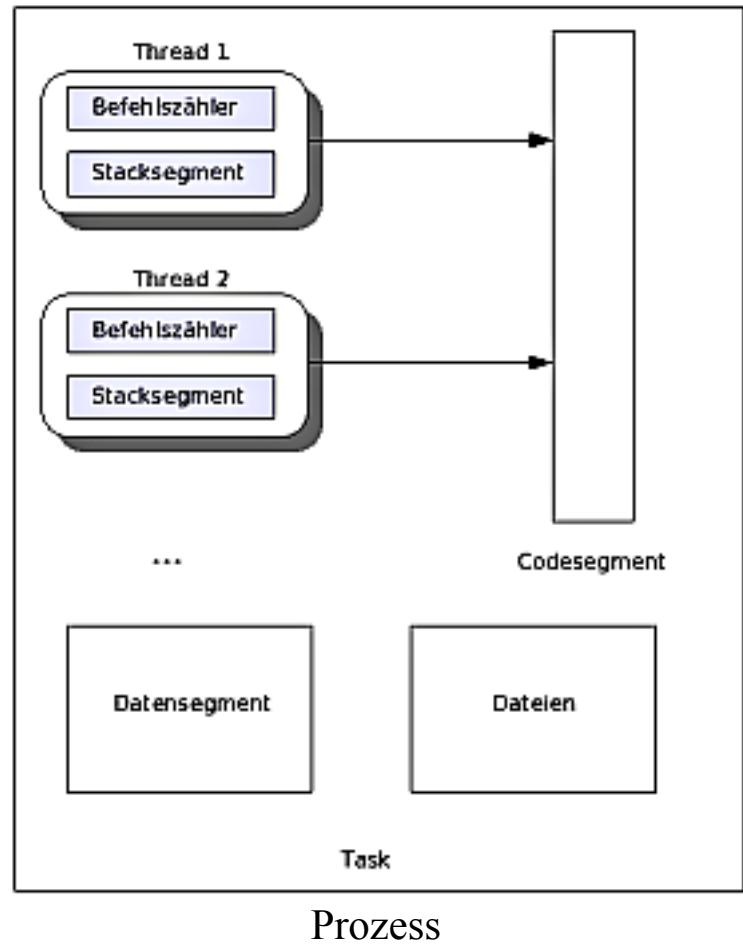
Threads

- In einem modernen OS mit Thread Support wird das Scheduling and Dispatching auf der Basis von Threads gemacht.
- Die meiste Status Information die für die Ausführung eines Threads verwendet werden sind in den Thread-Level Daten Strukturen abgelegt.
- Einen Prozess zu unterbrechen bedingt die Unterbrechung aller Threads (in diesem Prozess).
- Einen Prozess zu terminieren heisst, dass alle Threads im Prozess beendet werden.



Thread Kontext

- Ein Prozess besteht aus mindestens 1 Thread
- Threads vom gleichen Prozess teilen sich:
 - Codesegment
 - Datensegment
 - Dateideskriptoren
- Jeder Thread hat einen eigenen Stack
- Jeder Thread hat seinen eigenen Code Pointer

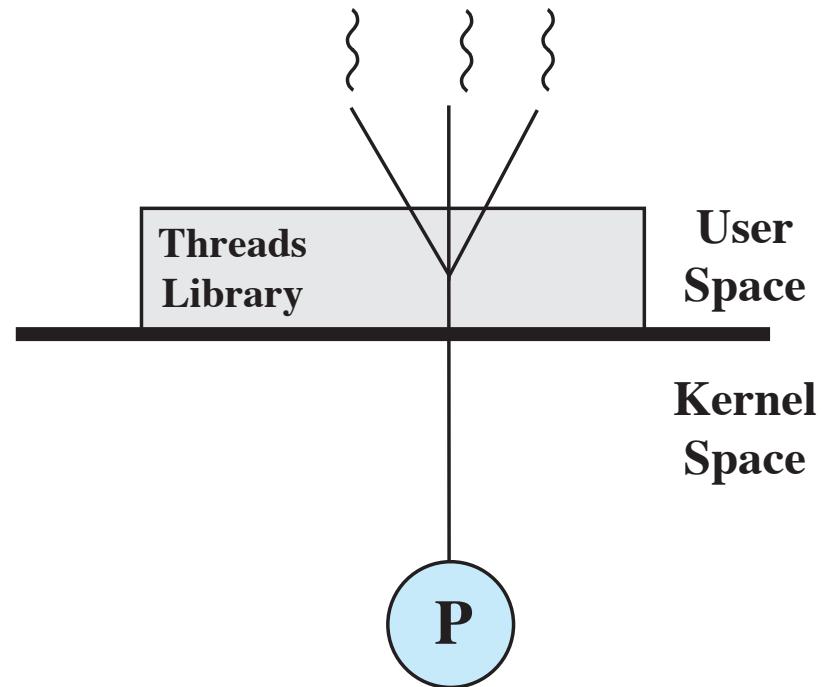


Es gibt 2 Arten von Threads

- User level Threads
- Kernel level Threads

User-Level Threads (ULTs)

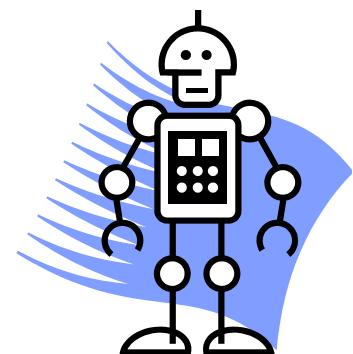
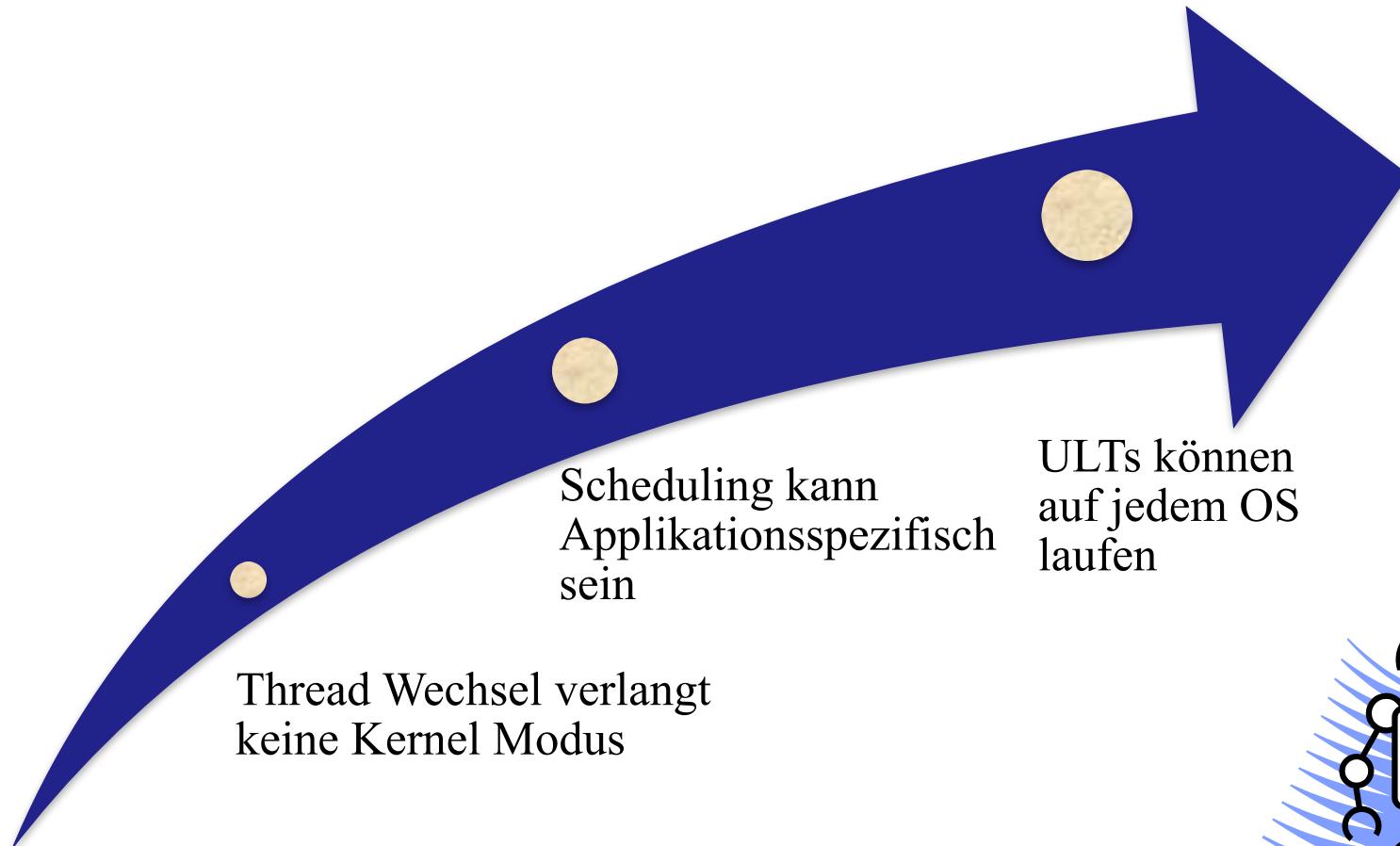
- Thread Management wird in der Applikation gemacht
- Der Kern hat keine Kenntnis von der Existenz eines Threads.



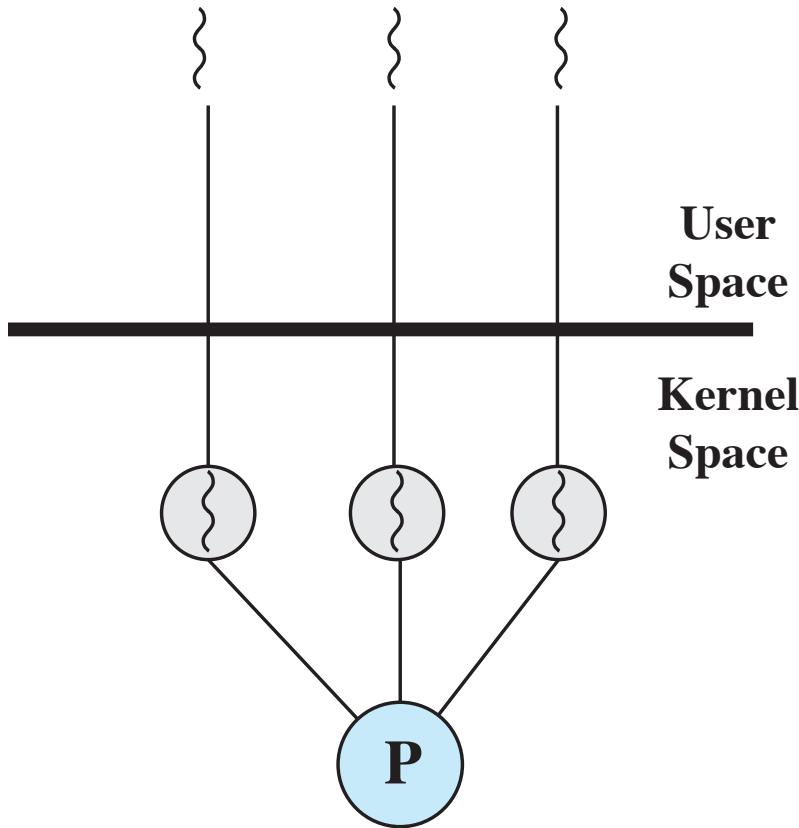
User Thread

- Werden durch eine Library (vom Programmierer) implementiert
- Der Kern hat keine Kenntnis davon
- Müssen Kontrolle von sich aus abgeben
- Neigen mehr zu kooperativem als zu preemptivem Scheduling
- Implementieren manchmal das Scheduling selber (z.B: frühe Versionen von Java)

Vorteile ULTs



Kernel-Level Threads (KLTs)

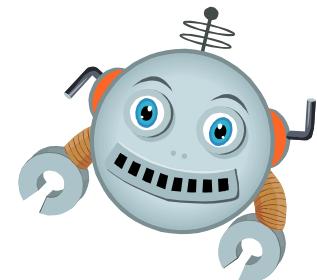


Thread Management wird vom Kern gemacht.

- Kein Thread management von der Applikation.
- Windows verfolgt diesen Ansatz.

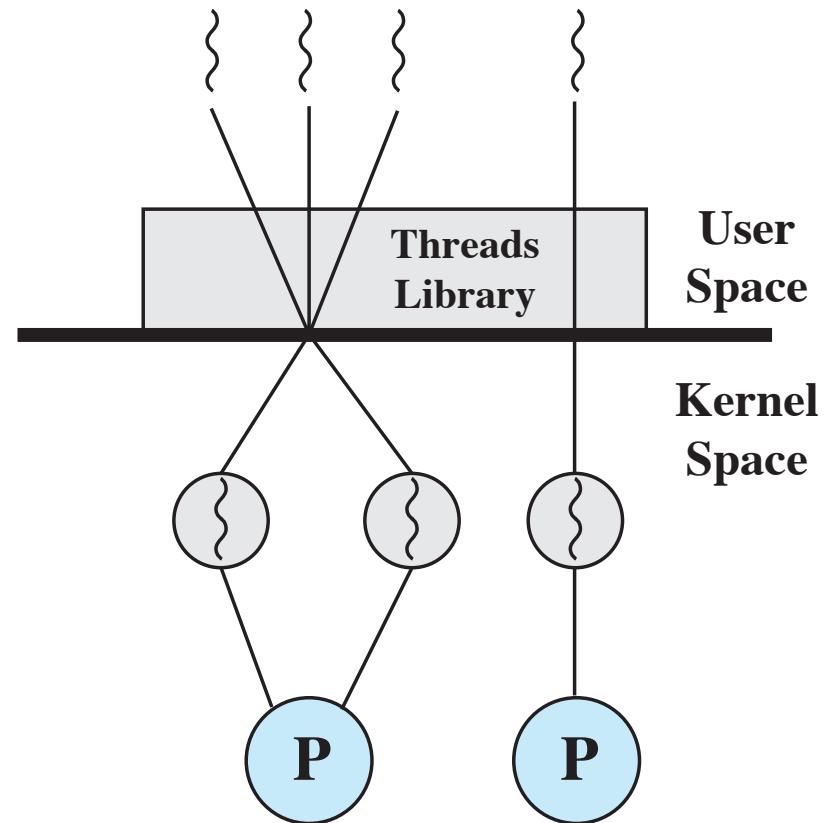
Advantages of KLTs

- Der Kern kann mehrere Threads vom gleichen Prozess auf mehrere Prozessoren verteilen (Scheduling)
- Wenn ein Thread blockiert ist kann der Kern einen anderen Thread vom gleichen Prozess berücksichtigen.
- Kern Prozeduren können multithreaded ausgeführt werden.



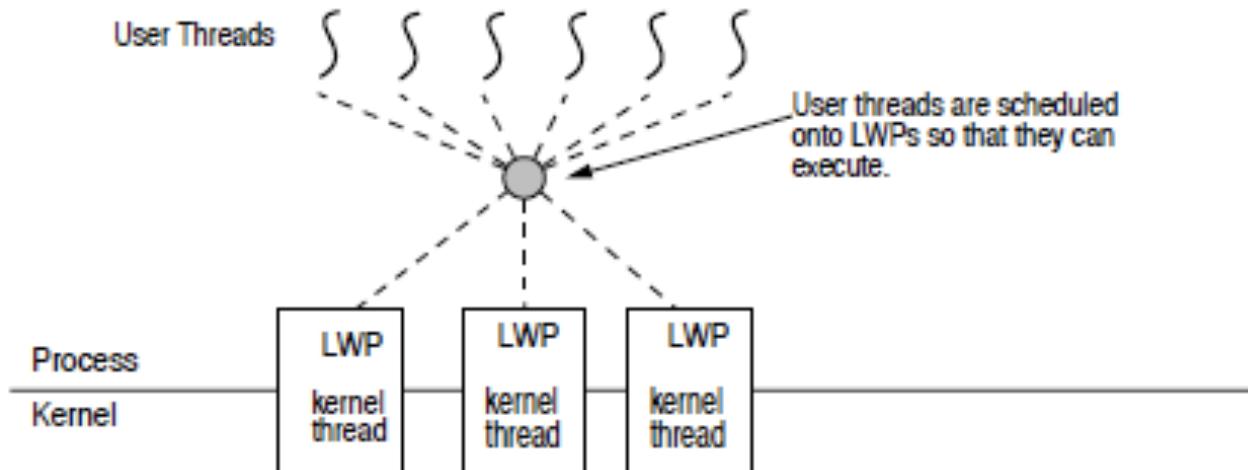
Kombinierter Ansatz

- Thread Erzeugung wird im User Modus gemacht.
- Der Grossteil vom Scheduling und Synchronisation vom Thread wird von der Applikation gemacht.
- Solaris ist ein Vertreter dieses Ansatzes.



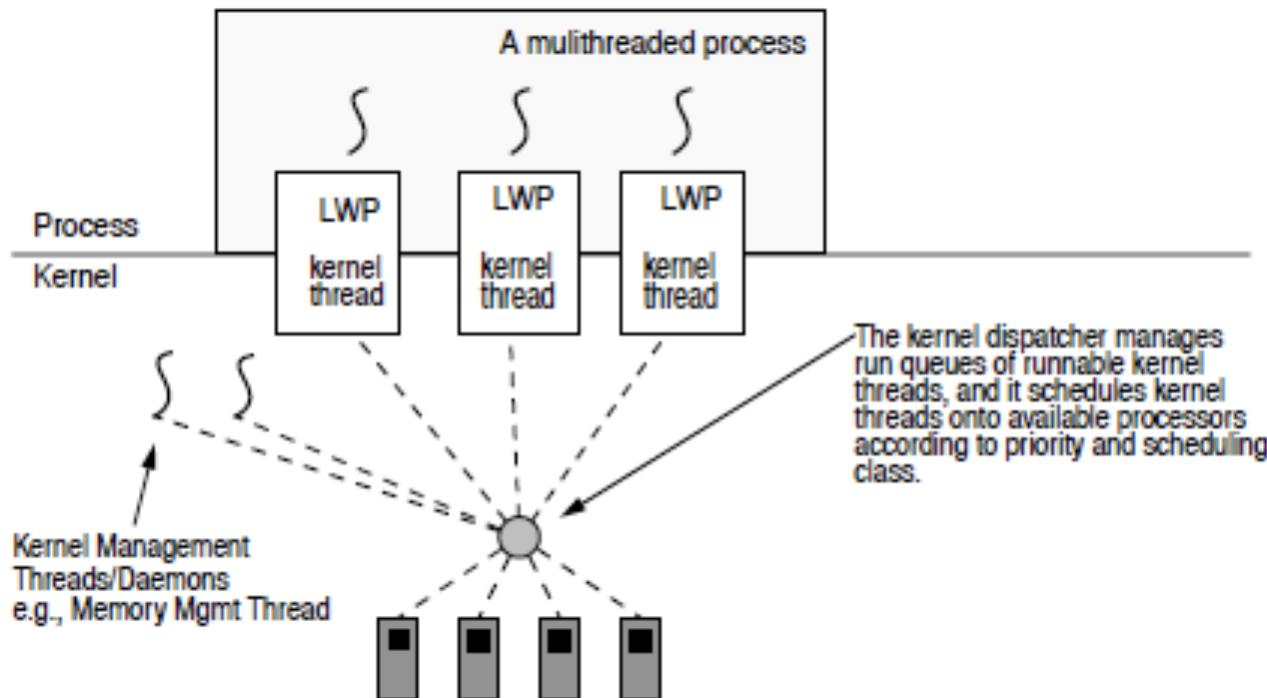
Zwei Level Thread Modell

- Obwohl Threads schnell verlegt werden, sind erzeugen/löschen teure Operationen und jeder Thread braucht einen Stack.
- Daher wurde weitere Schicht Thread Management eingeführt
- User Threads werden den LWPs zugeführt.
- Nur die Threads auf LWPs sind aktiv.

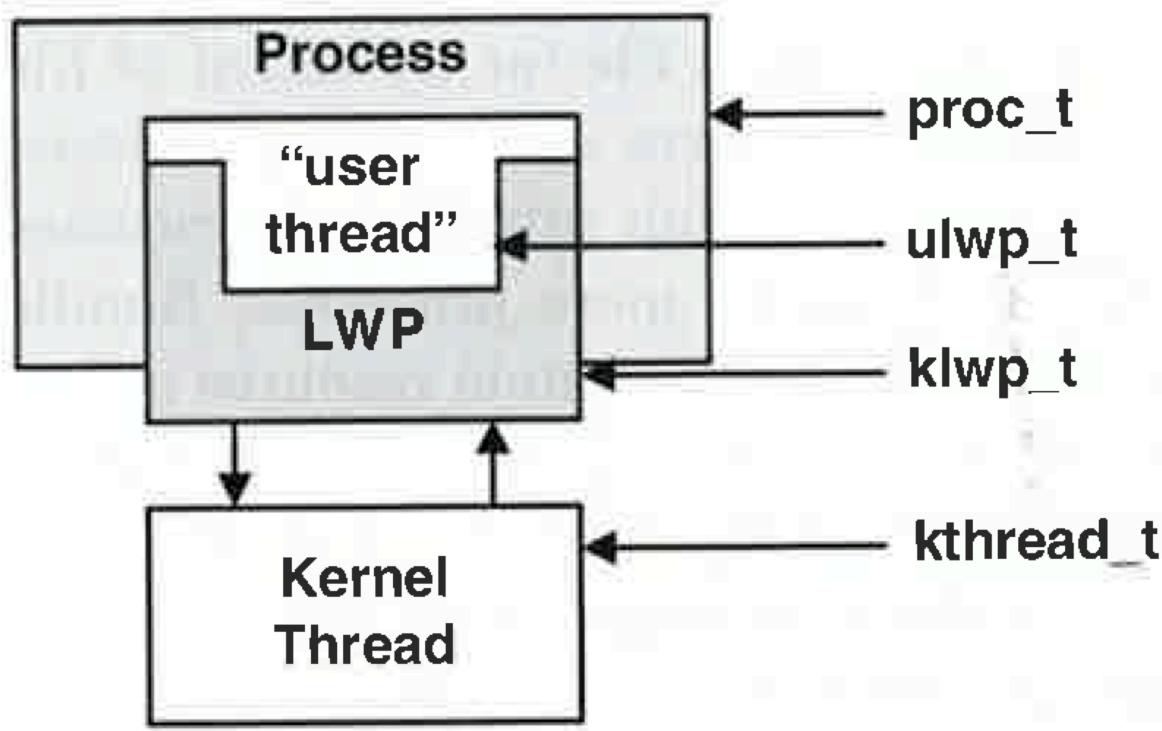


Kernel Threads

- Fundamentale Einheit die zum Prozessor verlegt wird.
- Switching Kernel Threads geht sehr schnell.
- Werden für Kernel Tasks verwendet (process execution, interrupt handling)

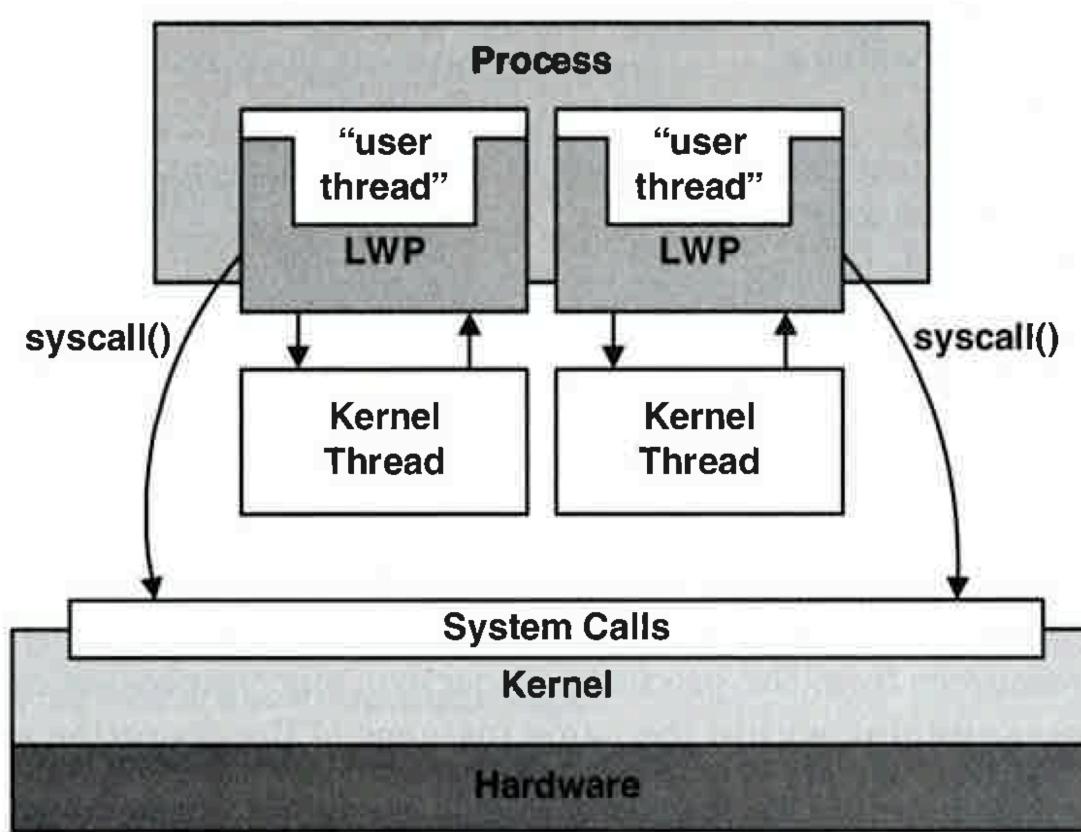


Thread Objekte



Für multithreaded execution verwendet das OS ausschliesslich Kernel Threads. Dazu formieren User-Threads zusammen mit LWPs (light weight process) Kern Objekte. Das erlaubt dem User-Thread ausgeführt zu werden, unabhängig von anderen Threads im gleichen Prozess.

System Calls



Das API direkt zwischen User-Prozessen und dem Kern wird Syscall genannt.

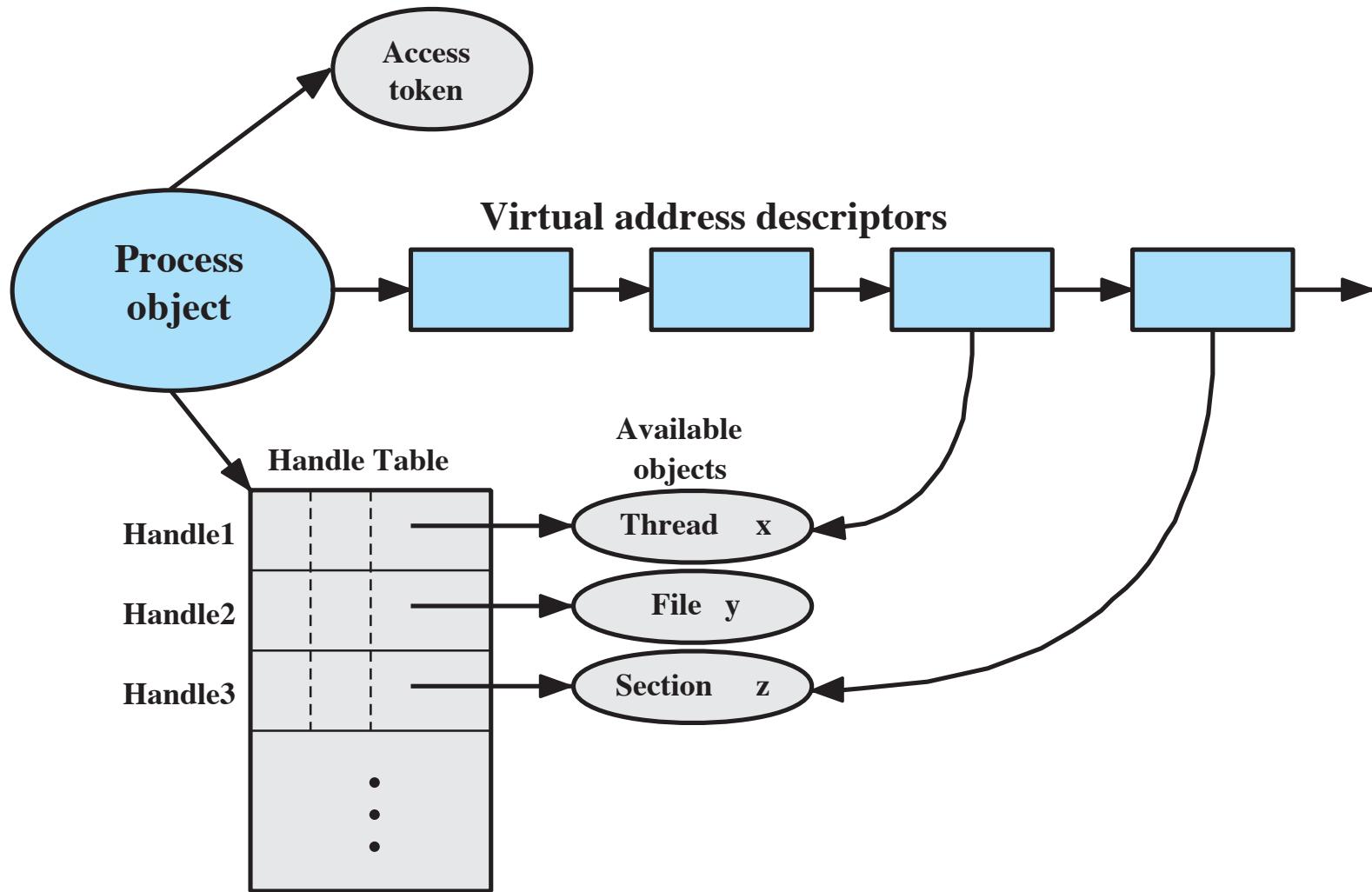
Windows Prozesse

Die vom Windows Kern zur Verfügung gestellten Prozesse und Services sind relativ simpel und generell nutzbar.

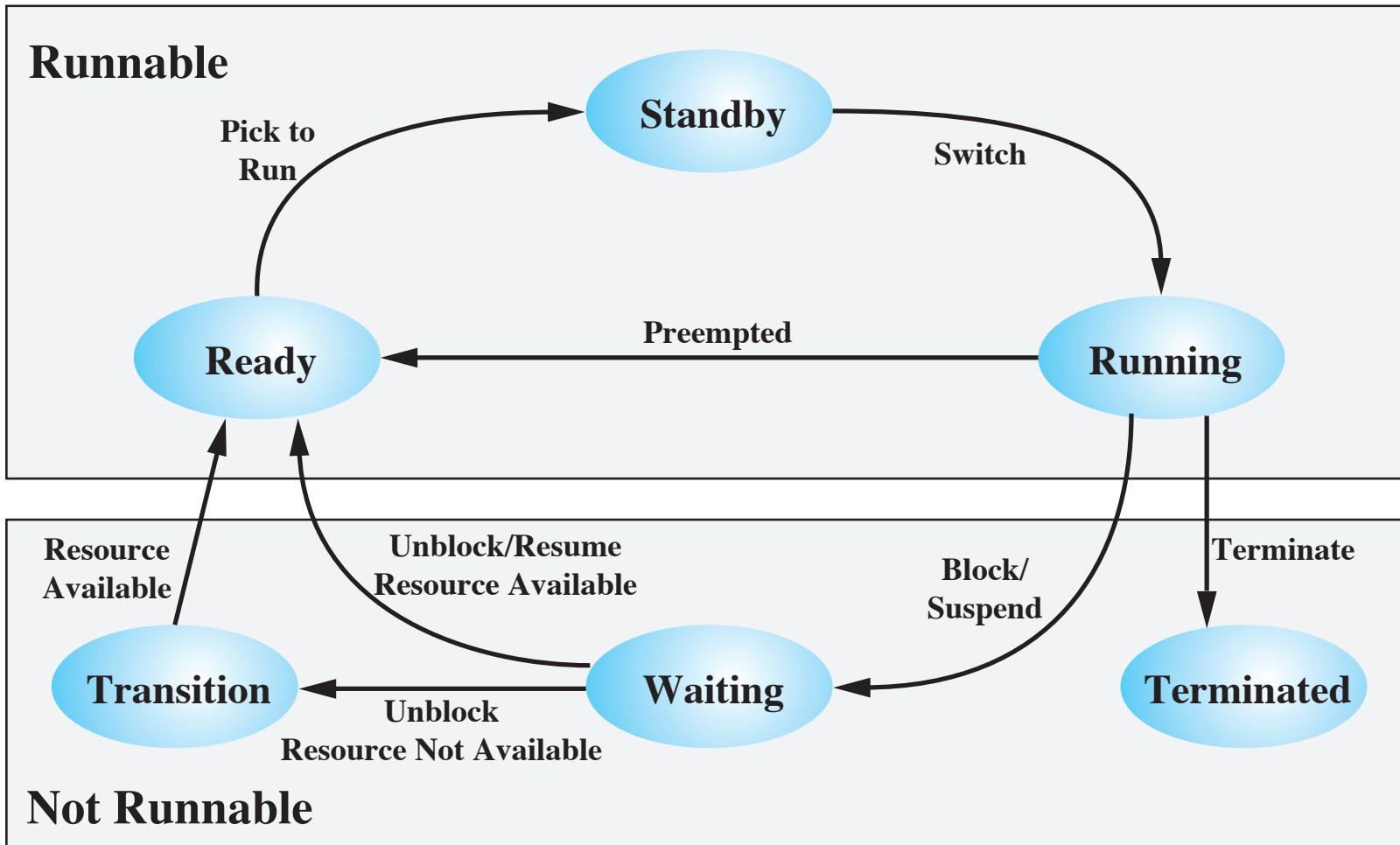
- Implementiert als Objekte.
- Werden als neue Prozesse oder als Kopie eines existierenden Prozesses erzeugt.
- Ein ausführbarer Prozess kann ein oder mehrere Thread beinhalten.
- Beide, Prozesse und Threads Objekte haben eingebaute synchronisations Mechanismen.



Windows Prozesse und deren Ressourcen



Windows Thread Status



Lucerne University of
Applied Sciences and Arts

**HOCHSCHULE
LUZERN**

Engineering & Architecture



Questions?

Referenzen

- Betriebssysteme, William Stallings, Pearson, ISBN: 3-8273-7030-2
- Solaris Internals, Mauro & McDougall, ISBN: 0-13-148209-2
- Solaris Performance & Tools, Mauro & McDougall, ISBN: 0-13-156819-1
- Fork and Exec:
<http://stackoverflow.com/questions/1653340/exec-and-fork>