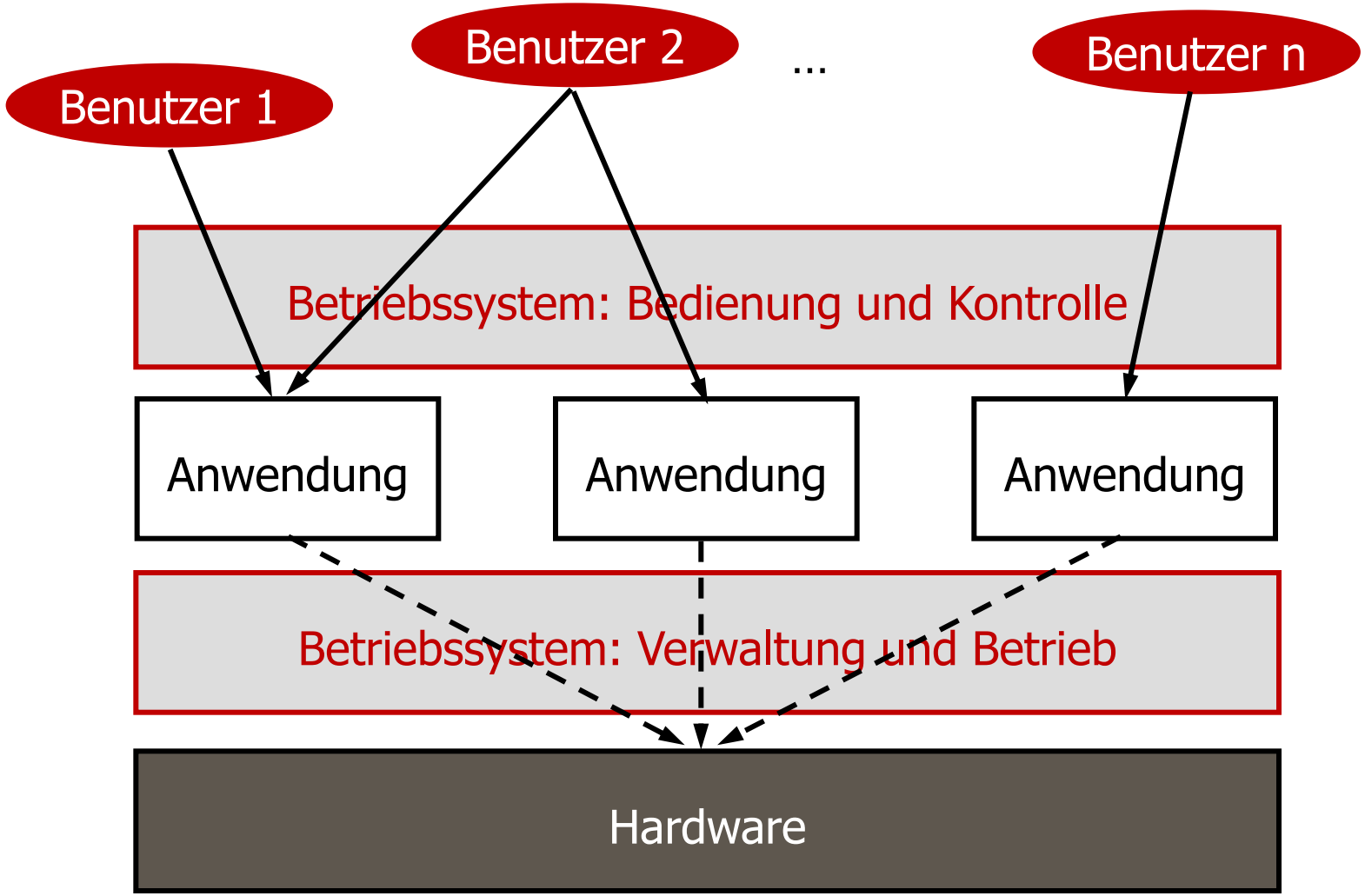
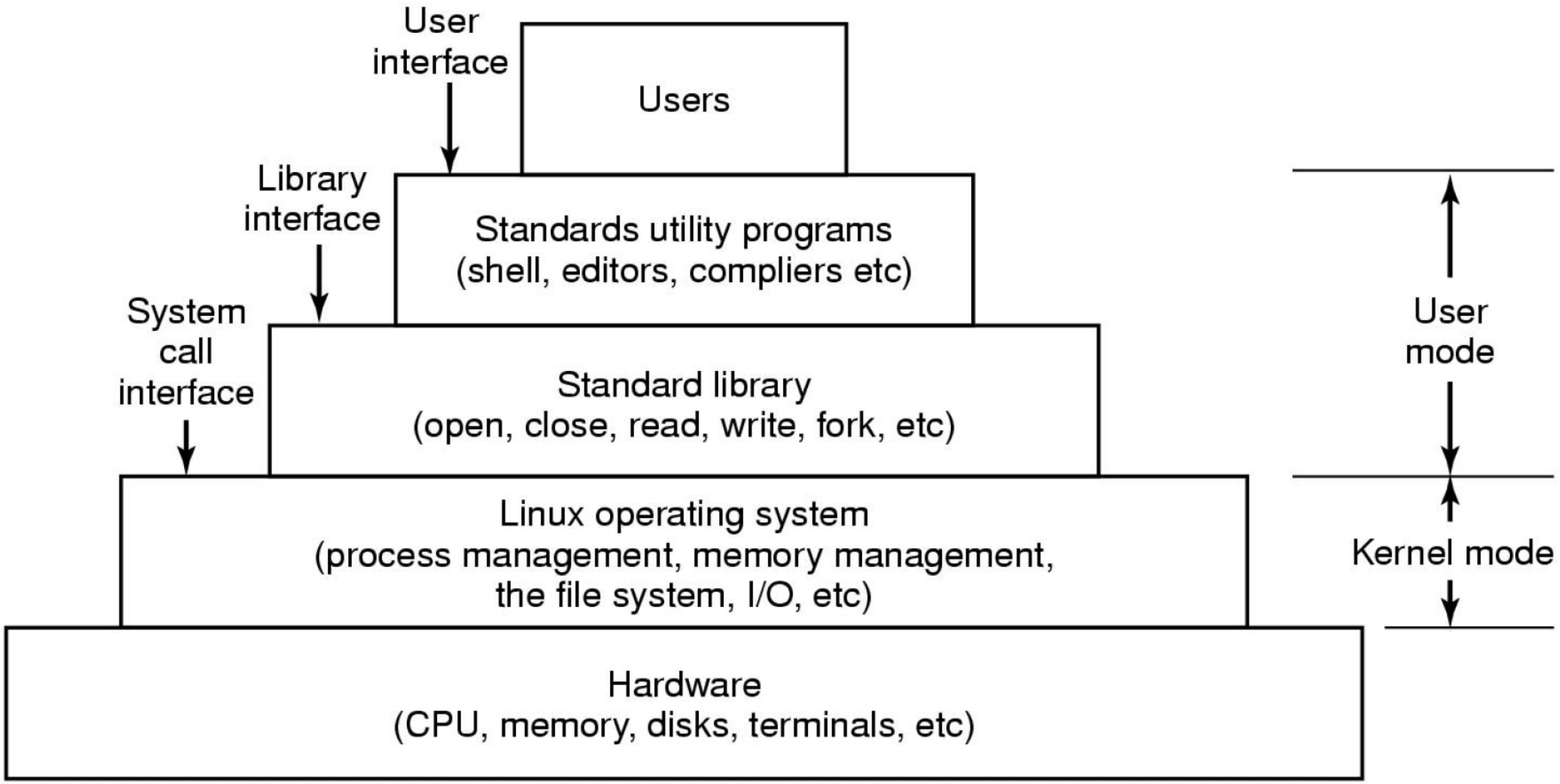
2. Betriebssysteme und Prozesse

- Überblick
  - 2.1 Systemaufrufe und Interrupts
  - 2.2 Prozesse
  - 2.3 Threads
  - 2.4 Prozesshierarchien
  - 2.5 Shell

Zusammenhang Hardware, Benutzer und Betriebssystem



Ebenen und Zugänge



Definition Betriebssystem

- BS als Mittler zwischen Programmen und Hardware
  - Bereitstellung von Hilfsmitteln für Benutzerprogramme
  - Abstraktion von HW-Eigenschaften und SW-Komponenten, wie z.B. Gerätetreiber
  - Koordination, Vergabe der Ressourcen an mehrere Benutzer
- Basiskatalog von Funktionen für verschiedene BS identisch, Unterschiede in Umfang und Art der Implementierung
  - Urbrechungsverarbeitung (interrupt handling)
  - Prozessumschaltung
  - Betriebsmittelverwaltung (resource management)
  - Programmallokation (program allocation)
  - Dateiverwaltung (file management)
  - Auftragsteuerung (Scheduling)
  - Zuverlässigkeit (reliability)

Mechanismen und Methoden (Policies)

- Wichtige Unterscheidung zwischen Mechanismen und Policies
  - Mechanismus: Wie wird eine Aufgabe prinzipiell gelöst?
  - Policy: Welche Vorgaben/Parameter werden im konkreten Fall eingesetzt?
- Beispiel: Zeitscheibenprinzip
  - Existenz eines Zeitgebers zur Bereitstellung von Unterbrechungen → Mechanismus
  - Entscheidung, wie lange die entsprechende Zeit für einzelne Anwendungen / Anwendungsgruppen eingestellt wird → Policy
- Trennung wichtig für Flexibilität
  - Policies ändern sich im Laufe der Zeit oder bei unterschiedlichen Plattformen → Falls keine Trennung vorhanden, muss jedes Mal auch der grundlegende Mechanismus geändert werden
  - Wünschenswert: Genereller Mechanismus, so dass eine Veränderung der Policy durch Anpassung von Parametern umgesetzt werden kann

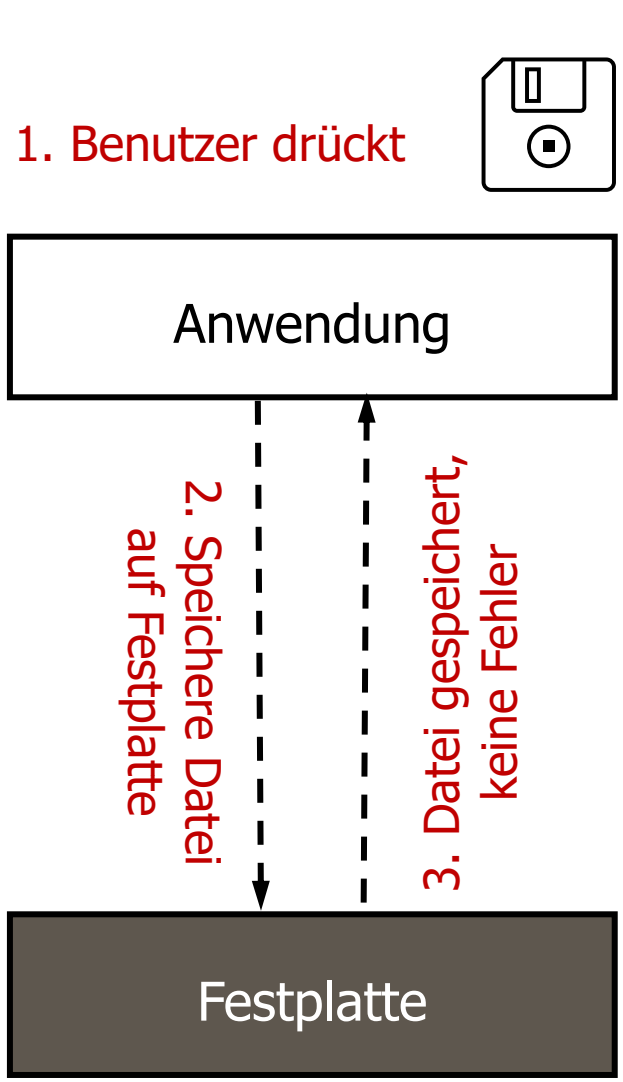
Benutzermodus vs. Systemmodus

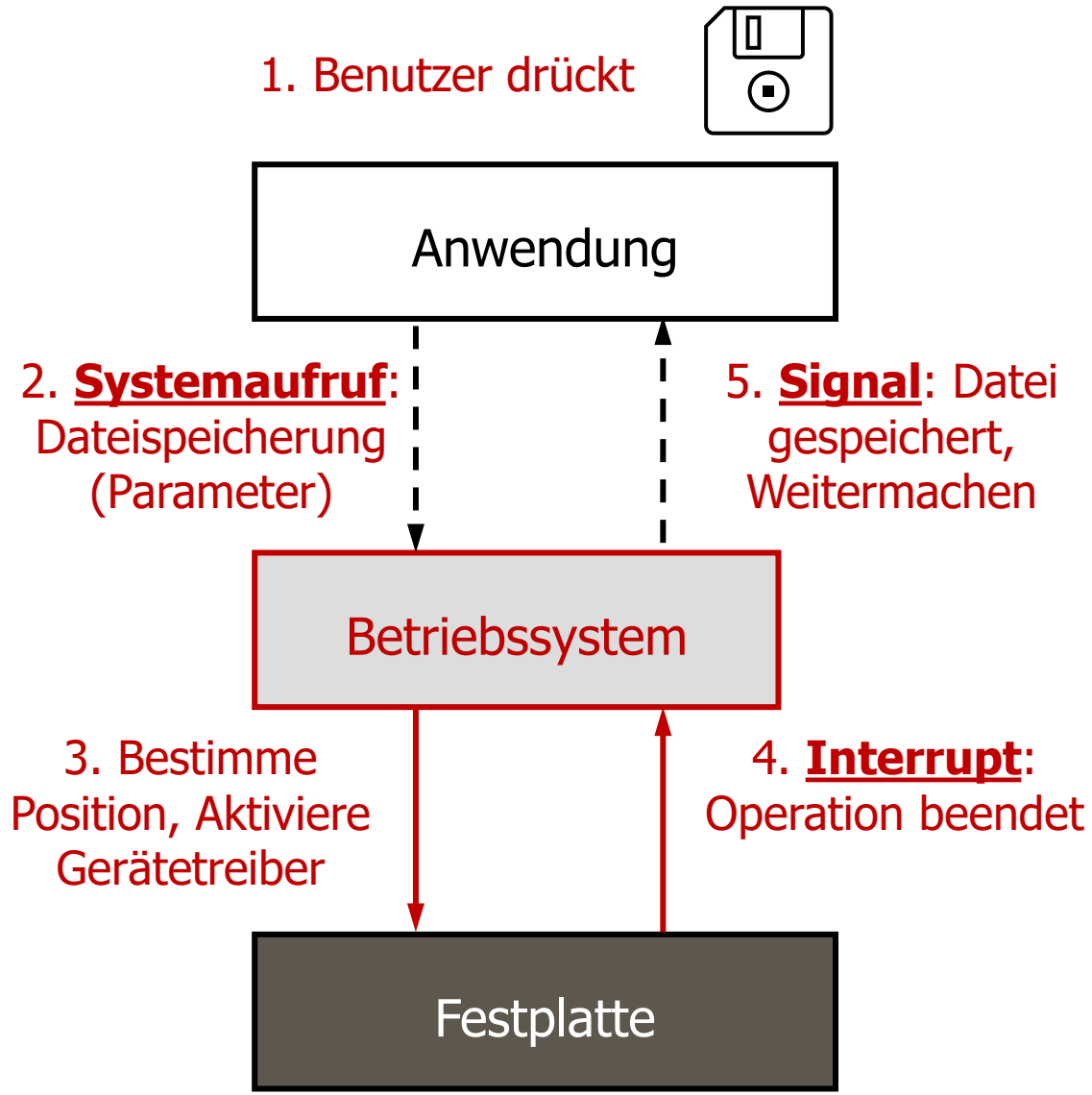
- Unterscheidung aus Sicherheitsgründen zwischen zwei Zuständen oder Modi (Bit im Prozessorstatusregister) der CPU und damit des Betriebssystems
  - Benutzermodus/unprivilegierter Zustand (user mode)
    - einige Instruktionen gesperrt
    - einige Register nicht zugreifbar
    - in der Regel für Benutzerprogramme
  - Systemmodus/privilegierter Zustand (system/supervisor mode, ...)
    - alle Instruktionen zulässig
    - alle Register benutzbar
    - in der Regel für das Betriebssystem

Wechsel zwischen den Modi

- unprivilegiert → privilegiert:
  - beim Auftreten einer Unterbrechung
  - beim Auslösen eines Fehlers (Division durch Null, Zugriffsversuch auf ein „Loch“ im Adressraum, verbotene Instruktion …)
  - durch explizite Instruktion (z. B. x86: sysenter, ARM: svc)
  - Ausführung wird an vom BS definierten Einsprungpunkten fortgesetzt
  - ursprünglicher Prozessorzustand (Register etc.) wird gesichert
- privilegiert → unprivilegiert:
  - jederzeit erlaubt
  - vom BS durchgeführt, um das unterbrochene Programm fortzusetzen

2.1 Kommunikation mit dem Betriebssystem





Systemaufruf

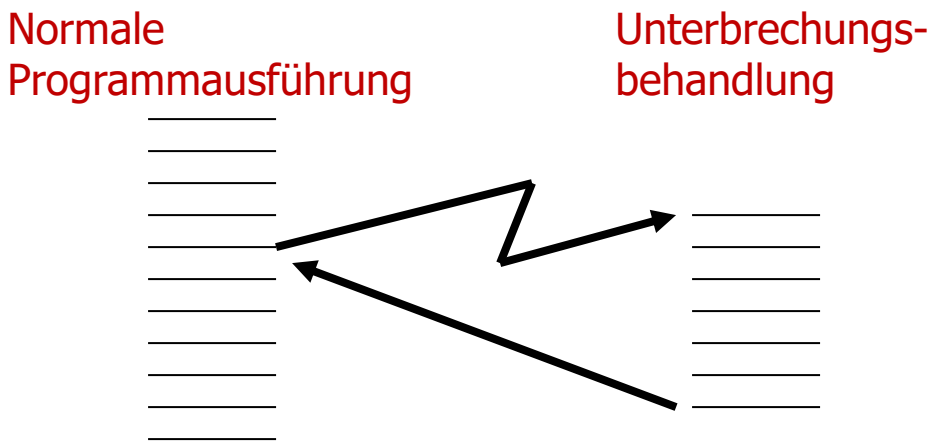
- BS bietet Funktionalität über „Systemaufruf“-Interface an
- Ablauf:
  1. Anwendung bereitet Systemaufruf vor (Register mit Parametern belegen, architekturspezifisch)
  2. Anwendung führt spezielle Instruktion aus (`svc`/...) ⟶ *Trap*
  3. Ausführung springt zu BS-Behandlungsroutine (⟶ privilegierter Modus!) für Systemaufrufe
  4. BS analysiert Parameter, identifiziert gewünschte Funktionalität
  5. BS prüft Berechtigung, Ressourcen, … führt ggf. gewünschte Funktion durch
  6. BS setzt Anwendung fort (Rückkehr in unprivilegierten Modus zur Instruktion, die der aus Schritt 2. folgt)

Beispielsignale UNIX

Signal	Bedeutung
SIGHUP	Hangup detected on controlling terminal or death of controlling process
SIGINT	Interrupt from keyboard; interactive attention signal.
SIGQUIT	Quit from keyboard.
SIGILL	Illegal instruction.
SIGTRAP	Trace/breakpoint trap.
SIGABRT	Abnormal termination; abort signal from abort(3).
SIGBUS	BUS error (bad memory access).
SIGKILL	Kill, unblockable.
SIGPIPE	„Broken pipe“: write to pipe with no readers.
SIGTERM	Termination request.
SIGCHLD	Child status has changed (stopped or terminated).
SIGSTOP	Stop process, unblockable.
SIGTTIN	Background read from tty.
SIGXCPU	CPU time limit exceeded.
SIGPWR	Power failure restart.
SIGSYS	Bad system call.

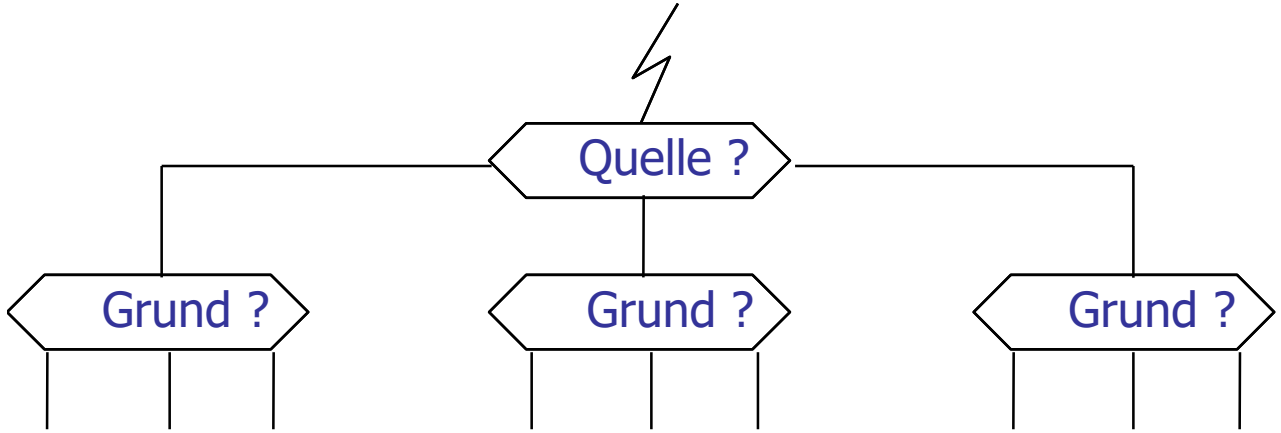
Kommunikation mit dem BS: Unterbrechungen (Interrupts)

- Interrupt: Signal informiert CPU über das Ende einer Aktivität
  - Der Bus verfügt über (mindestens) eine Unterbrechungsleitung. Prüfung nach jedem CPU-Befehl, ob ein Signal anliegt. Falls ja:
    - Sofortiger Sprung in eine Prozedur zur Auswertung der Unterbrechung
    - Abhängig von Auswertung werden die erforderlichen Aktionen durchgeführt / veranlasst
- Eine Unterbrechung kann zu jedem Zeitpunkt und in jeder Situation auftreten



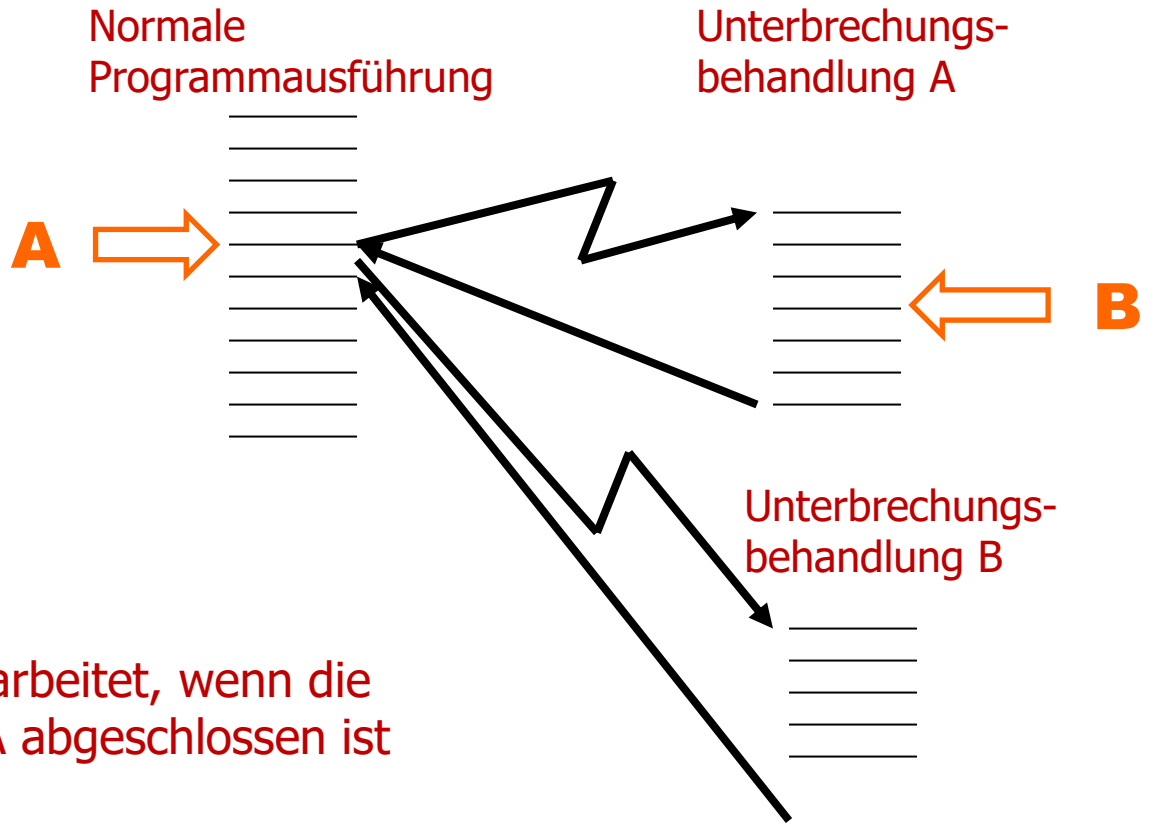
Unterbrechungsanalyse

- Unterbrechungssignal liegt vor
- Analyse mit dem Ziel, herauszufinden
  - wer (welches Gerät) die Unterbrechung verursacht hat (Quelle),
  - warum die Unterbrechung ausgelöst wurde (z.B. Ende der Übertragung, Fehler)
- Entscheidung: sequentielle vs. geschachtelte Behandlung
- Struktur der Unterbrechungsbehandlung



Abarbeitung: Sequentielle Unterbrechungsbehandlung

- Verbieten weiterer Unterbrechungen während der Unterbrechungsbehandlung (Unterbrechungssperre setzen, disable interrupt)
- Maskierung: Das Verbot wird auf bestimmte Unterbrechungstypen beschränkt

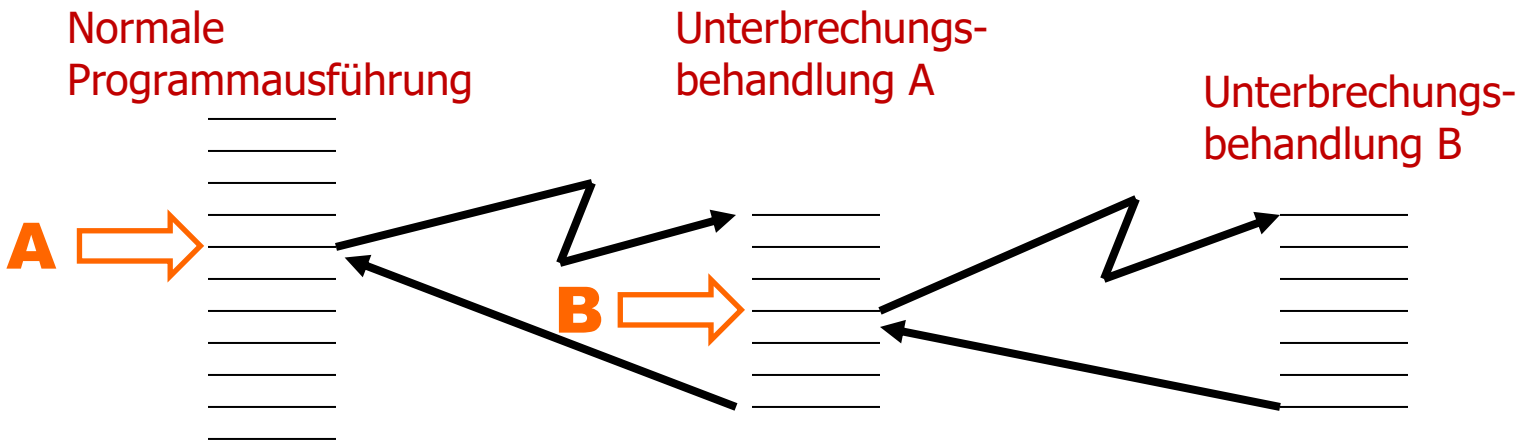


Zweite Unterbrechung B wird erst dann bearbeitet, wenn die Bearbeitung der aktuellen Unterbrechung A abgeschlossen ist

Abarbeitung: Geschachtelte Unterbrechungsbehandlung

- Klassifikation von Unterbrechungen in Prioritätsklassen (statisch)

Unterbrechungen höherer Priorität dürfen die Bearbeitung von Unterbrechungen geringerer Priorität unterbrechen



2.2 Prozesse

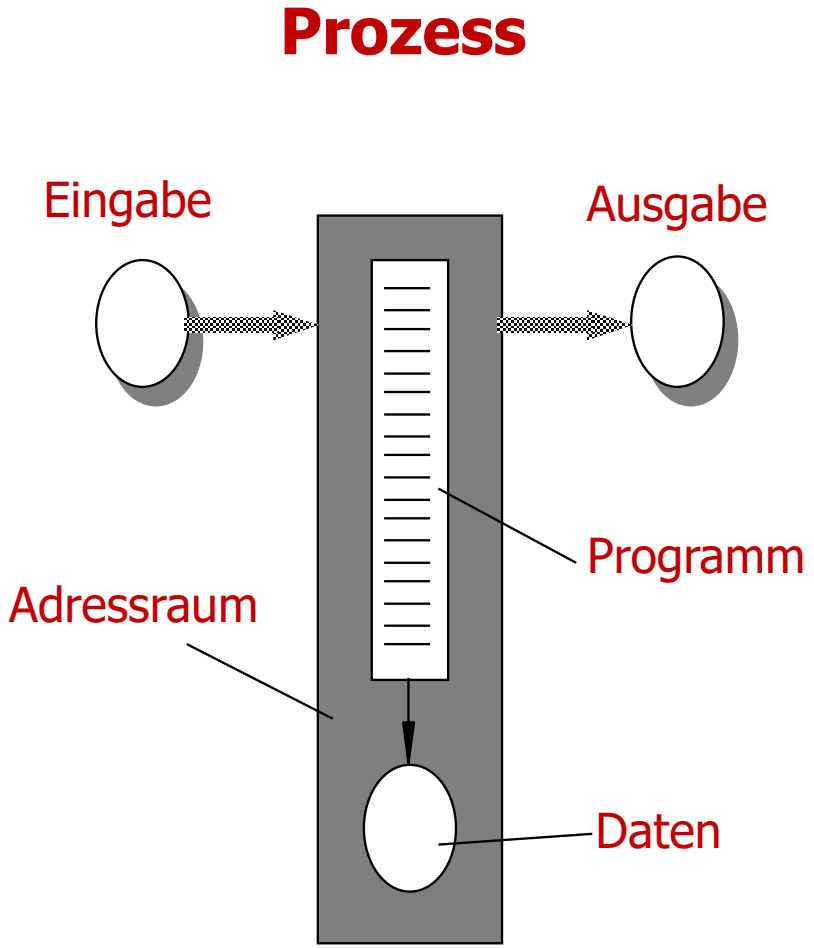
- Prozesse sind
  - dynamische Objekte, die Aktivitäten in einem System repräsentieren
  - funktionale und strukturierende Beschreibungseinheiten in System- und Anwendungssoftware

- Prozess = virtueller Rechner spezialisiert zur Ausführung eines bestimmten Programms (Instanz eines Programms, laufendes Programm)

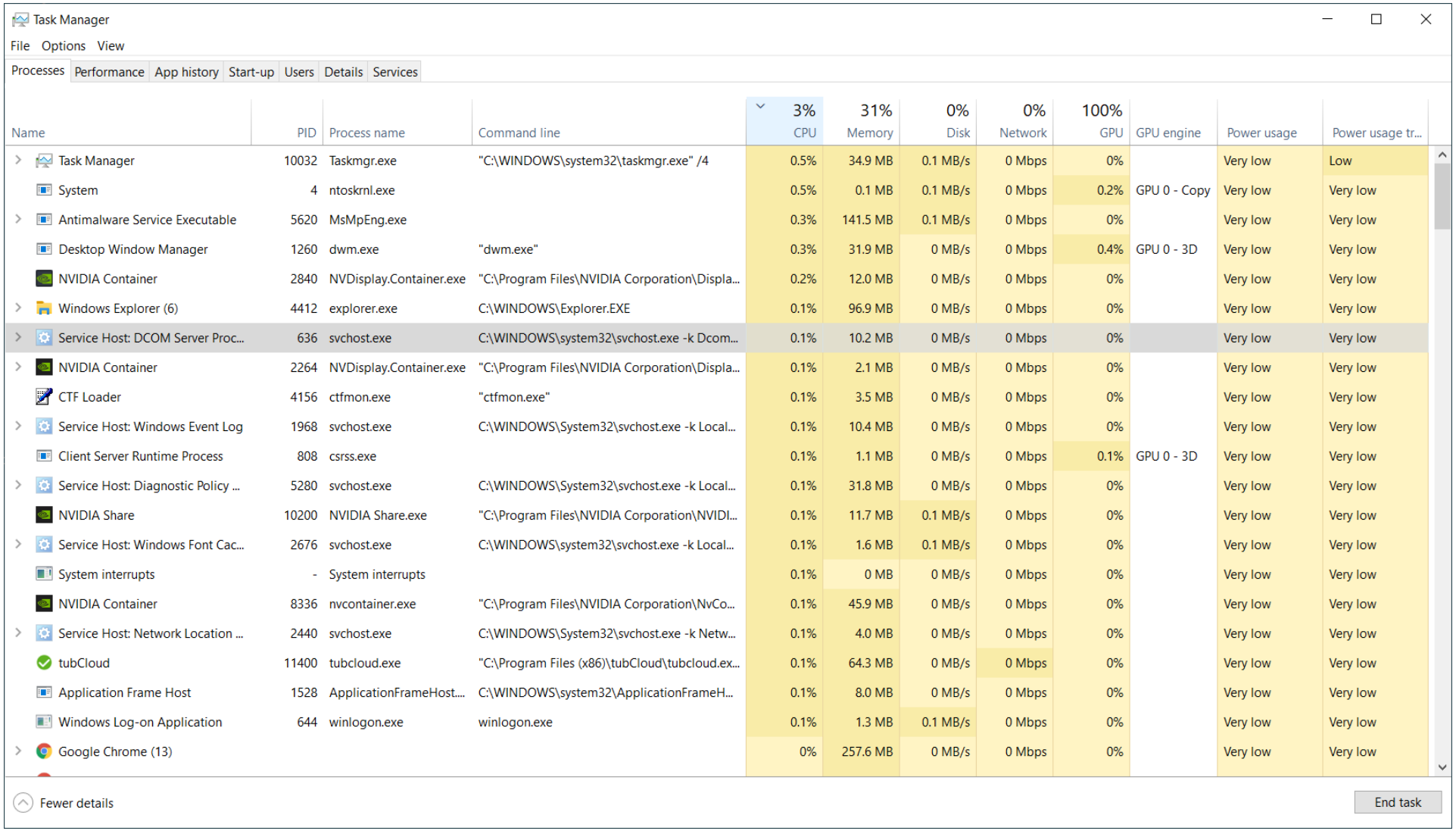
Beschreibungseinheit Prozess

- Ein Prozess (process, task) ist definiert durch
  - Adressraum und ggf. Besitz von weiteren Ressourcen
  - Auszuführendes Programm
  - Ein oder mehrere Threads (Aktivitätsträger)

- Ein Prozess bekommt i.d.R. Eingabedaten (Parameter) und erzeugt eine Ausgabe



Task manager



Name	PID	Process name	Command line	3% CPU	31% Memory	0% Disk	0% Network	100% GPU	GPU engine	Power usage	Power usage tr...
Task Manager	10032	Taskmgr.exe	"C:\WINDOWS\system32\taskmgr.exe" /4	0.5%	34.9 MB	0.1 MB/s	0 Mbps	0%		Very low	Low
System	4	ntoskrnl.exe		0.5%	0.1 MB	0.1 MB/s	0 Mbps	0.2%	GPU 0 - Copy	Very low	Very low
Antimalware Service Executable	5620	MsMpEng.exe		0.3%	141.5 MB	0.1 MB/s	0 Mbps	0%		Very low	Very low
Desktop Window Manager	1260	dwm.exe	"dwm.exe"	0.3%	31.9 MB	0 MB/s	0 Mbps	0.4%	GPU 0 - 3D	Very low	Very low
NVIDIA Container	2840	NVDisplay.Container.exe	"C:\Program Files\NVIDIA Corporation\Displa...	0.2%	12.0 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Windows Explorer (6)	4412	explorer.exe	C:\WINDOWS\Explorer.EXE	0.1%	96.9 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Service Host: DCOM Server Proc...	636	svchost.exe	C:\WINDOWS\system32\svchost.exe -k Dcom...	0.1%	10.2 MB	0 MB/s	0 Mbps	0%		Very low	Very low
NVIDIA Container	2264	NVDisplay.Container.exe	"C:\Program Files\NVIDIA Corporation\Displa...	0.1%	2.1 MB	0 MB/s	0 Mbps	0%		Very low	Very low
CTF Loader	4156	ctfmon.exe	"ctfmon.exe"	0.1%	3.5 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Service Host: Windows Event Log	1968	svchost.exe	C:\WINDOWS\System32\svchost.exe -k Local...	0.1%	10.4 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Client Server Runtime Process	808	csrss.exe		0.1%	1.1 MB	0 MB/s	0 Mbps	0.1%	GPU 0 - 3D	Very low	Very low
Service Host: Diagnostic Policy ...	5280	svchost.exe	C:\WINDOWS\System32\svchost.exe -k Local...	0.1%	31.8 MB	0 MB/s	0 Mbps	0%		Very low	Very low
NVIDIA Share	10200	NVIDIA Share.exe	"C:\Program Files\NVIDIA Corporation\NVIDI...	0.1%	11.7 MB	0.1 MB/s	0 Mbps	0%		Very low	Very low
Service Host: Windows Font Cac...	2676	svchost.exe	C:\WINDOWS\system32\svchost.exe -k Local...	0.1%	1.6 MB	0.1 MB/s	0 Mbps	0%		Very low	Very low
System interrupts	-	System interrupts		0.1%	0 MB	0 MB/s	0 Mbps	0%		Very low	Very low
NVIDIA Container	8336	nvcontainer.exe	"C:\Program Files\NVIDIA Corporation\NvCo...	0.1%	45.9 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Service Host: Network Location ...	2440	svchost.exe	C:\WINDOWS\System32\svchost.exe -k Netw...	0.1%	4.0 MB	0 MB/s	0 Mbps	0%		Very low	Very low
tubCloud	11400	tubcloud.exe	"C:\Program Files (x86)\tubCloud\tubcloud.ex...	0.1%	64.3 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Application Frame Host	1528	ApplicationFrameHost....	C:\WINDOWS\system32\ApplicationFrameH...	0.1%	8.0 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Windows Log-on Application	644	winlogon.exe	winlogon.exe	0.1%	1.3 MB	0.1 MB/s	0 Mbps	0%		Very low	Very low
Google Chrome (13)				0%	257.6 MB	0 MB/s	0 Mbps	0%		Very low	Very low

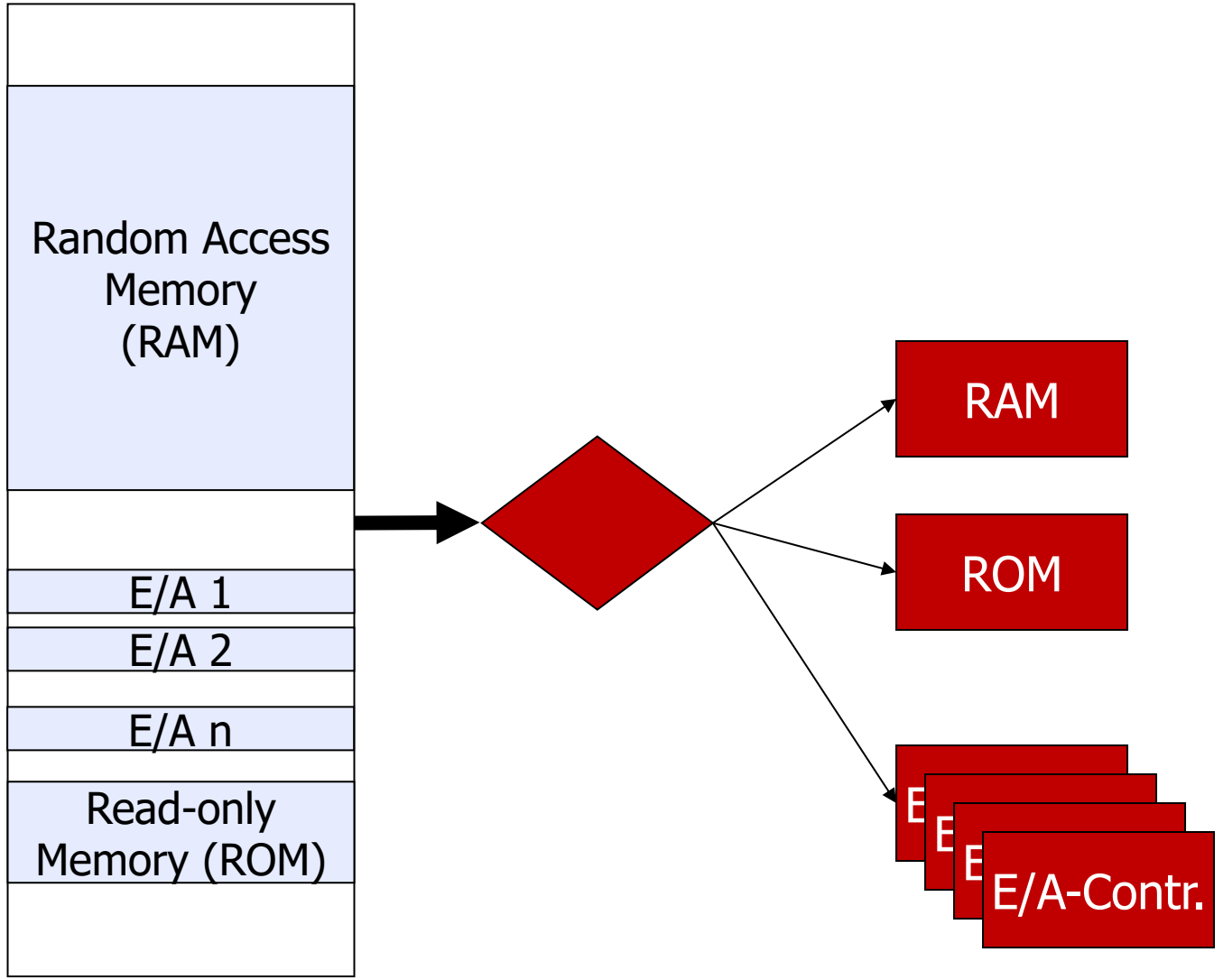
Adressräume

- Physischer Adressraum (einmal pro Maschine)
  - Hardwarekomponenten im Rechner und in Peripheriegeräten (RAM, Register, Controllerspeicher, …)
- Virtueller Adressraum (i.d.R. einmal pro Prozess)
  - vom Betriebssystem erzeugt und konfiguriert
  - Jeder Prozess hat einen eigenen virtuellen Adressraum, der i.d.R. viel größer ist als der tatsächlich vorhandene physische Adressreaum
  - enthält die für die Ausführung nötigen Instruktionen und Daten
- Teile des Adressraums können undefiniert sein → Zugriff darauf führt zu Fehler

Physischer Adressraum

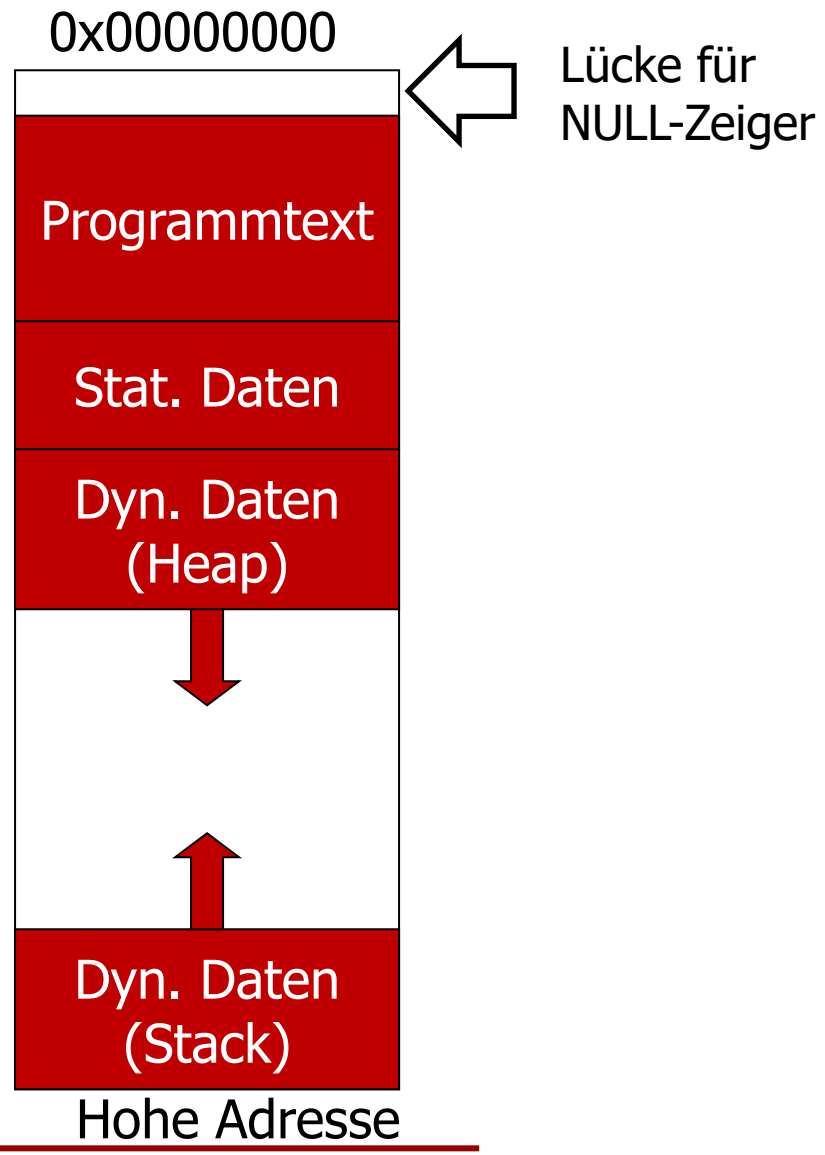
- Hauptspeicher (Arbeitsspeicher): Temporäre Speicherung der aktiven Prozesse und der dazugehörigen Daten

- Einblendung des Hauptspeichers (RAM), Read-Only-Speichers (ROM) und der E/A-Geräte in den physischen Adressraum

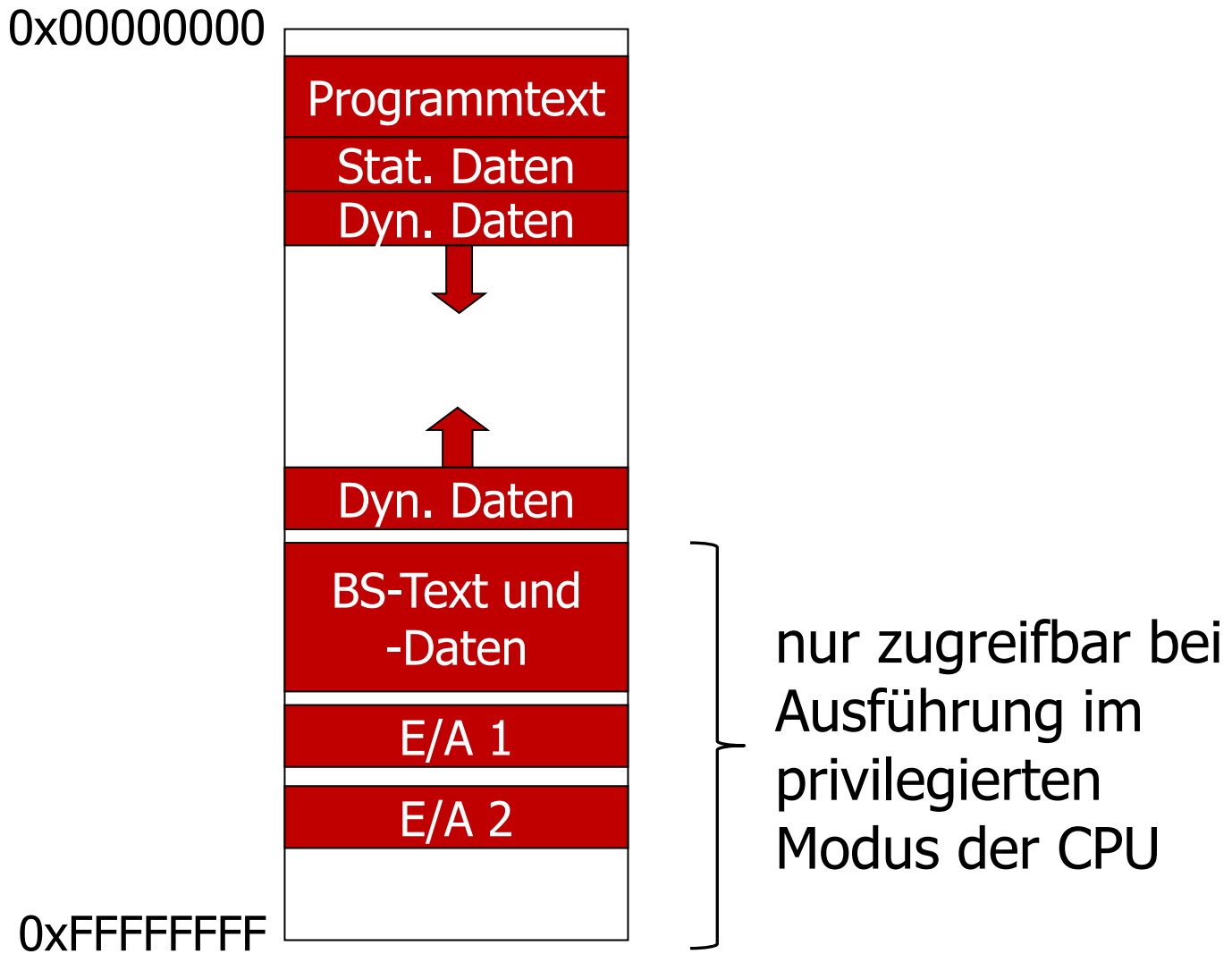


Virtueller Adressraum

- Programmtext: Instruktionen des Programms
- Statische Daten: globale Variablen, lokale Variablen mit static-Modifier
- Dynamische Daten (Heap)
  - zur Laufzeit explizit reservierbar Speicherbereich
  - wächst und schrumpft nach Bedarf
- Dynamische Daten (Stack)
  - je aufgerufene Funktion: lokale Variablen, Aufrufparameter
  - wächst, je tiefer die Aufrufkette ist (Rekursion!)
  - im Gegensatz zum Heap Benutzung „automatisch“



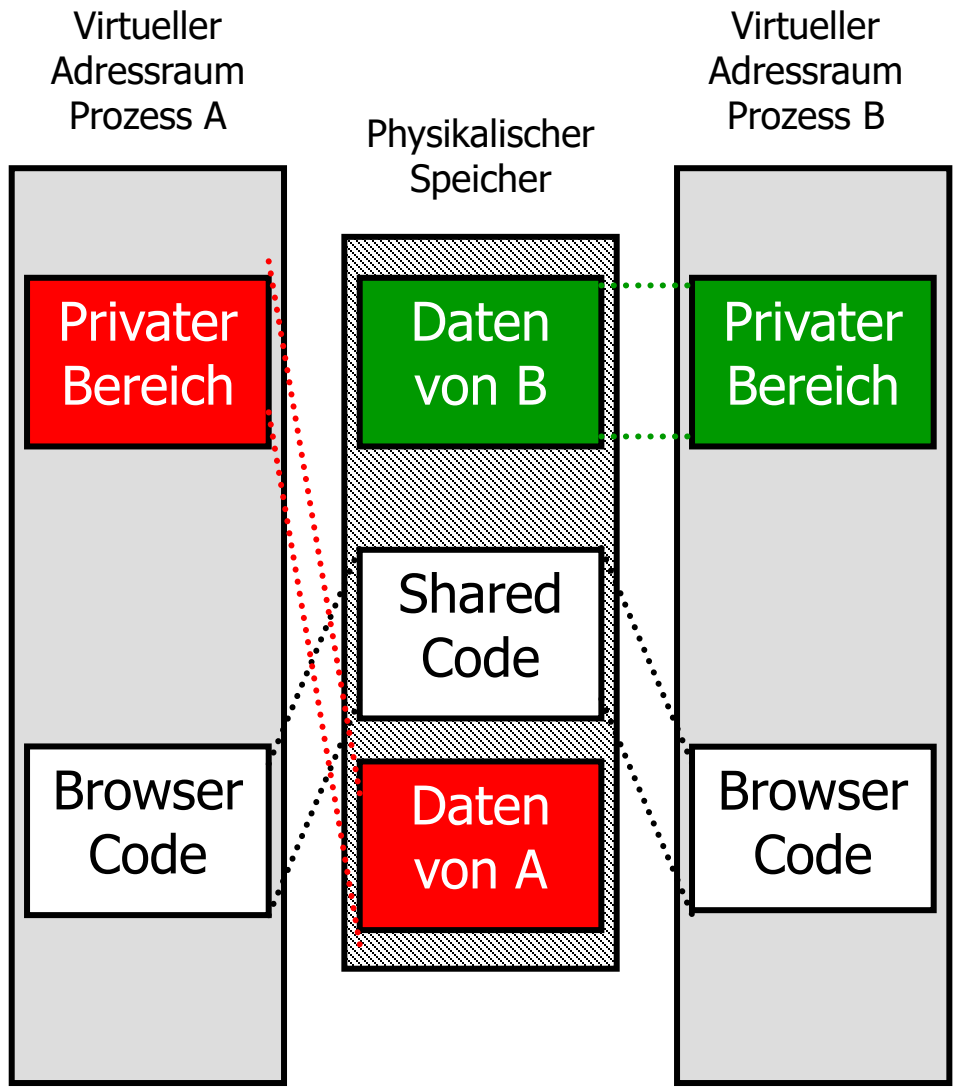
Virtueller Adressraum (Sicht des Betriebssystems)



Zusammenhang Prozesse und Programme

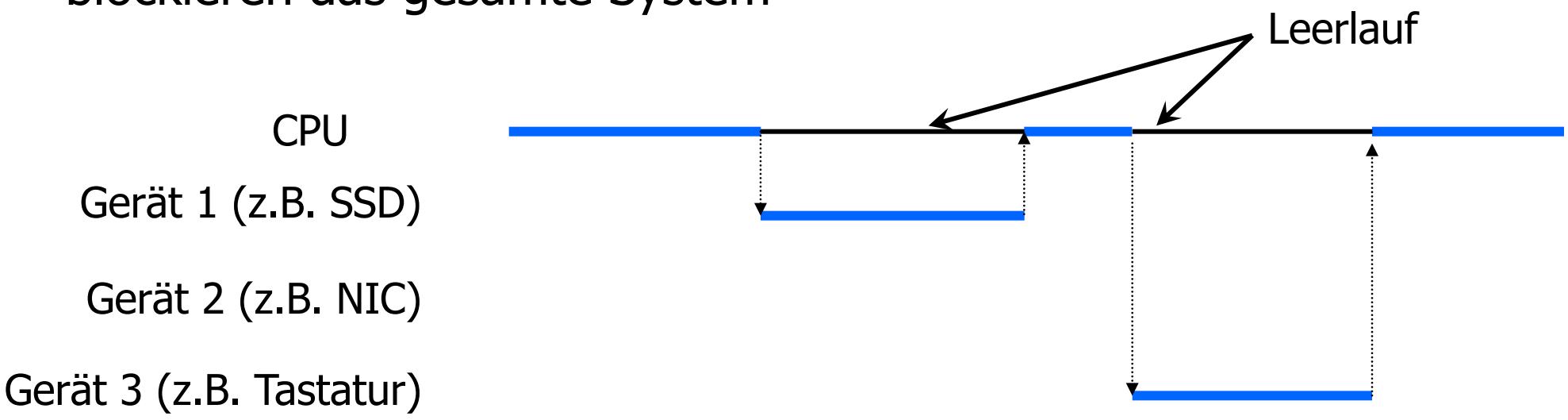
- Mehrere Prozesse können dasselbe Programm mit unterschiedlichen Daten ausführen

- Beispiel
  - Auf einem Server wird ein Webbrowser von zwei Benutzern gestartet
  - In beiden Fällen wird der gleiche Browsercode aber mit unterschiedlichen Parametern ausgeführt



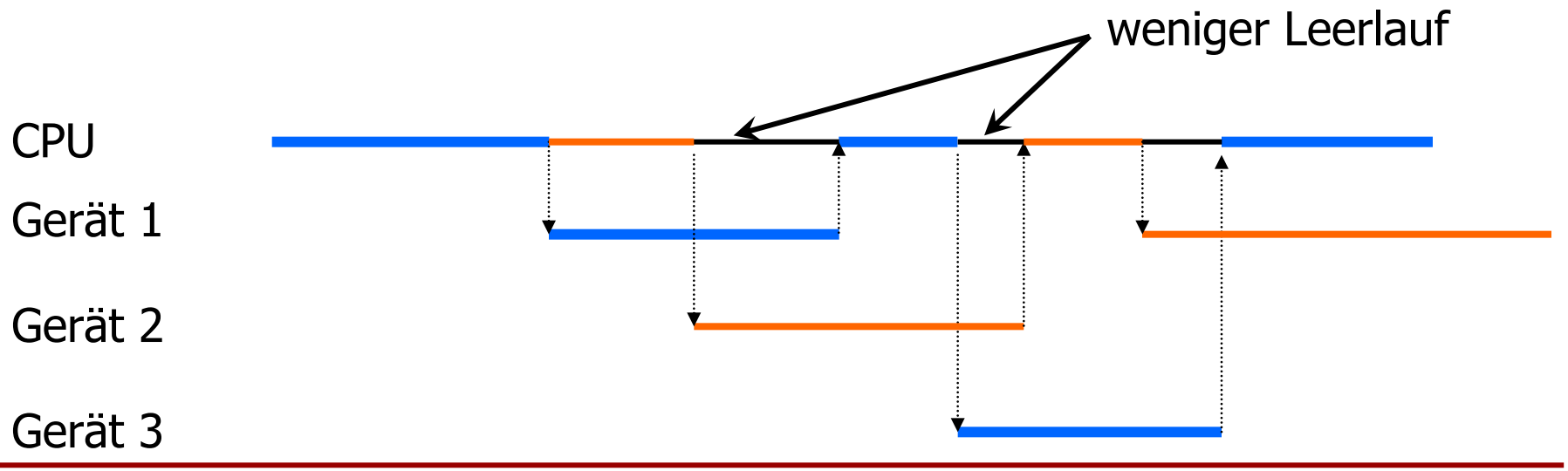
Ausführung von Prozessen

- Einfachste Rechnerbetriebsart → Stapelbetrieb (batch mode)
  - Der aktive Prozess wird unterbrechungsfrei – ohne Unterbrechung durch andere konkurrierende Prozesse – ausgeführt
  - Mehrere Prozesse werden sequentiell abgearbeitet
- Problem: Während der Kommunikation mit z.B. E/A-Geräten bleibt die CPU ungenutzt ⇒ Leerlaufzeiten und ineffiziente Ausführung, große Aufträge blockieren das gesamte System



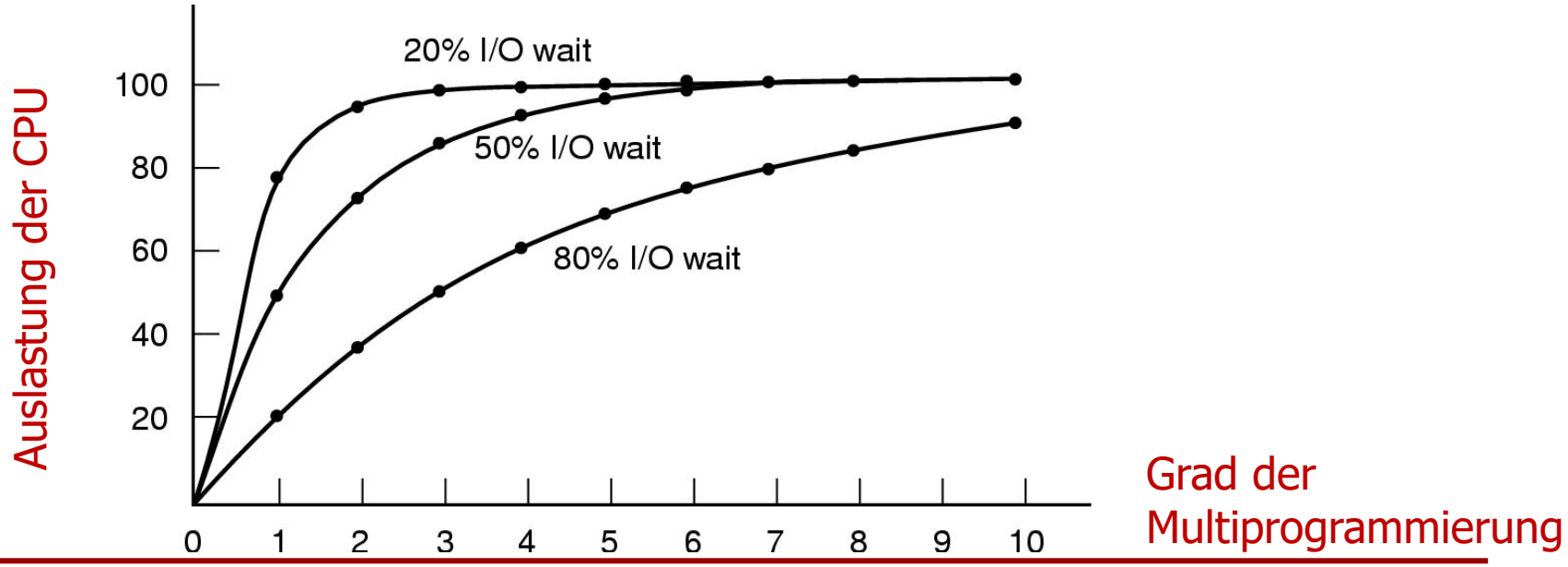
Ausführung von Prozessen

- Multiprogrammierung
  - Bei Warten auf Geräteantwort wird der nächste bereite Prozess gestartet/fortgesetzt (dispatcht)
  - Mehrere Prozesse werden verzahnt abgearbeitet (Nebenläufigkeit)
- Leerlauf wird somit weitgehend (eben so weit lauffähige Prozesse vorhanden sind) vermieden

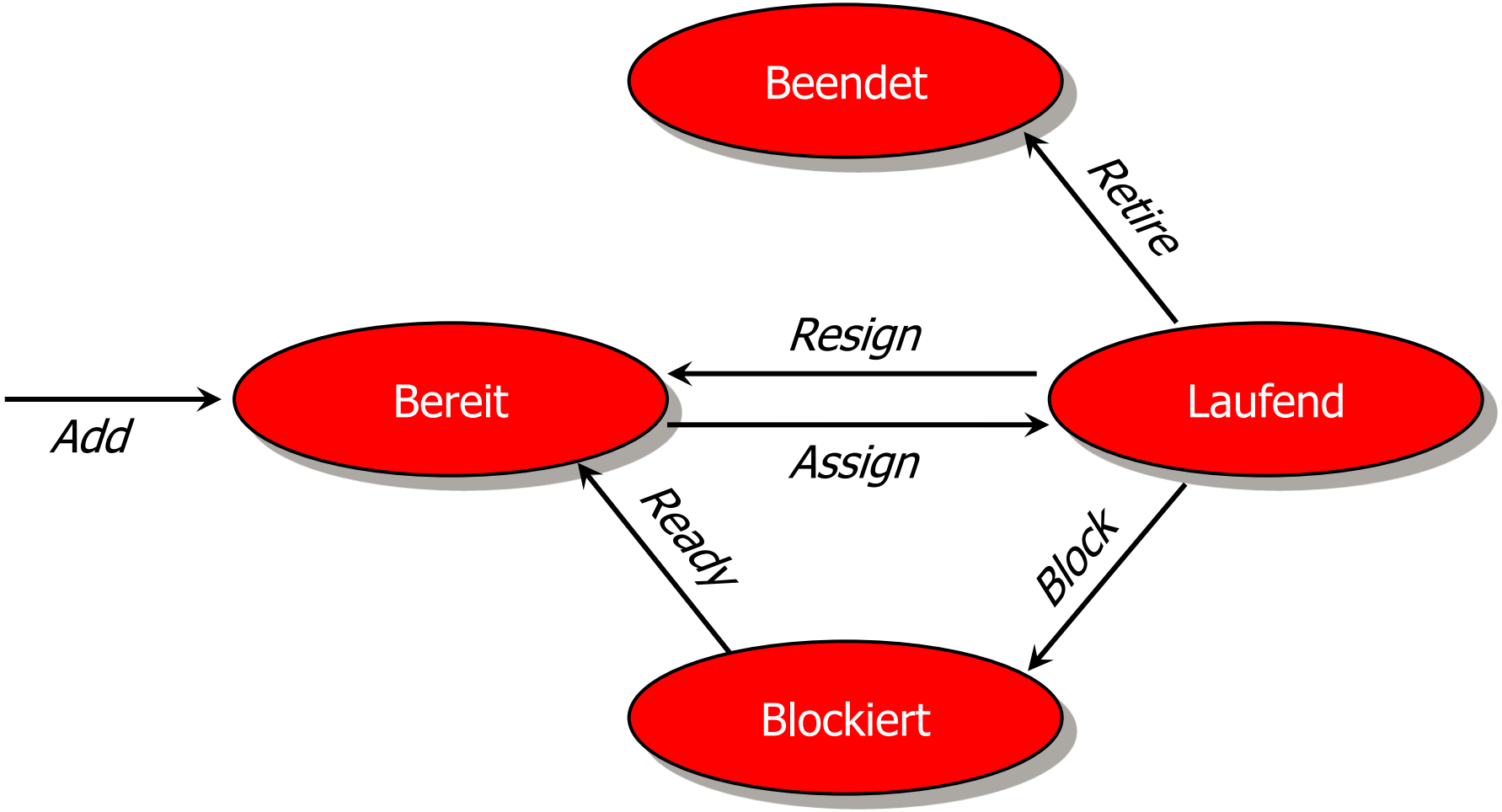


Modellierung der Multiprogrammierung

- Wie viele Prozesse sind für "genau richtige" Auslastung notwendig?
- Keine allgemeine Antwort möglich. Annahmen:
  - Ein Prozess verbringt einen Anteil p seiner Zeit mit Warten auf E/A-Operationen
  - Wahrscheinlichkeit p^n = n Prozesse warten gleichzeitig auf E/A-Ende
  - Ausnutzung der CPU: $A = 1 - p^n$
  - n = Grad der Multiprogrammierung (Degree of Multiprogramming)



Prozesszustände



Prozesszustände

- Ein Prozess kann sich – abhängig vom aktuellen Status – in unterschiedlichen Zuständen befinden
  - Rechnend, Laufend (Running): Der Prozess ist im Besitz des physikalischen Prozessors und wird aktuell ausgeführt
  - Bereit (Ready): Der Prozess hat alle notwendigen Betriebsmittel und wartet auf die Zuteilung des/eines Prozessors
  - Blockiert, Wartend (Waiting): Der Prozess wartet auf die Erfüllung einer Bedingung, z.B. Beendigung einer E/A-Operation und bewirbt sich derzeit nicht um den Prozessor
  - Beendet (Terminated): Der Prozess hat alle Berechnung beendet und die zugeteilten Betriebsmittel freigegeben

Zustandsübergänge

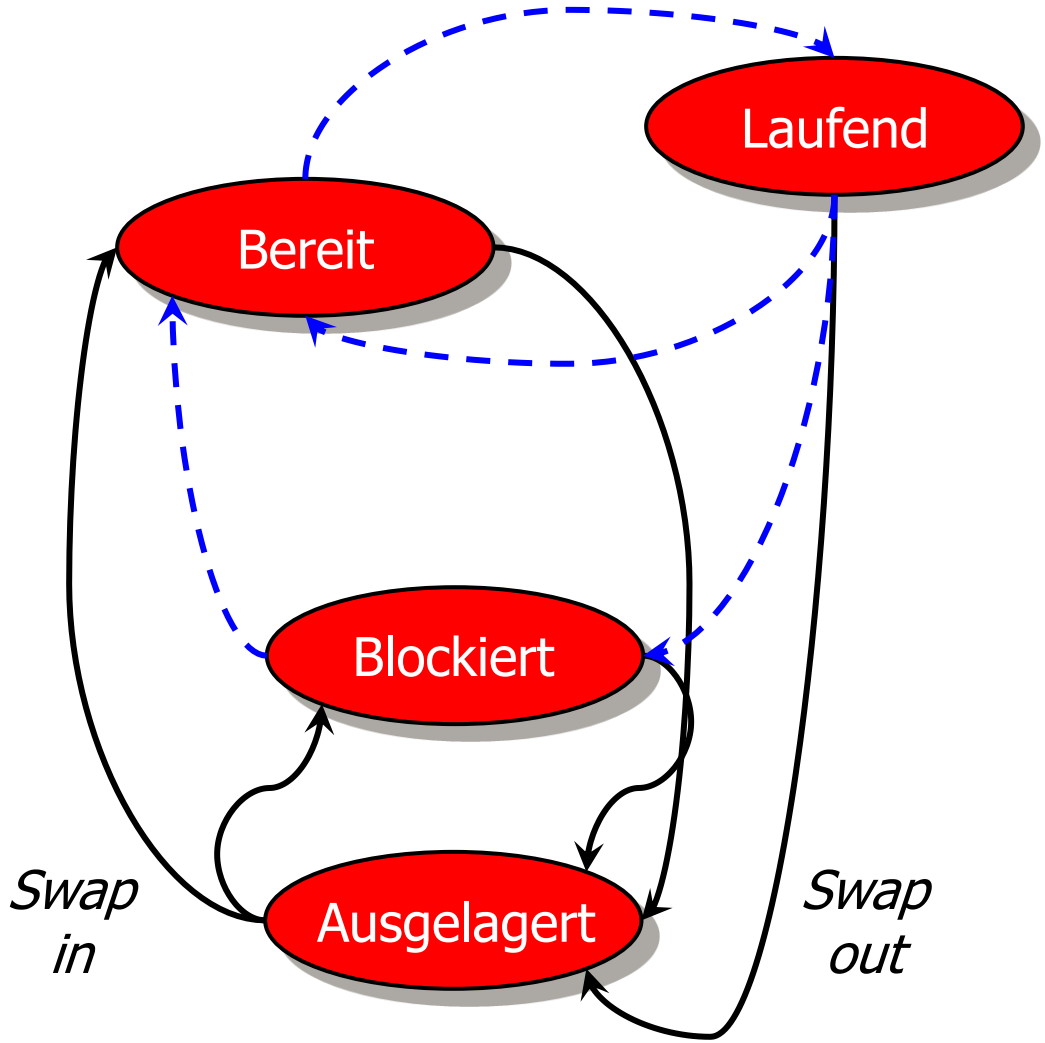
- Erlaubte Übergänge

Add:	Ein neu erzeugter Prozess wird in die Klasse Bereit aufgenommen
Assign:	Infolge des Kontextwechsels wird der Prozessor zugeteilt
Block:	Aufruf einer blockierenden E/A-Operation oder Synchronisation bewirkt, dass der Prozessor entzogen wird
Ready:	Nach Beendigung der blockierenden Operation wartet der Prozess auf erneute Zuteilung des Prozessors
Resign:	Einem laufenden Prozess wird der Prozessor – aufgrund eines Timer-Interrupts, z.B. Zeitscheibe abgelaufen – entzogen, oder er gibt den Prozessor freiwillig ab
Retire:	Der laufende Prozess terminiert und gibt alle Ressourcen wieder frei

Erweitertes Zustandsmodell

- Wegen Speichermangel werden oft ganze Prozesse (d.h. der Inhalt ihrer Adressräume) auf die Festplatte ausgelagert
  - Zusatzzustand Ausgelagert
  - Zusatzübergänge Swap in und Swap out
  - Mehr dazu in Kapitel 6 (Speicherverwaltung)

- Nach der Einlagerung kann der Prozess in den Zustand Bereit oder Blockiert wechseln, abhängig von aktuellen blockierenden Operationen



Prozessverwaltung in Betriebssystemen

- Implementierung von Prozessen in BS durch Datenstruktur Prozesskontrollblock (Process Control Block, PCB)
- PCB = verwaltungstechnischer Repräsentant des Prozesses
- Elemente eines PCB (unter anderem):
  - Prozessidentifikation (häufig: PID, Prozessnummer)
  - Identifikation des Besitzers (z.B. Nutzerkennung)
  - Bereich zur Sicherung der aktuellen Registerwerte, wenn Prozess nicht im Zustand Laufend
  - Zustandsvariable (Prozesszustand): Bereit / Laufend / …
  - Informationen über zugeteilte Betriebsmittel
  - Konfiguration des virtuellen Adressraums
  - Verweise auf Eltern- bzw. Kindprozesse

```
struct task_struct {
volatile long state;
long counter;
long priority;
unsigned long signal;
unsigned long blocked;
unsigned long flags;
int errno;
long debugreg[8];
struct exec_domain *exec_domain;
struct linux_binfmt *binfmt;
struct task_struct *next_task, *prev_task;
struct task_struct 0, *prev_run;
unsigned long saved_kernel_stack;
unsigned long kernel_stack_page;
int exit_code, exit_signal;
unsigned long personality;
int dumpable:1;
int did_exec:1;
int pid;
int pgrp;
int tty_old_pgrp;
int session;
int leader;
int groups[NGROUPS];
struct task_struct *p_opptr, *p_pptr, *p_cptr, *p_ysptr, *p_os
struct wait_queue *wait_chldexit;
unsigned short uid,euid,suid,fsuid;
unsigned short gid,egid,sgid,fsgid;
unsigned long timeout, policy, rt_priority;
unsigned long it_real_value, it_prof_value, it_virt_value;
unsigned long it_real_incr, it_prof_incr, it_virt_incr;
struct timer_list real_timer;
long utime, stime, cutime, cstime, start_time;
unsigned long min_flt, maj_flt, nswap, cmin_flt, cmaj_flt, cnswap;
int swappable:1;
unsigned long swap_address;
unsigned long old_maj_flt; /* old value of maj_flt */
unsigned long dec_flt;
unsigned long swap_cnt;
struct rlimit rlim[RLIM_NLIMITS];
unsigned short used_math;
char comm[16];
int link_count;
struct tty_struct *tty; /* NULL if no tty */
struct sem_undo *semundo;
struct sem_queue *semsleeping;
struct desc_struct *ldt;
struct thread_struct tss;
struct fs_struct *fs;
struct files_struct *files;
struct mm_struct *mm;
struct signal_struct *sig;
};
```

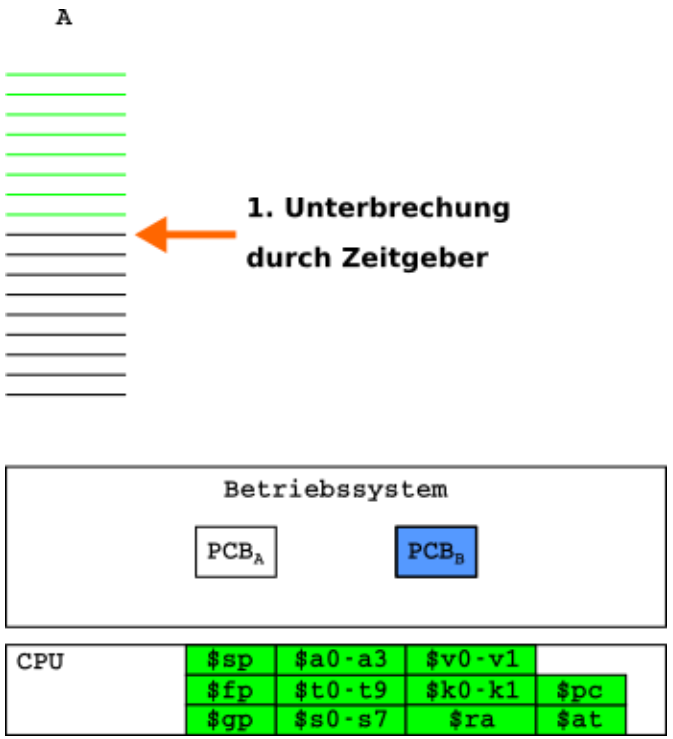
Beispiel eines Prozesskontrollblocks Linux 2.6.11

Prozessumschaltung

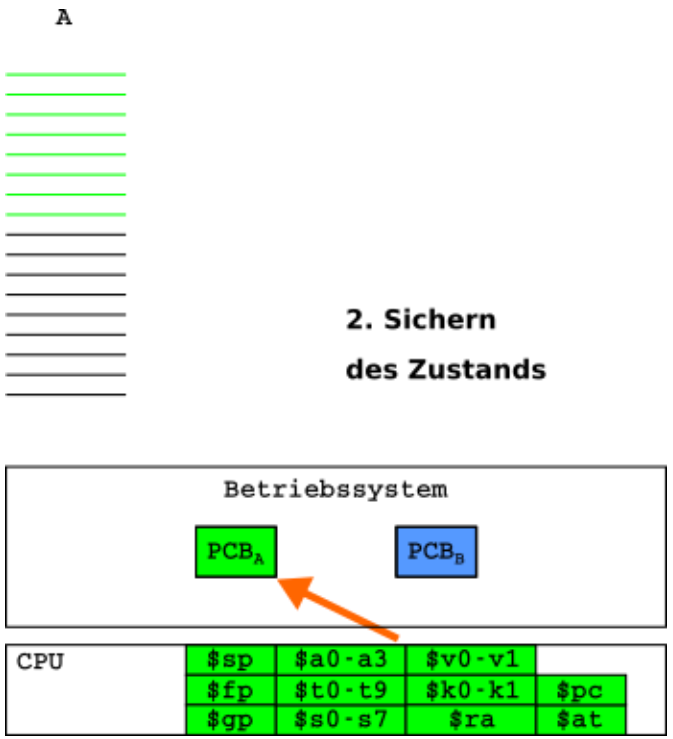
- Aktuell aktiver Prozess A wird aus Zustand Laufend in anderen Zustand versetzt (Grund z.B. Zeitscheibe verbraucht, Interrupt, blockierender Systemaufruf, …)
  - Registerinhalte in PCB_A ablegen (inkl. Ort, wo A unterbrochen wurde = Befehlszähler zum Zeitpunkt der Unterbrechung)
  - Prozesszustand aktualisieren (→ Blockiert/Bereit/…)

- Ein bereiter Prozess B wird in den Zustand Laufend versetzt
  - Prozesszustand aktualisieren (→ Laufend)
  - Umschalten des virtuellen Adressraums gemäß Konfiguration in PCB_B
  - Laden der Registerinhalte aus PCB_B
  - Fortsetzen von B an dessen gespeichertem Befehlszähler

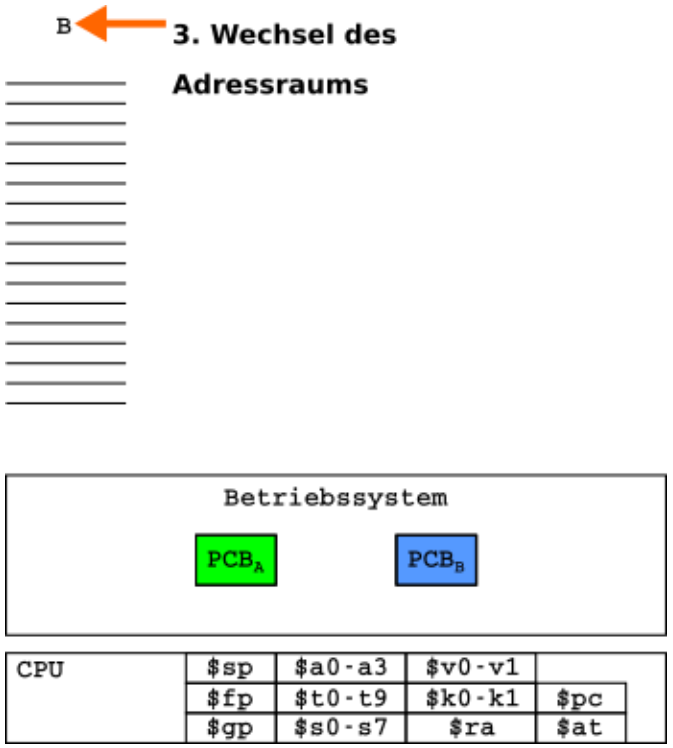
Bildsequenz: Illustration Prozessumschaltung



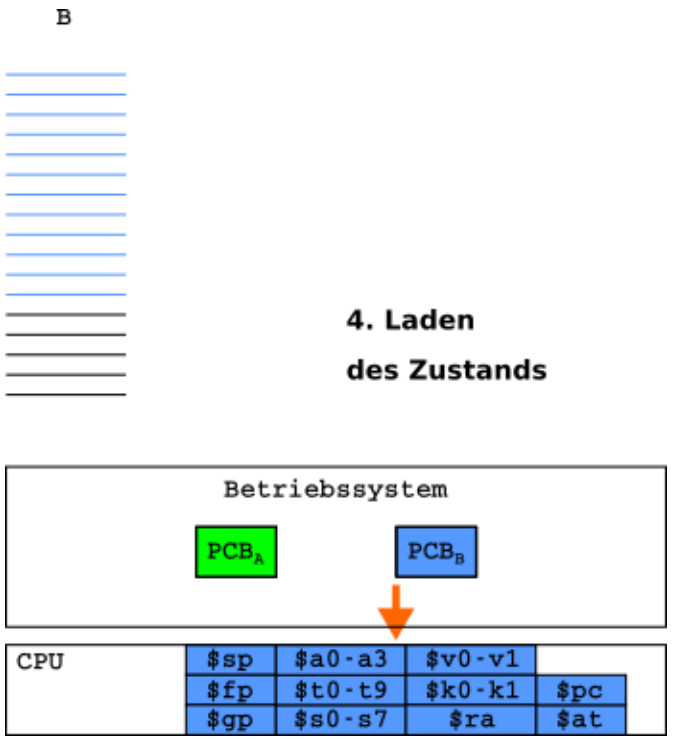
Bildsequenz: Illustration Prozessumschaltung



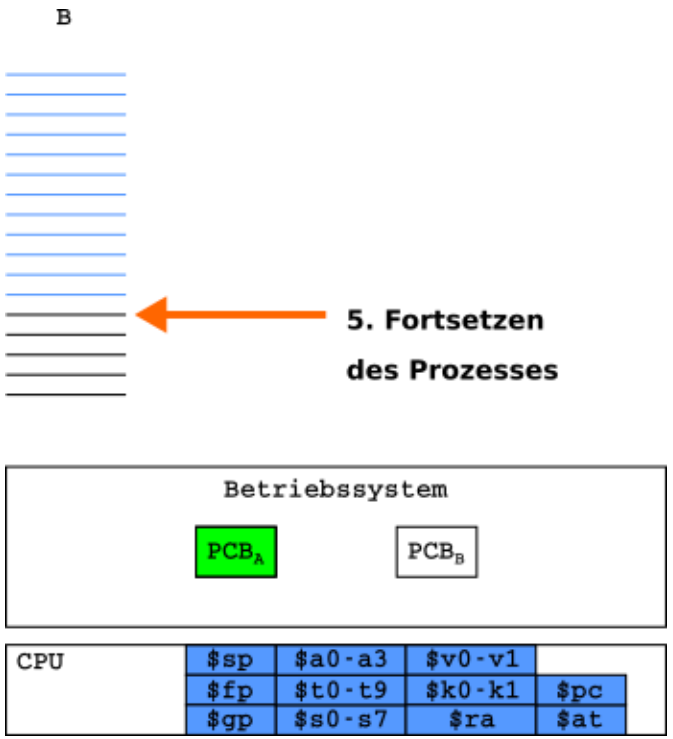
Bildsequenz: Illustration Prozessumschaltung



Bildsequenz: Illustration Prozessumschaltung



Bildsequenz: Illustration Prozessumschaltung



Auswahl des nächsten laufenden Prozesses

- Strategien zur Überführung der Prozesse Bereit → Laufend sind wichtig für die Effizienz eines Systems
- Auswahlprozess beinhaltet die dynamische Auswertung von verschiedenen Kriterien, z.B.
  - Prozessnummer (zyklisches Umschalten)
  - Ankunftsreihenfolge
  - Fairness und Priorität (Konstant / Dynamisch)
  - Einhaltung von geforderten Fertigstellungspunkten
- Nach der Wahl müssen die Attribute aller anderen Prozesse angepasst werden → Detailliert in Kapitel 3 („Scheduling“)

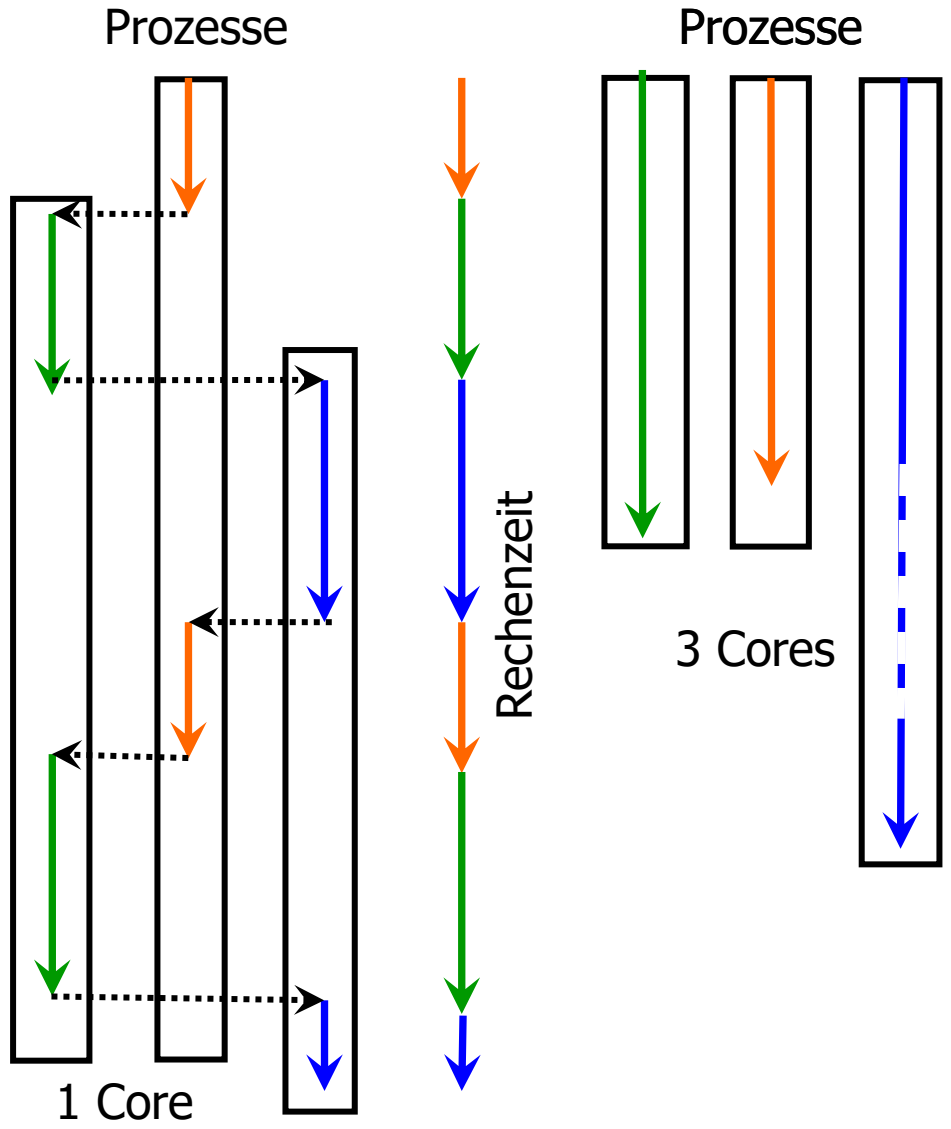
Nebenläufigkeit und Parallelität

- Nebenläufigkeit (Concurrency = Concurrent Execution)
  - Logisch simultane Verarbeitung von Operationsströmen
  - Eindruck erweckt, dass die Prozesse gleichzeitig ablaufen → Verzahnte Ausführung auf einem 1-CPU-System
- Parallelität
  - Die Operationsströme werden tatsächlich simultan ausgeführt
  - Mehrfache Verarbeitungselemente, d.h. Prozessoren oder andere unabhängige Architekturelemente, sind zwingend notwendig
- Bemerkungen
  - Nebenläufigkeit und Parallelität setzen einen kontrollierten Zugang zu gemeinsamen Ressourcen voraus
  - Nebenläufiges Programm auf Parallelsystem → paralleles Programm

Zusammenhang Nebenläufigkeit und Parallelität

- Nebenläufigkeit = Zuordnung mehrerer Prozesse zu mindestens einem Prozessor
- Parallelität = Zuordnung mehrerer Prozesse zu mindestens zwei Prozessoren
- Parallelität ist eine Teilmenge der Nebenläufigkeit
- Prozesse und Datentransfers werden nebenläufig ausgeführt (*Parallelität wird durch Warteoperationen ausgebremst*)

Grundlage für Threads!

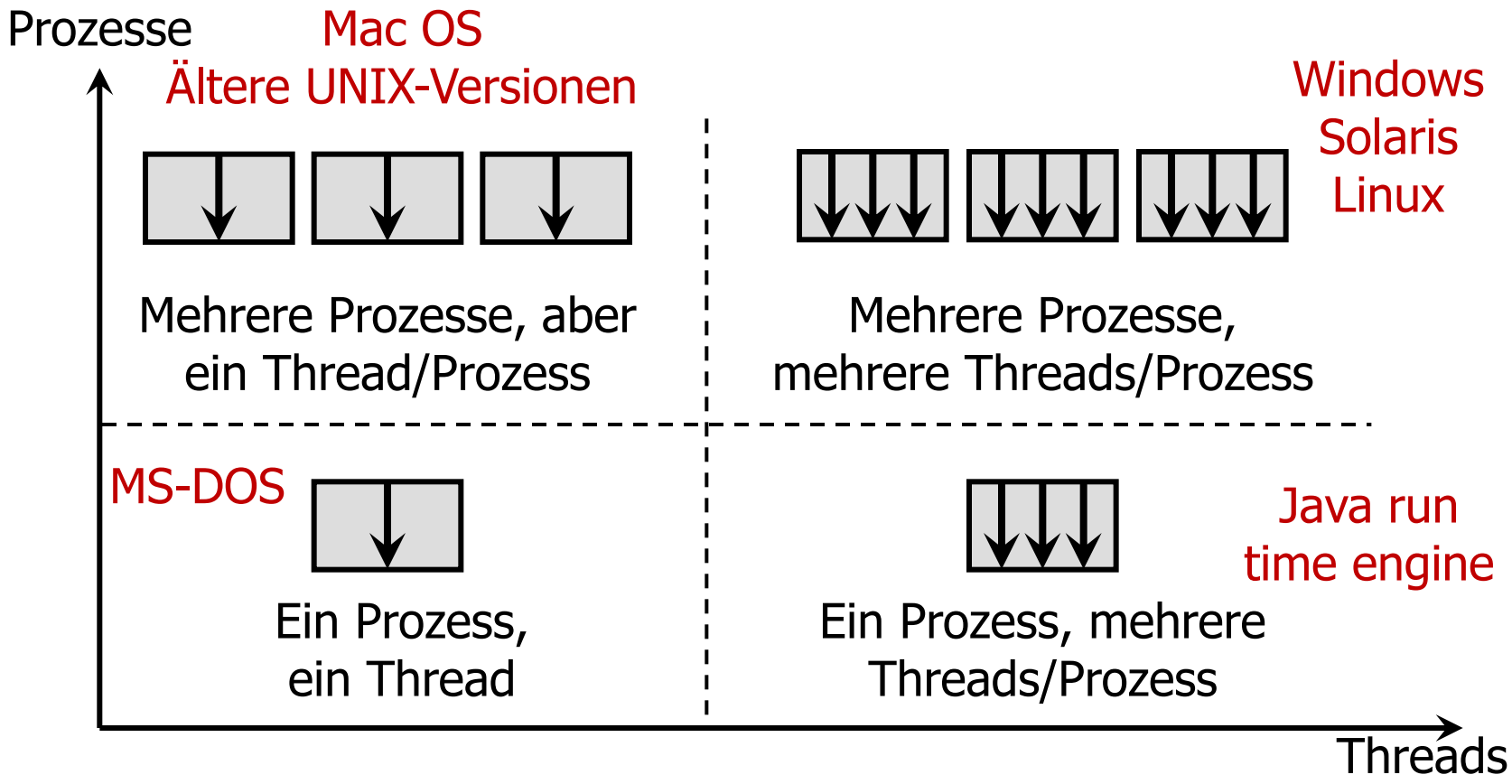


2.3 Thread (Leichtgewichtsprozess)

- Prozess bündelt virtuellen Adressraum, Quelltexte und Daten, Betriebsmittel wie E/A-Geräte, Dateien, …
- Thread
  - Gewöhnlich einem Prozess fest zugeordnet
  - Entspricht einem Kontrollfluss (von ggf. mehreren) dieses Prozesses
  - Operiert im selben virtuellen Adressraum und mit denselben Betriebsmitteln (Ausnahme: Stack) wie alle anderen Threads des Prozesses
  - Hat eigene Priorität, Zustand (bereit, laufend, blockiert, …), Befehlszähler, Registerwerte, die aber nicht in PCB, sondern in einer Threadtabelle gespeichert werden
- Erzeugung von neuem Prozess immer implizit mit einem Thread; Prozess selbst kann zur Laufzeit weitere anlegen

Zusammenhang Prozesse und Threads

- Single-Threaded: ein Thread pro Prozess (klassische Prozesse)
- Multi-Threaded: mehrere Threads innerhalb eines Prozesses



Beispiel: Webserver (single-threaded)

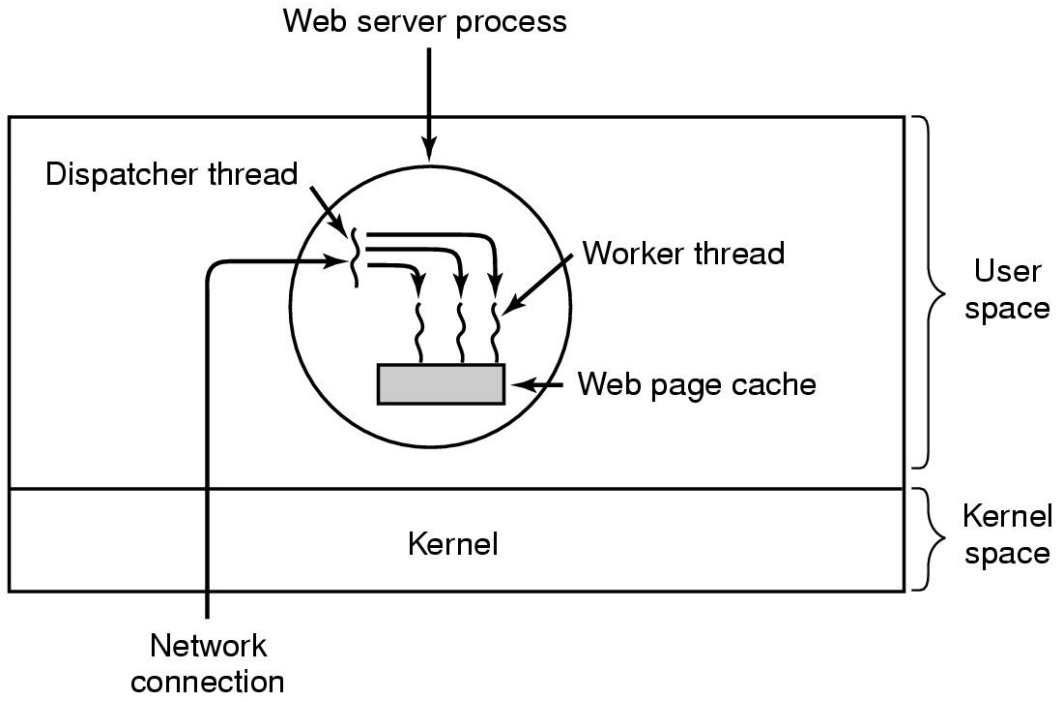
- Gegeben: Webserver auf einer dedizierten Maschine
  - Daten vergangener Anfragen werden in Cache solange aufbewahrt, bis der Speicher verbraucht ist
  - Älteste Datensätze werden durch neue ausgetauscht

- Realisierung mit nur einem Thread
  - Endlosschleife zur Annahme von Anfragen
  - Die Anfragen werden sequentiell bearbeitet
    - Sind die geforderten Daten im Cache → Kurze Antwortzeit
    - Andernfalls Systemaufruf zum Lesen der Daten von der Festplatte → Prozess blockiert

    → Leerlauf und geringe CPU-Auslastung

    → nächste Anfrage muss warten, selbst wenn deren Daten im Cache vorhanden wären

Beispiel: Webserver (multi-threaded)



Code Worker

```
while (TRUE) {
        wait_for_work(&buf);
        look_in_cache(&buf, &page);
        if(page_not_in_cache(&page);
                read_page_from_disk
                                (&buf, &page);
        return_page(&page);

}
```

Code Dispatcher

```
while (TRUE) {
                get_next_request(&buf);
                handoff_work(&buf);

}
```

Beispiel: Webserver (multi-threaded)

- Realisierung mit mehreren Threads
  - Thread Dispatcher: nimmt ankommende Anfragen entgegen
  - Thread Worker: bearbeitet eine einzelne Anfrage
- Ablauf
  - Dispatcher empfängt die Anfrage und erzeugt/weckt einen Worker
  - Worker wechselt sobald möglich in Laufend, überprüft Anfrage
    - Daten im Cache → Bearbeitung sofort
    - Daten auf Festplatte → startet Leseoperation → wird in Zustand Blockiert versetzt.

      Sobald Leseoperation beendet:

      → Wechsel in Zustand Bereit

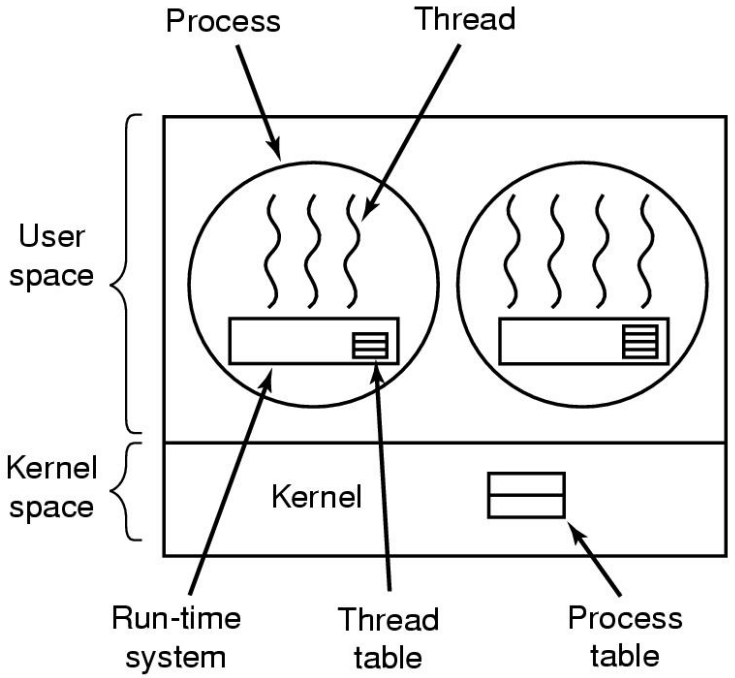
      → Worker bewirbt sich erneut um die CPU

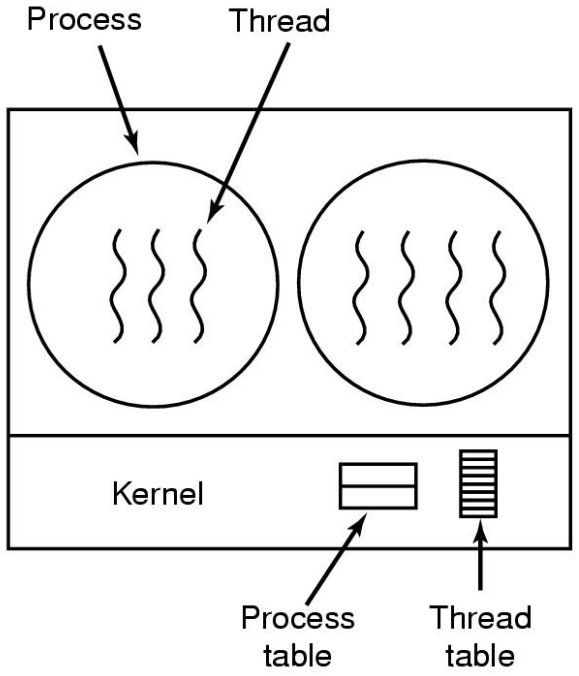
- Vorteil: Hohes Maß an Parallelität zwischen Lese- und Rechenzugriffen

Threadtypen

- Grundsätzlich werden Threads aufgeteilt in
  - Kernel-Level-Threads (KL-Threads): realisiert im Kernadressraum
  - User-Level-Threads (UL-Threads): realisiert im Benutzeradressraum
- Hybride Realisierung auch möglich

Realisierung im Benutzer-adressraum





Realisierung im Kern-adressraum

KL-Threads

- KL-Threads haben folgende Eigenschaften
  - Werden im Betriebssystem realisiert
  - Erzeugung, Umschaltung, Zerstörung erfordert Aufruf von Verwaltungsfunktionen des Betriebssystems → Systemaufruf
  - Jeder KL-Thread vom BS mit eigenem Kontrollblock verwaltet (Threadtabelle im BS)
  - KL-Threads werden vom BS einzeln dispatcht

- Beispiele für Betriebssysteme mit Unterstützung für KL-Threads
  - Windows, Linux, Solaris 2, BeOS, Tru64 (früher DigitalUNIX)

UL-Threads

- Vollständige Realisierung im Adressraum der Anwendung
- (mindestens ein) KL-Thread als Träger: UL-Threads sind dem Betriebssystem völlig unbekannt
  - BS „sieht" (und dispatcht) nur den KL-Thread
  - Zwischen den dem KL-Thread zugeordneten UL-Threads kann dann ohne Beteiligung des BS gewechselt werden
  - Speichern/Laden des Zustands wird von einer Threading-Hilfsbibliothek („Laufzeitumgebung") durchgeführt
- Erzeugung, Umschaltung, Zerstörung von UL-Threads ähnelt einem Prozeduraufruf
  → kein Wechsel der Privilegstufe: Operationen sehr schnell

UL-Threads: Laufzeitumgebung

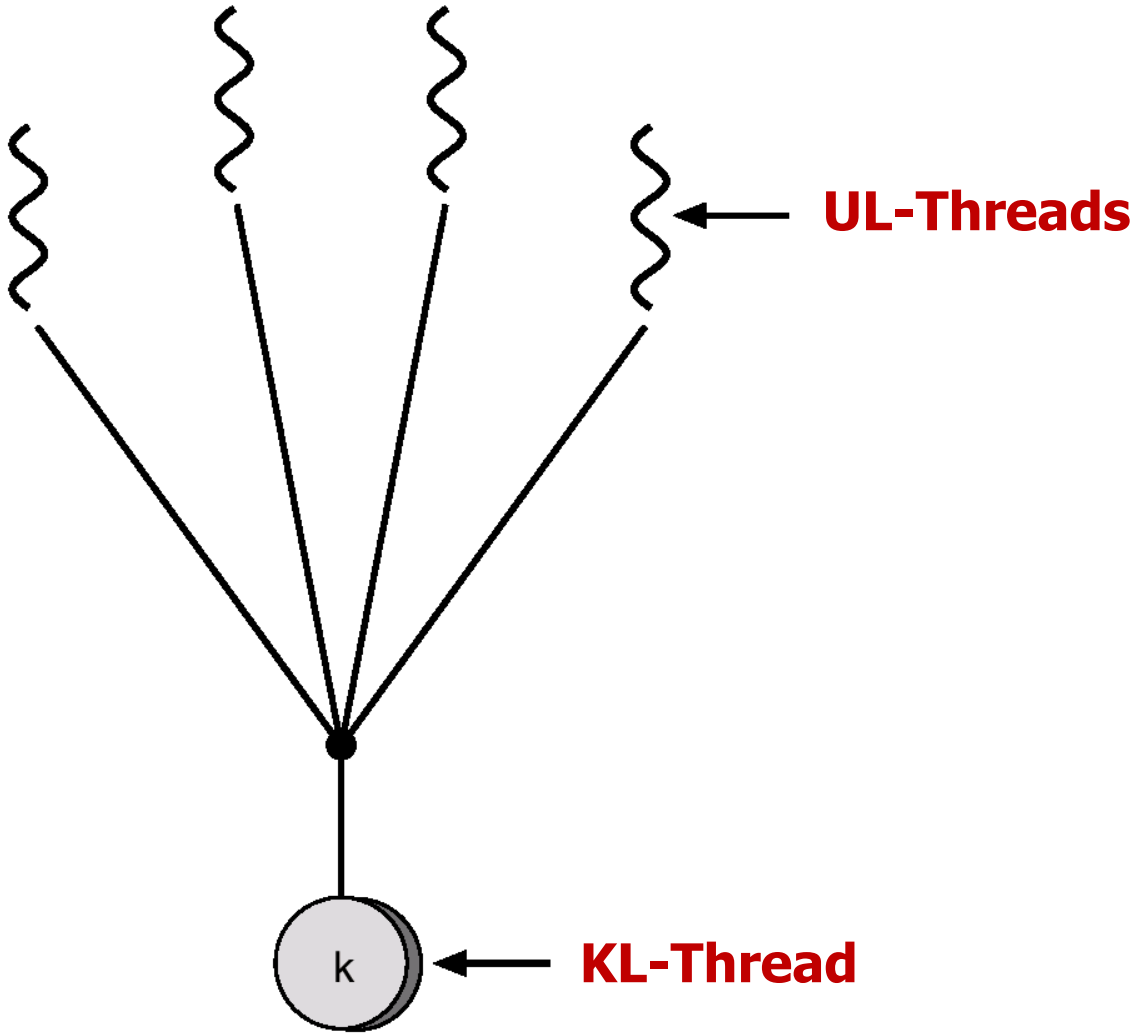
- Laufzeitumgebung zur Verwaltung der Threads
  - Aufgaben: Blockierung, Umschaltung, Scheduling, Erzeugung/Löschung, …
  - Verwaltung der Threadtabelle (Laden und Speichern von Register-werten und Zustand jedes Threads) im Datenbereich des Prozesses
  - Einsetzbar auch bei Betriebssystemen ohne Thread-Unterstützung

- Auch ein UL-Thread benötigt bestimmte Betriebsmittel (BM), z.B. eigenen Stack
  - Laufzeitumgebung kümmert sich um Allokation und fordert ggf. Ressourcen vom BS an

Multithreading-Modelle

- Mehrere Kombinationen aus UL- und KL-Threads möglich
  - UL-Threads: alle UL-Threads einem einzigen KL-Thread zugeordnet (many-to-one), Threadverwaltung durch die Laufzeitumgebung
  - KL-Threads: keine Laufzeitumgebung nötig, Verwaltung der Threads durch das BS
  - Hybrid: Zuordnung mehrerer UL- zu mehreren KL-Threads (many-to-many)

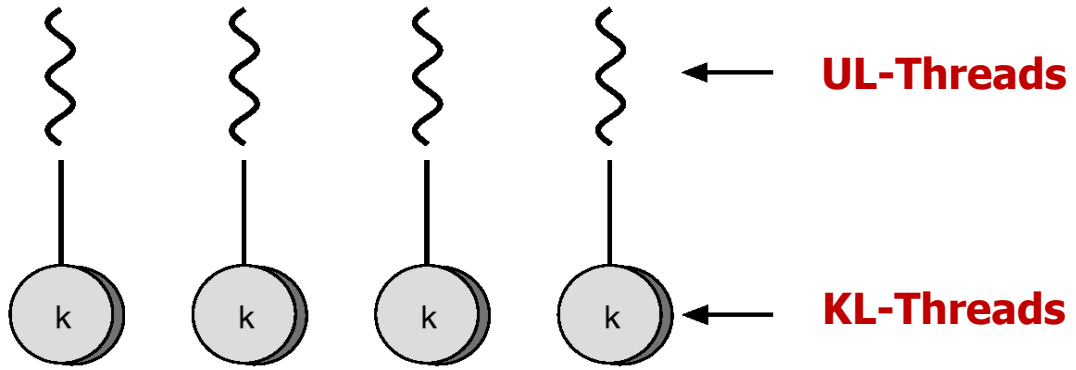
Multithreading-Modelle: User-Level (Many-to-One)



- KL-Thread als Träger für UL-Threads
- Vorteile:
  - Scheduling-Algorithmus beliebig
  - Sehr schnelle Verwaltung
- Nachteile:
  - Blockierender Systemaufruf blockiert alle UL-Threads
  - Nicht geeignet für Multicore (BS kann nur einen KL-Thread einer CPU zuordnen)
- Beispiele: Bibliotheken für BS ohne Threadunterstützung

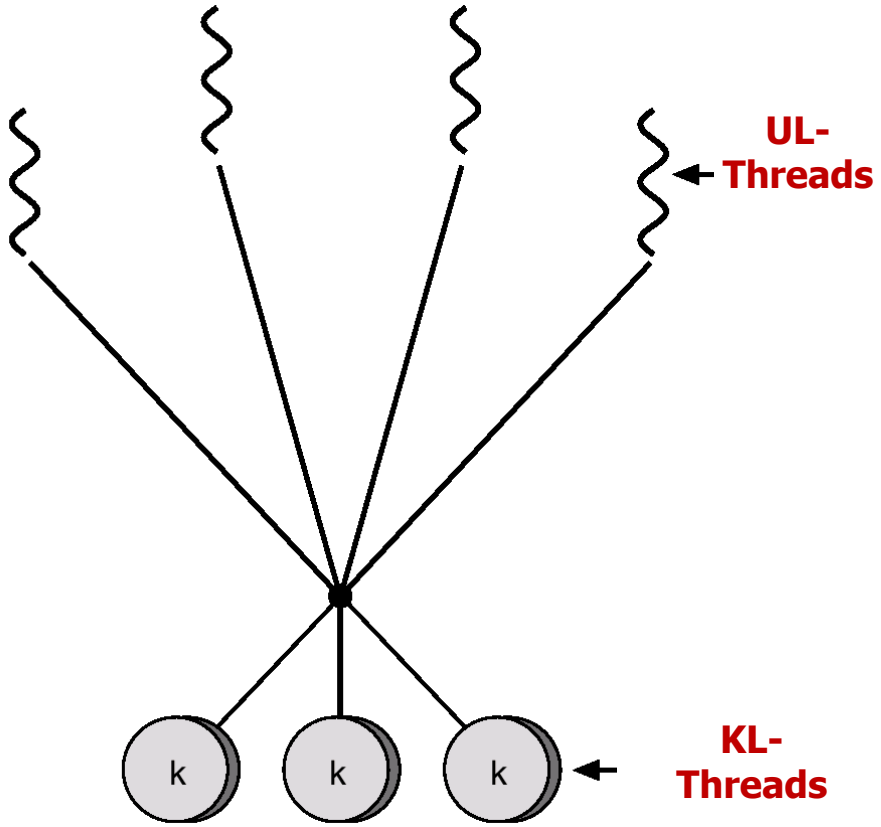
Multithreading-Modelle: Kernel-Level (One-to-One)

- Direkte Verwendung von KL-Threads bzw. Abbildung jedes UL-Threads auf genau einen KL-Thread
- Vorteile: blockierende Systemaufrufe ohne Einfluss auf übrige Threads → mehrere Threads parallel
- Nachteile: Verwaltung (Erzeugen, Umschalten, ...) von Threads ähnlich aufwendig wie entsprechende Prozessoperationen, nur ohne Adressraumwechsel
- Beispiele: Windows, Linux



Multithreading-Modelle: Hybrid (Many-to-Many)

- Mehrere UL-Threads auf gleich viele/weniger KL-Threads
  - Anzahl der KL-Threads wird durch die Anwendung oder in Abhängigkeit von Zielhardware (z.B. Anzahl CPUs) bestimmt
  - Kompromiss zwischen Modellen
    - Keine Blockierung durch Systemaufrufe
    - Parallele UL-Threads partiell möglich
    - Keine Beschränkung der Threadanzahl
    - Komplexe Randfälle (z.B. Ablauf der Zeitscheibe während des Wechsels zwischen zwei UL-Threads)
  - Beispiele: Solaris 2, HP-UX, Tru64 UNIX, IRIX



Verzweigen von Prozessen mit Fork / Join

- UNIX-Konzept fork/join (auch fork/wait) ermöglicht Erzeugung einer perfekten Kopie (Child) des aufrufenden Prozesses (Parent) mit folgenden Eigenschaften
  - Gleiches Programm
  - Gleiche Daten (gleiche Werte in Variablen)
  - Gleicher Programmzähler (nach der Kopie)
  - Gleicher Eigentümer
  - Gleiches aktuelles Verzeichnis
  - Gleiche Dateien geöffnet (selbst Schreib-, Lesezeiger ist gemeinsam)
  - Unterschiedliche Prozessnummer (PID)
- Zusammenführung der beiden Zweige mittels wait im Parent (im Pseudocode auch join)

Beispiel: fork

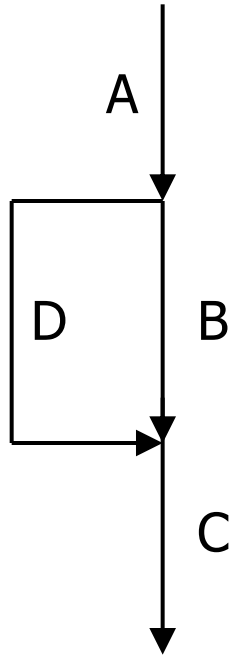
- Unterscheidungsmerkmal
  - fork()-Rückgabewert:
    - im Parent: PID des Childs, > 0
    - im Child: 0
    - (im Fehlerfall: <0)

```
pid_t p;          Parent
...
p = fork();
if (p == 0) {
/* child */ ...
}
else if (p > 0)
{
/* parent */ ...
}
```

```
pid_t p;          Child
...
p = fork();
if (p == 0) {
/* child */ ...
}
else if (p > 0)
{
/* parent */ ...
}
```

Beispielhafter Ablauf von `fork/join`

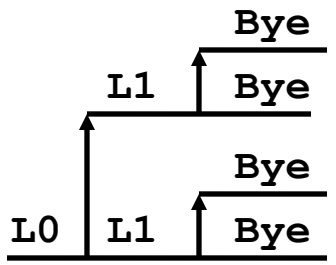
```
A
fork f
B
join f
C
end

f:  D
    end
```



Beispiel: fork (2)

- Sowohl Mutter als auch Kind können weiter forken

```
void fork2()
{
    printf("L0\n");
    fork();
    printf("L1\n");
    fork();
    printf("Bye\n");
}
```



2.4 Prozesshierarchien

- Prozesse können in diversen Beziehungen stehen:
  - Eltern-Kind-Beziehung: Ein Prozess erzeugt einen weiteren Prozess
  - Vorgänger-Nachfolger-Beziehung: Ein Prozess darf erst starten, wenn ein anderer Prozess beendet ist
  - Kommunikationsbeziehung: Zwei (oder mehr) Prozesse kommunizieren miteinander
  - Wartebeziehung: Ein Prozess wartet auf etwas, was von einem anderen Prozess kommt
  - Dringlichkeitsbeziehung: Ein Prozess ist wichtiger (dringlicher) als ein anderer
- … und viele andere mehr

Ereignisse zur Erzeugung von Prozessen

- Viele mögliche Ereignisse zur Erzeugung von Prozessen:
  - Initialisierung des Systems: Meistens Hintergrundprozesse (Daemons) wie Terminaldienst, Mailserver, Webserver, …
  - Prozesserzeugung durch andere Prozesse: Aufteilung des Prozesses in mehrere, nebenläufige oder parallele Aktivitäten, die als eigene KL-Threads initialisiert werden
  - Benutzerbefehle zum Starten eines Prozesses (Kommandozeile oder GUI)

- Technischer Ablauf in allen Fällen gleich
  - Bestimmter Prozess analysiert die Eingabe (z.B. von Benutzern oder Konfigurationsdateien)
  - Prozess sendet einen Systemaufruf zur Prozesserzeugung und teilt dem BS mit, welches Programm darin ausgeführt werden soll

Neue Prozesse in UNIX starten

- Prozesskopie mit fork() erzeugen, dann Code und Speicher wechseln mit z.B. int execl(char *path, char *arg0, char *arg1, ..., NULL)
- Lädt und startet ausführbares Programm
  - path ist der Pfad, wo sich die ausführbare Datei befindet
  - arg0 ist der Name des Prozesses (Programmname)
  - arg1, ... sind die eigentlichen Argumente (mit NULL terminiert)
  - Rückgabewert -1 im Fehlerfall

TODO: Fehlschlagen von fork() abfangen

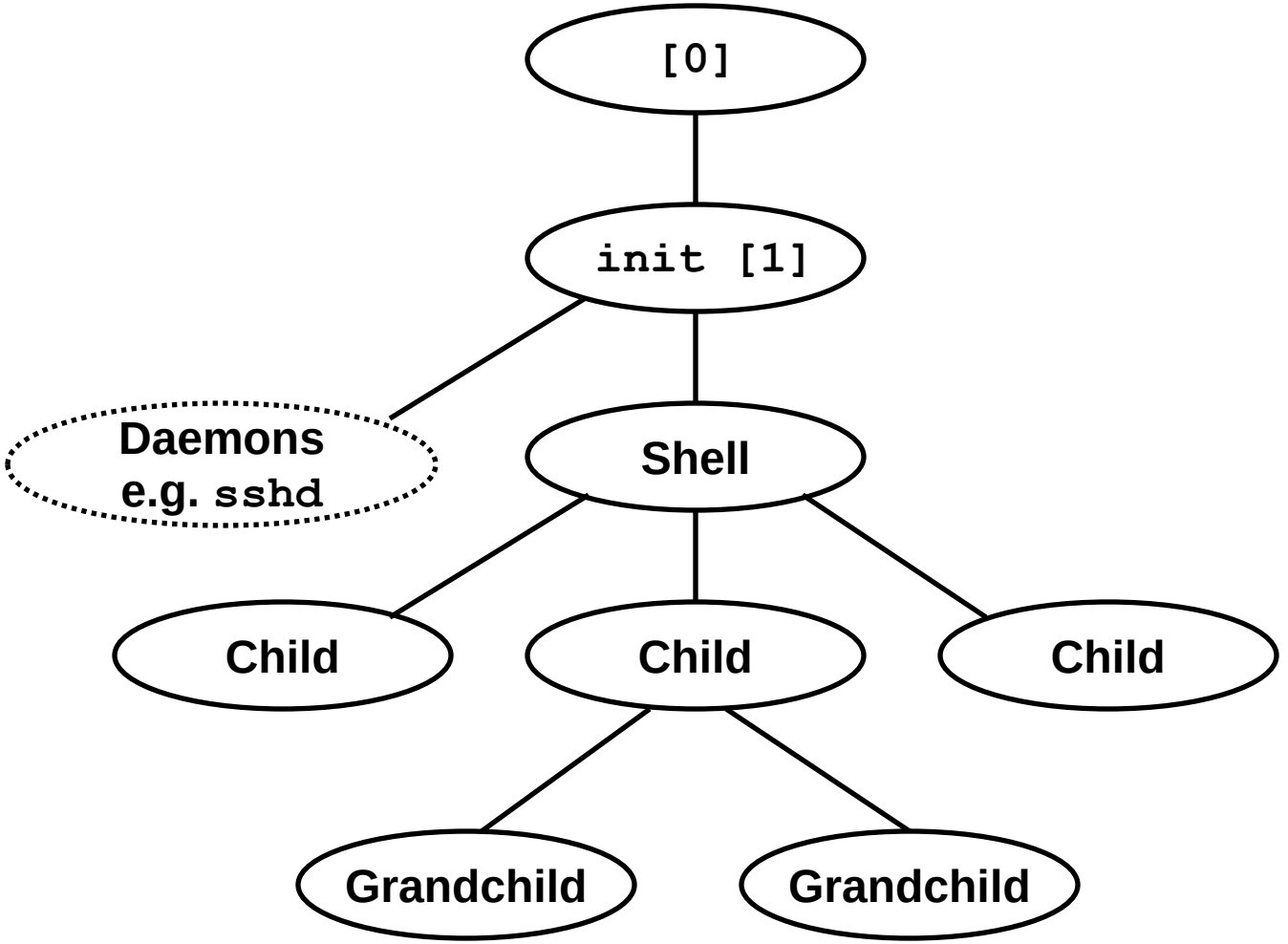
```
main() {
    if (fork() == 0) {
        execl("/usr/bin/cp", "cp", "a.txt", "b.txt", NULL);
        abort();
    }
    wait(NULL);
    printf("copy completed\n");
    exit();
}
```

Kind abbrechen, wenn execl() fehlschlägt

Prozesshierarchien

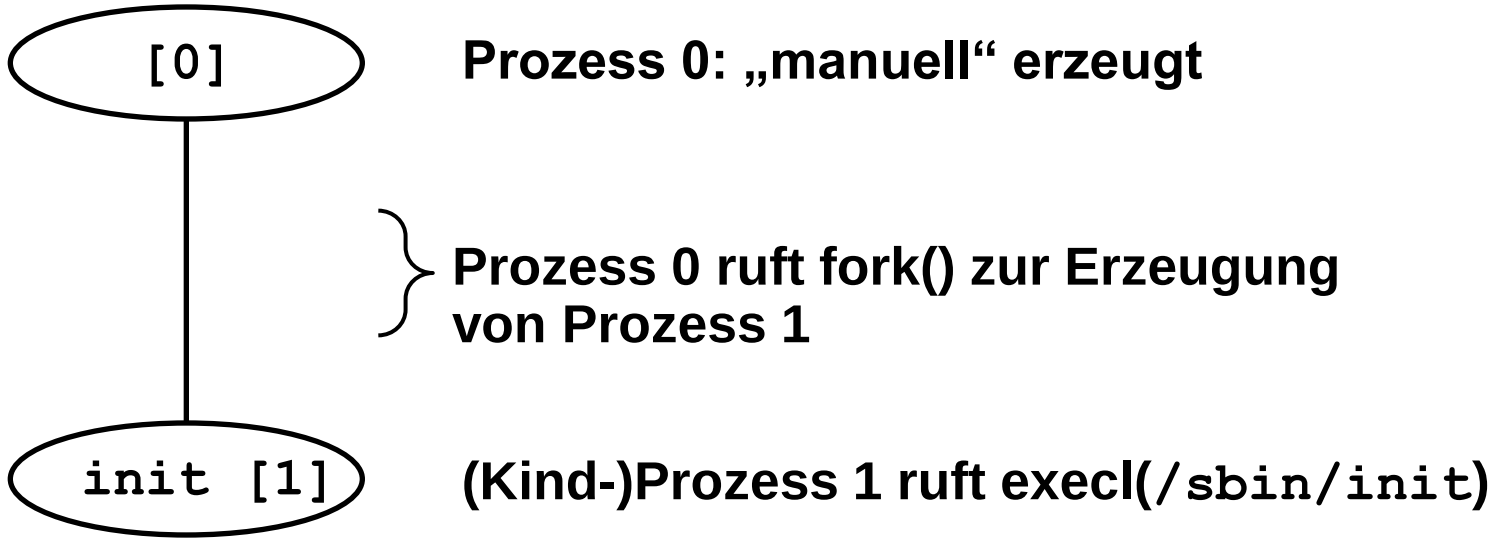
- Kindprozesse erzeugen weitere Prozesse (Kindkindprozesse) → Prozesshierarchie
- UNIX: Eltern- und Kindprozesse bilden eine Familie, z.B. kann ein Elternprozess Signale an seine Kindprozesse senden
  - Zombies: Kind terminiert vor Eltern → Platz in Tabelle belegt, freigegeben wenn Eltern auch terminieren
  - Weisen: (orphans): Elternprozess terminiert vor Kindprozess → Kind nicht mehr erreichbar
- UNIX-Initialisierung startet mit Prozess init
  - init startet weitere Hintergrunddienste, das Login-Programm und danach eine Shell
  - Shells erzeugen neue Prozesse bei Eingabe von Befehlen → Alle Prozesse gehören zu einem Baum mit init als Wurzel
- Windows: kein Konzept einer Prozesshierarchie
  - Elternprozess kann Kindprozess über ein Handle steuern, der auch an andere Prozesse weitergegeben werden darf → Prozesshierarchie wird außer Kraft gesetzt

Unix Prozesshierarchie

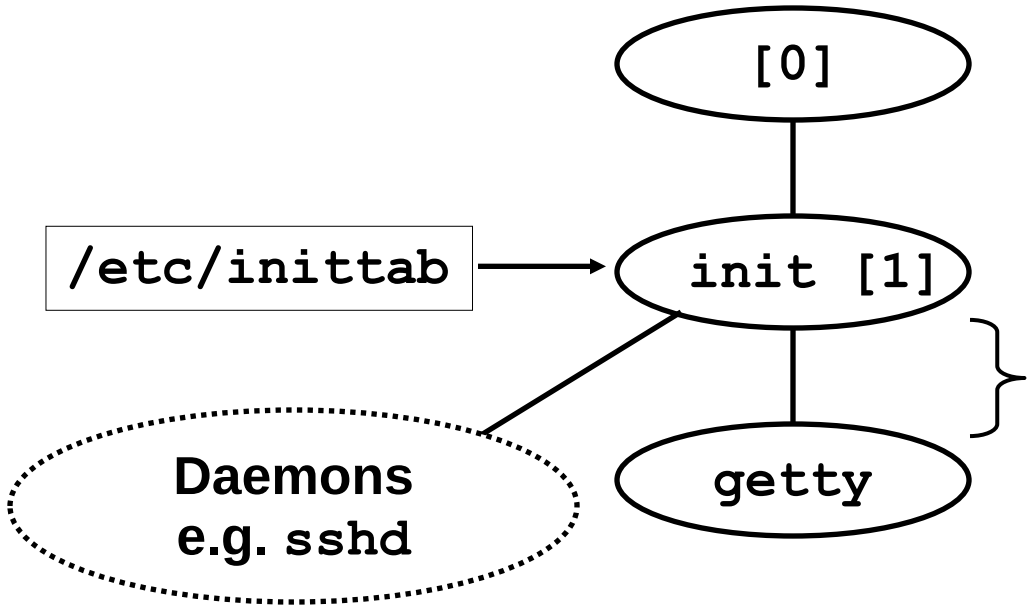


Unix Start: Schritt 1

- Drücken des Resetknopfs lädt den PC mit der Adresse eines kleinen Bootstrapprogramms (im ROM)
  - Bootstrapprogramm lädt den Bootblock (Plattenblock 0)
  - Bootblockprogramm lädt das Kernelbinärprogramm (z.B.: /boot/vmlinux)
  - Bootblockprogramm übergibt die Kontrolle an den Kernel
- Kernel kreiert „per Hand" die Datenstruktur (PCB) für Prozess 0

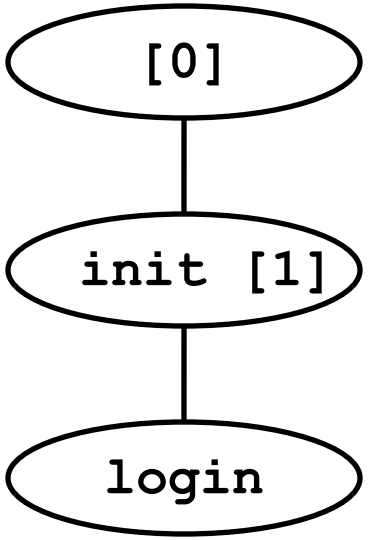


Unix Start: Schritt 2



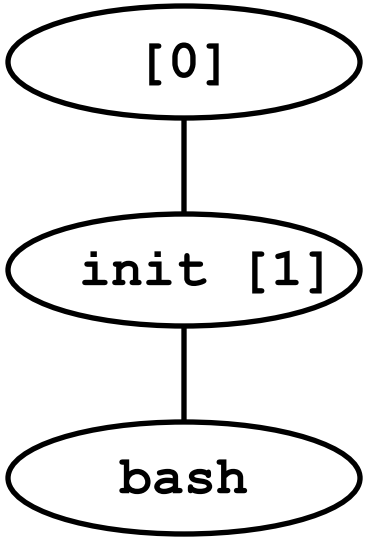
init forkt und startet Daemons wie in /etc/inittab deklariert, darunter u.a. ein getty-Programm für die Konsole

Unix Start: Schritt 3



Der getty-Prozess konfiguriert die Konsole und verwandelt (execl()) sich dann in ein login-Programm

Unix Start: Schritt 4



login liest Benutzername und Passwort. Wenn OK, execl() zur Shell des Benutzers (z. B. tcsh, bash, zsh...); wenn nicht OK, beendet sich login → init forkt neues getty

2.5 Shell

- Hilfsprogramm aus einzelnen Befehlen oder aus Shell-Programmen (genannt Script) zum Starten von Anwendungen
- Einsatzgebiete
  - Programme direkt ohne BS-GUI starten
  - Programme ohne eigene GUI starten (Server, Mikrocontroller, Container, ...)
  - Automatisierung von Prozessen und Workflows (hauptsächlich durch Skripte)
- Beispiele
  - Bourne-Shell (/bin/sh, 1977/78), Unix V7, Syntax als Grundlage für moderne Shells
  - C-Shell (csh, BSD, 1979) orientiert sich an C-Syntax
  - Bourne-again-shell (bash, GNU, Ende 80er), /bin/bash Standard-Shell auf Linux
  - Z-Shell (zsh, BSD): Mischung aus KornShell, bash, C-Shell, Standard unter MacOS
  - Windows PowerShell's basiert auf KornShell, aber mit vielen Erweiterungen

Basisfunktionen moderner Shells

- Erstellen und Starten von Scripts mit Befehlen
- Bedingungen (if, case) und Schleifen (while, for)
- Interne Befehle (cd, read) und Variablen ($HOME)
- Manipulation der Umgebungsvariablen für die neuen Prozesse
- Ein-/Ausgabeumlenkung, Starten mehrerer Prozesse, Verkettung über Pipes
- Starten von Prozessen im Hintergrund, Stoppen und erneutes Starten von Prozessen (job control)
- Vervollständigung von Befehlen, Dateinamen und Variablen (completion)
- Wiederholung und Editieren früherer Befehle (command history)
- Testen von Dateieigenschaften (test)
- Aufbau der Befehle: $ Befehl –Optionen Argumente (z.B. rm –r /temp)

Beispiel: Grundlegende Befehle bash

- ls (list): Inhalt von Ordnern bzw. Dateiattributen anzeigen
- pwd (print working directory): Pfad des aktuellen Ordners anzeigen
- cd (change directory): Ordner wechseln
  - Sondersymbole: ~ (Home-Ordner), . (aktueller Ordner), .. (übergeordneter Ordner)
- mkdir/rmdir (make/remove directory): Ordner erstellen/löschen
- rm (remove): Datei löschen
- cp (copy): Datei duplizieren
- mv (move): Datei verschieben
- Anwendung
  - ls /temp/
  - cp ~/beispiel/sysprog.txt .
  - rm –r /temp/
- Hilfe mit Befehlen: *man bash* oder z.B. *man cp* (man von manual)

Pipes und E/A-Umleitung

- Pipes verbinden zwei Shell-Befehle → Ausgabe des ersten Befehls dient als Eingabe für zweiten Befehl
  - ls | wc –l → Anzahl von Dateien und Verzeichnissen (wc = word count)
  - cat beispiel.txt | grep okao → Zeige Zeilen an, in denen okao vorkommt
- Jeder Befehl mit mehreren E/A-Optionen
  - Eingabe: stdin (0, Tastatur) oder Datei
  - Ausgabe: stdout (1, Bildschirm), stderr (2, Fehler) oder Datei
- E/A Umleitung: Änderung des Eingabe- (<) bzw. Ausgabemediums (>, >>)
  - wc < beispiel.txt → Zähle die Wörter in Datei beispiel.txt
  - wc < beispiel.txt > ergebnis.txt → wie oben, aber Ausgabe nicht in Shell, sondern in Datei ergebnis.txt
  - ./myCode < beispiel.txt 2 > fehlerlog.txt → Eingabe aus beispiel.txt, Ausgabe in shell, Fehlermeldungen in die Datei fehlerlog.txt

Variablen

- Vordefinierte Variablen
  - Skripte und Argumente: $0, ..., $9
  - PATH = Standardverzeichnisse, in denen nach Codes gesucht wird
  - HOME = Homeverzeichnis des aktuellen Benutzers
  - PPID = ProzessId des Elternprozesss
  - PWD = aktuelles Verzeichnis
- Operationen auf Shell-Variablen
  - Wertzuweisung mit =, z.B. NAME="Linux T" echo $NAME
  - Erweiterung, z.B. $PATH="$PATH:/home/sysprog/bin/"
- set: zeigt alle Variablen an
- export: Shell-Variablen werden zu globalen Umgebungsvariablen
- Befehl als Variable durch "$(Befehl)", z.B. echo Anzahl Dateien: "$(ls –l | wc –l)"

Scripts

- Ausführbare Shell-Programme (hier mycode) mit folgendem Inhalt
  - #!/usr/bin/bash → Festlegen des Interpreters (bash shell)
  - $1, $2, $3, … → Variablen für Eingabeparameter, z.B. mycode "Hello World" belegt $1 mit "Hello World”, während mycode "Hello world" erzeugt $1=Hello, $2=World
  - Sonderfälle: $0 ist der Name des Scripts, $@ bezeichnet alle Argumente
  - Auflistung von Befehlen, z.B

```
if [ "${1}" = "${2}" ]; then echo "Parameter identisch"
else echo "Parameter nicht identisch"
fi
echo mycode endet
```

  - chmod + x mycode / chmod 777 mycode → Datei mycode als ausführbar definieren
  - ./mycode Hello World ausführen

Befehle und Vergleiche

- Vergleich Strings
  - = (gleich) und (!=) nicht gleich
- Vergleich ganzer Zahlen
  - -eq (gleich), -ne (nicht gleich), -gt (größer als), -ge (größer gleich), -lt (kleiner als) …
- Klassische Befehle
  - if <Befehl>; then <Befehl> else <optionaler Befehl> fi
  - for <Variable> in <Liste> do <Befehle> done (*for j in $(seq 3 6) do sum=$(($j+$j)) echo $sum done*)
  - while <Bedingung = wahr> do <Befehle> done
  - until <Bedingung = falsch> do <Befehle> done

Bash Command Cheat Sheet

Navigating the File System

cd [directory]	Change directory
pwd	Print working directory
ls [options] [directory]	List directory contents
mkdir [directory]	Create a new directory
rmdir [directory]	Remove a directory
cp [source] [destination]	Copy files or directories
mv [source] [destination]	Move or rename files or directories
rm [options] [file]	Remove files or directories
touch [file]	Create an empty file

Archiving and Compression

tar [options] [files/directories]	Create or extract tar archives
gzip [file]	Compress a file
gunzip [file.gz]	Decompress a gzipped file
zip [archive.zip] [files/directories]	Create a zip archive
unzip [archive.zip]	Extract files from a zip archive

File Manipulation

cat [file]	Output the contents of a file
head [options] [file]	Output the first lines of a file
tail [options] [file]	Output the last lines of a file
less [file]	View the contents of a file interactively
grep [pattern] [file]	Search for a pattern in a file
wc [options] [file]	Count the number of lines, words, or characters in a file

Permissions

chmod [permissions] [file]	Change the permissions of a file or directory
chown [user:group] [file]	Change the owner and group of a file or directory
chgrp [group] [file]	Change the group of a file or directory
umask [mask]	Set the default file permissions for newly created files

Process Management

ps [options]	Display information about active processes
kill [process_ID]	Terminate a process
top	Display and manage the top processes
bg [job_ID]	Move a job to the background
fg [job_ID]	Bring a background job to the foreground

https://linuxstans.com/bash-cheat-sheet/