

Betriebssysteme und Netzwerke

Vorlesung 4

Artur Andrzejak

Open House Day

@SAP HANA Student Campus

- *Where?*

SAP Headquarters Walldorf/Germany

- *When?*

May 15th, 2019 09:30 - 17:00

- *Who?*

Students, PhD Students, Faculty Members

- *About What?*

Database Research and Development

- *More Information and Registration:*

tinyurl.com/hcod19

Internships,
Theses

Career
Opportunities

Free Lunch
and Snacks ☺



Scan me

Wiederholung Vorlesung 3

Programm vs. Prozess?

fork()?

Was passiert bei Prozesswechsel?

Was macht `X=$(cmd)`?

Geschichte und Typen von Shells

execve, waitpid?

Wie macht man eine Mini-Shell?

Process Control Block (PCB)?

Umfragen: <https://pingo.coactum.de/301541>

Prozesse - Verwaltung

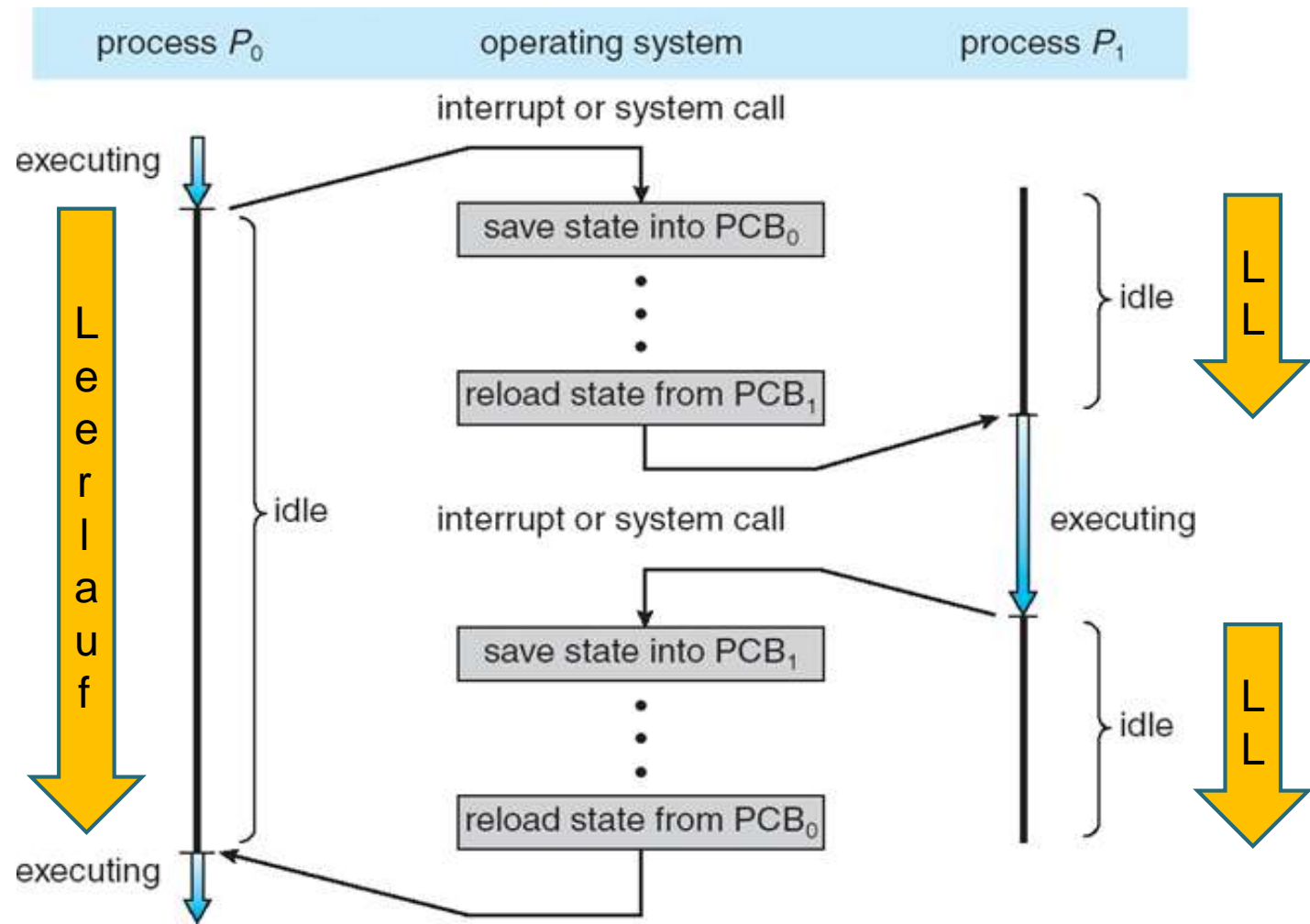


Prozesse und Multiprogrammierung

- ▶ Erinnerung: die Multiprogrammierung (zuerst bei OS/360) war ein erheblicher Fortschritt
- ▶ Es bedeutete zugleich einen **10-100 fachen Zuwachs an BS-Komplexität** – warum?
- ▶ Viele neue Funktionen waren notwendig:
- ▶ Verwaltung und Scheduling (Ablaufsteuerung) von Prozessen
- ▶ Kommunikation der Prozesse untereinander
- ▶ Schutz voneinander => komplexe Speicherverwaltung

Wechsel (Switch) zwischen Prozessen

- ▶ Prozesswechsel ist recht komplex
 - ▶ Aufwand ist abhängig von der CPU und dem BS



Prozesswechsel kostet (viel) Zeit!

1. Speichere den Kontext des Prozessors inklusive des Programmzählers und anderer Register
2. Aktualisierte den PCB des Prozesses, der gerade im Zustand „running“ ist
3. Verschiebe diesen PCB in die korrekte Warteschlange
4. Wähle einen anderen Prozess zur Ausführung aus
5. Aktualisierte den PCB des ausgewählten Prozesses
6. Aktualisiere die Datenstrukturen für Speichermanagement
7. Stelle den Kontext des ausgewählten Prozesses wieder her

Und: CPU-Caches werden geleert!

Wann soll man zwischen Prozessen wechseln?

- ▶ Clock Interrupt
 - ▶ Das Zeitbudget („Zeitscheibe“) einer unterbrochenen Ausführung eines Prozesses ist aufgebraucht
- ▶ I/O-Anfrage (z.B. für das Lesen einer Diskdatei)
- ▶ I/O-Interrupt
 - ▶ Daten für einen anderen Prozess sind bereit
- ▶ Prozess macht einen Systemaufruf
 - ▶ z.B. Erzeugt ein Kindprozess
- ▶ Fehler
 - ▶ Z.B. Prozess versucht, die Daten eines anderen zu lesen

Zustände **Waiting** und **Ready**

- ▶ Einige Ereignisse, die zu einer Unterbrechung führen
 1. P hat eine Anfrage an ein I/O-Gerät geschickt
 2. P hat ein Kindprozess erzeugt und wartet auf seine Terminierung
 3. Die Zeitzeuteilung (Zeitscheibe) von P ging zu Ende, und P wurde vom BS-Scheduler unterbrochen
- ▶ Fälle 1 und 2 unterscheiden sich vom Fall 3 – wie?
- ▶ Bei 1 und 2 wartet der Prozess auf etwas, und kann grundsätzlich nicht ausführen
 - ▶ Er ist im Zustand **waiting** (wartend)
- ▶ Bei 3 könnte der Prozess weiter ausführen
 - ▶ Er ist im Zustand **ready** (bereit)

Prozesszustände

Es ist sinnvoll, jedem Prozess einen **Zustand** zu geben

Neu: Prozess wird erstellt

Beendet (terminated):
Der Prozess hat die Ausführung beendet

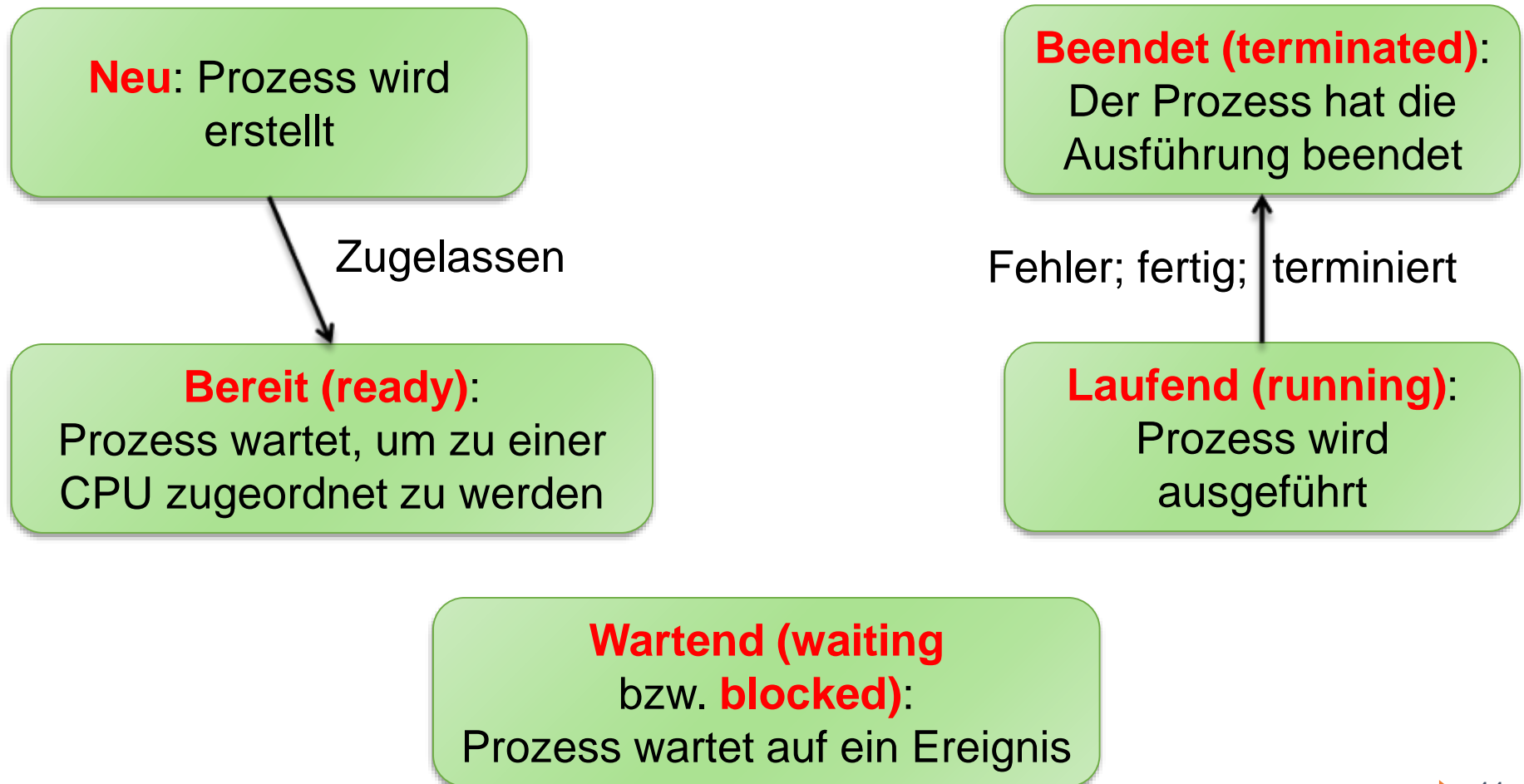
Bereit (ready):
Prozess wartet, um zu einer CPU zugeordnet zu werden

Laufend (running):
Prozess wird ausgeführt

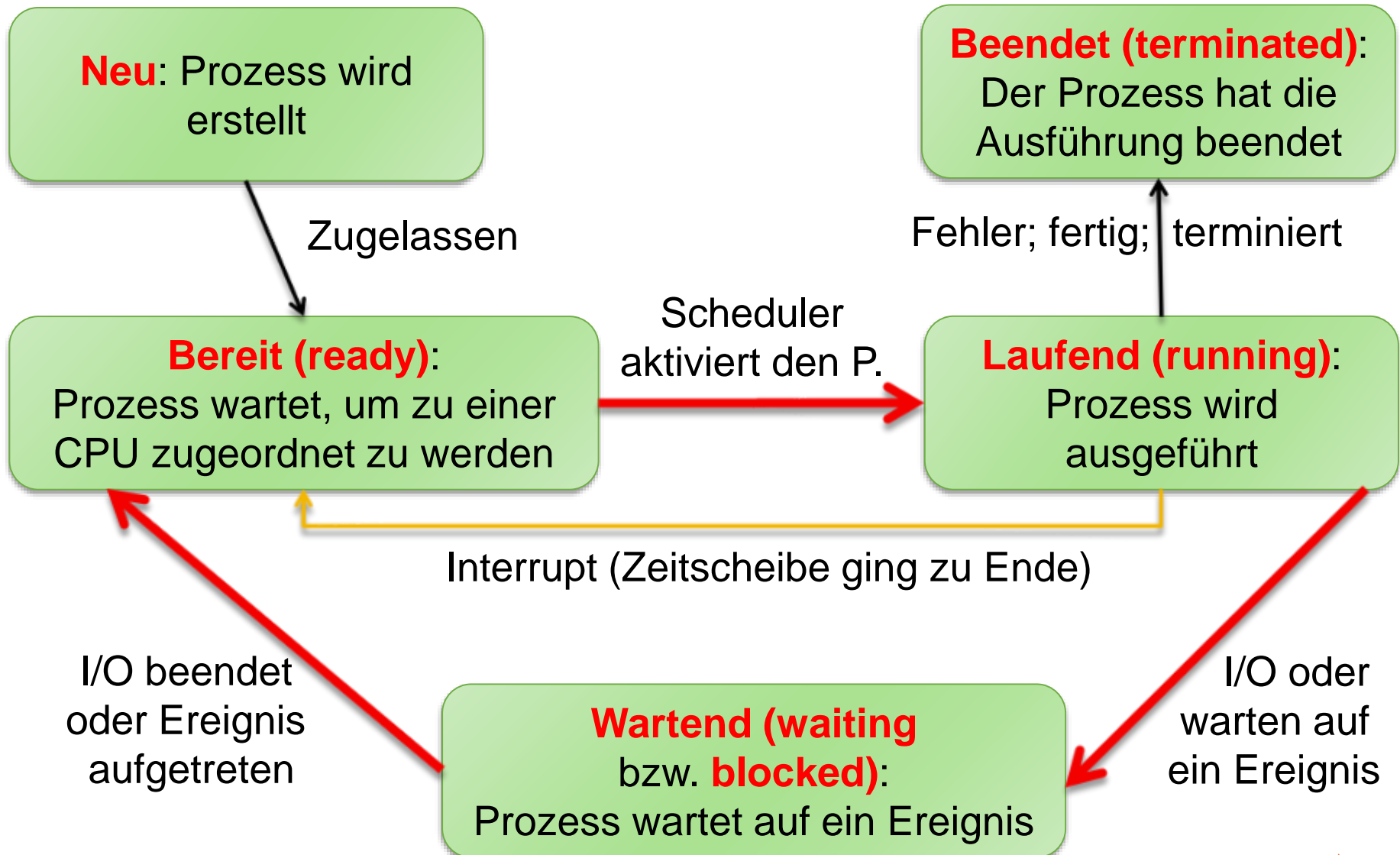
**Wartend (waiting
bzw. blocked):**
Prozess wartet auf ein Ereignis

Übergänge zwischen Prozesszuständen /1

- ▶ Welche Übergänge sind sinnvoll, und wodurch können sie ausgelöst werden?



Übergänge zwischen Prozesszuständen /2



Umfrage: Musiker im Restaurant

- ▶ In einem Restaurant essen zugleich Luciano Pavarotti, Helge Schneider, und Helene Fischer.
- ▶ Im Moment passiert folgendes:
 - ▶ Helene möchte bestellen, *wartet* auf den Kellner, und beginnt zu singen
 - ▶ Der Kellner *läuft* um den Tisch von Luciano *herum* und nimmt seine Bestellung auf. Da klingt das Handy von Luciano, er nimmt ab.
 - ▶ Helge hat sich noch nicht entschieden, er tüftelt statt dessen an seiner neuen Sakraloperette „I’m sowas von *blocked*, baby!“

Umfrage: Musiker im Restaurant

▶ ...

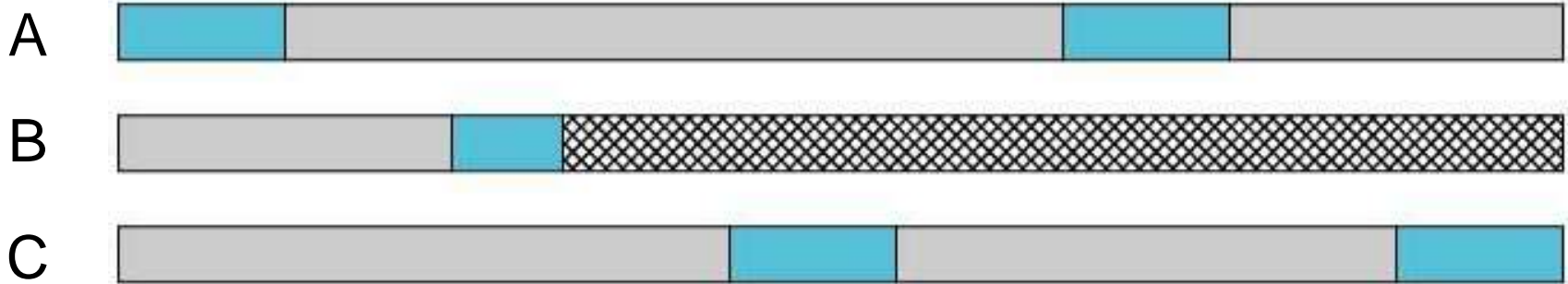
Welches Verhalten des Kellners K wäre korrekt in Analogie zur Prozessverwaltung?

- ▶ A. K geht zur Helge und bittet um sowie Bestellung (aber H. erzeugt nur unverständliche Geräusche)
- ▶ B. K geht zur Helene, und nimmt ihre Bestellung auf, während sie zu singen aufhört
- ▶ C. K bleibt bei Luciano und wartet, bis er sein Gespräch endet, dann notiert weiter
- ▶ D. K nimmt den Feuerlöscher und bedeckt Helene mit gleichmäßigen weißen Schaum, damit sie verstummt, er nimmt aber ihre Bestellung nicht auf

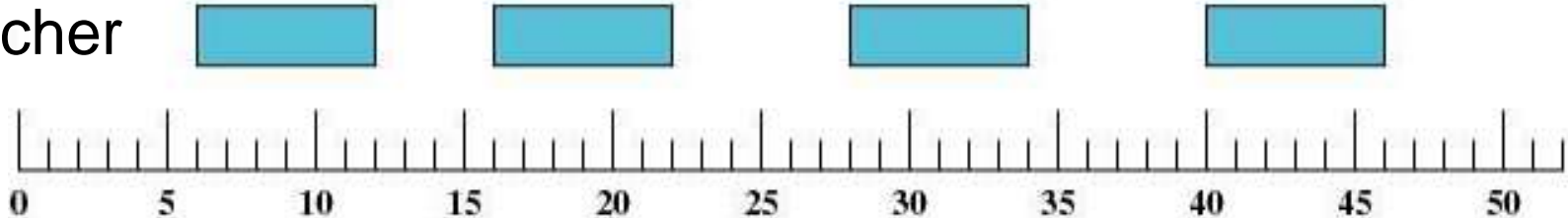
Prozessverwaltung: Interessante Details




Ablauf der Zustandwechsel

Prozess



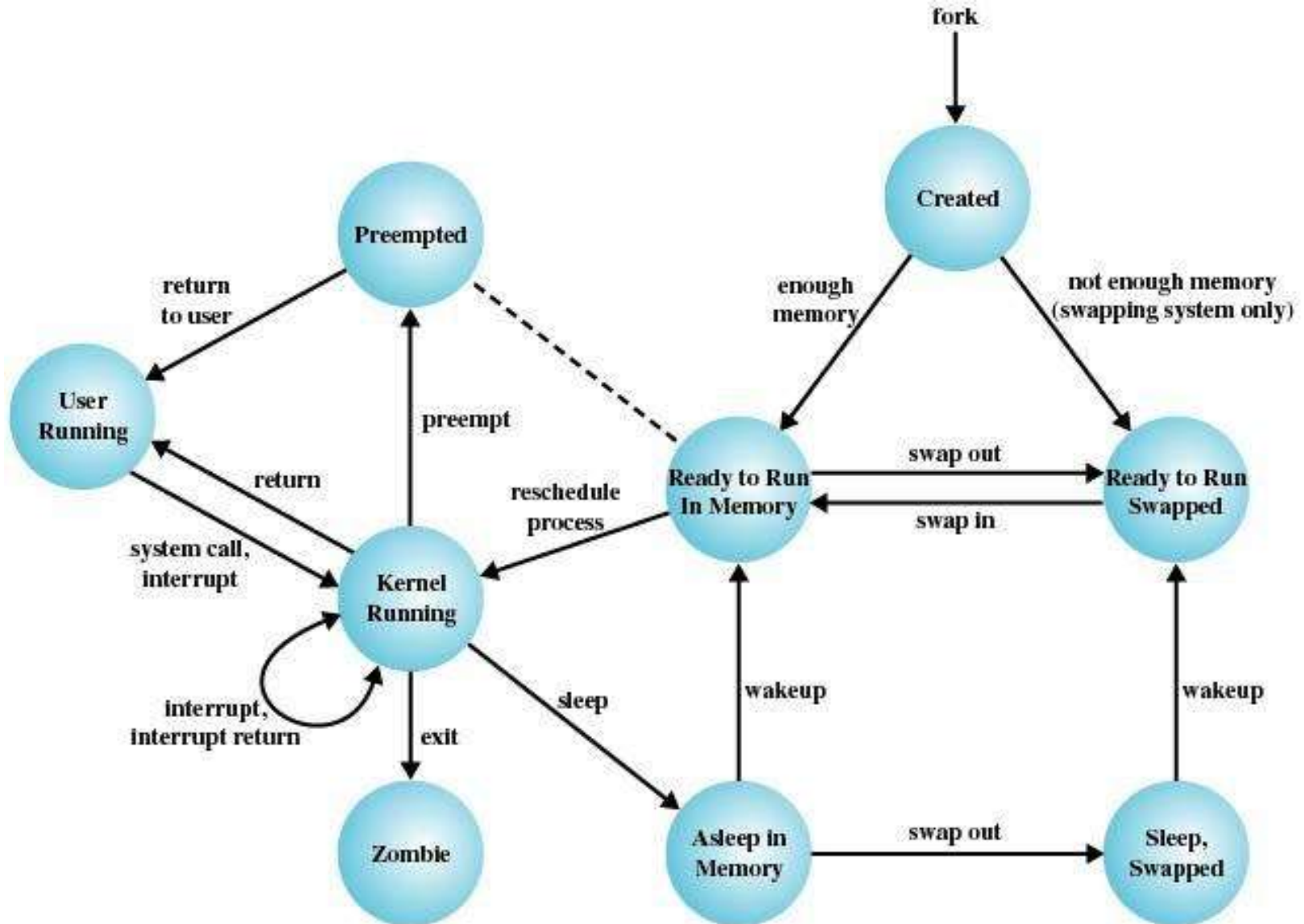
Dispatcher



 = Running  = Ready  = Blocked

Dispatcher = Rechenzeitverteiler
to dispatch = (ent)senden, erledigen, abfertigen

Wirkliche Prozesszustände (UNIX) /1

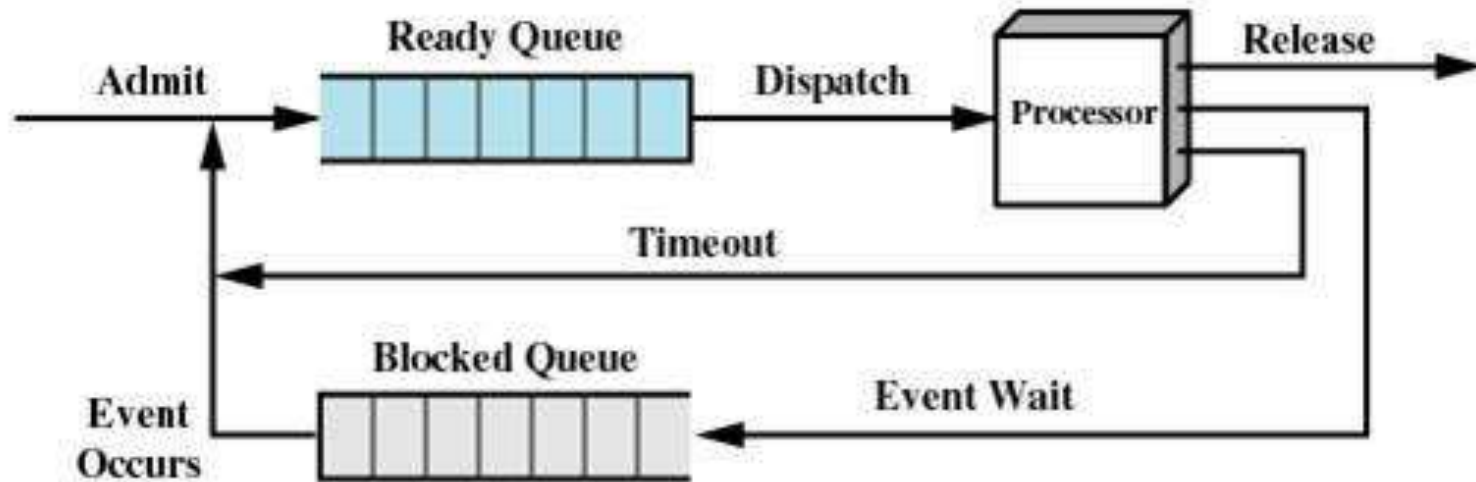


Wirkliche Prozesszustände (UNIX) /2

Zustand	Erläuterung
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 (blocked)	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

Warteschlangen /1

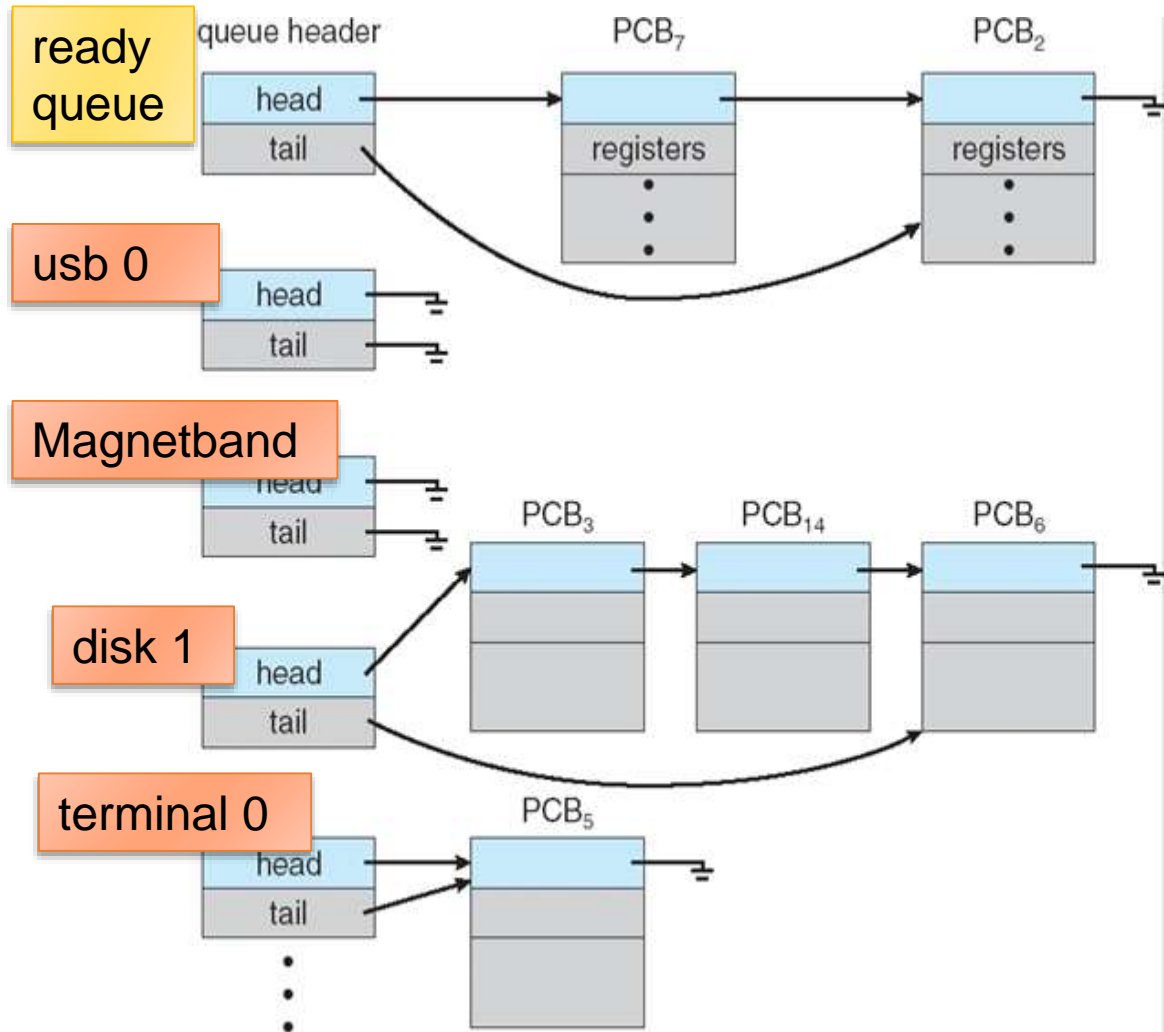
- ▶ Wie erkennt das BS, welche Prozesse bereit und welche wartend (blockiert) sind?
- ▶ Hier helfen die sog. **Warteschlangen (Queues)**
 - ▶ **Ready Queue**: alle Prozesse, die bereit sind
 - ▶ **Blocked Queue**: alle blockierten Prozesse



- ▶ Man kann das effizienter machen ...

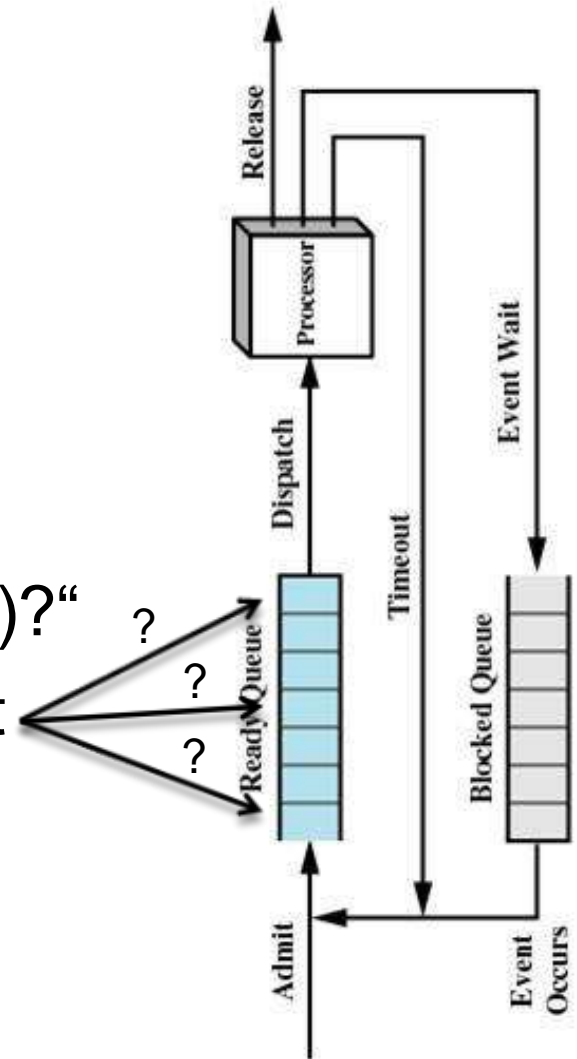
Warteschlangen /2

- ▶ Man führt die **device queues** ein: Warteschlange pro Ereignistyp
- ▶ Bei einem Ereignis von einem Gerät (z.B. USB, Netzwerk) weiß man ohne Suchen, welche Prozesse in den Zustand „bereit“ versetzt werden können



Prozessscheduling

- ▶ **Scheduling** bedeutet Ablaufsteuerung
 - ▶ „Steuerung der Zuteilung von Ressourcen an Prozesse“
- ▶ Bei der Ressource „CPU“ ist Scheduling das Entscheidende: „Welcher Prozess soll jetzt ausgeführt werden (und wie lange)?“
- ▶ Einfachstes Vorgehen: Man nimmt immer den Prozess am Kopf der Ready-Queue
 - ▶ In Wirklichkeit ist das komplizierter



Scheduler: Definition und zwei Typen

- ▶ **Scheduler** sind Implementationen von Algorithmen, die über **Scheduling** (= Ablaufsteuerung) entscheiden
- ▶ **CPU-Scheduler** bzw. **Kurzzeit-Scheduler**
 - ▶ Entscheidet, welcher der Prozesse als nächster ausgeführt wird, und ob ggf. der laufende Prozess unterbrochen wird
 - ▶ Wird sehr häufig (alle 10-100 Millisekunden) aufgerufen
- ▶ **Langzeit-Scheduler**
 - ▶ Wählt die Prozesse aus, die überhaupt in die „ready“-Schlange aufgenommen werden
 - ▶ Wird sehr selten (alle paar Sekunden, Minuten) aufgerufen
 - ▶ Von Bedeutung in Cluster-Systemen, Supercomputern usw.

Prozesserzeugung und Prozesshierarchien

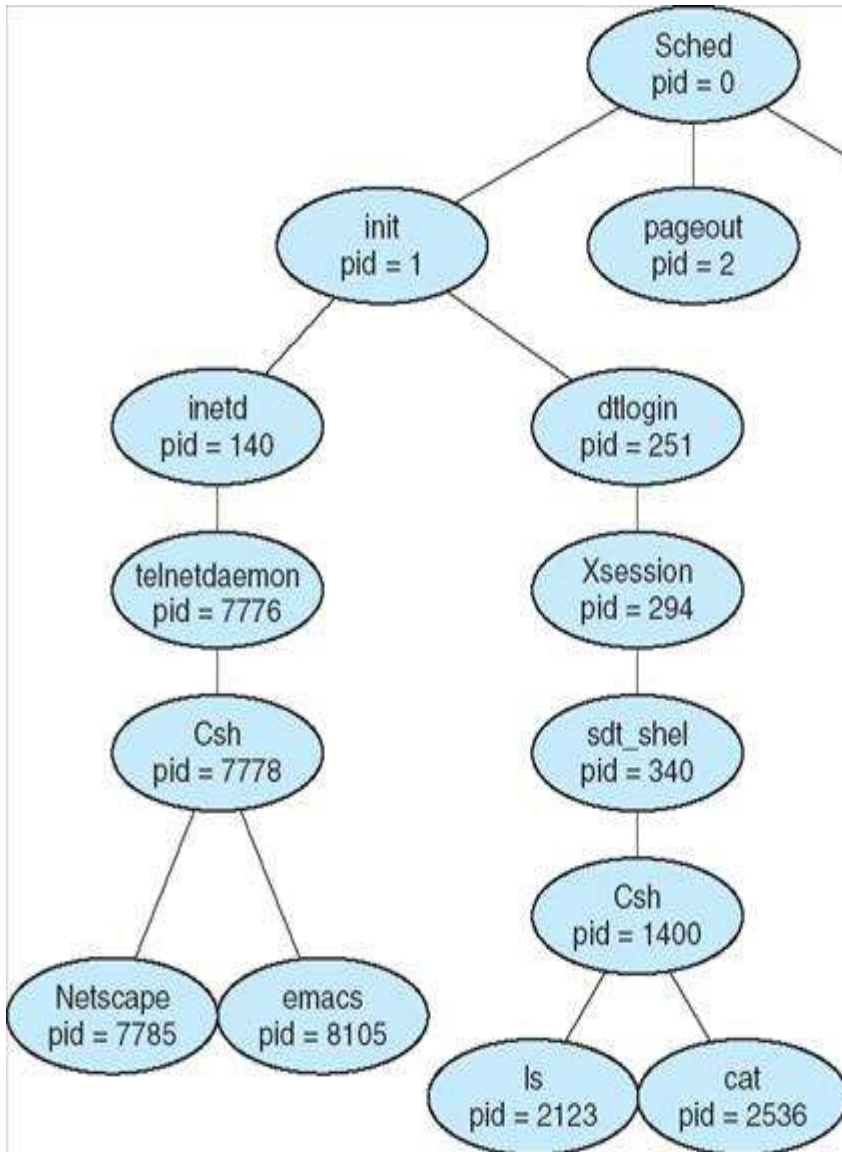
Prozess init

- ▶ Unter Linux / UNIX / erzeugt der Kernel nur wenige Prozesse
- ▶ Die restlichen erzeugt der Prozess **init**
- ▶ **init** führt alle Startup-Skripte aus und ist "Vater" aller Prozesse, die nicht vom Kernel erzeugt wurden
 - ▶ **init** hat **Process ID (PID) 1**
- ▶ Es erzeugt eine **Prozesshierarchie**

Prozesshierarchie Linux: `ps tree -p`

```
init(1)-+-NetworkManager(1063)---{NetworkManager}(1131)
|-accounts-daemon(1384)---{accounts-daemon}(1385)
|-at-spi-bus-laun(1482)-+-dbus-daemon(1486)
...
|-console-kit-dae(1401)-+-{console-kit-dae}(1402)
|                          |-{console-kit-dae}(1403)
...
|-cron(1090)---cron(12263)---sh(12264)---getydataLinux.s(12265)---java(12266)-+-{java}(12267)
|                                                                    |-{java}(12268)
...
|-rsyslogd(906)-+-{rsyslogd}(937)
|                |-{rsyslogd}(938)
|                `-{rsyslogd}(939)
|-rtkit-daemon(1654)-+-{rtkit-daemon}(1655)
|                    `-{rtkit-daemon}(1656)
|-sshd(898)---sshd(7057)---sshd(7223)---bash(7226)---pstree(7364)
|-ubuntu-geoip-pr(1687)-+-{ubuntu-geoip-pr}(1688)
|                        `-{ubuntu-geoip-pr}(1689)
```

Beispiel Prozesshierarchie Solaris



- ▶ „**sched**“: „Urprozess“
 - ▶ erzeugt Systemprozesse **init**, **pageout**, **fsflush**
- ▶ **init**: die „Wurzel“ aller anderen Prozesse
- ▶ **inetd**: verantwortlich für alle Netzwerkdienste
- ▶ **dtlogin**: erzeugt Benutzerlogin „Maske“
- ▶ **Xsession**: X-windows Hauptprozess
- ▶ ...

Video

- ▶ **Linux Architecture Overview**

- ▶ <https://www.youtube.com/watch?v=2a05xjNceP0&list=PLqE63EN7m04eoD84hdrHK7rt9-zsZHe78&index=14>
- ▶ Von 18:28 (min:sec) bis 20:29 [04a]

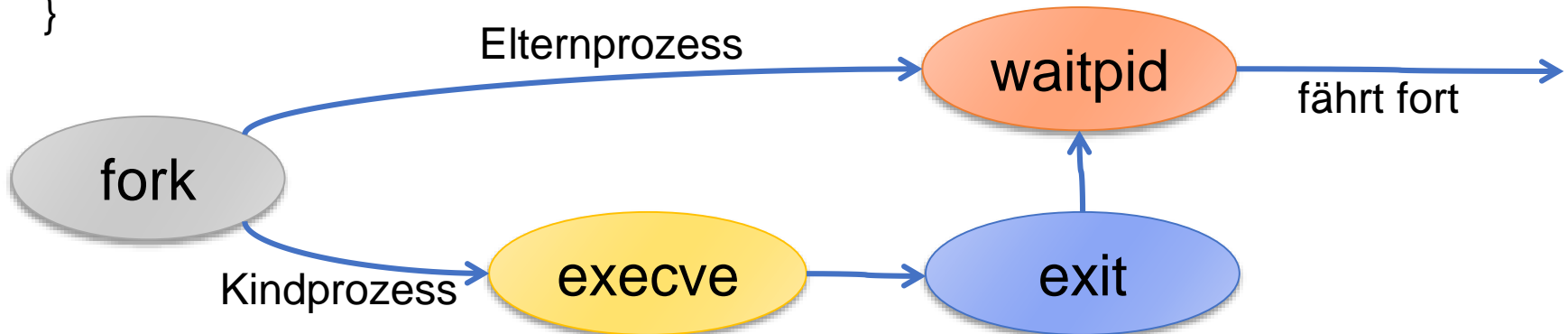
- ▶ Auch sehr interessant in diesem Video:

- ▶ "What happens when linux boots"
- ▶ Von 12:00 bis 18:28

Prozesserzeugung POSIX - Wiederholung

```
#define TRUE 1
while (TRUE) {
    type_prompt();           /* Prompt ausgeben */
    read_command (command, parameters); /* Befehl einlesen */

    if ( fork() != 0 ) {    /* Kindprozess erzeugen */
        /* Code des Elternprozesses */
        waitpid (-1, &status, 0); /* Auf Ende des Kindprozesses warten */
    } else {
        /* Code des Kindprozesses */
        execve (command, parameters, 0); /* Befehl command ausführen */
    }
}
```



Fork() – was wird vererbt?

- ▶ Speicher: Code (Text), Halden (Heaps), Stack
- ▶ Alle offenen Dateideskriptoren ("Zeiger" auf Dateien)
- ▶ Arbeits- und Wurzelverzeichnis
- ▶ Umgebungsvariablen
- ▶ Ressourcenlimits
- ▶ Dateierstellungsmaske; Signalmaske und Handler
- ▶ UID, EUID, GID, EGID, Prozessgruppen-, Session-, Setuser- und Setgroup-ID
- ▶ Steuerterminal (CTTY)
- ▶ Shared-Memory-Segmente (falls vorhanden)

"Fortgeschrittene"
Details, nur der
Vollständigkeit halber

Prozesserzeugung in Windows /1

```
#include <stdio.h>
#include <windows.h>
int main(VOID)
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    // allocate memory
    ZeroMemory(&si, sizeof (si)) ;
    si.cb = sizeof (si) ;
    ZeroMemory(&pi, sizeof(pi));
```

Prozesserzeugung in Windows /2

```
// create child process
if (!CreateProcess (
    NULL,    // use cmd line
    "C:\\WINDOWS\\system32\\mspaint.exe", // cmd line
    NULL,    // don't inherit process handle
    NULL,    // don't inherit thread handle
    FALSE,   // disable handle inheritance
    0,       //no creation flags
    NULL,    // use parent's environment block
    NULL,    // use parent's existing directory
    &si, &pi))
```

Ressource
Sharing

Prozesserzeugung in Windows /3

```
{ /* if-body: 0 returned by CreateProcess */  
    fprintf (stderr, "Create Process Failed");  
    return -1;  
}  
  
// parent will wait for the child to complete  
WaitForSingleObject (pi.hProcess, INFINITE);  
printf ("Child Complete");  
  
// close handles  
CloseHandle (pi.hProcess);  
CloseHandle (pi.hThread);  
}
```


Interprozess-Kommunikation (IPC)

Kooperierende Prozesse

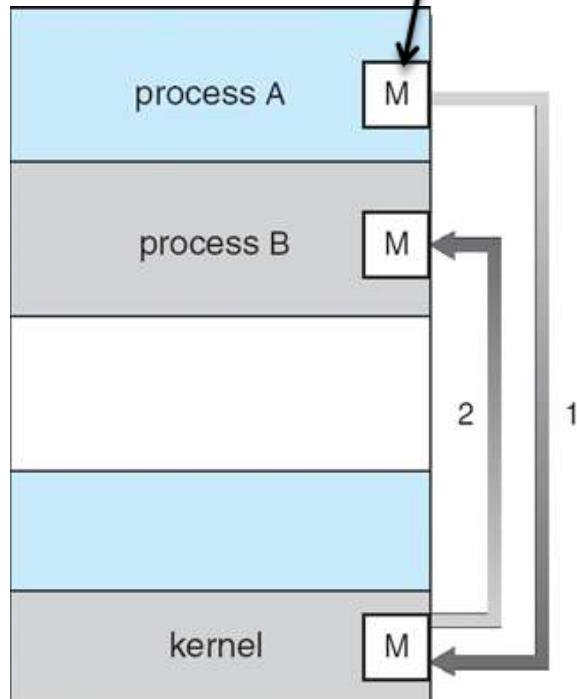
- ▶ Prozesse laufen i.A. unabhängig und können sich gegenseitig nicht beeinflussen
- ▶ Kooperation von Prozessen hat aber Vorteile – z.B.?
- ▶ **Mitbenutzung von Information**, z.B. mehrere Benutzer teilen sich ankommende Sensordaten (**information sharing**)
- ▶ **Beschleunigung der Ausführung**, z.B. wenn eine Aufgabe in mehrere (kooperierende) Teilaufgaben aufgeteilt wird
- ▶ **Modularität** von Programmen
- ▶ Höherer **Komfort** für den Benutzer: Word kann mit Excel „sprechen“

Inter-Prozess Kommunikation (IPC)

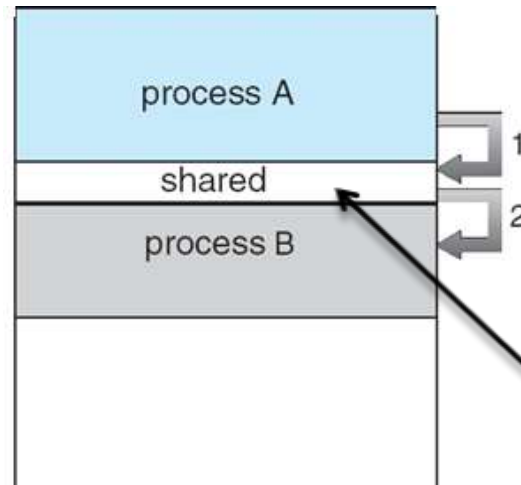
- ▶ Es gibt zwei prinzipielle Arten von IPC
 - ▶ **1: Nachrichtenübermittlung - message passing (MP)**
 - ▶ **2: gemeinsamer Speicher - shared-memory (SM)**

Ein kleiner Speicherbereich von A (Nachricht M) wird durch das BS in den Adressraum von B kopiert

1



2



Was ist effizienter?
Was wird häufiger benutzt?

Der gemeinsame Speicher kann in den Adressraum mehrerer Prozesse eingeblendet werden, als ob dieser ein Teil des Speichers des Prozesses wäre

IPC hat Verschiedene Formen

- ▶ **Datenströme:** Kanäle, die die Daten **sequentiell** von einem Prozess zu einem anderen schicken
 - ▶ Pipes, Sockets, Message Queues, Terminal, ...
- ▶ **Ereignisse:** Spezielle Nachrichten, ggf. ohne Daten
 - ▶ Interrupts, Signale, Windows DDE, Apple Events
- ▶ **Entfernte Funktionsaufrufe:** Aufruf von Funktionen innerhalb anderer Prozesse
 - ▶ Remote Procedure Call/Remote Method Invocation, CORBA, DCOM, Java RMI
- ▶ **Shared Memory:** APIs für gemeinsamen Speicher mehrerer Prozesse

Interprocess Kommunikation in Android

- ▶ Einfachere Mechanismen
- ▶ **Sandboxing**
 - ▶ Gleiches “privates Dateisystem” für mehrere Prozesse
 - ▶ Video: #2 Android Interprocess Data Exchange Part 1 [HD 1080p], <https://www.youtube.com/watch?v=4u-xpB1RFfs>, von 0:15 bis ca. 3:30 (min:sec)[04b]
- ▶ **Binder** ist eine ressourcenschonende Form von IPC mittels Shared Memory
 - ▶ Programme tauschen per Nachrichten lediglich Referenzen auf Objekte aus, die im Shared Memory abgelegt sind

Zusammenfassung

- ▶ Prozessverwaltung
 - ▶ Wechsel zwischen den Prozessen
 - ▶ Zustände der Prozesse (Vereinfachung: fünf Zustände)
 - ▶ Prozesserzeugung und Prozesshierarchien
- ▶ Inter-Prozess Kommunikation (IPC)
 - ▶ Via Nachrichtenübermittlung (message passing)
- ▶ Quellen
 - ▶ Silberschatz et al., Kapitel 3
 - ▶ Wikipedia

Danke schön.

Zusatzfolien

Prozessterminierung (mehrere BS)

- ▶ Prozess terminiert sich selbst via **exit**
 - ▶ Kann Daten an seinen Elternteil übermitteln (Lesen via wait())
 - ▶ Seine Ressourcen (Hauptspeicher, offene Dateien, Fenster, ...) werden vom BS automatisch freigegeben
- ▶ Elternteil kann ein Kindprozess terminieren
 - ▶ **#include <signal.h>**
 - ▶ **int kill(pid_t pid, int sig)**
- ▶ Falls ein Elternteil terminiert, werden in manchen BS auch alle Kinder terminiert
 - ▶ **Kaskadierte Terminierung** (cascading termination)
 - ▶ z.B. in Virtual Memory System (VMS) von DEC / HP, heute OpenVMS

Message Passing: Nachrichtenübermittlung

- ▶ Nochmals: Es werden kleine „Speicherblöcke“ (Nachrichten) zwischen den Prozessen kopiert
- ▶ Es gibt einige Varianten
 - ▶ **Direkte** oder **indirekte** Kommunikation
 - ▶ **Synchrone** oder **asynchrone** Kommunikation
 - ▶ **Automatische** oder **explizite Pufferung** der Nachrichten
 - ▶ **Uni-** oder **bidirektionale** Verbindung

Message Passing: Direkte Kommunikation

- ▶ Prozesse müssen einander explizit benennen
 - ▶ **send (P, message)** - Senden an Prozess P
 - ▶ **receive (Q, message)** – Empfangen einer Nachricht von Q
- ▶ Eigenschaften einer Verbindung
 - ▶ Verbindung wird automatisch eingerichtet
 - ▶ Verbindung besteht nur zwischen den beiden kommunizierenden Prozessen
 - ▶ Die Verbindung ist (normalerweise) bidirektional
- ▶ Problem?
 - ▶ Änderung der Kennung eines Prozesses kann bewirken, dass alle anderen Teile umgeschrieben werden müssen

Message Passing: Indirekte Kommunikation

- ▶ Wir lösen dieses Problem (wieder) durch Indirektion
- ▶ Nachrichten werden über **Postfächer (Mailboxes)** zugestellt (auch als **Ports** bezeichnet)
 - ▶ Jedes Postfach hat eine eindeutige Kennung (ID)
- ▶ Protokoll
 - ▶ **send (A, message)** - Senden einer Nachricht an Postfach A
 - ▶ **receive (A, message)** - Holen einer Nachricht vom Postfach A
- ▶ Eigenschaften?
 - ▶ Prozesse können nur kommunizieren, nachdem sie ein Postfach eingerichtet haben (-)
 - ▶ Ein Postfach kann mehrere Prozesse verbinden (+)
 - ▶ Jedes Paar von Prozessen kann über mehrere Wege kommunizieren (+)