

Betriebssysteme Vorlesung 2

Prozeßmanagement

Datenstrukturen, Vorgänge zur Verwaltung von
Prozessen

Literatur: Stallings Kapitel 3

Übersicht

Ein Prozeß ist ein in Bearbeitung befindliches Programm.

Ein Betriebssystem behandelt mehrere Prozesse gleichzeitig

-> Multiprogramming

Zugehörige Fragestellungen sind dann:

- Was passiert bei der Bearbeitung eines Prozesses ?
- Welche Situationen (Zustände) eines Prozesses muss ein OS unterscheiden ?
- Welche Informationen muß ein OS haben / verwalten, damit eine Prozeßbearbeitung unterbrochen und zu einem späteren Zeitpunkt fortgesetzt werden kann ?

Nutzen

- ◆ Sie lernen mit den Prozessen ein Konzept kennen, mit dem parallele Anwendungen auf einem Rechner (und in einem Rechnernetz) realisiert werden können.
- ◆ Sie erfahren, wie ein Betriebssystem es schafft, mehrere Aufgaben quasi simultan zu bearbeiten.

Prozeß (auch Task)

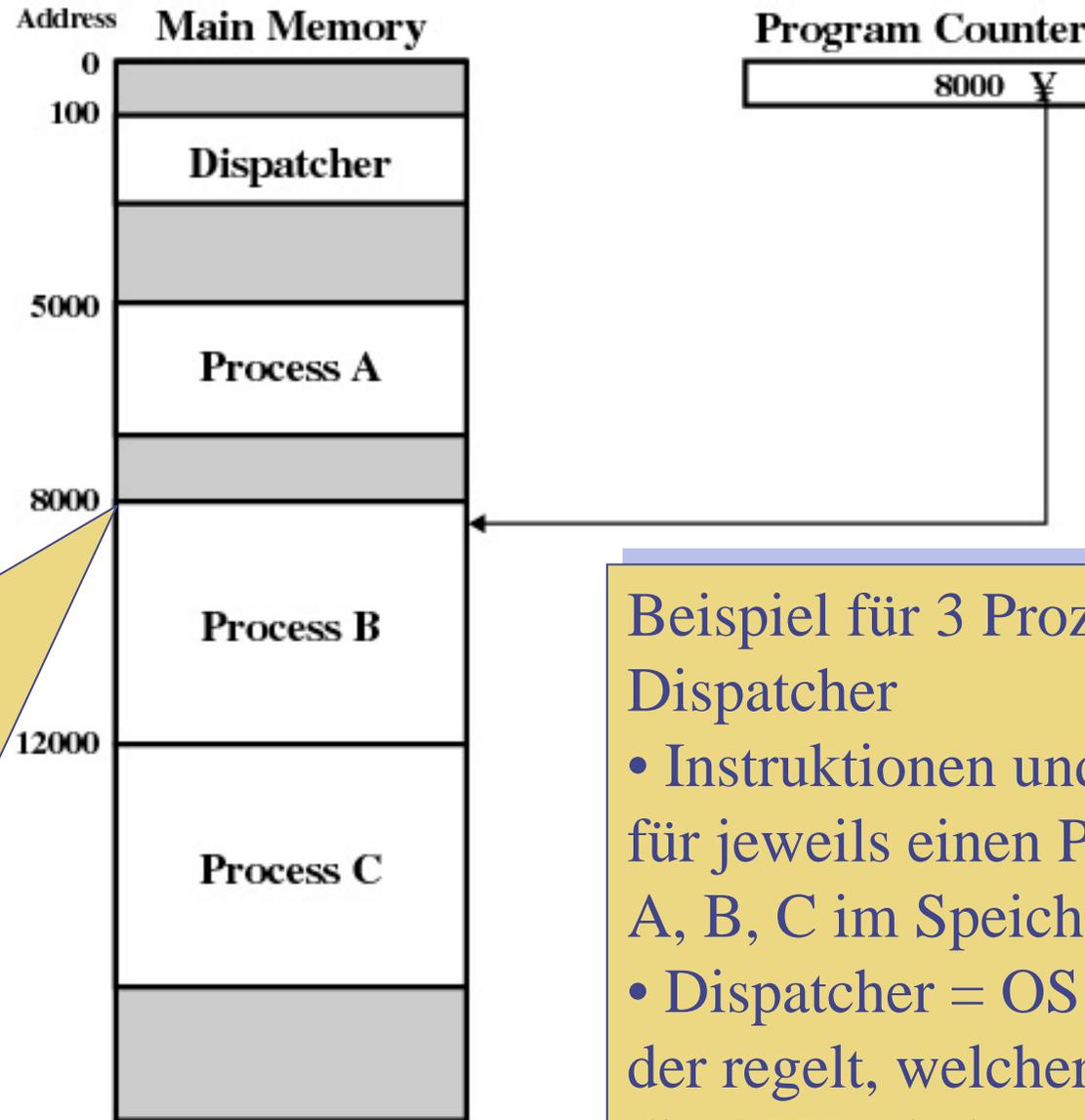
- ◆ Ein Prozeß ist ein in Ausführung befindliches Programm.
- ◆ Unterschied zu "Programm": dasselbe Programm kann mehrfach aufgerufen werden. Dies führt zu mehreren individuellen Prozessen.

Für ein Betriebssystem bedeutet dies,

- dass mehrere Prozesse versetzt ausgeführt werden müssen, um den Prozessor auszulasten wobei gleichzeitig akzeptable Antwortzeiten für einzelne Prozesse eingehalten werden müssen.
- dass Ressourcen (CPU, Speicher, E/A Geräte, Dateien) für diese Prozesse verfügbar gemacht werden müssen, wobei Konflikte erkannt und behandelt werden müssen.
- dass Prozesse ggfs selbst Prozesse erzeugen und zwischen Prozessen eine Kommunikationsmöglichkeit bestehen muss.
- ◆ Was passiert eigentlich bei der Bearbeitung eines Prozesses ?

Bearbeitung
von
Prozessen

Instruktions-
reihenfolge
ausgehend von
Anfangsadresse,
dann
„Spaghetticode“,
durch
Schleifen,
bedingte/
unbedingte
Sprünge



Beispiel für 3 Prozesse und
Dispatcher

- Instruktionen und Daten für jeweils einen Prozeß A, B, C im Speicher.
- Dispatcher = OS Prozeß der regelt, welcher Prozeß die CPU erhält

Figure 3.1 Snapshot of Example Execution (Figure 3.3)
at Instruction Cycle 13

Abarbeitung
von Prozessen

prozeßlokale
Sicht

5000
5001
5002
5003
5004
5005
5006
5007
5008
5009
5010
5011

(a) Trace of Process A

8000
8001
8002
8003

(b) Trace of Process B

12000
12001
12002
12003
12004
12005
12006
12007
12008
12009
12010
12011

(c) Trace of Process C

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

Trace (=Spur): Folge abgearbeiteter Befehle

hier: Abläufe für Prozesse A, B, C: sequentielle Folge abgearbeiteter Adressen, (gleichmäßige Zahlenfolgen häufig aber nicht ausschließlich, warum ?)

Abarbeitungsfolge
aus CPU Sicht

100 = Anfangsadr
des Dispatchers

1. Spalte zählt
Instruktionszyklen

2. Spalte gibt
Adresse des Befehls
an

Timeouts stammen
vom Dispatcher

I/O Anforderungen
stammen vom
jeweils laufenden
Prozeß A, B, C

1 5000
2 5001
3 5002
4 5003
5 5004
6 5005

-----Time out

7 100
8 101
9 102
10 103
11 104
12 105

13 8000
14 8001
15 8002
16 8003

-----I/O request

17 100
18 101
19 102
20 103
21 104
22 105

23 12000
24 12001
25 12002
26 12003

27 12004
28 12005

-----Time out

29 100
30 101
31 102
32 103
33 104
34 105

35 5006
36 5007
37 5008
38 5009
39 5010
40 5011

-----Time out

41 100
42 101
43 102
44 103
45 104
46 105

47 12006
48 12007
49 12008
50 12009
51 12010
52 12011

-----Time out

Prozeßmanagement

◆ die Verwaltung von Prozessen umfaßt

- Erzeugung von Prozessen (auch dynamisch durch Prozesse selbst)
 - ◆ Bereitstellen von physikalischem Speicherplatz, Initialisieren von Datenstrukturen zur Darstellung des Prozesses, Erzeugung eines Prozeßimages
- Bearbeitung von Prozessen
 - ◆ Abarbeiten der Befehle im Programmcode gemäß der vorgegebenen Befehlsfolge, Eingabeparameter, vorliegender Daten und Signale / Nachrichten anderer Prozesse
- Terminierung von Prozessen
- Wechsel bei der Bearbeitung von Prozessen
 - ◆ Dispatcher regelt und bewirkt Wechsel
(Der Dispatcher ist auch nur ein Programm!)
 - ◆ Datenbedarf für spätere Weiterbearbeitung ???

Anforderungen

- ◆ Nebenläufige Ausführung von mehreren Prozessen
 - Ziel: möglichst viele Prozesse, möglichst schnell bearbeiten,
(bei 1 CPU meist mittels maximaler Prozessorauslastung angestrebt)Nebenbedingung:
 - bei Echtzeitanforderungen: rechtzeitige Terminierung
 - bei interaktiven Programmen: akzeptable Antwortzeiten
- ◆ Zuordnung von Ressourcen zu Prozessen
Nebenbedingungen:
 - keine Deadlocks/Verklemmungen, d.h. jeder Prozeß erhält irgendwann angeforderte Ressourcen und arbeitet weiter
 - Fairneß, d.h. Ressourcen werden nach Bedarf und angemessen verteilt
(ggfs. unter Berücksichtigung von Prioritäten, Deadlines, ...)
- ◆ Unterstützung für eine dynamische Prozeßerzeugung durch Anwendungsprozesse und die Kommunikation zwischen Prozessen
 - Übermittlung von Signalen (Signals) und Daten (Pipes, Messages, Shared Memory) zwischen Prozessen

Prozeßerzeugung - Wie entstehen Prozesse ?

Typische Ereignisse zur Erzeugung

- ◆ Batchbetrieb: es wird ein neuer Batch Job aus einem passendem Eingabestrom eingelesen
- ◆ Interaktives Logon: ein Anwender startet eine Sitzung
- ◆ das OS erzeugt eine Prozeß, um Dienst zu erbringen, z.B. Drucken
- ◆ ein (Anwender-) Prozeß erzeugt einen Prozeß, z.B. ein Doppelklick auf ein Dateisymbol veranlaßt Windowmanagement Editorprozeß zu erzeugen, der die passende Datei lädt

Das Betriebssystem erzeugt einen Prozess, indem es

- ◆ eine eindeutige Prozeß Id (Identifizier, Bezeichner) zuweist,
- ◆ Speicherplatz zur Darstellung allokiert,
- ◆ Prozeß Kontroll Block(PCB), Eintrag in Prozesstabelle initialisiert
- ◆ den Prozeß in die OS Datenstruktur zur Prozeßverwaltung integriert
 - z.B.: Einfügen in Scheduling Queue
- ◆ weitere Datenstrukturen erzeugt/ergänzt, z.B. Accounting, Performance Monitoring

Prozeßterminierung

Eine Auswahl an Ursachen:

- ◆ Normalfall: im Programmcode wurde die Befehlsfolge gemäß vorliegender Parameter, Daten und Nachrichten komplett abgearbeitet, der Prozeß terminiert
 - z.B. meist erreicht die Hauptfunktion einen return() Funktionsaufruf, oder Batch Job erreicht speziellen Befehl: *Halt*,
- ◆ Anwender logged sich aus, der Elterprozeß terminiert
- ◆ Anwendung terminiert (Quit, Exit), Prozeß erhält Terminierungssignal
- ◆ Fehler- und Ausfallbehandlung, OS terminiert den Prozess zwangsweise
 - kein Speicherplatz, Speichergrenzen verletzt, fehlerhafte Instruktion wg Operandendaten, ...

Das Betriebssystem terminiert den Prozess, indem es

- ◆ den Status des Prozesses auf terminiert setzt
- ◆ belegte Ressourcen freigibt, Dateien schließt, Accounting durchführt
- ◆ Ergebnisse / Returnwerte an den Elterprozeß übermittelt
- ◆ das Prozeßimage zerstört (Speicher wird dem freien Speicher zugeordnet, Prozess wird aus Verwaltungstabellen ausgetragen)

Welche Situationen/Zustände muß OS unterscheiden?

- ◆ Running

= aktuell in Bearbeitung

- ◆ Ready

= für Bearbeitung bereit

- ◆ Blocked

= blockiert, Ressourcen nicht verfügbar oder Bedingung nicht erfüllt

- ◆ New

= neuer Prozeß, z.B. noch nicht im Hauptspeicher verfügbar

- ◆ Exit

= terminierter Prozeß, dessen Prozeßimage noch existiert

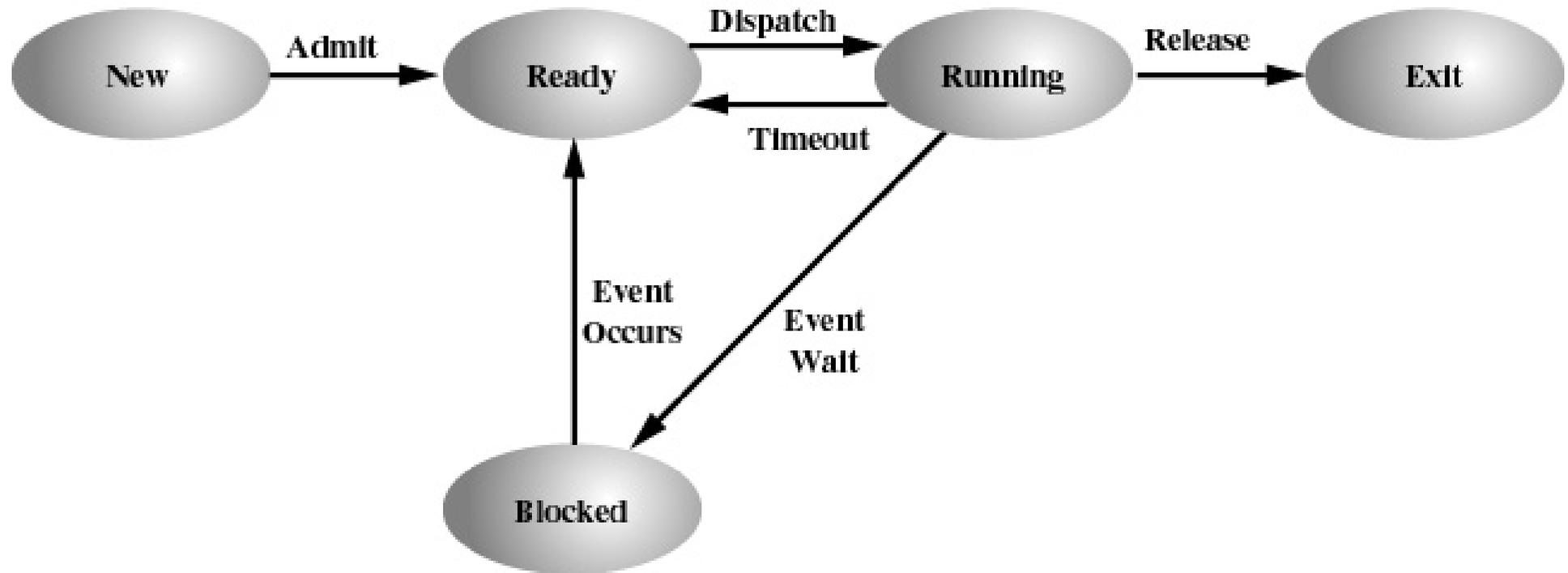


Abb. 3.5 aus Stallings: Prozeßmodell
mögliche Zustände und Zustandsübergänge eines Prozesses
Eine Unterscheidung von genau 5 Zuständen ist nicht zwingend!

Zeitverlauf

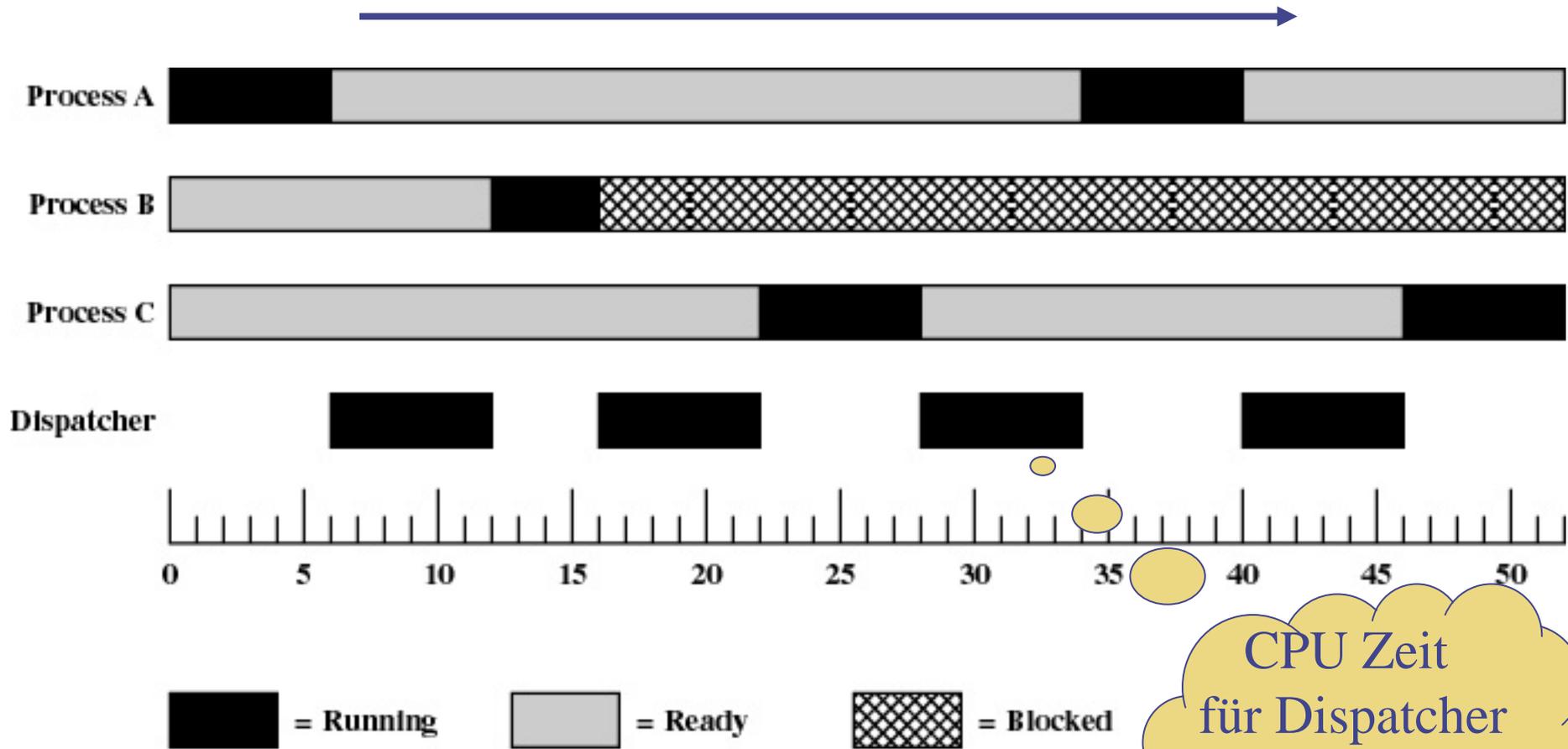
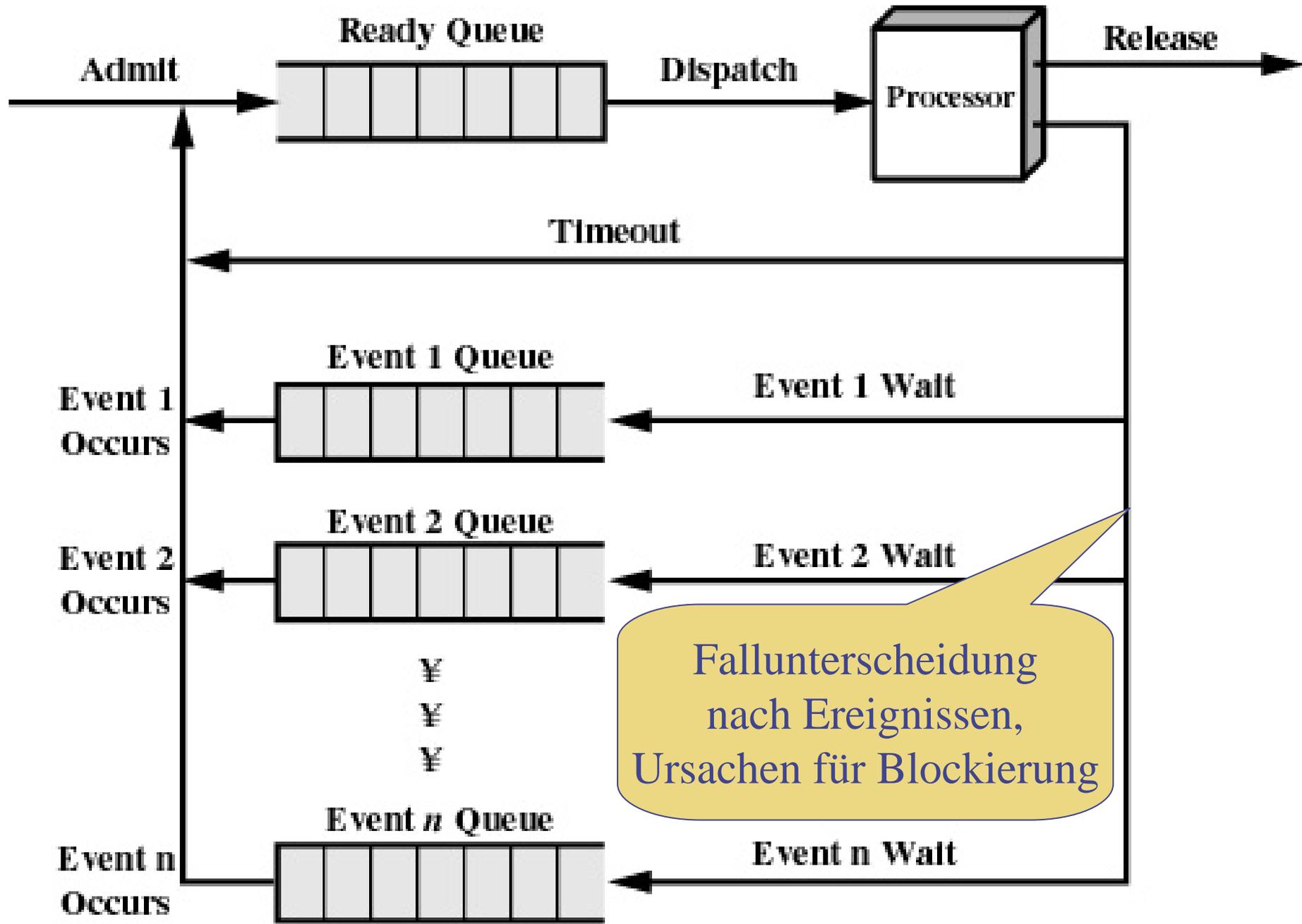


Figure 3.6 Process States for Trace of Figure 3.3



Warteschlangenmodell mit mehreren Queues für blockierte Prozesse

Feinere Unterscheidung für reales OS,
z.B. UNIX : Swapping und User/Kernel Modus

Swapping:

- ◆ Motivation: die CPU ist wesentlich schneller als E/A Geräte, so daß viele/alle Prozesse auf E/A warten können
- ◆ Idee: Auslagern wartender Prozesse auf Plattenspeicher schafft freien Hauptspeicher für einen größeren/weiteren Prozeß
- ◆ Swapping impliziert 2 neue Zustände im Zustandsdiagramm
 - Blocked/sleeping, swapped
 - Ready, swapped

User/Kernel Modus:

- ◆ Unterscheidung nach Art des Prozesses, Anwendungsprozeß oder OS Prozeß, bzw nach den aktuellen Rechten des Prozesses

UNIX Prozeßmodell

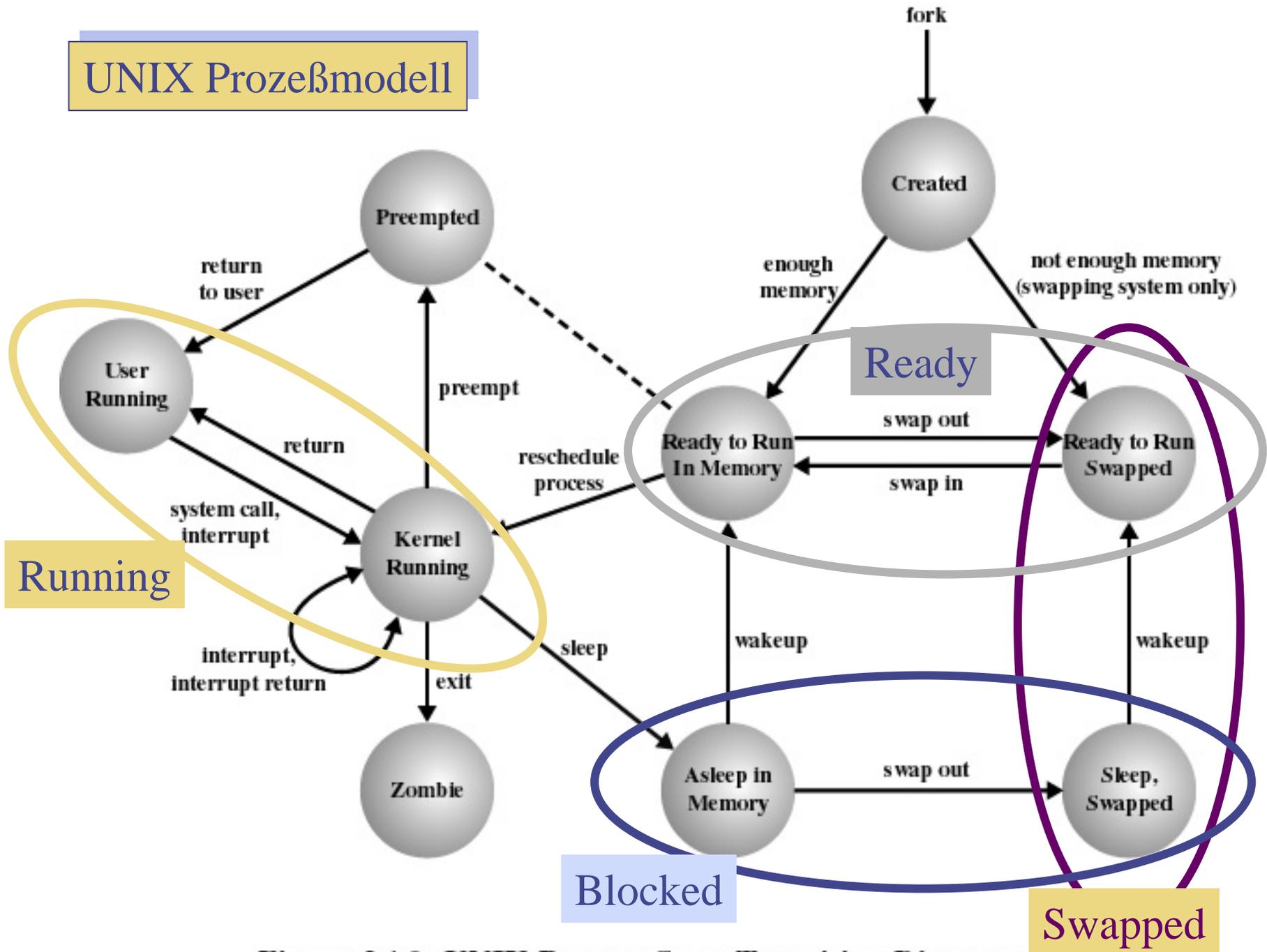
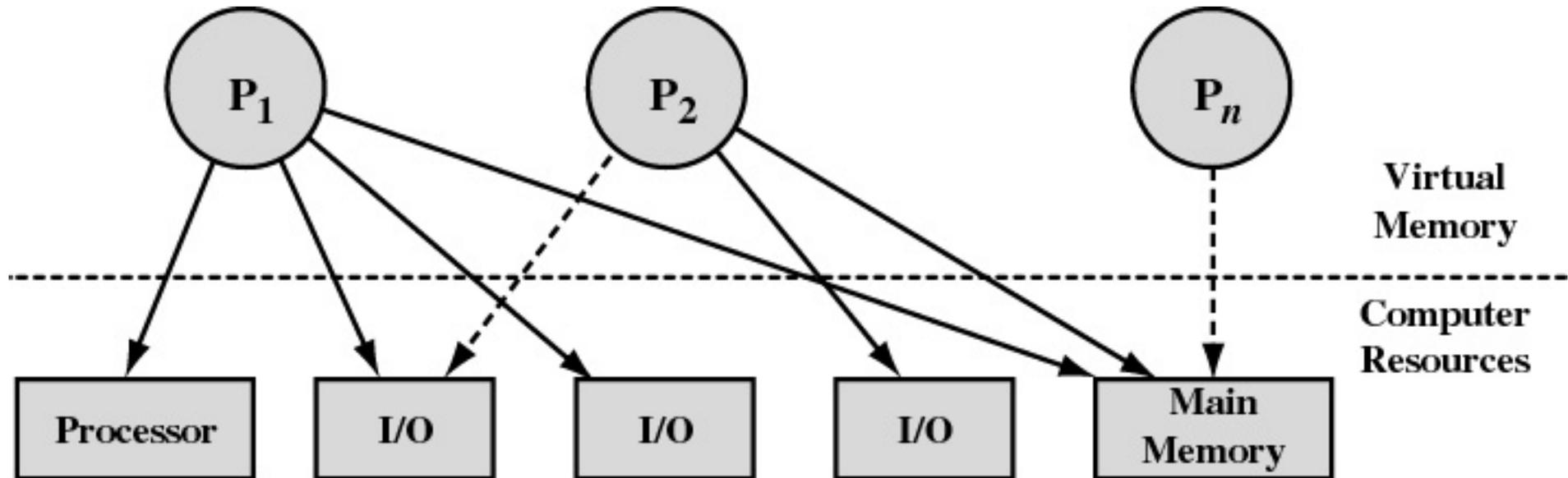


Figure 3.16 UNIX Process State Transition Diagram

Zustandsmodelle nicht einheitlich

- ◆ Je nach Betriebssystem unterschiedliche Anzahlen von Zuständen und unterschiedliche Bezeichnungen
- ◆ Grundlegende Ideen stets gleich
 - 1 Prozeß in Bearbeitung, bei Mehrprozessorsystemen entsprechend der Prozessorzahl sind auch mehrere Prozesse in Bearbeitung
 - Auswahl an bearbeitbaren, verfügbaren Prozessen
 - Prozesse, die zur Zeit nicht bearbeitet werden können
 - ◆ mangels Daten, Speicher, Ressourcen
 - ◆ für diese Klasse von Prozessen lassen sich unterschiedliche feine Unterklassen finden, hier entsteht im wesentlichen die Vielfalt der Zustandsdiagramme

OS regelt nicht nur CPU Zuteilung sondern allg. Ressourcenverwaltung



Prozesse und Ressourcen: Snapshot der Ressourcenzuordnung zu einem Zeitpunkt

Welche Information braucht OS zur Prozeßkontrolle und zur Ressourcenverwaltung ?

OS Kontrollstrukturen

◆ Informationen über den aktuellen Status aller Prozesse und Ressourcen (CPU, Speicher, E/A Geräte, Dateien, ...)

◆ für jede verwaltete Entität werden Tabellen angelegt, d.h.

- | | |
|------------------------------|-------------------|
| ■ Memory, | Speicherplatz |
| ■ I/O, Input/Output devices, | Ein/Ausgabegeräte |
| ■ Files, | Dateien |
| ■ Processes | Prozesse |

oder Informationen werden im Prozessimage lokal abgelegt.

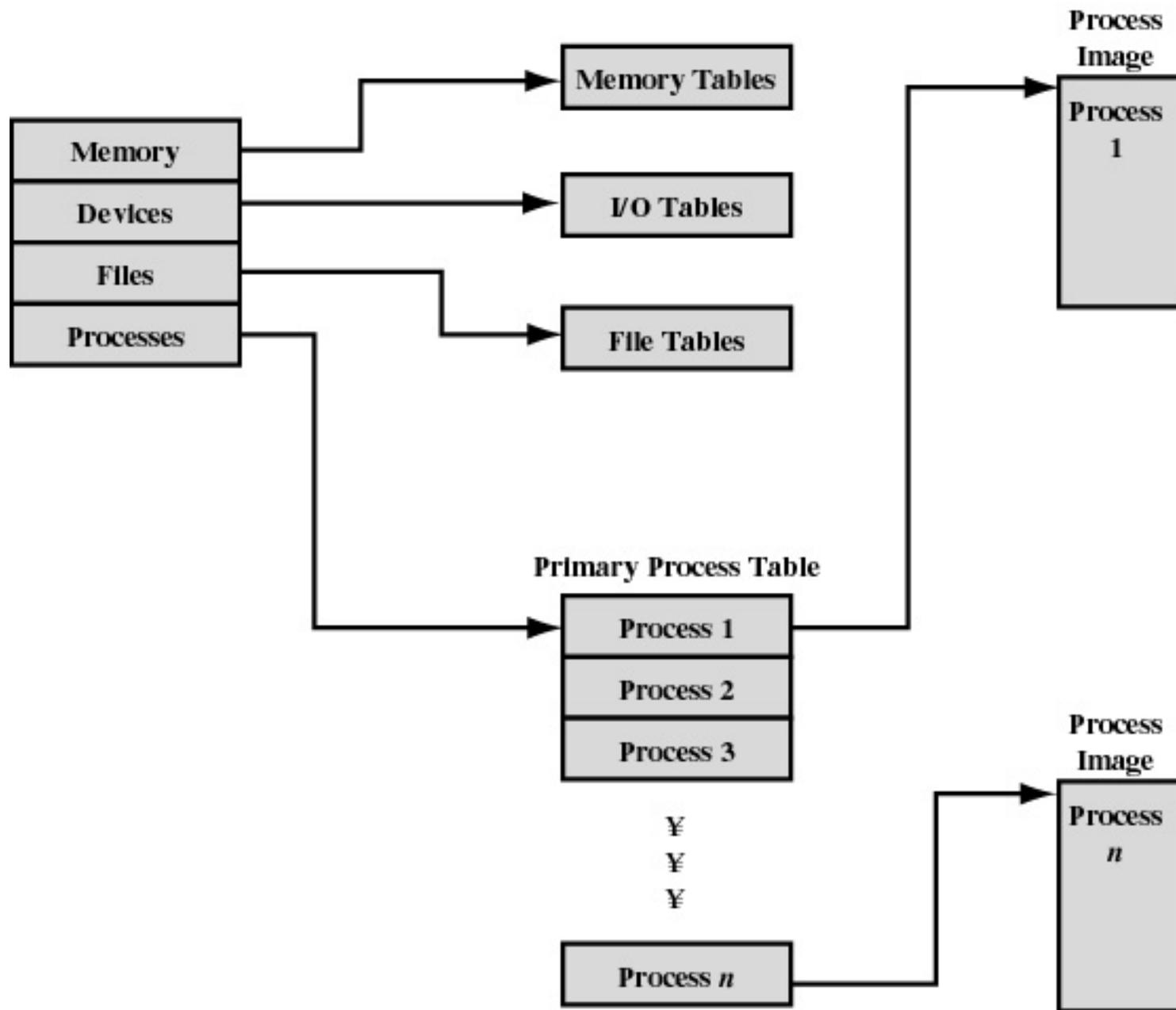


Figure 3.10 General Structure of Operating System Control Tables

Speichertabellen

- ◆ beschreiben, welche Speicherbereiche existieren und wem die Inhalte gehören, Zuordnung:
 - Hauptspeicherplatz zu Prozeß, Plattenplatz zu Prozeß
- ◆ beinhalten Schutzattribute für geteilte Bereiche
- ◆ Information für "virtuellen" Speicher,
 - Aufenthaltsort und Zustand von Daten,
Daten zur Abbildung virtueller -> physikalischer Adressraum

Tabellen für Geräte und Dateien

- ◆ I/O Tabellen beschreiben, welche Geräte es gibt, Verfügbarkeit, Zuordnung zu Prozessen, Status einer I/O Operation
- ◆ Dateitabellen beschreiben, welche Dateien es gibt, wo sie gespeichert sind, wem sie gehören, Status, Attribute
 - je nach OS in eigener Dateiverwaltung
 - Filemanagement wird später noch behandelt!

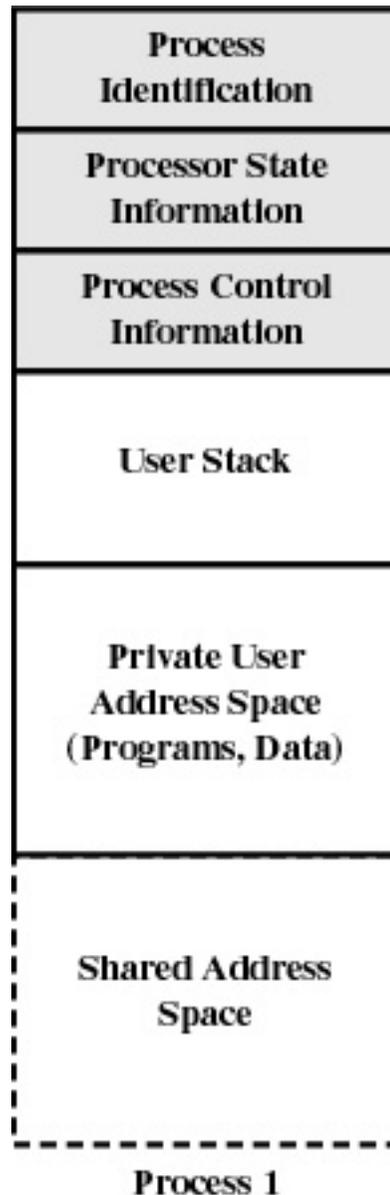
Prozeßtabellen

- ◆ beschreiben, welche Prozesse existieren, deren Zustand, Attribute
- ◆ Speicheradresse
- ◆ Attribute zur Verwaltung eines Prozesses
 - Prozeß ID (Identifizier)
 - Prozeß Zustand
 - Speicherort, Größe etc

Prozeßtabellen mit / und Prozeß Control Blocks bilden Datenstruktur zur Prozeßverwaltung

Process Control Block (PCB): Sammlung von Prozessattributen

- komplett als Eintrag in der Prozesstabelle implementiert
- nur in Teilen in der Prozesstabelle, restliche Teile im Prozessimage implementiert



Der PCB = Process Control Block beinhaltet Verwaltungsinformation für das OS:

- 1) Identifizierung, (wer ist das?)
- 2) CPU Status (wo weitermachen?)
- 3) Kontrollinformation (wann auswählen?)

Aufrufstack der Funktionen/Methoden im aktuellen Zustand der Bearbeitung

Speicherbereich für Daten/Objekte, Code

Prozeßimage bildet logischen Block, der in der Realität aus einzelnen Speicherseiten (Pages) zusammengesetzt sein kann.

Je nach OS kann noch zusätzlich ein Kernelstack im Prozeßimage auftreten.

Prozeßmanagement arbeitet mit Prozeßtabelle und den PCBs der einzelnen Prozesse.

Process Control Block

Identifizierung eines Prozesses

- Numerische Bezeichner:
 - i.d.R. für den Prozeß selbst, für den erzeugenden Prozeß(parent), für den Benutzer (effektive Benutzer ID \neq User ID wg Privilegien)

Prozessor Status Information

- User-Visible Register
 - ◆ Instruktionen aus dem Code des Prozesses (in Maschinensprache) können diese Register referenzieren. Anzahl: typisch 8-32, einige RISC Architekturen haben über 100 Register
- Control und Status Register, PSW=Prozessor Status Word

Eine Reihe von Registern werden zur Steuerung der CPU genutzt:

- ◆ Program counter (PC): Adresse der nächsten zu ladenden Instruktion
- ◆ Conditions Codes: Ergebnisse arithmetischer oder logischer Operationen (z.B. Vorzeichen, NULL, Übertrag, Gleichheit, Over/Underflow)
- ◆ Status Information: z.B. Interrupt enabling/disabling flags, Ausführungsmodus (Kernel, User Mode, ...)

PSW ist Grundlage für Prozeßwechsel !!!

Process Control Block

◆ Prozessor Status Information

- User-visible Register, Control und Status Register, PSW
- Stack Pointer
 - ◆ Je Prozeß ein (bei Threads mehrere) last-in-first-out (LIFO) System Stacks. Ein Stack (=Stapel) speichert aufgerufene Prozeduren/Methoden mit ihren Parametern, lokalen Variablen, Registerinhalten und Rücksprungadressen.
 - ◆ Stack Pointer zeigt auf oberstes Element (= zuletzt hinzugefügt, als nächstes zu entfernen)

◆ Prozeß Kontroll Information

- Scheduling und Status Information
 - OS braucht diese Information zum Scheduling, z.B.
 - ◆ Prozeß Zustand: zur Ausführung bereit ? (running, ready, blocked, ...)
 - ◆ Priorität: Ein/mehrere Felder für Rangfolgewert
 - ◆ je nach Scheduling Strategie weitere Informationen, z.B. Wartezeit, Deadlines, bereits erhaltene Rechenzeit
 - ◆ Ereignis: Identifier eines Ereignisses auf das ein Prozeß wartet

Process Control Block

weitere Einzelheiten zu Prozeß Kontroll Information

- Datenstrukturen zur Verknüpfung von Prozeßimages
 - ◆ Ein Prozeß kann mit anderen Prozessen in einer Warteschlange (Queue) , einem Ring, o.ä. verbunden sein, z.B. alle wartenden Prozesse einer Prioritätsstufe. Parent/Child Relationen mittels Zeigern implementiert.
- Interprozeß Kommunikation
 - ◆ Flags, Signale und Messages können zur Prozeßkommunikation zwischen unabhängigen Prozessen genutzt werden.
- Prozeß Privilegien, Rechte
 - ◆ Privilegien betreffen die Speicherbereiche, auf die ein Prozeß zugreifen kann und die Art der Instruktionen, die er ausführen darf. Ferner zur Nutzung von System Utilities und Diensten.
- Memory Management
 - ◆ Adressen von Segment- und/oder Seiten- Tabellen für Zuordnung virtueller - physikalischer Speicher
- Ressourcenbesitz und -nutzung
 - ◆ aktuell genutzte Ressourcen, z.B. geöffnete Dateien. Daten über die bisherige CPU Nutzung (oder auch anderer Ressourcen), "historische Daten", werden evtl. vom Scheduler genutzt.

Ausführungsmodi

◆ User Modus

- Nicht privilegiert
- typisch für Anwendungsprogramme

◆ System/Kontroll/Kernel Modus

- für den OS Kernel, zum Schutz OS Code + Daten
- Kernelcode meist ständig im Hauptspeicher, absolute Adressen
- Privilegiert, z.B.
 - ◆ Prozeßverwaltung: Erzeugung, Terminierung, Scheduling/Dispatching, Wechsel, PCB Manipulation
 - ◆ Speicherverwaltung: Allokation, Swapping, Seiten/Segmentverwaltung
 - ◆ I/O Verwaltung: Pufferverwaltung, Gerätezuordnung
 - ◆ Interrupt Behandlung, Accounting, Monitoring
- Systemfunktionsaufrufe für Anwendungsprozesse erfolgen im Kernel Modus, dadurch erhalten auch Anwendungen temporär diesen Status

Wann erfolgt ein Prozesswechsel ?

◆ Clock Interrupt

- Prozeß wurde maximal erlaubte Zeit von CPU bearbeitet (Zeitscheibe erschöpft)

◆ E/A Unterbrechungssignal (I/O Interrupt)

◆ Speicherfehler (Memory Fault, Page Miss)

- virtuelle Adresse hat zur Zeit keine physikalischen Gegenpart im Hauptspeicher (Datentransfer Festplatte -> Hauptspeicher)

◆ Trap

- ein Fehler ist aufgetreten,
- kann zu Prozeßterminierung führen (EXIT)
- kann zu Recoverymaßnahmen, Benachrichtigung des Anwenders führen

◆ Supervisor Call, System Call

- z.B. Öffnen einer Datei im Kernel Mode

Wie erfolgt ein Prozesswechsel ?

1. CPU Kontext/Register, PSW speichern
2. PCB aktualisieren
3. PCB in Warteschlange verschieben, je nach Status:
ready, blocked
4. anderen Prozeß auswählen
5. PCB des ausgewählten Prozesses aktualisieren
6. Memory Management Datenstrukturen aktualisieren
7. Kontext rekonstruieren (laden) für den ausgewählten Prozess

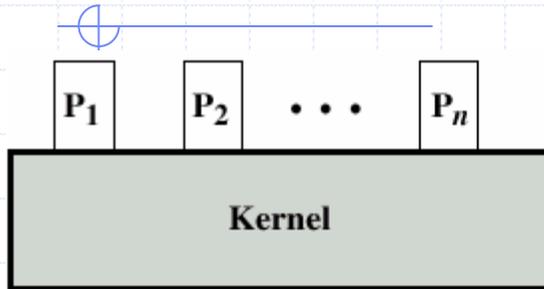


Aufwand!!!

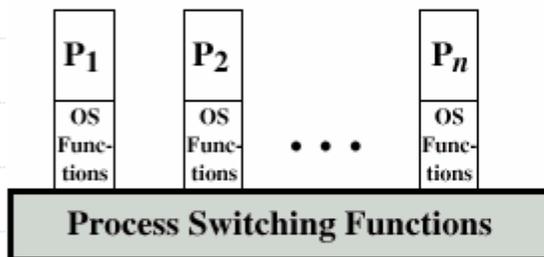
Einfachere Variante: Moduswechsel User-Kernel Modus

erfordert nur Schritte 1 & 7, um die Bearbeitung des Anwendungsprozesses zu unterbrechen, eine OS Routine (z.B. Interrupthandler und Dispatcher) durchzuführen und anschliessend mit demselben Anwendungsprozeß weiterzumachen.

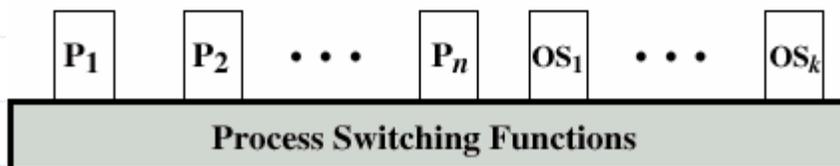
Wer kontrolliert eigentlich die Ausführung des OS ?



(a) Separate kernel



(b) OS functions execute within user processes



(c) OS functions execute as separate processes

Non-process Kernel (traditionell)

- Kernel wird außerhalb eines Prozesses ausgeführt
- OS Code separat in privilegiertem Modus

Ausführung im Anwendungsprozeß

- OS Software im Kontext einer Anwendung, Kernel Stack
- Anwendungsprozeß führt System Code im privilegierten Modus aus
- eher auf PCs, Workstations, zB Unix
- Kernel Code/Daten werden geteilt

Prozeßbasiertes OS: Kollektion von System Prozessen

- ◆ z.B. für Multiprozessorsysteme

(Stallings, Fig.3.14)

Wodurch können Prozesse kommunizieren ? z.B. in UNIX

◆ Pipe (Röhre)

- spezielle Datei, in die der sendende Prozess schreibt, aus der empfangende Prozess liest. Dateigröße/Buffergröße ist fest.

◆ Messages (Nachrichten)

messagesend

- Nachricht ist eine Menge von Bytes (Text), der mittels msgsd versendet und mittels msgrcv empfangen werden kann. Jeder Prozess hat einen Nachrichtenpuffer (Briefkasten), in dem Nachrichten bis zur Annahme abgelegt werden.

◆ Shared Memory (geteilter Speicher)

- Prozesse lesen und schreiben Daten von/in einem gemeinsamen Speicherbereich, erfordert Zugriffe im wechselseitigen Ausschluß (Mutual exclusion, Synchronisation), z.B. mittels Semaphorkonzept, Rechte der Prozesse können für Lese-Schreiboperationen eingeschränkt sein.

Wodurch können Prozesse kommunizieren ? z.B. in UNIX

◆ Signal

- Softwaremechanismus zur Benachrichtigung von Prozessen, ähnlich zu Hardwareinterrupts, aber ohne Prioritäten
- Prozesse, aber auch der Kernel können Signale versenden
- Signale werden bei Eintreffen in einem Feld des zugehörigen Prozesses als 1 Bit gesetzt; sie transportieren daher keine Parameter, bilden keine Warteschlange.
- Signale werden beim „Aufwecken“ des Prozesses ausgewertet oder bei der Rückkehr von einem Systemaufruf.
- Mögliche Reaktionen auf ein Signal
 - ◆ Terminierung (default)
 - ◆ Ausführen einer speziellen Signalbehandlungsroutine
 - ◆ Ignorieren
- Beispiele für Signale
 - ◆ SIGHUP (hang up), SIGQUIT (halten, core file erzeugen)
 - ◆ SIGILL (illegal instruction), SIGTRAP (trace trap, tracing)
 - ◆ SIGFPE (floating point exception), SIGKILL (terminieren)
 - ◆ SIGBUS (Bus error), SIGSEGV (Segmentation violation)
 - ◆ ...

Zusammenfassung

- ◆ Programm -- Prozeß
- ◆ Information zur Darstellung eines Prozesses
 - Tabellen
 - Prozeßimage mit Process Control Block (PCB)
- ◆ Prozeßerzeugung
- ◆ Prozeßbearbeitung
 - Prozeßzustände und Übergangsdigramme
 - Vorgänge beim Dispatching, Prozeßwechsel (Switch)
- ◆ Prozeßterminierung
- ◆ Bearbeitungsmodi: Usermodus vs Kernelmodus
- ◆ Interprozeßkommunikation

- ◆ Offene Punkte, z.B.
 - Parallele Programme, Threads