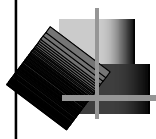
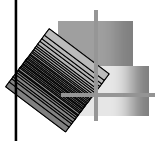


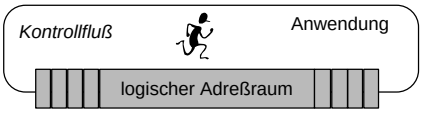
Kontrollflüsse

Einführung

1



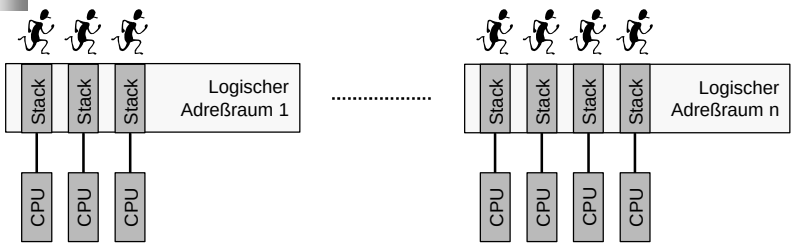
Motivation



- Kontrollfluß (Thread) = CPU führt Instruktionen aus
- Was charakterisiert einen Kontrollfluß?
  - Programmzähler
  - Registerinhalte
  - Aufrufkeller (Stack)
  - Adreßraum
- Adreßraum + Kontrollfluß = Prozeß

2

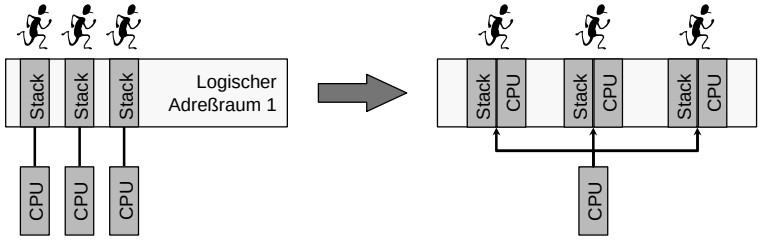
Beliebig viele Prozessoren



- Von der tatsächlichen Prozessoranzahl abstrahieren
  - Nebenläufigkeit bei der Anwendungsentwicklung ausnutzen
  - Maximalen Parallelitätsgrad ausnutzen
- Mehr Freiheitsgrade
  - Maximale Prozessoranzahl bei Start anfordern
  - Zusätzliche Hardware leicht nutzbar
  - Priorisierung von Prozessen u.ä. möglich

3

Virtuelle Prozessoren

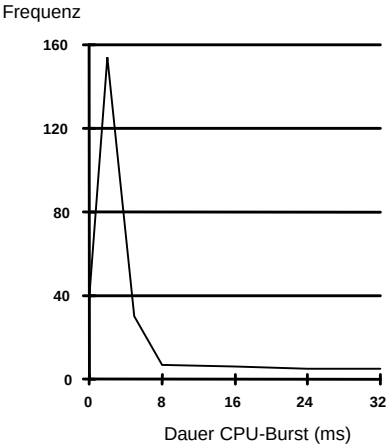


- Bessere (transparente) Nutzung vorhandener Hardware
  - Zeitmultiplexen der realen CPU
  - Verwendung mehrerer Prozessoren (Multiprozessorsystem)
- Vorteile:
  - Gleichzeitige Unterstützung mehrerer Anwendungen
  - Bessere Hardware-Auslastung

4

CPU- und I/O-Bursts

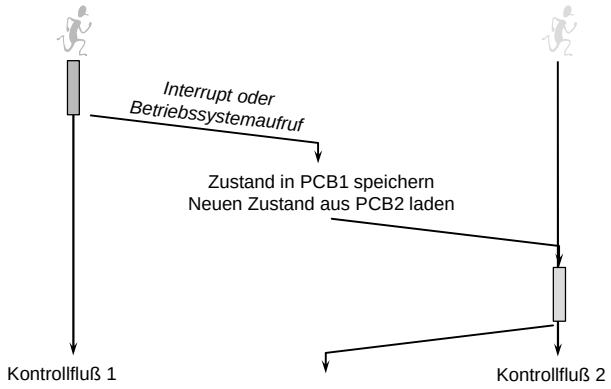
- Typisches Programmverhalten: alternierende CPU- und I/O-Bursts:
  - Lesen von Datei (I/O)
  - Daten bearbeiten (CPU)
  - Lesen von Datei (I/O)
  - Daten bearbeiten (CPU)
  - ...
- I/O-Burst:
  - Hardware wird aktiv
  - Anwendung blockiert
- CPU kann anderes Programm ausführen



5

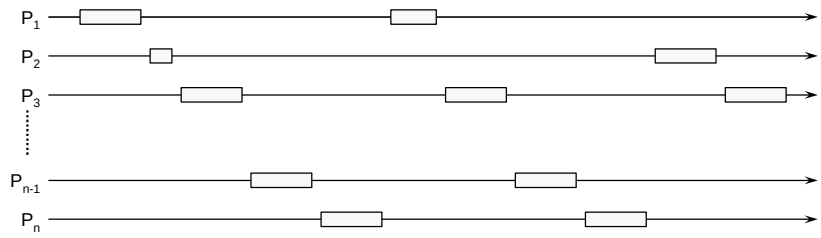
Kontextwechsel

- Zustand eines virtuellen Prozessors speichert ein Prozeßkontrollblock (PCB)



6

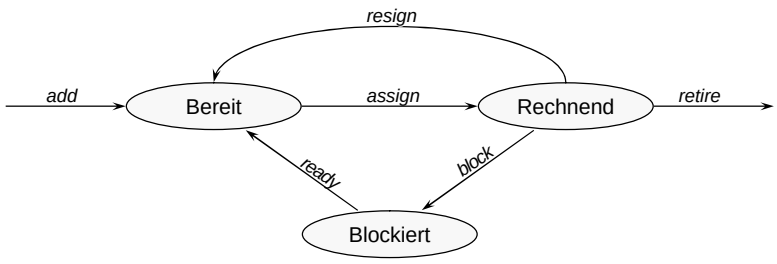
Multitasking



- Zeitmultiplexen auf einem Prozessor
  - Kontextwechsel wird durch Timer-Interrupt angestoßen
  - Zeitscheibe: Maximale Zeitintervall zwischen zwei Timer-Interrupts
  - Typische Werte 10 - 20 Millisekunden
- wird von allen gängigen Betriebssystemen unterstützt

7

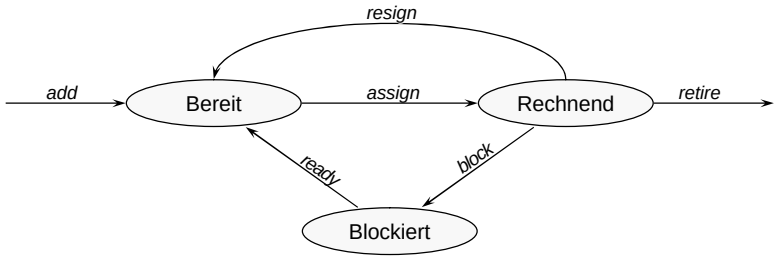
Prozeß-Zustandsmodell



- Verwaltung der PCB's in 3 Listen:
  - **Rechnend**:
    PCB's der Prozesse, die im Besitz eines realen Prozessors sind
    (1 Element bei Monoprozessor, n bei Multiprozessor)
  - **Bereit**:
    Rechenwillige Prozesse, denen kein realer Prozessor zugeordnet ist
  - **Blockiert**:
    Prozesse, die einen realen Prozessor zur Zeit nicht sinnvoll nutzen können

8

Scheduler und Dispatcher

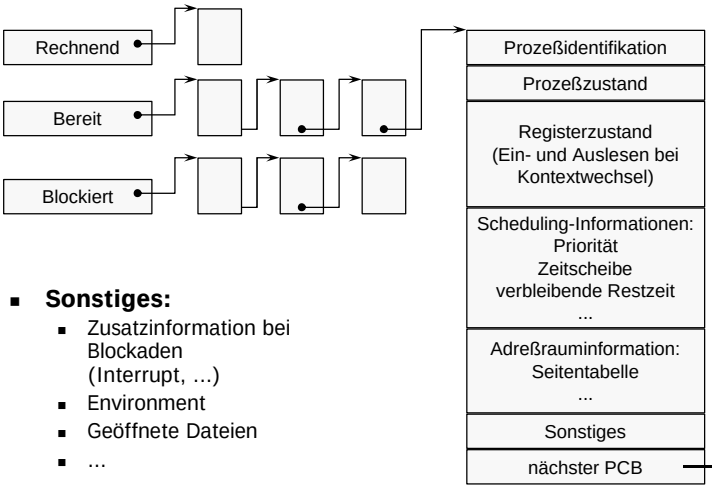


Scheduler
- Wählt aus der Bereit-Liste bei Bedarf einen Kandidaten für die Rechnend-Liste
- Viele Schedulervarianten

Dispatcher
- Führt Übergänge im Prozeß-Zustandsmodell aus
- Meist Assembler-Routinen

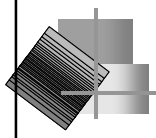
9

Der Prozeßkontrollblock (PCB)



- **Sonstiges:**
  - Zusatzinformation bei Blockaden (Interrupt, ...)
  - Environment
  - Geöffnete Dateien
  - ...

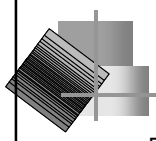
10



Zustandsübergänge

- Übergang “Rechnend” - “Bereit”:
  - Prozeß hat sein Zeitlimit überschritten (Multi-Tasking)
  - Höher-priorer Prozeß wurde bereit (z.B. durch Interrupt)
- Übergang “Rechnend” - “Blockiert”:
  - Seitenfehler
    Blockade bis Seite(n) eingelagert
  - Prozeß liest von Datei / Netzwerk / Tastatur
    Blockade bis die gewünschten Daten vorliegen
  - Prozeß wartet auf den Eintritt einer Synchronisations-bedingung, die durch einen anderen Prozeß hergestellt werden kann
    Blockade, bis Synchronisationsbedingung eintritt

11



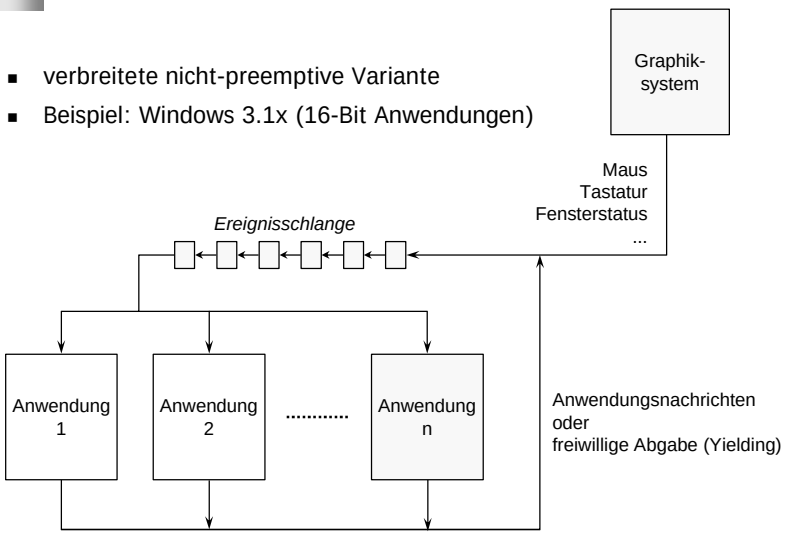
Preemptives und nicht-preemptives Scheduling

- Preemptives Scheduling
  - Prozessor wird einer Anwendung “unfreiwillig” entzogen
  - Preemption typischerweise bei abgelaufener Zeitscheibe (Timer)
- Vorteile:
  - einzige Möglichkeit, um eine Monopolisierung zu verhindern
  - Gleichmäßige Aufteilung der Rechenleistung möglich
  - im Mittel schnelle Reaktion möglich
- Nachteile:
  - Zusätzliche Kontextwechsel
  - Programmlaufzeiten unbestimmt
- Nicht-preemptives Scheduling
  - Anwendung gibt den Prozessor freiwillig ab
  - Implizit z.B. bei blockierenden Aufrufen
  - Explizit
- Vorteile:
  - maximale Programmlaufzeiten abschätzbar (Echtzeitsysteme)
- Nachteile:
  - Monopolisierung möglich
  - Schnelle Reaktion nicht immer garantiert

12

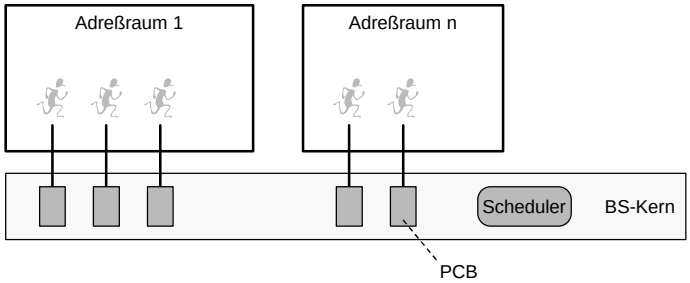
Kooperatives Scheduling

- verbreitete nicht-preemptive Variante
- Beispiel: Windows 3.1x (16-Bit Anwendungen)



13

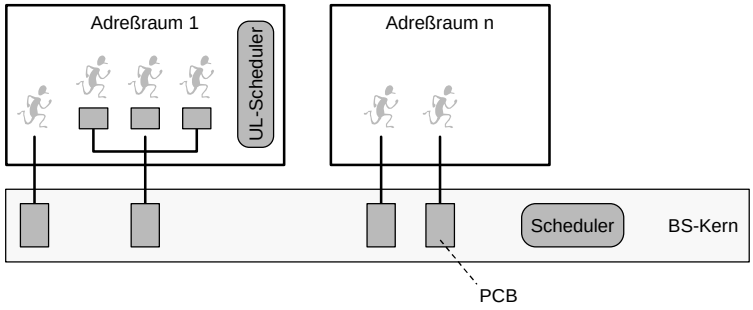
Kernel-Level-Threads



- **Vorteile:**
  - Betriebssystem kann bei Blockade rechenbereiten Thread wechseln
  - Scheduling-Problematik auf Anwendungsebene nicht sichtbar
- **Nachteile:**
  - u.U. bei Wechsel Thread eines anderen Adreßraums
  - Jede E/A-Operation u.a. muß eine Kernfunktion aufrufen
  - Schwergewichtig: Adreßraum-wechsel (kalter Cache)

14

User-Level-Threads



- **Vorteile:**
  - Scheduling durch Anwendung
  - effizientes Scheduling, kein Betriebssystemaufruf
  - Leichtgeweichtig: Kein Adreßraum-wechsel (warmer Cache)
- **Nachteile:**
  - (Scheduling durch Anwendung)
  - Blockierende Aufrufe für Kernel-Level-Thread vermeiden

15