

# Betriebssysteme

## 2. Prozesse und Threads

Prof. Dr. Harald, Kornmayer  
Institut für Informatik, DHBW Mannheim, Germany

# Prozesse und Threads

- Agenda
  - Prozesse
  - Threads
  - Interprozess-Kommunikation
  - Klassische Probleme bei Interprozess-Kommunikation
  - Scheduling
  - Deadlocks /Verklemmungen

## Aufgabe

- Definieren Sie den Begriff Prozess!
    - Was ist der Unterschied zwischen Programm und Prozess?
      - Suchen Sie ein Analogon!
  - Was kann das Betriebssystem durch Prozesse realisieren?
  - Welche Daten brauchen Sie um einem Prozess zu beschreiben?
- 3 er Team  
 – 5 Minuten

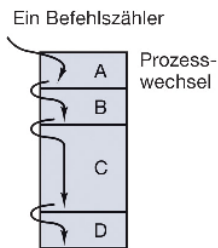


## Prozesse

- Das **zentrale Konzept** in jedem Betriebssystem
  - **Prozess** = ein Programm in Ausführung
  - Prozesse modellieren Nebenläufigkeit
    - unterstützen (Quasi-)Parallelität
  - Multiprogrammierung
    - Schnelles Hin und Herschalten zwischen Prozessen
    - Umladen der Register notwendig!
- Prozessmodell
  - „*Das Betriebssystem ist als Menge von (sequenziellen) Prozessen organisiert*“
  - jeder Prozess umfasst eine virtuelle CPU mit eigenem Adressraum
    - Befehlszähler, Registerinhalte, Belegung von Variablen
    - Und einiges mehr!

## Prozesse

- Prozesse haben einen Adressraum
  - Inklusive logischer Befehlszähler



- Konsequenz:
  - Laufzeit eines Programms ist nicht mehr reproduzierbar!
- Programme dürfen keine Annahmen über den Zeitablauf enthalten

## Prozesse

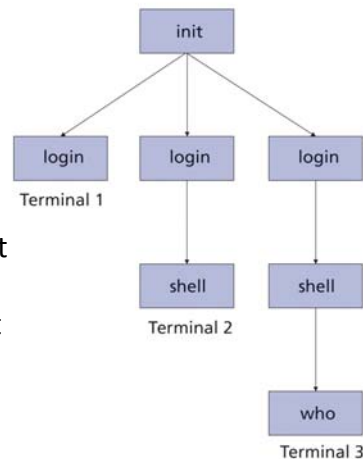
- Unterschied Programm und Prozess
  - Vergleich:
    - Programm ist das Kuchenrezept mit allen Anweisungen
    - Prozess ist die Aktivität des Backen
    - Beachte: Ein Programm 2x starten, ergibt 2 Prozesse!
- Wie wird ein Prozess erzeugt?
  - Durch das Betriebssystem
    - Initialisierung
      - Als Hintergrundprozess
    - Durch einen anderen Prozess
      - Systemaufruf

# Prozesse

## • Prozesserzeugung

### – Bsp: Linux starten

- init liest Datei /etc/init/tty?.conf
- Startet dann ein login für jedes Terminal
- Nach der Anmeldung startet eine Shell, von der aus die nächsten Prozesse gestartet werden können.

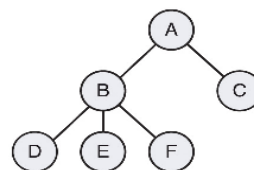


# Prozesse

## • Beziehungen zwischen Prozessen

### – Eltern-Kind Beziehung

- Entstehung von Prozess-Familien bzw. Prozess-Hierarchien



### – UNIX

- Signale werden an alle Mitglieder der Familie verschickt
  - Prozess entscheiden wie auf Signal reagiert werden!

### – Windows

- Kein Konzept der Prozesshierarchie
- Alle Prozesse sind gleichwertig
  - Beziehungen werden durch spezielle Tokens (Handle) gesteuert!
    - » Weitergabe von Handles möglich (Enterbung)

## Prozesse und Threads



- Prozessbeendigung
  - Normales Beenden (freiwillig)
    - Wenn Aufgabe erledigt ist! (exit bzw. exitProcess)
  - Beenden aufgrund eines Fehlers (freiwillig)
    - Der Prozess stellt einen Fehler fest, der nicht vom Programm verursacht wurde (z.b. Datendatei nicht vorhanden)
  - Beenden aufgrund eines schwerwiegenden Fehlers (unfreiwillig)
    - Der Prozess selbst verursacht einen Fehler
    - Ausführen eines unzulässigen Befehl
    - Zugriff auf ungültige Speicheradresse
  - Beenden durch anderen Prozess (unfreiwillig)
    - (kill bzw. TerminateProcess)

27. September 2012

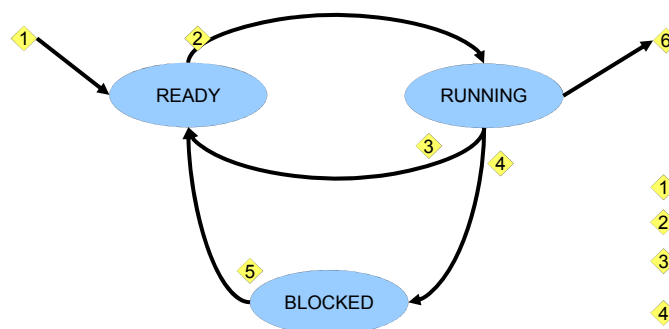
Prof. Dr. Kornmayer

133

## Prozesse



- Prozess-Zustände
  - Wichtig für die Verwaltung der Ressourcen durch BS



- 1 Start nach Erzeugung
- 2 Auswahl durch Scheduler
- 3 Scheduler wählt anderen Prozess
- 4 Prozess blockiert, weil er auf Eingabe wartet
- 5 Eingabe vorhanden
- 6 Beendigung des Prozesses

Verwaltung der Übergänge durch Scheduler

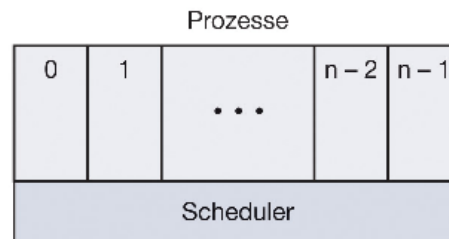
27. September 2012

Prof. Dr. Kornmayer

134

# Prozesse

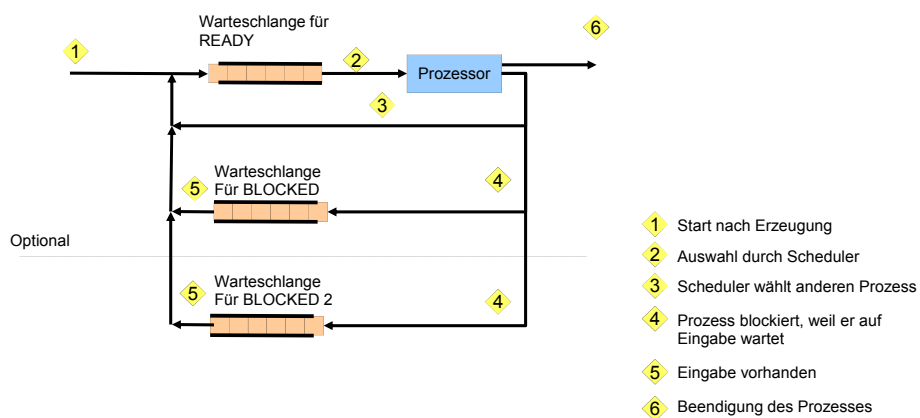
- Scheduler
  - Verwaltet die Prozesse
  - Unterste Schicht des Betriebssystems



- behandelt
  - Unterbrechungen (Interrupts)
  - Starten und Stoppen von Prozessen
  - Warteschlangen für Prozesse

# Prozesse

- Scheduler
  - Warteschlangen



# Prozesse



27. September 2012 Prof. Dr. Kornmayer 137

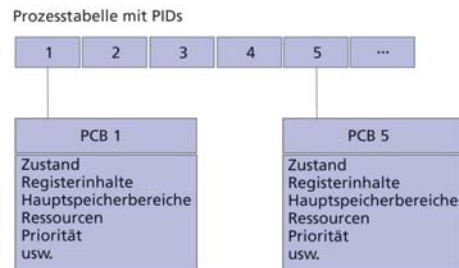
# Prozesse

- Datenmodell zur Verwaltung von Prozessen
  - **Prozesstabelle**
    - Ein Eintrag pro Prozess (Prozesskontrollblock (PCB))
    - PCB enthält alle Information über Prozess
      - Alle Informationen → Datenmodell
      - Zur **Prozessverwaltung**
        - » Register, Befehlszähler, PSW, Stack
        - » Prozessidentifikation
          - Kennung des Prozesses (P-ID) und des Elternprozesses
          - Benutzerkennung
        - » Zustandsinformation
          - Priorität, verbrauchte CPU-Zeit, Signale,...
      - Zur **Speicherverwaltung**
        - » Zeiger auf Textsegment, Datensegment, Stacksegment
      - Zur **Dateiverwaltung**
        - » Wurzelverzeichnis, Arbeitsverzeichnis, offene Dateien, Benutzer-ID, Gruppen-ID, ...

27. September 2012 Prof. Dr. Kornmayer 138

# Prozesse

- Prozesswechsel
  - mit Hilfe der Prozesstabelle
    - Bsp: Interrupt von E/A-Gerät („Daten stehen bereit“)



- Wechseln zwischen zwei PCB

# Prozesse

- Prozesswechsel
  - Ablauf
    - Sichern des Befehlszählers, Prozessorstatus, etc
      - Prozess auf „Blocked“ setzen
    - Ursache der Unterbrechung ermitteln
    - Ereignis (z.B. Ende der E/A) entsprechend behandeln
      - Blockierte Threads auf „READY“ setzen
    - Sprung zum Scheduler
    - Entscheidet welcher Prozess als nächstes läuft



## Prozesse

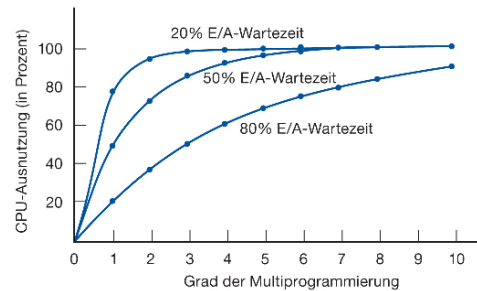
- Modell der Multiprogrammierung

- Vorteil: Verbesserung der CPU Auslastung

- N: Prozesse die gleichzeitig im Speicher gehalten werden

- P: E/A- Anteil eines Prozesses

- z.B. 20% der Zeit warten auf Daten/Festplatte



CPU-Ausnutzung =  $1 - P^N$

Mehr Speicher ermöglicht  
mehr Prozesse

→ Verbesserung der Auslastung

27. September 2012

Prof. Dr. Kornmayer

141

## Aufgabe

- Im bisherigen Prozessmodell hält jeder Prozess Information über seinen Adressraum und über Dateien
- Ein Textverarbeitungsprogramm soll in der Lage sein Eingaben über die Tastatur entgegenzunehmen, den formatierten Text auf dem Bildschirm auszugeben und gleichzeitig eine Sicherung auf die Festplatte zu schreiben.
- Was sind die Vor- und Nachteile des bisherigen Prozessmodells für dieses Szenario?

- 3er Team
- 5 Minuten



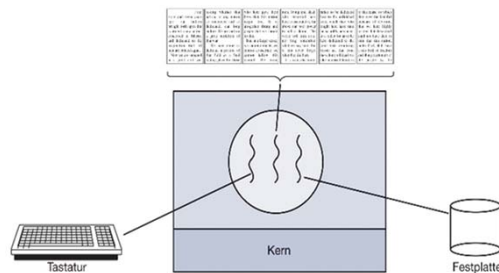
27. September 2012

Prof. Dr. Kornmayer

142

# Prozesse und Threads

- Prozessmodell
  - Jeder Prozess hat **eigenen Adressraum**
  - mit einen **Ausführungsfaden** (thread of control)
  - Fragen:
    - Wieso nicht quasi-parallel den Adressraum bearbeiten?
    - Wieso nicht mehrere Ausführungsfäden im selben Adressraum ablaufen lassen?



27. September 2012

Prof. Dr. Kornmayer

143

# Prozesse und Threads

- Threads
  - Erweitertes Prozessmodell
    - Mehrere Ausführungsfäden mit **gleichem Adressraum**
      - Daten gemeinsam benutzen
    - Ausführung im Benutzermodus möglich
    - Erstellung ist 10-100 mal schneller als Prozesse
  - Konzept der Ressourcen-Bündelung für Prozesse
    - Adressraum, geöffnete Dateien, Signale, Kindprozesse, ...
  - Konzept der Ausführung von Prozessen (Threads)
    - Befehlszähler, Register für lokale Variablen, Stack,
    - Mehrere Ausführungsfäden sind nun möglich (Multi-Threading)
      - Hardware-Unterstützung (schnelles Umschalten in nsec)

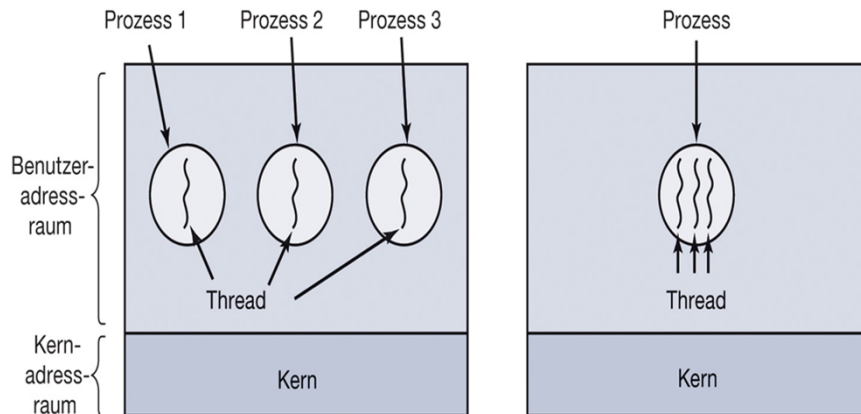
27. September 2012

Prof. Dr. Kornmayer

144

# Prozesse und Threads

## • Threads



27. September 2012

Prof. Dr. Kornmayer

145

# Prozesse und Threads

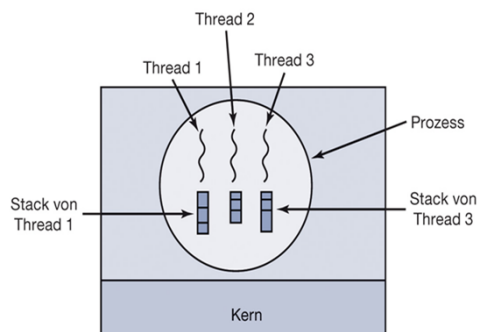
## • Threads

### – Thread-Zustände

- Gleiche Zustände wie Prozesse
  - RUNNING, BLOCKED, READY

### – Thread-Verwaltung

- Thread-Tabelle
- Befehlszähler, Register, Stack und Zustand
  - Jeder Thread hat seinen eigenen Stack



27. September 2012

Prof. Dr. Kornmayer

146

# Prozesse und Threads

## • Threads

### – Lebenszyklus

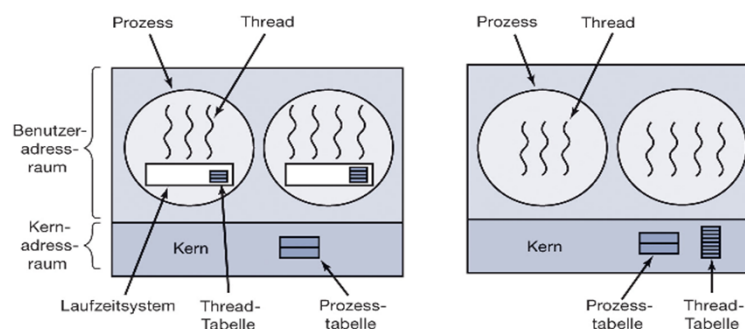
- Erzeugung von Threads
  - Prozess startet einen Thread, der neue Threads erzeugt (create)
    - » Argument: Name der Prozedur, die ausgeführt werden soll
    - » Keine Angaben des Adressraums notwendig
      - Neuer Thread läuft im Adressraum des erzeugenden Threads
- Beendigung
  - Nach der Beendigung der Aufgabe durch den Thread selbst (exit)
- Synchronisation
  - Warten auf die Beendigung eines bestimmten Threads (join)
- Freiwilliger Verzicht
  - Ressourcen an die CPU zurückgeben (yield)
    - » Scheduler beauftragen anderen Thread zu starten


# Prozesse und Threads

## • Threads

### – Wo läuft der Thread-Scheduler/die Thread-Verwaltung?

- Zwei Varianten
  - Im Benutzeradressraum/-modus
  - Im Kernadressraum/-modus






## Prozesse und Threads

---

- Threads
  - Wo läuft der Thread-Scheduler/die Thread-Verwaltung?
- Benutzermodus
  - Pro
    - Schneller Wechsel
    - Angepasster Scheduler möglich
  - Con
    - Umgang mit blockierenden Systemaufrufen
    - Ein Thread kann den gesamten Prozess blockieren
      - z.B. bei Seitenfehlern
    - Threads sollen Programme ermöglichen, die oft blockieren
- Kernmodus
  - Pro
    - Kein Laufzeitsystem
    - Effizienter Umgang mit blockierende Systemaufrufen
  - Con
    - Höher Kosten bei Thread- Wechsel durch Systemaufrufe
    - Umgang mit Signalen

- Threads werden meist im Kernmodus realisiert!  
 - Spannendes Thema in Betriebssystementwicklung

27. September 2012
Prof. Dr. Kornmayer
149



## Prozesse und Threads

---

- Threads
  - Threadwechsel
    - erfolgt immer dann, wenn BS die Kontrolle erhält
      - Bei Systemaufrufen
        - » Thread gibt Kontrolle freiwillig ab!
      - Bei Ausnahme
      - Bei Interrupt
        - » Periodischer Timer-Interrupt stellt sicher, daß kein Thread die CPU monopolisiert
  - Scheduler des Betriebssystems entscheidet, welcher Thread als nächstes rechnen soll
    - Falls nötig, wird dann auch der Prozess gewechselt
    - Scheduler kann diese Kosten berücksichtigen

27. September 2012
Prof. Dr. Kornmayer
150

# Prozesse und Threads



- Zusammenfassung
  - Prozessmodell
    - Prozess: Einheit der Ressourcenverwaltung, Schutzseinheit
      - Adressraum, geöffnete Dateien, Signale, Prioritäten, ...
    - Thread: Einheit der Prozessorzuteilung
      - Befehlszähler, CPU-Register, Stack, Thread-Zustand, ...
    - mehrere Threads pro Prozess möglich
    - Prozess/Thread-Zustände
      - RUNNING, READY, BLOCKED
      - Warteschlangen

27. September 2012

Prof. Dr. Kornmayer

151

# Prozesse und Threads



- Zusammenfassung
  - Realisierungsvarianten für Threads
    - Im Kernmodus (gängig) , im Benutzermodus
    - Threadwechsel
      - Bei Systemaufrufen, Ausnahmen oder Interrupts
      - Umladen des Prozessorkontext
        - Beim Prozesswechsel auch Wechsel des Speicherabbilds
  - Programmier-Schnittstellen
    - POSIX-Threads
    - JAVA Threads

27. September 2012

Prof. Dr. Kornmayer

152

# Betriebssysteme

## Ergänzungen zur Übung 2

Prof. Dr. Harald, Kornmayer  
Institut für Informatik, DHBW Mannheim, Germany

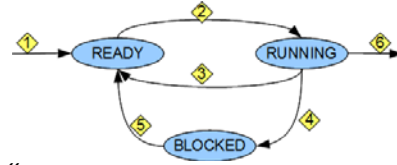
## Übung 2

- Systemaufrufe
  - fork()
  - exec()
- POSIX Thread library
- JAVA Threads

## Systemaufrufe

- **fork()**

- Erzeugung von Processen
- man 2 fork
- „creates a new process by duplicating the calling process“

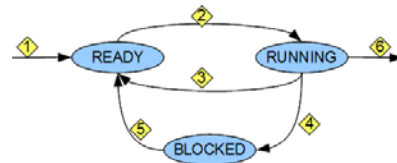


- benötigt **#include** <unistd.h>  
(**#include** <sys/types.h> für pid\_t)
- Der Rückgabewert unterscheidet zwischen Kind und Vater
  - Fehlerfall: -1
  - Vater: > 0
  - Kind: 0

## Systemaufrufe

- **exec()**

- Erzeugung von Prozessen
  - führt ein andere Datei aus!
  - mit neuen Code!



- **6 Varianten**

- Argumente als Liste: `execl, execl, execlp`
- Argumente als Vector: `execv, execve, execvp`
  - „e“ - Environment
  - „p“ - Path
- Bsp: `execlp( prog_name, prog_name, Arg1, Arg2, ... 0 ) ;`
  - `execlp(„sort“, „sort“, „-n“, „studentenliste“, 0 ) ;`
  - „sort“ wird aus historischen Gründen zweimal angegeben!
  - 0 terminiert die Eingabe Liste



## Systemaufrufe



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
int main(void) {
    printf("Schau mer mal"); /* prints Schau mer mal! */
    int pid1, pid2;
    pid1 = getpid();
    printf(" Process ID %d", pid1);

    pid2 = fork();
    if (pid2 < 0) {
        perror("\n Fork war nicht erfolgreich");
        exit(1);
    }
    else if (pid2 == 0) {
        // Kind prozess
        printf("\nKIND Prozess");
    }
    else if (pid2 > 0) {
        // Vater prozess
        printf("\nVATER Prozess ");
    }
    return EXIT_SUCCESS;
}
```

27. September 2012

Prof. Dr. Kornmayer

157

## POSIX Threads



- POSIX = Portable Operating System Interface
  - Standardisiertes (API) Application Programming Interface für UNIX-artige Systems
  - Umfasst
    - Systemaufrufe und
    - Kommandozeileninterpreter und Hilfsprogramme
    - Schnittstelle zwischen Anwendung und Betriebssystem
  - Threads sind ein Beispiel für ein API
    - IEEE POSIX 1003.1c
    - Implementierungen werden als POSIX threads oder Pthreads bezeichnet

27. September 2012

Prof. Dr. Kornmayer

158

## POSIX Threads



- C-Library
  - #include <pthread.h> und „gcc .... -pthread ...“
- > 60 Funktionsaufrufe
- Wichtigste Befehle
 

– pthread_create	Erzeugt neuen Thread
– pthread_exit	Beendet den aufrufenden Thread
– pthread_join	Wartet auf die Beendigung eines bestimmten Threads
– pthread_yield	Gibt CPU frei, damit andere Threads laufen können
– pthread_attr_init	Erzeugt und initialisiert eine Attributsstruktur
– pthread_attr_destroy	Löscht die Attributsstruktur
- Unter Linux u.U. Probleme mit Manpages → Web!

27. September 2012

Prof. Dr. Kornmayer

159

## JAVA Threads



- Integraler Bestandteil der Sprache
- Interface Runnable bzw. Klasse Thread
  - Überschreiben der Methode run()
- Weitere Methoden
  - start(): ein Thread wird gestartet
    - Beendet sich mit dem Ende der Methode run()
  - yield(): ein Thread gibt den Prozessor freiwillig ab. Der Scheduler verteilt anschliessend die CPU an wartende Threads
  - sleep(): Unterbrechung des Threads für eine vorgegebene Zeit
  - join(): warten auf Terminierung eines Threads
  - wait(): Thread wartet auf eine Benachrichtigung
  - notify(), notifyAll(): Versenden einer Benachrichtigung
  - synchronized: Lock-Mechanismus (später: Semaphore)

27. September 2012

Prof. Dr. Kornmayer

160

## Aufgabe

---



- Übungsblatt 2
  - Siehe Moodle-Webseite!
- Viel Spass beim Programmieren!