IT-Security

DI Hans Mühlehner
Wintersemester 2016/2017

Betriebssystem Internals

ILV Betriebssystem Internals
Stoff

- Inhalt der Vorlesung:
  - Aufbau und Funktionsweise verschiedener Betriebssystem-komponenten
  - Vorbereitung auf die Übungen
- Inhalt der Übungen:
  - System-nahe Programmierung

ILV Betriebssystem Internals

- Notenfindung
  - 33% Vorlesungsprüfung Theorie
  - 33% Übungsbenotung
    - Abgegebene Programme (Dokumentation)
    - Erklärung der Programme in den Übungsstunden (laufende Beurteilung der Mitarbeit)
  - 34% Praktische Abschlussübung, bei der im Laufe des Semesters entwickelte Programme erweitert werden müssen

    (Alle Teile müssen positiv sein)

ILV Betriebssysteme - Materialien

- Foliensätze
- POSIX Dokumentation (Online)
- Linux Treiberhandbuch (O'Reilley ebook)
- Literaturempfehlungen:
  - Tanenbaum, *Moderne Betriebssysteme*, Pearson Studium

  oder
  - W. Stallings, Modern Operating Systems, Pearson Prentice Hall

Teil 0

Einführung in die Thematik

Historischer Überblick

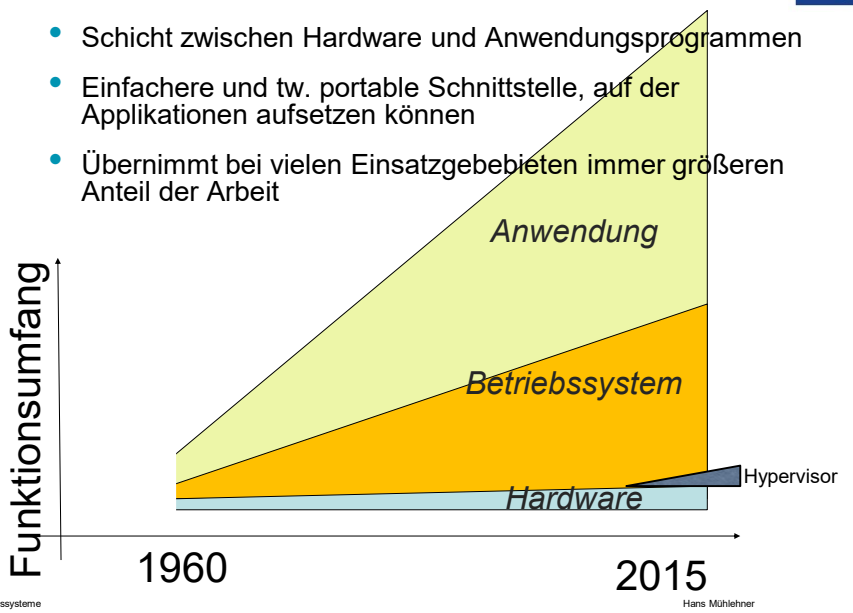
Charakteristik typischer Produkte

Was ist ein Betriebssystem

- Als **Betriebssystem** (operating system) bezeichnet man alle Programme, die die grundlegende Infrastruktur für die Ausführung von Anwendungssoftware bilden
- Das Betriebssystem bildet eine Abstraktion von Hardwareeigenschaften und ist für die Steuerung und Überwachung von Anwendungsprogrammen zuständig

Warum Betriebssysteme?

- Schicht zwischen Hardware und Anwendungsprogrammen
- Einfachere und tw. portable Schnittstelle, auf der Applikationen aufsetzen können
- Übernimmt bei vielen Einsatzgebebieten immer größeren Anteil der Arbeit



Betriebssystemaufgaben

- Prozessverwaltung – Programmausführung
- Verwaltung und Zuteilung von Memory
- Dateiverwaltung - Datenspeicherung
- Prozesskommunikation
- Hardwareverwaltung
- Sicherheit und Benutzerumgebungen
- Resourceabrechnung
- Monitoring
- Netzwerkunterstützung
- …….

Betriebssysteme wofür? (1)

- Mainframe (alt, proprietär, Stapelverarbeitung, Datendurchsatz…)
  - IBM OS 370, AS400, Siemens BS2000, ….
- Server (Datenverwaltung, Netzwerk, Benutzerverwaltung, wenigUI…)
  - Novell-Netware, Unix, Windows-NT, VM- und Speicherlösungen
- Client (Arbeitsstation) (Schwerpunkt UI)
  - X-Terminals, DOS und Windows
- PC-Systeme (zuerst billige „Personal-Rechner“ mit wenig Funktion, heute leistungsstarke Rechner mit „abgespecktem“ Server-OS sowohl Client- als auch Serverfunktionen grossteils enthalten)
  - Windows NT (Vista, XP, 7…), Unix (Linux, MacOS, Solaris,…)

Betriebssysteme wofür? (2)

- Echtzeit Systeme (oft proprietär, garantierte Reaktions- und Antwortzeiten, für „harte“ oder „weiche“ Echtzeitkriterien, Einsatzgebiete in industrieller Fertigung, Roboter, Verkehrssteuerung, Medizintechnik, …)
  - RTOS, LynxOS, RTLinux, ….
  - Echtzeitkomponenten in div. Unix/Linux Derivaten
- Embeded Systems (oft proprietär, tw. eher klein, für ganz bestimmte Art der Anwendung gebaut, zB: für Mautsysteme, Motormanagement, Verkehrssteuerungen, Handy, …)
  - Windows CE, Embeded Linux, PalmOS, Handy- (zB Android), Foto- od. Navi-Betriebssysteme, …
- OS für Smartcards (noch mehr Spezialisiert, Sicherheitsfunktionen)
  - ACOS , TCOS, CARDOS, …

Die neue Schicht zw. OS und HW: Virtualisierung

- Aus Sicherheitsgründen → Wunsch nach Aufteilung verschiedener Funktionen auf verschiedene Rechner
- Rechnerleistung so groß, dass oft nicht mehr ausgelastet
- → Virtualisierung (mehrere logische Rechner auf 1 Hardware)

Hypervisor-Lösungen (Virtualisierung zwischen HW und OS)	Prozess-Lösungen (Virtualisierung durch Prozess im OS)	Container-Lösungen (Virtualisierung mit shared OS-Kernel)
VM-Ware ESX	VM-Ware Server & WS	Solaris Zonen
Linux-XEN / Citrix	MS Virtual PC	Linux vServer
Microsoft Hyper-V	Sun Virtual Box	Windows 2016 Container
….	…..	…..

Betriebssystementwicklung History

- 50er Jahre: 1 Programm wird von 1 Prozessor abgearbeitet
  → BS unterstützt E/A und Umwandlung Zahlen / Zeichen
- 60er Jahre:
  Erste Konzepte von Parallelität: E/A-Prozessoren, Multiprogramming,
  Einführung von Prozessen als virtuelle Prozessoren,
  Virtueller Speicher,
  Prozesse als internes Strukturierungsmittel für BS,
  Interaktiver Mehrbenutzerbetrieb (Timesharing),
  Erste Ansätze von Großrechner-BS entstehen (OS/360, BS2000,...)

Betriebssystementwicklung History

- 70er Jahre:
- Prozess wird zur Schutzumgebung mit eigenem, abgeschottetem Adressraum und Rechten
- Forderung nach Unterstützung von modularer Programmierung, abstrakten Datentypen und Objektorientierung
- Einerseits entstehen sehr komplexe (und auch fehlerhafte) Betriebssysteme mit sehr viel Funktionalität
- Andererseits entsteht Unix nach dem Prinzip “simple is beautiful” auf einfacher Hardware (PDP-11), aber nicht unbedingt für Endbenutzer gedacht
- Einsatz höherer Programmiersprachen für BS-Implementierung statt nur Assembler

Betriebssystementwicklung History

- 80er Jahre:
- Beginnende Verbreitung von PCs und damit MSDOS / PCDOS(IBM)
- Integration von Netzwerken
- Erste verteilte Systeme kommen auf
- Unix entwickelt sich mit den Arbeitsplatzrechnern weiter
- Standardisierungsprozesse (POSIX Standard für UNIX basierende Systeme und MSDOS/PCDOS zwar nicht als Standard definiert, aber durch enorme Verbreitung auf billigen Endbenutzergeräten auch zum defacto Standard geworden)

Betriebssystementwicklung History

- 90er Jahre:
- Billige Microprozessoren bringen in Zusammenschaltung mehr Leistung als Großrechner
- Schwerpunkt Parallelverarbeitung in Weiterentwicklung
- Multimediaunterstützung
- Betriebssysteme für Embeded Systems
- Heterogene Systeme
- Mobile Geräte
- Internetintegration
- Am OS Markt für PCs übernimmt MS von IBM mit Windows 95 endgültig die Vormachtstellung (OS/2 wird eingestellt), Linux verbreitet sich als kostenlose UNIX-Version vorerst am Server Markt

Betriebssystementwicklung History

- 2000 - aktuell:
- Integration von Multimedia (Voice, Video, Kommunikation etc…)
- Billige Hardware führt auch in embeded Systmen zu 32-bit Architekturen → Einsatz von moderneren Betriebssystemen und Programmiersprachen
- Verteilte Anwendungen, GRID Computing
- Virtualisierung
- Steigende Bedeutung von Internet & Netzwerk
- Am Servermarkt konkurrieren Unix und Microsoft, bei der Verwaltung von Betriebssystemfunktionen im Netzwerk bring MS mit Windows 2000 das Active Directory als LDAP Datenbank mit Kerberos und übernimmt damit in vielen Betrieben auch die Serverrollen. Mainframes verschwinden langsam.
- Cloud Computing lässt Apps & z.T. das ganze OS vom Anwender zu Cloud Service Provider übersiedeln.
  Betriebssystemfunktionen müssen **Cloud-** und **on premises** Funktionalität kombinieren können (zB Authentifizierungsfunktionalität wie Federation Services, SSO-Lösungen).

Betriebsarten

- Einprogrammbetrieb (Singletasking) / Mehrprogrammbetrieb (Multitasking)
- Einprozessorbetrieb / Mehrprozessorbetrieb (Multiprocessing)
- Einbenutzerbetrieb / Mehrbenutzerbetrieb
- Stapelbetrieb / interaktiver (Dialog-) Betrieb
- Client Server Betrieb, Peer-to-Peer Betrieb

Betriebsarten

- Echtzeitbetrieb

  innerhalb einer garantierten Bearbeitungszeit müssen Anforderungen erfüllt werden

  oft als Embeded Systems

  Beispiele: Industriesteuerungen (zB Kraftwerke), Tranportsteuerungen (Bahn, Auto)

Betriebsarten

- **Rechnerverbundsysteme** (Cluster) ist ein Zusammenschluss mehrerer autonomer Rechner
  - die nach Außen wie 1 Rechner (Dienstanbieter) auftreten
  - zwecks Lastverteilung / Durchsatz (“Performance Cluster”)
  - zwecks Ausfallssicherheit (“Fault Tolerance Cluster”)
  - oder beides zusammen

Microsoft Betriebssysteme

- MS-DOS
  - Im Auftrag vom Marktführer IBM für PCs entwicklelt (PCDOS): 1980, textbasiert, single user, single tasking
- Windows (3.x)
  - Entwicklung Mitte der 80er Jahre, ab Version 3.0 durchschlagender Erfolg (10 Mio Verkäufe in 2 Jahren) trotz Konkurrenzprodukten von Apple, Attari, Commodore, die tw. leistungsfähiger waren.
    Kooperatives Multitasking, nicht selbständig bootfähig, “eigentlich ein MS-DOS Programm”

Microsoft Betriebssysteme

- Windows 95 / 98 / ME
  - MS-DOS sehr gut versteckt, 16-bit und 32-bit Programmunterstützung, neue graphische Oberfläche, Netzwerkintegration, Plug'n Play Untertützung, aber kein Speicherschutz, erweitertes FAT-16, ab 98 FAT-32
- Windows NT (3.0, 3.5, 4.0)
  - ab Anfang der 90er Jahre parallel zu Windows entwickelt, komplett neues 32-bit OS für mehrere HW Plattformen, Arbeitsplatz- & Serverversion, modernes Multitasking- , später auch Multiusersystem, sehr stabil, netzwerkweite Benutzer- und Berechtigungsverwaltung (Benutzer Domänen), modernes Filesystem (NTFS)

Microsoft Betriebssysteme

- Windows 2000 (NT 5.0)
  - Komplettüberarbeitung und Einführung des Active Directory dadurch enorme Verbesserung in netzwerkweiter Verwaltung von Betriebssystemaufgaben (Benutzer, Berechtigungen, Arbeitsumgebungskonfigurationen, Netzwerkkonfigurationen, Softwareverteilung…)
- Windows XP / Server 2003 (NT 5.1)
  - hauptsächlich Oberflächenredesign & "Fast Boot"
- Windows Vista / 7 / Server 2008 (NT 6.x)
  - Core-Versionen, Netzwerksicherheit, neue Oberfläche, neuer Netzwerkstack (IP6 Integration, Direct Access), Mobility,..

Microsoft Betriebssysteme

- Windows 8 / 8.1 / 10 / Server 2012 / Server 2016 (NT 6.x, NT 10)
  - Neue GUI (Kacheloberfläche) mit dem Ziel, am Desktop-Rechner und am Mobile-Gerät die gleiche GUI zu haben
  - Vollständiger Power Shell Support
  - unzählige Erweiterungen bei Netzwerk-Funktionalität, Authentifizierungsmechanismen (zB DAC), Virtualisierung, Cluster-Funktionalität und Cloud-Integration
  - Windows Container
  - OS wird als “Service” verkauft → Zwang zum Update

UNIX Betriebssysteme

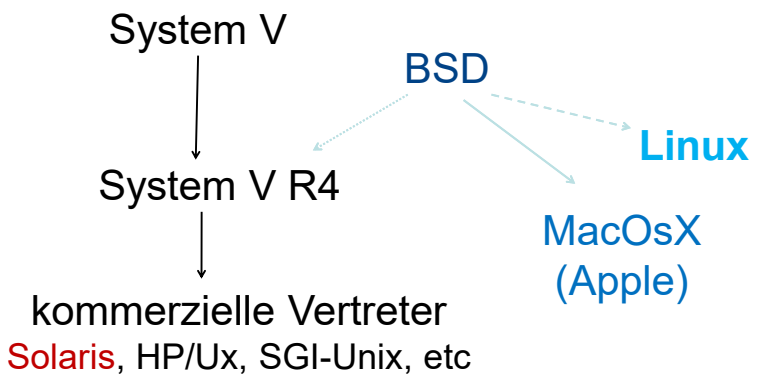
Unix, Linux, Solaris, MacOSx, AIX, HPUx, …

- Entwicklung im Auftrag von AT&T in den Bell-Labs aus Multics ab Mitte 1960;
- Offiziell verfügbar ab 1970
- Unix gibt es auf fast allen Maschinenarchitekturen
  vermutlich am weitesten verbreitet als Handy- oder TV-OS;
  auch im Workstation- und Serverbereich
- Unix gibt es auf Rechnern aller Größenordnungen
- Schwächen im Bereich Büroapplikationen (im Vergl. zu MS)

Unix Betriebssysteme Charakteritstika

- Mehrprogrammbetrieb
- Mehrbenutzerbetrieb
- Mehrprozessorbetrieb
- Stapelverarbeitung/interaktive Bearbeitung
- Rechnerverbundsysteme
- Netzwerkunterstützung („TCP/IP ist drauf erwachsen geworden“)
- Wenig End-Benutzerfreundlich, dafür sehr beliebt bei „Freaks“
  - stimmt nicht mehr, wo es ganz versteckt läuft (iOS, Android, TV…)

Unix Versionen, Hauptlinien/Ähnlichkeiten:



Linux

- heute populärstes Unix
- Linux Kernel – Verwendung fast überall:
  - Desktop/Server Distributionen (GNU + X)
    - Ubuntu, Fedora/RHEL, Gentoo, Debian, SUSE, ……
  - Android
  - Fernseher, Drucker, Router, Firewalls, Spielkonsolen,….

Teil 1

Mutitasking, Scheduling & Dispatching,

Problematik der Parallelität beim Resourcezugriff

Deadlocks

Programmausführung

Wesentliche Aufgaben im modernem OS:

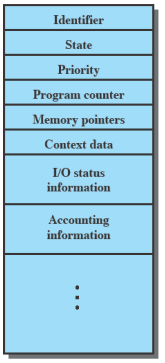
- Starten und Laden des Programms in den Hauptspeicher (ev. nur die Teile, die gerade gebraucht werden)
- Zur Verfügung Stellen von Resourcen wie Dateien, Geräten, etc.
- Einführung von (scheinbarer) Parallelität („Multitasking“)
- Verwendung von mehreren Prozessoren („Multiprocessing“)

Prozess

- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system instructions

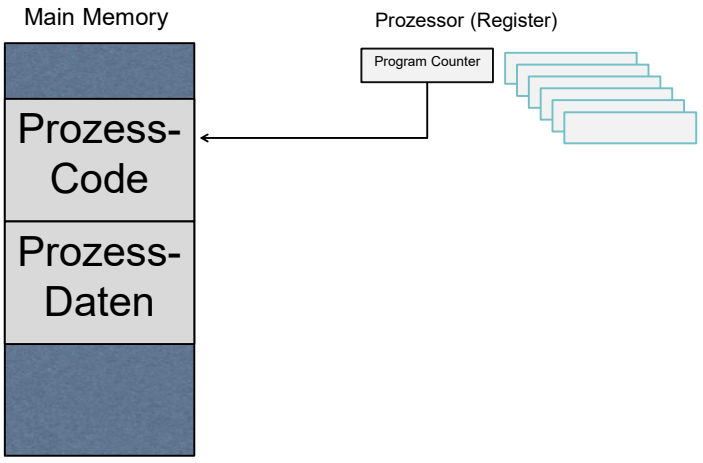
Woraus besteht ein Prozess?

- Identifier
- State
- Priority
- Program counter
- Memory pointers
- Context data
- I/O status information
- Accounting information

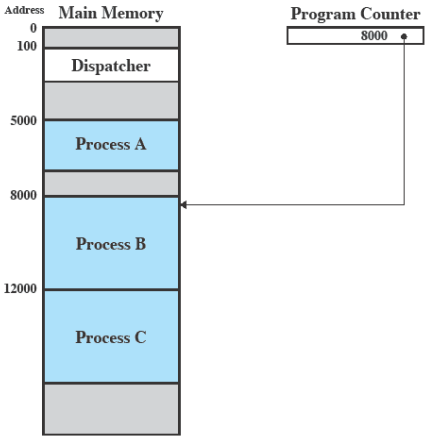


Example Process Control Block
aus: William Stallings: Modern Operating Systems

Die Ausführung eines Prozesses am Rechner



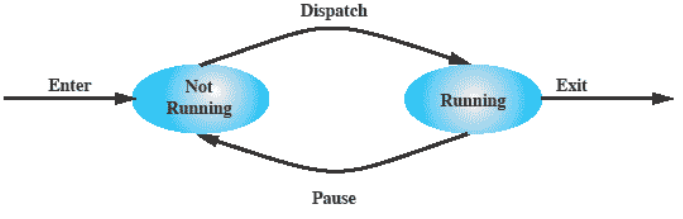
Ausführung mehrerer Prozesse



Example Execution Snapshot
aus: William Stallings: Modern Operating Systems

Two-State Process Model

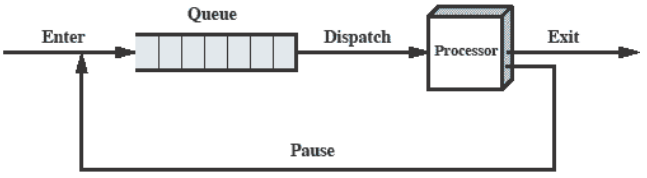
- Process may be in one of two states
  - Running
  - Not-running



(a) State transition diagram

aus: William Stallings: Modern Operating Systems

Prozesse, die nicht laufen, müssen warten → Warteschlangen (Queues)



(b) Queuing diagram

aus: William Stallings: Modern Operating Systems

Wann / Warum kommt ein anderer Prozess dran?

- Non-Preemptives (kooperatives) Multitasking
  - Der Prozess gibt die Kontrolle freiwillig ab (ruft den Dispatcher auf)
- Preemptives Multitasking
  - Das Betriebssystem (der Dispatcher) übernimmt die Kontrolle auf Grund von:

–Time-Slice aufgebraucht (Clock Interrupt)
–Interruptbehandlung
–Systemaufruf, der länger braucht, zB I/O-Operation → "block"
–Suspendierung (Trap) → "block"
–Resource-Anforderung → "block"
–Speichereinlagerung (Memory Fault)→ "block"
–Terminierung

A Five-State Model

- Running
- Ready
- Blocked
- New
- Exit

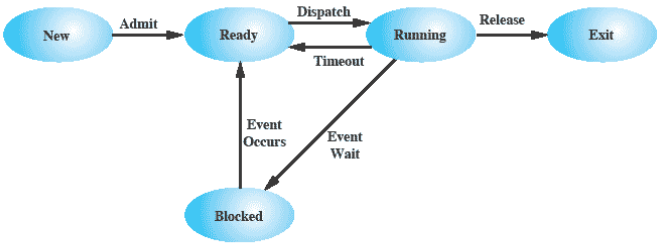


Figure 3.6 Five-State Process Model

aus: William Stallings: Modern Operating Systems

Process States

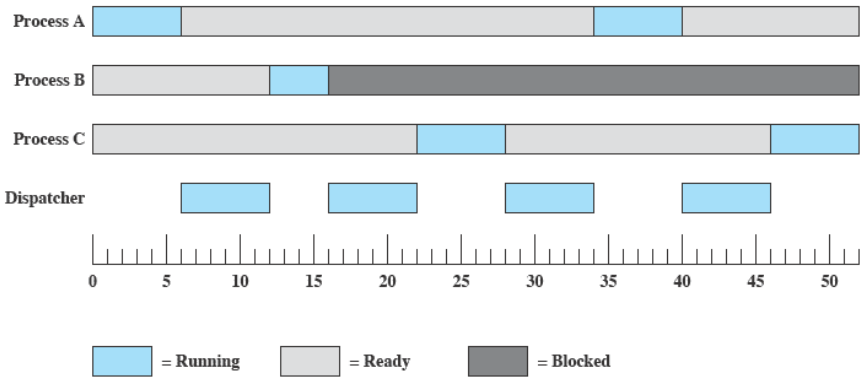
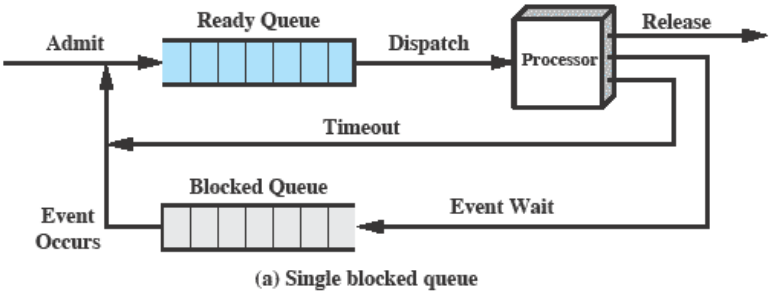


Figure 3.7 Process States for Trace of Figure 3.4

aus: William Stallings: Modern Operating Systems

Using Two Queues



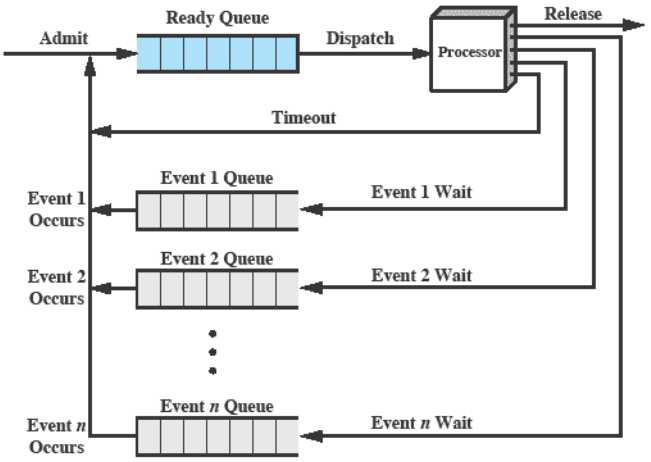
(a) Single blocked queue

aus: William Stallings: Modern Operating Systems

Context-Switch

- Save context of processor including program counter and other registers
- Update the process control block of the process that is currently in the Running state
- Move process control block to appropriate queue – ready; blocked; ready/suspend
- Select another process for execution
- Update the process control block of the process selected
- Update memory-management data structures
- Restore context of the selected process

Multiple Blocked Queues



(b) Multiple blocked queues

aus: William Stallings: Modern Operating Systems

Weitere Prozesszustände durch…

- Prozesse können temporär aus dem Hauptspeicher ausgelagert werden ("swapped")
  - wenn sie gerade blockieren
  - obwohl sie lauffähig wären
- Programmcode kann im Kernelmodus (voller Prozessor Befehlsumfang verfügbar) oder Usermodus (eingeschränkter Befehlssatz) laufen

UNIX Process State Transition Diagram

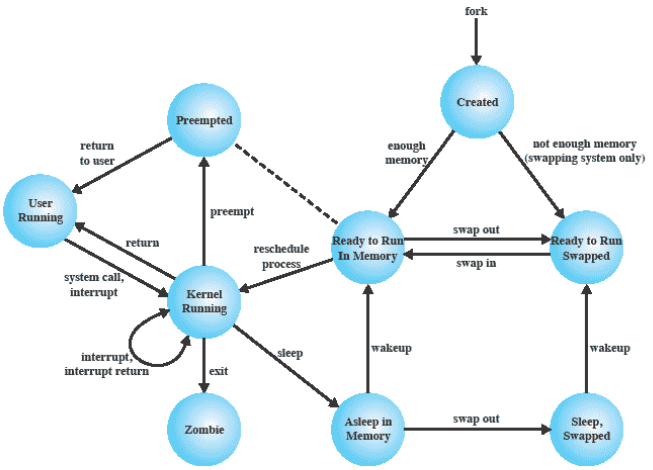


Figure 3.17 UNIX Process State Transition Diagram

aus: William Stallings: Modern Operating Systems

Scheduling / Displatching

Wer kommt als nächster dran?

Die Ziele:

All systems

Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems

Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

Interactive systems

Response time - respond to requests quickly
Proportionality - meet users' expectations

Real-time systems

Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

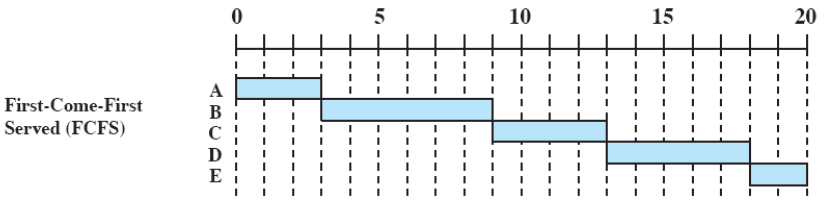
Scheduling Methoden

- FCFS (First Come First Served)
- Shortest Process Next
- Shortest Remaining Time Next
- HRRN (Highest Response Ratio Next)
- Round Robin
- Priority based
- Feedback
- Fair Share

FCFS

Beispiel:

Process	Startzeit	Servicezeit
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

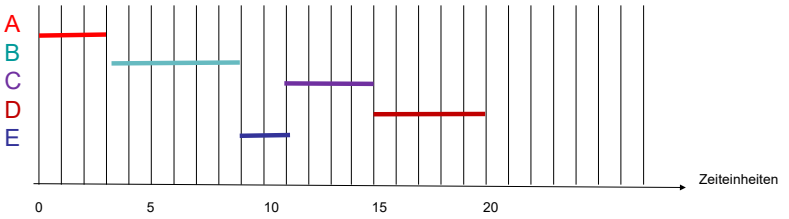


Wer am längsten in der Ready Queue ist, kommt dran.
Kurze Prozesse müssen überdimensional lange warten
(schlechte “Response Ration”)

Shortest Process Next (ohne Preemption)

Beispiel:

Process	Startzeit	Servicezeit
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



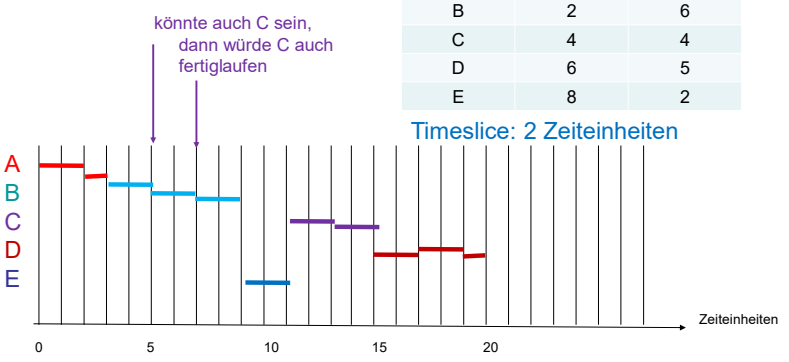
Prozesstabelle wird klein gehalten
Starvation-Gefahr für lange/große Prozesse!

Shortest Remaining Time Next (mit Preemption)

Beispiel:

Process	Startzeit	Servicezeit
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Timeslice: 2 Zeiteinheiten



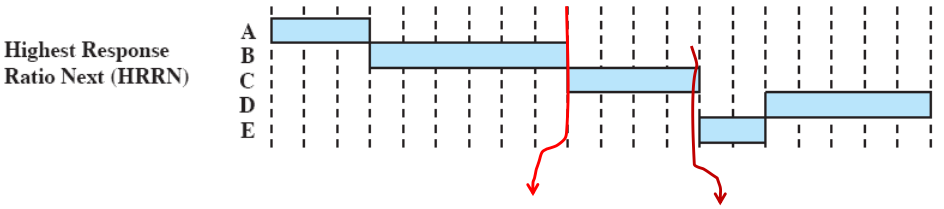
Starvation-Gefahr für lange/große Prozesse!

HRRN
(Highest Response Ratio Next)

$$Ratio = \frac{time\ spent\ waiting + expected\ service\ time}{expected\ service\ time}$$

Beispiel:

Process	Startzeit	Servicezeit
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



Wer weiß die “Servicezeit voraus?”
Versucht Antwortzeit in Bezug auf Aufwand gerecht zu verteilen.

Zeitpunkt 9:
C: (4+5)/4 = 2,25
D: (5+3)/5 = 1,6
E: (2+1)/2 = 1,5

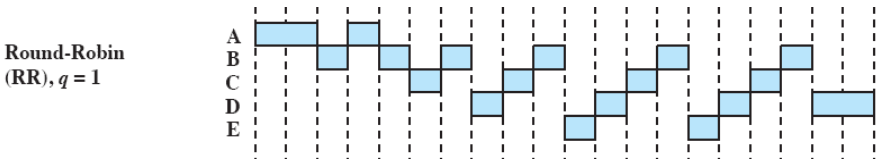
Zeitpunkt 13:
D: (5+7)/5 = 2,4
E: (2+5)/2 = 3,5

Round Robin

Beispiel:

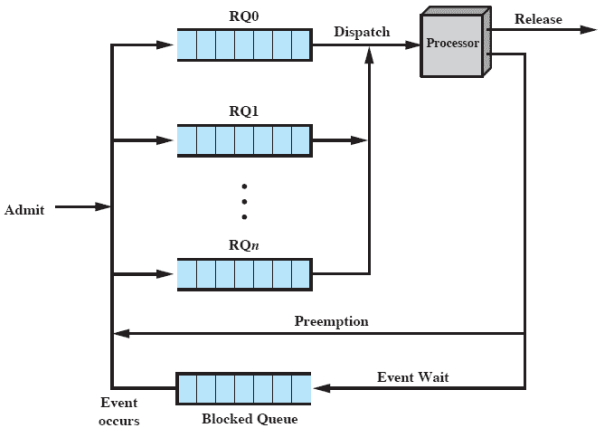
Process	Startzeit	Servicezeit
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Timeslice: 1 Zeiteinheit



Als klassisches “Time Slicing” bekannt.
ei periodischen Clock Interrupts wird Prozess unterbrochen und hinten in die Ready-Queue eingehängt.

Priority Based Scheduling
(Ready Queue pro Priorität)



aus: William Stallings: Modern Operating Systems

Feedback Scheduling
(zuletzt verbrauchte Prozessorzeit entscheidet)

Jobs, die viel Zeit gerechnet haben, werden bestraft

Kein Wissen über “remaining time” nötig

Kann mit (User-)Prioritäten gut kombiniert werden (zB beeinflusst das Prozessornutzungsverhalten die Einreihung in die Priority Queue indem als Bonus die Priorität erhöht oder als Strafe die Priorität verringert wird)

aus: William Stallings: Modern Operating Systems

Fair Share Scheduling

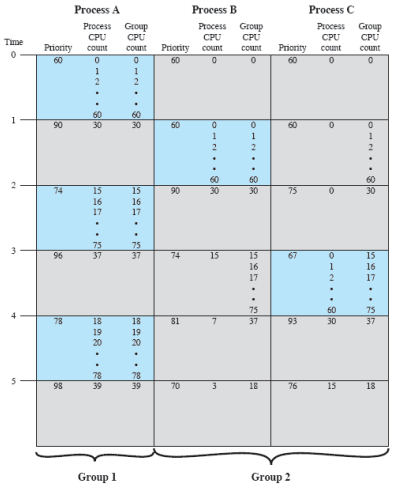
Eine Gruppe von Prozessen / Threads (eine Application)
sollte anderen Applicationen gegenüber “gerecht” behandelt werden.

Für Prozess *j* der Gruppe *k* gilt die Festlegung:

CPU*j* *(i) = CPUj (i-1) / 2*
CPU Auslastung von Prozess j im Intervall i.

GCPU*k* *(i) = GCPUk (i-1) /2*
CPU Auslastung von Gruppe k im Intervall i.

P*j* *(i) = Basis + CPUj (i-1) /4 +*
GCPUk (i-1) /4 x Wk
**W*k* *ist* das Gewicht der Gruppe



aus: William Stallings: Modern Operating Systems

Traditional Unix Scheduling

- Multilevel feedback using round robin within each of the priority queues
- If a running process does not block or complete within 1 second, it is preempted
- Priorities are recomputed once per second
- Base priority divides all processes into fixed bands of priority levels

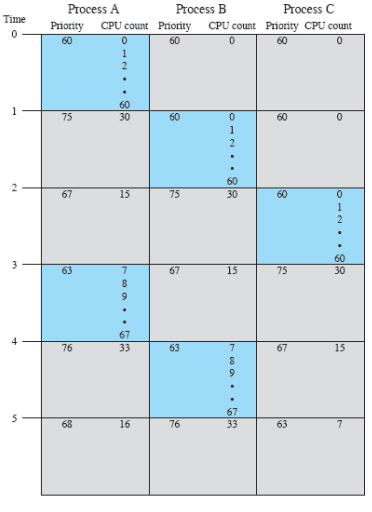
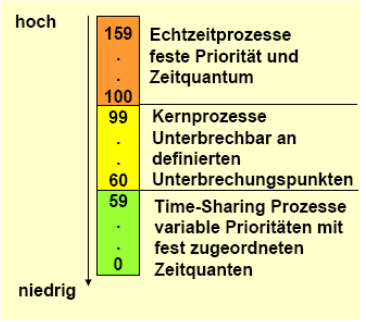


Figure 9.17 Example of Traditional UNIX Process Scheduling

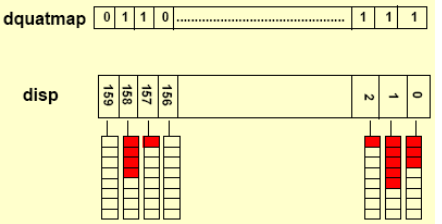
aus: William Stallings: Modern Operating Systems

Scheduling in Unix SVR4

Prioritätsklassen und Prioritätsstufen



Organisation der Warteschlange



Innerhalb der Time-Sharing Klasse variable Prioritäten und Zeitquanten. Das Zeitquantum hängt von der Prioritätsstufe ab.

aus: J. Kaiser VO Betriebssyteme Magdeburg

Scheduling in W2K

Festlegung der Basis-Prioritätsklassen

SetPriorityClass

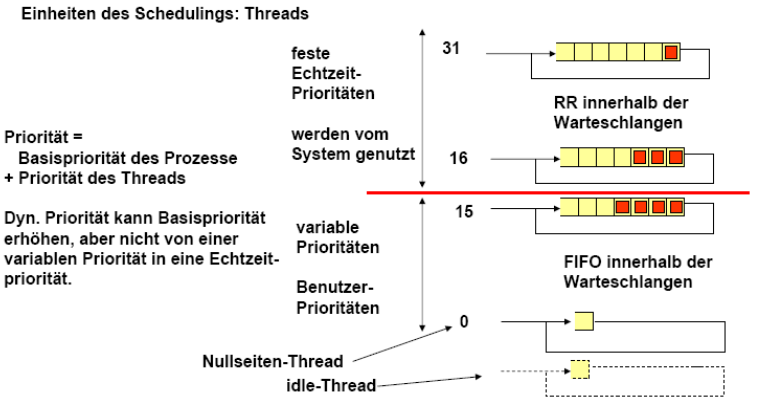
Prozessklassen- (Basis-) Prioritäten

Thread-Prioritäten	Echtzeit	hoch	über normal	normal	unter normal	idle
zeitkritisch	31	15	15	15	15	15
höchste	26	15	12	10	8	6
über normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
unter normal	23	12	9	7	5	3
niedrigste	22	11	8	6	4	2
idle	16	1	1	1	1	1

SetThreadPriority

aus: J. Kaiser VO Betriebssyteme Magdeburg

Scheduling in W2K



aus: J. Kaiser VO Betriebssyteme Magdeburg

Linux Scheduling

- ab Kernel 2.6.23: „Complete Fair Scheduler“
  (ab 2.6.24 inkl. CFS Task Group Scheduling)

  arbeitet mit 1 Red-Blak-Tree statt mit mehreren Priority-Queues

- unterstützt verschiedene Scheduling Policies (Classes)

  SCHED_OTHER or SCHED_NORMAL: The default policy with dynamic Priority (CPU usage + Nice Value)

  SCHED_BATCH: Similar to SCHED_OTHER, but with a throughput orientation

  SCHED_IDLE: A lower priority than SCHED_OTHER

  SCHED_FIFO: A first in/first out **realtime** policy

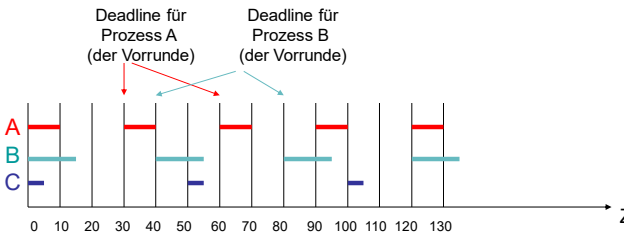
  SCHED_RR: A round-robin **realtime** policy

- chrt command

Echtzeitscheduling

Mehrere konkurrierende Prozesse müssen Deadlines einhalten

Beispiel:
Prozess A läuft 33 mal pro Sek. (alle 30 ms) für 10 ms
Prozess B läuft 25 mal pro Sek. (alle 40 ms) für 15 ms
Prozess C läuft 20 mal pro Sek. (alle 50 ms) für 5 ms

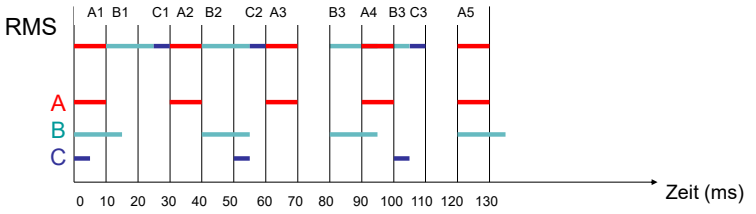


geht sichs aus?
10/30+15/40+5/50 =0,803
→ ca 80% CPU Zeit

Scheduling für Multimedia
Raten Monotones Scheduling (RMS)

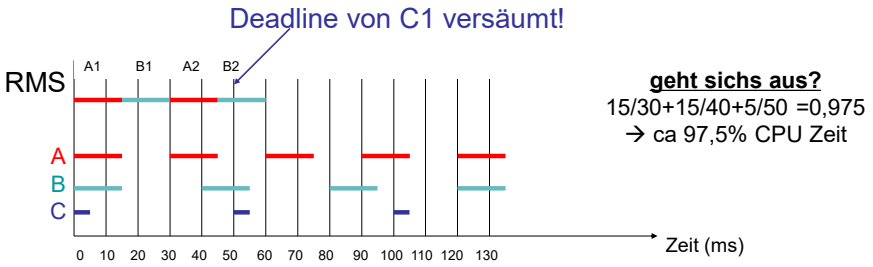
Priorität der Prozesse ergibt sich aus ihrer Rundenfrequenz

Beispiel 1:
Prozess A läuft 33 mal pro Sek. (alle 30 ms) für 10 ms → Priorität 33
Prozess B läuft 25 mal pro Sek. (alle 40 ms) für 15 ms → Priorität 25
Prozess C läuft 20 mal pro Sek. (alle 50 ms) für 5 ms → Priorität 20



Scheduling für Multimedia
Raten Monotones Scheduling (RMS)

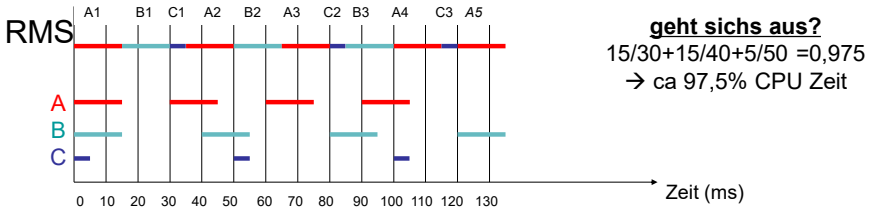
Beispiel 2:
Prozess A läuft 33 mal pro Sek. (alle 30 ms) für 15 ms → Priorität 33
Prozess B läuft 25 mal pro Sek. (alle 40 ms) für 15 ms → Priorität 25
Prozess C läuft 20 mal pro Sek. (alle 50 ms) für 5 ms → Priorität 20



geht sichs aus?
15/30+15/40+5/50 =0,975
→ ca 97,5% CPU Zeit

Scheduling für Multimedia
Earliest Deadline First (EDF)

Beispiel 2:
Prozess A läuft 33 mal pro Sek. (alle 30 ms) für 15 ms
Prozess B läuft 25 mal pro Sek. (alle 40 ms) für 15 ms
Prozess C läuft 20 mal pro Sek. (alle 50 ms) für 5 ms



geht sichs aus?
15/30+15/40+5/50 =0,975
→ ca 97,5% CPU Zeit

Prozesse und Threads

- Prozess
  - Resource Container
    - Adressraum, globale Variablen, Signalinfo,
    - Benutzerinfo, offene Dateien, …
- Thread
  - Scheduling Unit
    - Befehlszähler, Register, Zustand
    - Stack

Multithreading

- einfacheres Programmiermodell
- „billigere“ Parallelität
- Performance
- Ausnutzen von Multiprozessormaschinen
- Handling paralleler blockierender Systemcalls bei Operationen auf denselben Daten übernimmt das OS und muss nicht die Anwendung koordinieren

Realisierungsmöglichkeiten für Threads

- Threads im Benutzeradressraum
  - Threadtabelle und Thread-Laufzeitsystem im Benutzer-Space
  - schneller Wechsel,
  - Problem mit parallelen blockierenden Calls
- Threads im Kernel-Space
  - „teurer“
  - Scheduler im Kernel arbeitet mit Threads
- hybride Implementierung
  - Kernelthreads bestehen aus Benutzerthreads

Thread Realisierungen in der Praxis

- Windows NT Systeme:

  Thread und *Fiber*
- POSIX:

  Thread hat einen „*Contention Scope*“
  Process oder *System*
- Solaris, ..nix:

  Light-weight Process und *Thread*

Multi-thread Fähigkeit

- globale Variablen pro Thread / pro Prozess
- Ablaufinvarianz (zB Problem der globlalen Variablen in Prozedur!)
- Signalhandling im Thread / Prozess
- Speicherreservierung des OS bei Stackoverflow (→ mehrere Stacks!)
- Unterscheidung zw. Thread-save und Thread-unsave Funktionen

Probleme beim Zugriff auf gemeinsame Resourcen

- Race Condition
  - Zugriff auf gemeinsame Daten
  - Endergebnis abhängig von genauer Laufzeit der Prozesse/Threads

zur Vermeidung:

- Wechselseitiger Ausschluss von kritischen Abschnitten

Wechselseitiger Ausschluss durch aktives Warten

- TSL (Test and Set Lock)

```
while ( tsl(lock) == EsWarSchonGesperrt )
                    // tsl schaut, was drin steht und setzt Wert auf
    ;               // „gesperrt“, falls „frei“ war
// kritischer Abschnitt …
lock=1;             // gibt die Sperre wieder frei
```

aktives Warten verbraucht unnötig Rechenzeit!!

Betriebssysteme (oder Laufzeitumgebungen) bieten Methoden der Synchronisation, bei denen wartende Prozesse blockieren

- Typiche Primitive:
  - Semaphore
  - Mutex
  - Monitor
  - Message Passing
  - Condition Variable

Wo liegt der Vorteil gegenüber aktivem Warten?

Wechselseitiger Ausschluss kann zm Deadlock führen!!

Deadlock möglich?

- Prozess A
  - blockiere Resource A
  - blockiere Resource B
  - Arbeite
  - gib Resource B frei
  - gib Resource A frei

- Prozess B
  - blockiere Resource A
  - blockiere Resource B
  - Arbeite
  - gib Resource B frei
  - gib Resource A frei

Deadlock möglich?

- Prozess A
  - blockiere Resource A
  - blockiere Resource B
  - Arbeite
  - gib Resource B frei
  - gib Resource A frei

- Prozess B
  - blockiere Resource B
  - blockiere Resource A
  - Arbeite
  - gib Resource A frei
  - gib Resource B frei

Deadlock Bedingungen

- Wechselseitiger Ausschluss

  Resourcen sind verfügbar oder von Prozess besetzt

- Hold and Wait

  Prozesse, die schon Resourcen haben warten auf weitere

- Ununterbrechbarkeit

  Resourcen können einem Prozess nicht gewaltsam genommen werden

- Zyklische Wartebedingung

  zyklische Kette von wartenden Prozessen

Deadlocks ignorieren

Beispiel:

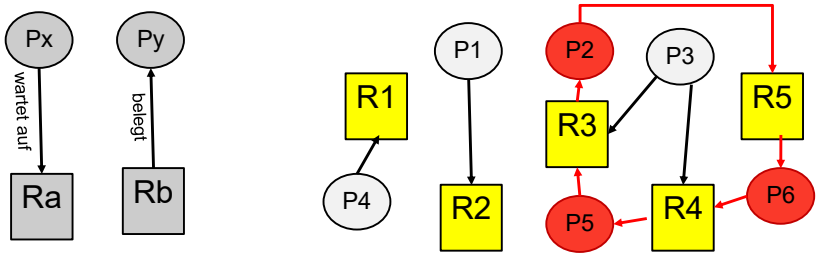
- In einem OS können maximal 100 Dateien geöffnet werden (Betriebssystemgrenze)
- 10 Prozesse öffnen je 10 Dateien und jeder will noch eine öffnen
- Alle probieren immer wieder in einer Schleife nach einiger Zeit die 11. Datei doch zu öffnen

Deadlock Erkennen und Beheben

- Erkennen grundsätzlich möglich, aber aufwendig, wenn vor jeder Resource-Anforderung durchgeführt
- Behebungsmöglichkeiten fragwürdig:
  - Resource einem Prozess für bestimmte Zeit entziehen ??
  - Teilweise Wiederholung – Rollback zu Checkpoint vor Resourceanforderung
  - Prozessabbruch

Deadlocks Erkennen ist relativ einfach:

- Mit Prozessen und Ressourcen einen Graphen bilden, der Belegung und Anforderung (Warten auf die Resource) beschreibt:



- **Kommt ein Zyklus zustande, sind die beteiligten Prozesse in einem Deadlock!**

Deadlock oder nicht?

(Wir haben pro Resource Mehrere Exemplare)

Bsp mit 4 Prozessen und 4 Resourcen

Resource Vector = { 4, 5, 3, 6 }

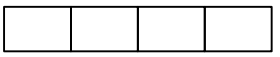
Belegungsmatrix =

{ { 2, 0, 2, 1 } //Prozess A
{ 1, 1, 0, 2 } //Prozess B
{ 0, 3, 0, 0 } //Prozess C
{ 1, 1, 0, 1 } } //Prozess D

Anforderungsmatrix =

{ { 1, 0, 1, 1 } //Prozess A
{ 1, 0, 0, 2 } //Prozess B
{ 0, 2, 0, 3 } //Prozess C
{ 0, 0, 1, 2 } } //Prozess D

Verfügbar?



kann damit einer weiterlaufen?

Deadlock oder nicht?

Resource Vector = { 4, 5, 3, 6 }

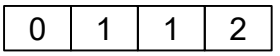
Belegungsmatrix =

{ { 2, 0, 2, 1 } //Prozess A
{ 1, 1, 0, 2 } //Prozess B
{ 0, 3, 0, 0 } //Prozess C
{ 1, 1, 0, 1 } } //Prozess D

Anforderungsmatrix =

{ { 1, 0, 1, 1 } //Prozess A
{ 1, 0, 0, 2 } //Prozess B
{ 0, 2, 0, 3 } //Prozess C
{ 1, 0, 1, 2 } } //Prozess D

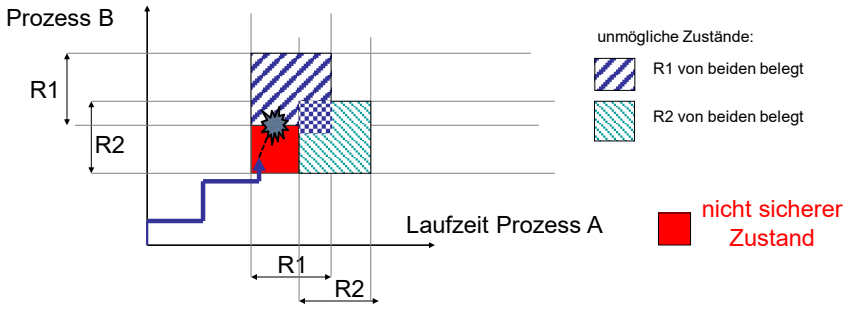
Verfügbar?



kann damit einer weiterlaufen?

Deadlocks Verhindern

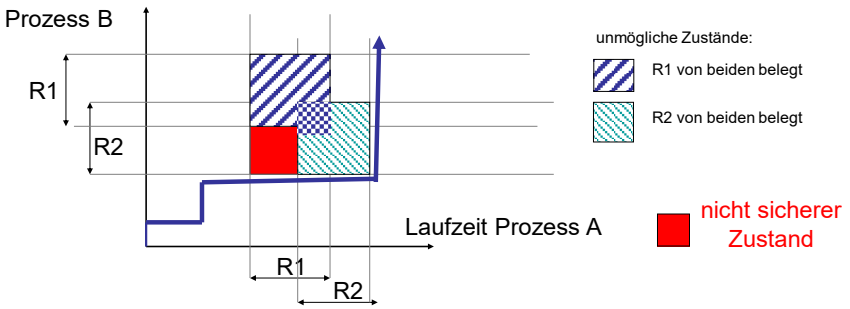
- Bankier Algorithmus
  - Arbeitet mit sicheren Zuständen: Es gibt immer eine Scheduling Reihenfolge, in der alle Prozesse abgearbeitet werden können!



Diese Scheduling-Entscheidungen führen in den unsicheren Zustand und schließlich zum Deadlock!

Deadlocks Verhindern

- Bankier Algorithmus



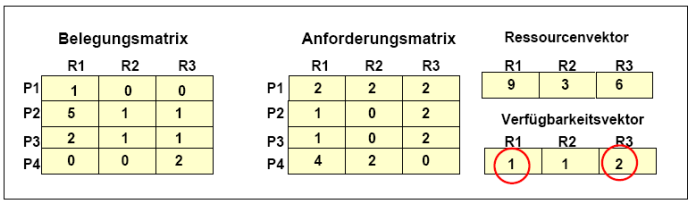
Diese Scheduling-Entscheidungen führen am unsicheren Zustand vorbei und es kommt nicht zum Deadlock!

Problem: Für jeden Prozess muss von vorne herein bekannt sein, welche Resourcen gebraucht werden können.

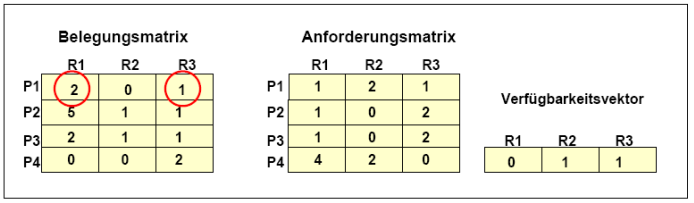
Bankier Algorithm

- **Ein sicherer Zustand ist ein Zustand, von dem aus mindestens eine Ausführungsreihenfolge existiert, die nicht zu einem Deadlock führt, d.h. alle Prozesse können bis zum Abschluss ablaufen.**

Sicherer Zustand



Unsicherer Zustand

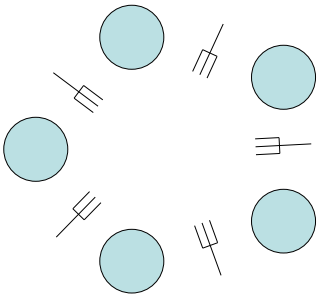


Deadlock Vermeiden

Deadlock Bedingung	Vermeidungsversuch
Wechselseitiger Ausschluss	Alles spoolen
Hold and Wait	Alle Resourcen zu Beginn anfordern
Ununterbrechbar	Resource wegnehmen
Zyklsiches Warten	Resourcen nummerieren und nur in dieser Reihenfolge reservieren

Das klassische IPC Problem:

Die dinierenden Philosophen



Deadlock

Aushungern

Performance

keine Lösungen des „Problems der Denker“

```
#define ANZAHL 5
#define links nr
#define rechts (nr+1)%ANZAHL

void denker (int nr)
{
        while (true)
        {
                think();
                takefork(links);
                takefork(rechts);
                eat();
                putfork(links);
                putfork(rechts);
        }
}
```

```
void denker (int nr)
{
      bool gotforks;
      while (true)
      {
            think();
            gotforks=false;
            do
            {
                  takefork(links);
                  if ( trytakefork(rechts))
                        gotforks=true;
                  else
                        putfork(links);
            } while ( ! gotforks );
            eat();
            putfork(links);
            putfork(rechts);
      }
}
```

Unperformante Lösung des „Problems der Denker“

```
void denker (int nr)
{
        while (true)
        {
                think();
                sem_wait(sem); // hole die Semaphore
                takefork(nr);
                takefork((nr+1)%N);
                eat();
                putfork(nr);
                putfork((nr+1)%N);
                sem_post(sem); // gib die Semaphore frei
        }
}
```


Mögliche Lösung des „Problems der Denker“

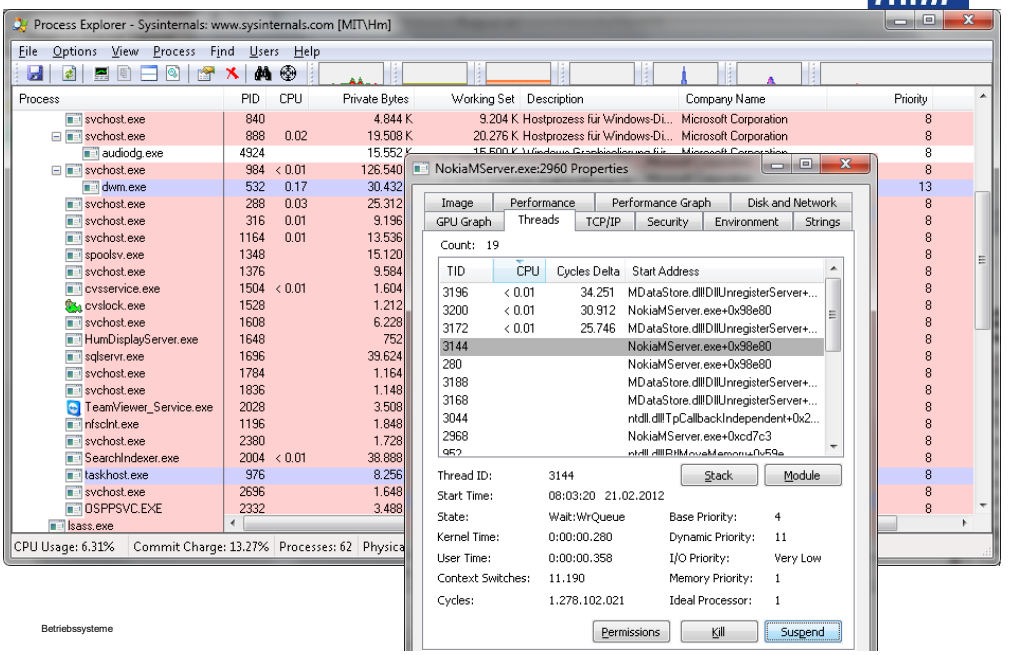
```
int state[N];

void denker (int nr)
{
  while (true)
    {
      think();
      takeforks(nr);
      eat();
      putforks(nr);
    }
}
```

```
mutex m;
semaphore s[N]

void takeforks (int nr)
{
  mutex_lock(m);
  state[nr]=HUNGRY;
  test(nr)
  mutex_unlock(m);
  sem_wait(s[i]);
}
```

```
#define LEFT (nr+N-1)%N
#define RIGHT (nr+1)%N

void putforks (int nr)
{
  mutex_lock(m);
  state[i]=THINK;
  test(LEFT);
  test(RIGHT);
  mutex_unlock(m);
}
```

```
void test(int i)
{
        if (state[i]==HUNGRY && state[LEFT]!=EAT && state[RIGHT]!=EAT)
        {  state[i]=EAT; sem_post(s[i]); }
}
```

Prozessinfos in Windows NT



Dispatcherinfos in Solaris

```
sunb# { dispadmin -l; dispadmin -c TS -g ; } | more
CONFIGURED CLASSES
==================

SYS      (System Class)
TS       (Time Sharing)
FX       (Fixed Priority)
IA       (Interactive)
# Time Sharing Dispatcher Configuration
RES=1000

# ts_quantum  ts_tqexp  ts_slpret  ts_maxwait ts_lwait  PRIORITY LEVEL
       200         0         50          0         50        #     0
       200         0         50          0         50        #     1
       200         0         50          0         50        #     2
       200         0         50          0         50        #     3
       200         0         50          0         50        #     4
       200         0         50          0         50        #     5
       200         0         50          0         50        #     6
       200         0         50          0         50        #     7
       200         0         50          0         50        #     8
       200         0         50          0         50        #     9
       160         0         51          0         51        #    10
--Weiter--
```

Prozessinfos in Unix

```
sunb# ps -cl                                                            [~]
 F S    UID   PID  PPID  CLS PRI     ADDR     SZ    WCHAN TTY         TIME CMD
 0 S      0  3324  3274   RT 115        ?    428        ? pts/1       0:00 sleep
 0 O      0  3341  3274   TS  49        ?    455          pts/1       0:00 ps
 0 S      0  3274  3263   TS  49        ?   1115        ? pts/1       0:01 zsh
sunb# ps -l                                                             [~]
 F S    UID   PID  PPID   C PRI NI      ADDR     SZ    WCHAN TTY         TIME CMD
 0 S      0  3324  3274   0   0 RT         ?    428        ? pts/1       0:00 sleep
 0 O      0  3342  3274   0  50 20         ?    455          pts/1       0:00 ps
 0 R      0  3274  3263   0  60 20         ?   1115          pts/1       0:01 zsh
sunb# /usr/ucb/ps -l                                                    [~]
 F   UID   PID  PPID %C PRI NI   SZ  RSS    WCHAN S TT        TIME COMMAND
 0     0   376     7  0  59 20 2960 1248 30007ac6452 S console   0:00 /usr/lib/saf/ttymon -g -d /dev
/c
 0     0  3274  3263  0  49 20 8920 3784 30007ee1086 S pts/1     0:00 -zsh
 0     0  3324  3274  0 115  0 3424 1144 3000924fba6 S pts/1     0:00 sleep 500
 0     0  3343  3274  0  49 20 3648 1128          O pts/1     0:00 /usr/ucb/ps -l
 0     0 24486 24472  0  59 20 5312 1824          T pts/3     0:00 login as: is101002
 0     0 16725 16662  0  59 20 5312 2648          T pts/4     0:00 login
 0     0 22268 22212  0  59 20 5264 2136          T pts/17    0:00 su
sunb#                                                                   [~]
```


Performance / System Monitoring

- Microsoft: perfmon, resmon
- Unix/Linux: top, prstat, sar, vmstat

```
sunb# vmstat                                                            [~]
 kthr      memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr rm s0 s1 s2   in   sy   cs us sy id
 0 0 0 341536 79616   1   8  0  0  0  0  0 -0  0 -1  0  241 19046 137  4  3 94
sunb# sar -up 2 2                                                       [~]

SunOS sunb 5.10 Generic_139555-08 sun4u    02/21/2012

09:45:55    %usr    %sys    %wio   %idle
          atch/s  pgin/s ppgin/s  pflt/s  vflt/s slock/s

09:45:57       0       1       0      99
            0.00    0.00    0.00    1.48    6.40    0.00

09:45:59       0       2       0      98
            0.00    0.00    0.00    0.00    0.00    0.00
```

Linux Process Concepts

- fork – Hierarchie → copy-on-write [1970: par.exec. unit:process!]
- Software-Interrupts == Signals
- Byte-Pipes (System or Named)
- Stack for Kernel-Mode and Stack for User Mode
- Threads are organized in Task Structs
  - Scheduling parameters, memory image pointers, signalinfos, registers, syscall-states, file-descriptor table, accounting info, kernel stack, zuid, gid, umask ….
- clone and pthread_create
- CFS [in previous Versions: O(1)] internal Prio: 100-139
- RT FIFO and RT RR internal Prio: 0 - 99

Linux Process Concepts

- only runable tasks are in a CPU runque (taskstruct queue)
  - 1 queue per CPU → CPU Affinity / Syscalls for Affinity management
- waiting tasks are in waitqueues
  - 1 queue per Event includes a spinlock (→ can be used by process and kernel)
- Sync Mechanisms:
  - spinlocks, memory barieres, mutex and semaphore operations (incl. trylocks)
  - RCU (read-copy-update), Condition-Variables,

Windows Process Concepts

- CreateProcess → Initialize Process Struct [198x: par.exec. unit: thread!]
  (~100 Syscalls for Process/Thread Handling! APCs)
- Jobs [Resource Linits and Quotas] – Processes [Resource Container] – Threads [Kernel Sched. Unit] – Fibers [User Sched Unit]
- Thread-Pools
- Stack for Kernel-Mode and Stack for User Mode
- Process/Thread Data is stored in
  - PEB (Process Env. Block), TEB (Thread Env Block) and User Shared Data
    (readable by process, writeable by kernel → no CS to read it!)
- Communication:
  - System- and named- Byte-Pipes and Message-Pipes, Mailslots (1-way pipes), Sockets, RPC (over TCP/IP or LPC/ALPC), Shared Objects

Windows Process Concepts

- Windows Scheduling:
  - CPU Affinity, NUMA Support, Base Priorities with Boosts for:
    - Completion of IO-Operation: (1: disk, 2: serial, 6: keyboard)
      On Return from Waiting on Events like Semaphore ...: (2: foreground, 1: otherwise)
      Consuming entire time slice: -1
    - Autoboost: Move higher Priority from higher priority waiting thread to lower priority producer thread
    - no boost for Real-Time Threads!
- Sync Mechanisms for Apps (all are operating on threads!):
  - Mutex and Semaphore operations (incl. trylocks), [kernel mechanism]
  - Critical Sections [like mutex but in user mode]
  - Notification Event: when signaled all waiting threads are released
  - Synchronisation Event; 1 waiting thread is released