

2 Treiber mit DMA (3)

■ Große Systeme mit mehreren DMA-Kanälen und vielen Platten

- ◆ es muss ein freier DMA-Kanal gesucht werden und evtl. auf einen freien gewartet werden bevor der Auftrag ausgeführt werden kann
- ◆ Anforderung kann parallel zur Plattenpositionierung erfolgen

■ Mainframe-Systeme

- ◆ Steuereinheit fasst mehrere Platten zu einem Gerät zusammen
- ◆ mehrere Steuereinheiten hängen an einem Kanal zum Hauptspeicher
- ◆ zum Zugriff auf die eigentliche Platte muss erst die Steuereinheit und dann der Kanal belegt werden (Teilwegbelegung)

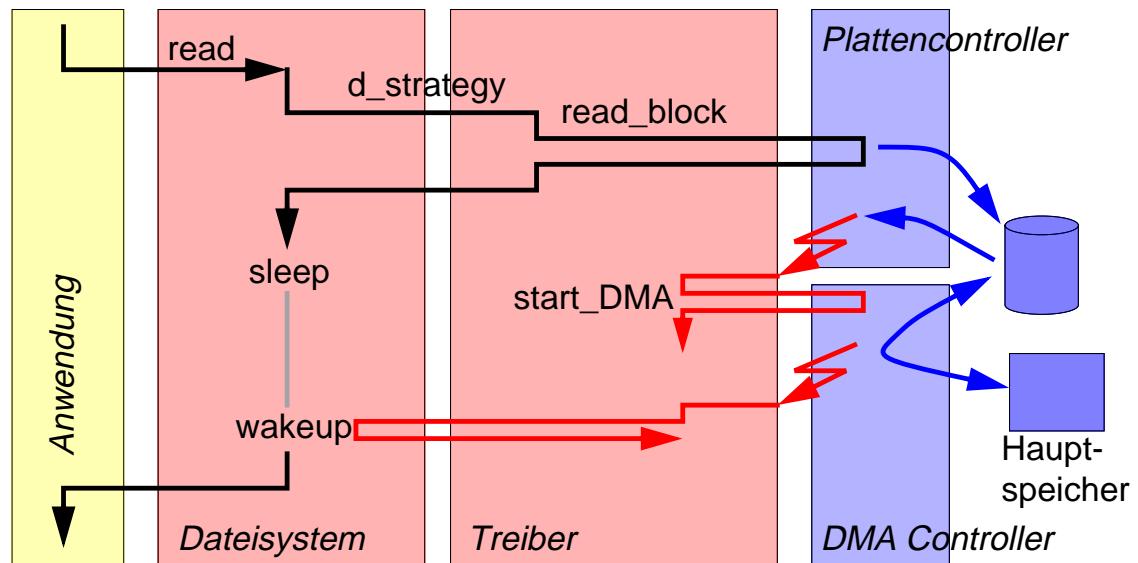
■ DMA und Caching

- ◆ heutige Prozessoren arbeiten mit Datencaches
- ◆ DMA läuft am Cache vorbei: Betriebssystem muss vor dem Aufsetzen von DMA-Transfers Caches zurückschreiben und invalidieren

3 Treiber für intelligente Platte

■ Intelligente Platten besitzen eigenen Prozessor für

- ◆ das Umsortieren von Aufträgen (interne Plattenstrategie)
- ◆ eigene Bad block-Erkennung, etc.

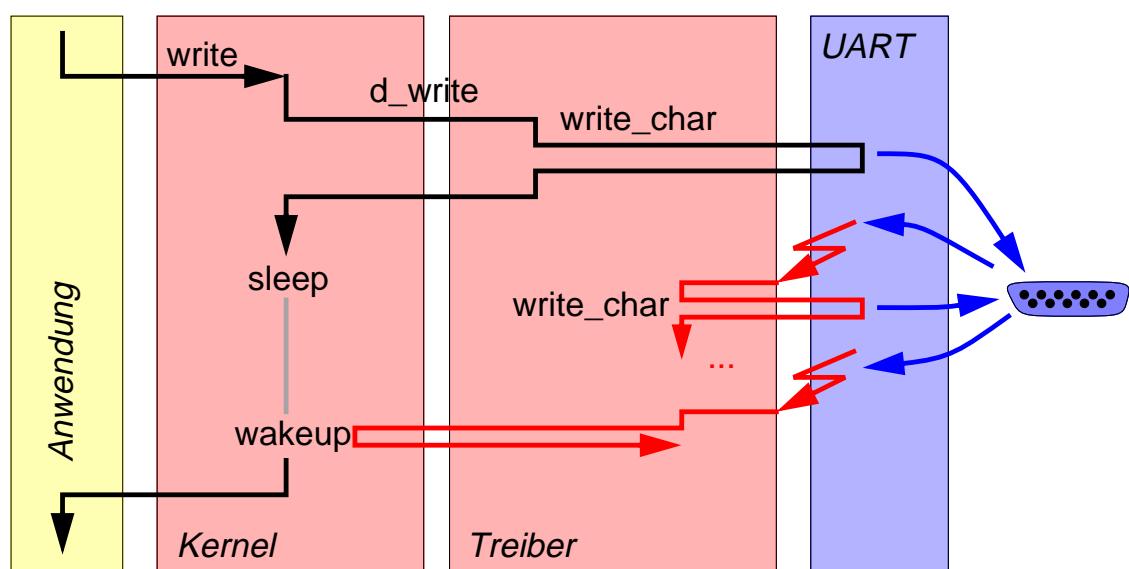


G.3 Treiber für serielle Schnittstellen

- Einsatz serieller Schnittstellen (z.B. RS-232)
 - ◆ Terminals
 - ◆ Drucker
 - ◆ Modems
- Datenübertragung
 - ◆ zeichenweise seriell (z.B. Startbit, Datenbits, Stopbits)
 - ◆ getaktet in bestimmter Geschwindigkeit (Bitrate, z.B. 38.400 Bit/s), im Vergleich zu Platten relativ langsam
 - ◆ Flusskontrolle (d.h. Empfänger kann Datenfluss bremsen)
 - ◆ bidirektional
- Treiber
 - ◆ zeichenorientiertes Gerät
 - ◆ vom Prinzip her ähnlich dem Plattentreiber

1 TTY-Treiber

- TTY-Treiber (*Teletype, Fernschreiber*) und der Ablauf eines Schreibauftrags



- ◆ UART = Universal Asynchronous Receiver / Transmitter

1 TTY-Treiber (2)

- Enger Zusammenhang zwischen Ein- und Ausgabe
 - ◆ Echofunktion (getippte Zeichen werden angezeigt)
 - eingelesene Zeichen werden gleich wieder ausgegeben
 - ◆ Flusskontrolle (bestimmtes Zeichen in der Eingabe hält Ausgabe an: ^S)
 - wird ^S eingelesen wird Ausgabe angehalten bis ^Q eingelesen wird
- Zeilenorientierte Treiber
 - ◆ Anwendung will Zeichen zeilenweise, z.B. Shell
 - ◆ Treiber blockiert Prozess bis Zeilenende erkannt
 - ◆ Treiber erlaubt das Editieren der Zeile (Backspace, etc.)
- Signale
 - ◆ bestimmte Zeichen lösen Signale an korrespondierende Prozesse aus

2 TTY-Treiber in UNIX

- Konfigurierbar
 - ◆ Repräsentation einer seriellen Schnittstellen als zeichenorientiertes Gerät
 - ◆ durch Aufruf von ioctl kann Treiber konfiguriert werden

```
int ioctl( int fildes, int request, /* arg */ );
```
 - ◆ Kommando zum Lesen der Konfiguration: Übergabe einer Strukturadresse

```
struct termios t;
ioctl( fd, TCGETS, &t );
```
 - ◆ Kommando zum Schreiben einer Konfiguration:

```
ioctl( fd, TCSETS, &t );
```
 - ◆ Struktur enthält Bitfelder für verschiedene Einstellungen
 - ◆ Bitmasken sind als Makros verfügbar
 - ◆ näheres: „**man termios**“ und „**man ioctl**“

3 Einstellung der physikalischen Parameter

■ Bitrate einer seriellen Schnittstelle

- ◆ **B2400** 2400 Bit/s
- ◆ **B4800** 4800 Bit/s
- ◆ **B9600** 9600 Bit/s
- ◆ **B19200** 19200 Bit/s
- ◆ **B38400** 38400 Bit/s
- ◆ **B57600** 57600 Bit/s

■ Zeichengröße, Parität, Stopbits

- ◆ **CS7** 7 Bits
- ◆ **CSTOPB** zwei Stopbits sonst eins
- ◆ **PARENB** Parität einschalten
- ◆ **CRTSCTS** Hardware-basierte Flusskontrolle einschalten

4 Einstellung der Ein-, Ausgabeverarbeitung

■ Festlegen der Zeichen mit Sonderbedeutung

- ◆ Erase-Character: löscht letztes Zeichen (Backspace)
- ◆ Kill-Character: löscht ganze Zeile (^K)

■ Eingabeverarbeitung

- ◆ **ICRNL** CR-Zeichen wird als NL-Zeichen gelesen
- ◆ **ICANON** kanonische Eingabeverarbeitung (Zeileneditierung)
- ◆ **IXON** erlaube Flusskontrolle mit ^Q und ^S

■ Ausgabeverarbeitung

- ◆ **ECHO** schaltet Echofunktion ein
- ◆ **ECHOE** Echo von Backspace als Backspace, Leerzeichen, Backspace
- ◆ **ONLCR** NL-Zeichen wird als CR, NL ausgegeben

5 Signalauslösung und Jobkontrolle

■ Signalauslösung

- ◆ **ISIG**: Schaltet Signale ein
- ◆ **INTR**-Zeichen: sendet **SIGINT**-Signal (^C)
- ◆ **QUIT**-Zeichen: sendet **SIGQUIT**-Signal (^|)

■ Signal wird an ganze Prozessgruppe geschickt

- ◆ alle Prozesse der Gruppe empfangen Signal
- ◆ Beispiel: `cat /etc/passwd | grep Mueller | sort`
- ◆ alle Prozesse erhalten **SIGINT** bei ^C

■ Prozessgruppe

- ◆ Prozessgruppen-ID wird wie eine Prozess-ID (PID) bezeichnet
- ◆ Prozess mit gleicher PID und Prozessgruppen-ID ist Gruppenführer
- ◆ Shell sorgt dafür, dass im Beispiel **cat**, **grep** und **sort** in der gleichen Prozessgruppe sind (**sort** wird Gruppenführer)

5 Signalauslösung und Jobkontrolle (2)

■ Vordergrund- und Hintergrundprozesse

- ◆ Hintergrundprozesse erhalten keine Signale.
- ◆ Bei Shells mit Jobkontrolle kann zwischen Vorder- und Hintergrundprozessen umgeschaltet werden.

■ Sessions

- ◆ Shell öffnet eine Session, die mehrere Prozessgruppen enthalten kann (spezieller systemabhängiger Systemaufruf).
- ◆ Shell wird Sessionführer.
- ◆ Shell erzeugt Prozesse und Prozessgruppen.
- ◆ Ein TTY wird Controlling-Terminal für alle Prozessgruppen der Session.
- ◆ Unterbrechen der Terminalverbindung (**SIGHUP**) wird dem Sessionführer zugestellt.

5 Signalauslösung und Jobkontrolle (3)

■ Vordergrundprozess

- ◆ Eine Prozessgruppe der Session kann zur Vordergrundprozessgruppe gemacht werden.
- ◆ **SIGINT** und **SIGQUIT** sowie die Eingabe vom Terminal werden nur der Vordergrundprozessgruppe zugestellt.

■ Hintergrundprozesse

- ◆ Alle Hintergrundprozesse bekommen keine Eingabe vom Terminal und werden gestoppt, wenn sie lesen wollen (Shell wird benachrichtigt).

■ Jobkontrolle

- ◆ Shell kann zwischen Vorder- und Hintergrundprozessgruppen umschalten
- ◆ Benutzer kann Vordergrundprozesse stoppen und gelangt zur Shell zurück

5 Signalzustellung und Jobkontrolle (4)

■ Beispiel: Stoppen und wiederaufnehmen eines Vordergrundprozesses

```
prompt> cc -o test.c
^Z
Suspended
prompt> jobs
[1] Suspended cc -o test.c
prompt> fg %1
```

- ◆ Realisiert mit einem Signal namens **SIGTSTP**, das die Prozessgruppe stoppt
- ◆ Shell bekommt dies mit über ein **waitpid()**
- ◆ Shellkommando fg sendet ein Signal **SIGCONT** und die Prozesse fahren fort

5 Signalzustellung und Jobkontrolle (5)

- Beispiel: Stoppen eines Vordergrundprozesses, Umwandlung in einen Hintergrundprozess

```
prompt> cc -o test.c
^Z
Suspended
prompt> bg
[1] Running cc -o test.c
prompt>
```

- ◆ Wie auf vorheriger Folie, aber:

Shell schaltet die Prozessgruppe in den Hintergrund und wartet nicht mehr auf deren Beendigung.

5 Signalzustellung und Jobkontrolle (6)

- Beispiel: Starten eines Hintergrundprozesses und Umwandlung in einen Vordergrundprozess

```
prompt> cc -o test.c &
prompt> jobs
[1] Running cc -o test.c
prompt> fg %1
```

- ◆ Shell startet eine Hintergrundprozessgruppe und nimmt Kommandos entgegen
- ◆ **fg** Kommando schaltet die Hintergrundgruppe in eine Vordergrundprozessgruppe um und wartet auf deren Beendigung mit **waitpid()**

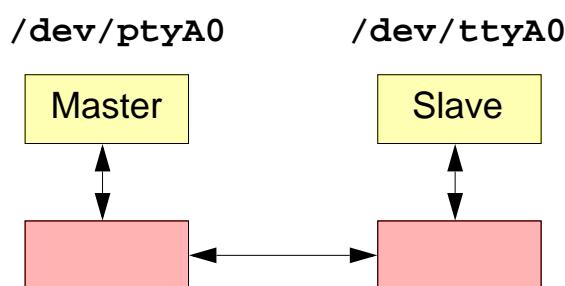
6 Pseudo-Terminals

■ Pseudo-TTY-Treiber (PTY)

- ◆ keine echte serielle Schnittstelle vorhanden
- ◆ Shell und andere Prozesse benötigen aber ein TTY für
 - Flusskontrolle,
 - Echofunktion,
 - Job-Kontrolle etc.
- ◆ fungiert als gewohnte Schnittstelle von Anwendungsprozessen
- ◆ Einsatz beispielsweise bei einem Fenstersystem (xterm-Programm)
 - xterm-Programm bedient die Masterseite eines PTTY
 - Shell und Anwendungsprogramme sehen xterm-Fenster wie ein TTY (Slavesseite)

6 Pseudo-Terminals (2)

■ Master- und Slaveside sehen wie ein normales TTY-Device aus



- ◆ Slaveside besitzt Modul zur Flusskontrolle, Eingabeeditierung, Signalzustellung, Flusskontrolle etc.

- Bisher: Lese- oder Schreibaufrufe blockieren
 - ◆ Was tun beim Lesen von mehreren Quellen?
- Alternative 1: nichtblockierende Ein-, Ausgabe
 - ◆ `O_NDELAY` beim `open()`
 - ◆ Pollingbetrieb: Prozess muss immer wieder `read()` aufrufen, bis etwas vorliegt

- Alternative 2: Blockieren an mehreren Filedeskriptoren
 - ◆ Systemaufruf:


```
int select( int nfds, fd_set *readfds, fd_set *writefds,
            fd_set *errorfds, struct timeval *timeout);
```
 - ◆ `nfds` legt fest, bis zu welchem Filedeskriptor `select` wirken soll.
 - ◆ `xxxfds` sind Filedeskriptoren, auf die gewartet werden soll:
 - `readfds` — bis etwas zum Lesen vorhanden ist
 - `writefds` — bis man schreiben kann
 - `errorfds` — bis ein Fehler aufgetreten ist
 - ◆ Timeout legt fest, wann der Aufruf spätestens deblockiert.
 - ◆ Makros zum Erzeugen der Filedeskriptormengen
 - ◆ Ergebnis: in den Filedeskriptormengen sind nur noch die Filedeskriptoren vorhanden, die zur Deblockade führten

G.4 Bildschirmtreiber

- Bildspeicher
 - ◆ zeichenorientiert
 - ◆ pixelorientiert
- Aufgaben des Treibers
 - ◆ Bereitstellen von Graphikprimitiven (z.B. Ausgabe von Text, Zeichnen von Rechtecken, etc.)
 - ◆ Ansprechen von Graphikprozessoren (schnelle Verschiebeoperationen, komplexe Zeichenoperationen, 3D Rendering, Textures)
 - ◆ Einblenden des Bildspeichers in Anwendungsprogramme (z.B. X11-Server)
- Bildspeicher
 - ◆ spezieller Speicher, der den Bildschirminhalt repräsentiert
 - ◆ Dual ported RAM (Videochip und Prozessor können gleichzeitig zugreifen)

G.5 Netzwerktreiber

- Beispiel: Ethernet
 - ◆ schneller serieller Bus mit CSMA/CD
(*Carrier sense media access / Collision detect*)
zu deutsch: es wird dann gesendet, wenn nicht gerade jemand anderes sendet; Kollisionen werden erkannt und aufgelöst
 - ◆ spezieller Netzwerkchip
 - implementiert unterstes Kommunikationsprotokoll
 - erkennt eintreffende Pakete
- Netzwerktreiber
 - ◆ wird von höheren Protokollen innerhalb des Betriebssystems angesprochen, z.B. von der IP-Schicht

G.5 Netzwerktreiber (2)

■ Senden

- ◆ Treiber übergibt dem Netzwerkchip eine Datenstruktur mit den notwendigen Informationen: Sendeadresse, Adresse und Länge von Datenpuffern
- ◆ Netzwerkchip löst Unterbrechung bei erfolgtem Senden aus

■ Empfangen

- ◆ Treiber übergibt dem Netzwerkchip eine Datenstruktur mit Adressen von freien Arbeitspuffern
- ◆ erkennt der Netzwerkchip ein Paket (für die eigene Adresse), füllt er das Paket in einen freien Puffer
- ◆ der Puffer wird in eine Liste von empfangenen Paketen eingehängt und eine Unterbrechung ausgelöst
- ◆ Treiber kann die empfangenen Pakete aushängen

G.5 Netzwerktreiber (3)

■ Übertragung der Daten erfolgt durch DMA

- ◆ evtl. direkt durch den Netzwerkchip

■ Intelligente und nicht-intelligente Netzwerkhardware

- ◆ intelligente Hardware: kann evtl. auch höhere Protokolle, Filterung etc.
- ◆ nicht-intelligente Hardware: benötigt mehr Unterstützung durch den Treiber (Prozessor)

G.6 Andere Geräte

■ Uhr

- ◆ Hardwareuhren (z.B. DCF 77, GPS Empfänger)
- ◆ Systemuhr fast immer in Software (wird mit Hardwareuhren synchronisiert)
- ◆ UNIX: `getitimer`, `setitimer`
 - vier Intervalltimer pro Prozess: Signal SIGALRM nach Ablauf
 - Ablauf konfigurierbar:
Realzeit, Virtuelle Zeit, Virtuelle Zeit (einschl. Systemzeit des Prozesses)

■ Bandlaufwerk

- ◆ zeichenorientiertes Gerät
- ◆ Spuloperationen durch `d_ioctl` realisiert

G.6 Andere Geräte (2)

■ CD-ROM

- ◆ wird wie Platte behandelt (eigener Treiber)
- ◆ nicht beschreibbar
- ◆ spezielle Treiber für Audio-Tracks möglich

■ Maus und Tastatur

- ◆ meist über serielle Schnittstellen und bestimmtes Protokoll implementiert

■ Floppy-Disk

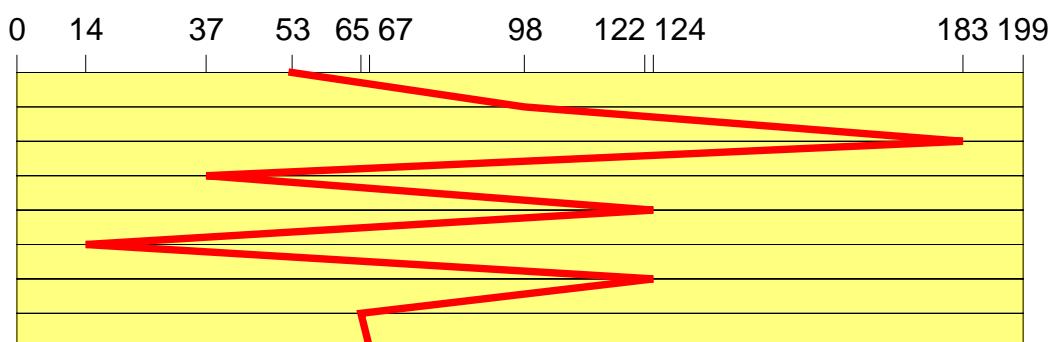
- ◆ wird im Prinzip wie Platte behandelt (eigener Treiber)
- ◆ spezielle Dateisysteme zur Realisierung von FAT Dateisystemen unter UNIX

G.7 Disk-Scheduling

- Plattentreiber hat in der Regel mehrere Aufträge in seiner Warteschlange
 - ◆ Warteschlange wird z.B. in UNIX durch Aufruf der Funktion `d_strategy()` gefüllt
 - ◆ eine bestimmte Ordnung der Ausführung kann Effizienz steigern
 - ◆ Zusammensetzung der Bearbeitungszeit eines Auftrags:
 - Positionierzeit: abhängig von der aktuellen Stellung des Plattenarms
 - Latenzzzeit: Zeit bis der Magnetkopf den Sektor bestreicht
 - Übertragungszeit: Zeit zur Übertragung der eigentlichen Daten
- ★ Ansatzpunkt: Positionierzeit

1 FCFS-Scheduling

- Bearbeitung gemäß Ankunft des Auftrags
 - ◆ Referenzfolge (Folge von Zylindernummern):
98, 183, 37, 122, 14, 124, 65, 67
 - ◆ Aktueller Zylinder: 53



- ◆ Gesamtzahl der Spurwechsel: 640
- ◆ Weite Bewegungen des Schwenkarms: mittlere Bearbeitungsdauer lang

2 SSTF-Scheduling

- Es wird der Auftrag mit der kürzesten Positionierzeit vorgezogen (*Shortest Seek Time First*)
 - ◆ Gleiche Referenzfolge
(Annahme: Positionierzeit proportional zum Zylinderabstand)



- ◆ Gesamtzahl von Spurwechseln: 236
- ◆ ähnlich wie SJF kann auch SSTF zur Aushungerung führen
- ◆ noch nicht optimal

3 SCAN-Scheduling

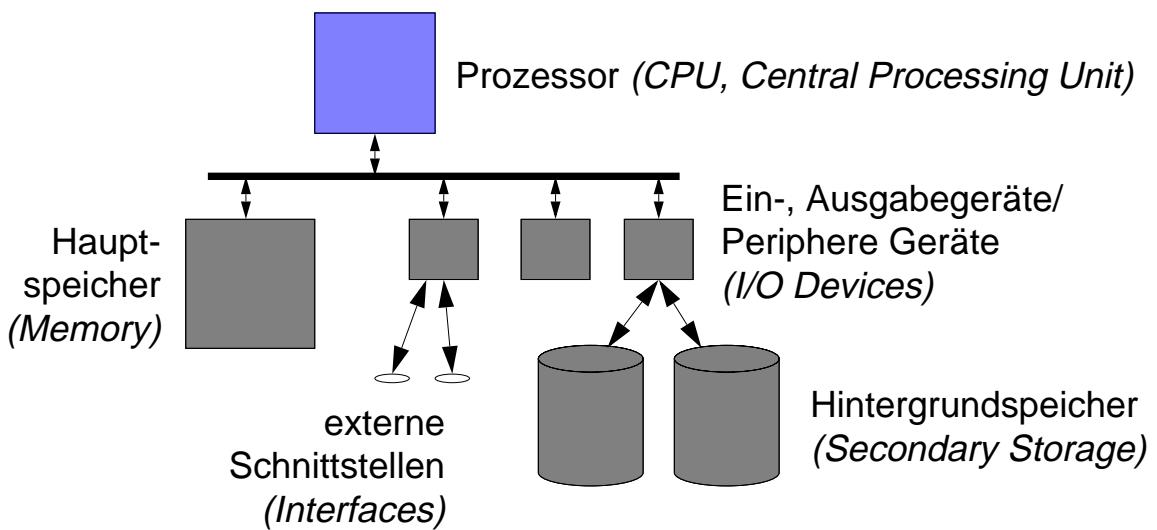
- Bewegung des Plattenarm in eine Richtung bis keine Aufträge mehr vorhanden sind (Fahrstuhlstrategie)
 - ◆ Gleiche Referenzfolge (Annahme: bisherige Kopfbewegung Richtung 0)



- ◆ Gesamtzahl der Spurwechsel: 208
- ◆ Neue Aufträge werden miterledigt ohne zusätzliche Positionierzeit und ohne mögliche Aushungerung
- ◆ Variante C-SCAN (*Circular SCAN*): Bewegung nur in eine Richtung

H Verklemmungen

■ Einordnung:

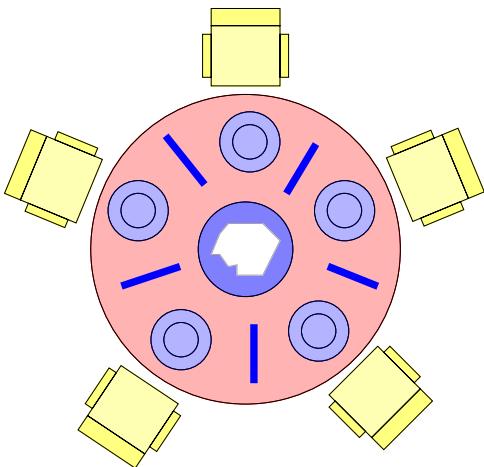


- ◆ Verhalten von Aktivitätsträgern / Prozessen

H.1

Motivation

■ Beispiel: die fünf Philosophen am runden Tisch



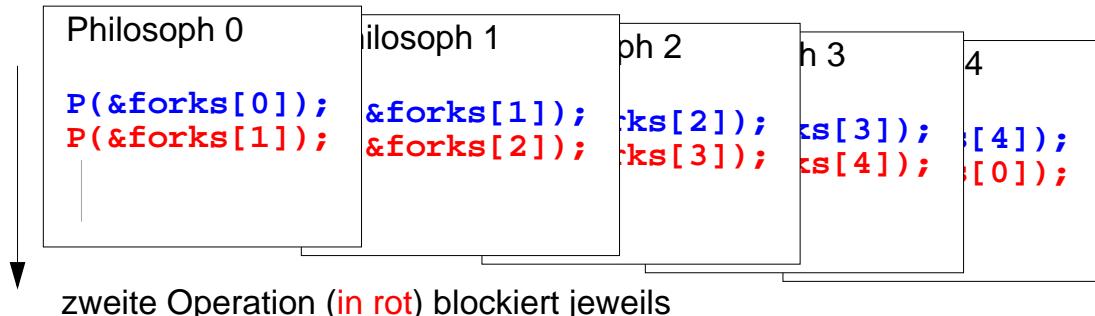
- ◆ Philosophen denken oder essen
"The life of a philosopher consists of an alternation of thinking and eating." (Dijkstra, 1971)
- ◆ zum Essen benötigen sie zwei Gabeln, die jeweils zwischen zwei benachbarten Philosophen abgelegt sind

■ Philosophen können verhungern, wenn sie sich „dumm“ anstellen.

H.1 Motivation (2)

■ Problem der Verklemmung (*Deadlock*)

- ◆ alle Philosophen nehmen gleichzeitig die linke Gabel auf und versuchen dann die rechte Gabel aufzunehmen



- ◆ System ist **verklemmt**: Philosophen warten alle auf ihre Nachbarn

■ Problemkreise:

- ◆ Vermeidung und Verhinderung von Verklemmungen
- ◆ Erkennung und Erholung von Verklemmungen

H.2 Betriebsmittelbelegung

■ Betriebsmittel

- ◆ CPU, Drucker, Geräte (Platten, CD-ROM, Floppy, Audio, usw.)
- ◆ nur elektronisch vorhandene Betriebsmittel der Anwendung oder des Betriebssystems, z.B. Gabeln der Philosophen

■ Unterscheidung von Typ und Instanz

- ◆ Typ definiert ein Betriebsmittel eindeutig
- ◆ Instanz ist eine Ausprägung des Typs
(die Anwendung benötigt eine Instanz eines best. Typs, egal welche)
 - **CPU**: Anwendung benötigt eine von mehreren gleichartigen CPUs
 - **Drucker**: Anwendung benötigt einen von mehreren gleichen Druckern
(falls Drucker nicht austauschbar und gleichwertig, so handelt es sich um verschiedene Typen)
 - **Gabeln**: jede Gabel ist ein eigener Betriebsmitteltyp

1 Belegung

■ Belegung erfolgt in drei Schritten

◆ Anfordern des Betriebsmittels

- blockiert evtl. falls Betriebsmittel nur exklusiv benutzt werden kann
- **Gabel:** nur exklusiv
- **Bildschirmausgabe:** exklusiv oder nicht-exklusiv

◆ Nutzen des Betriebsmittels

- **Gabel:** Philosoph kann essen
- **Drucker:** Anwendung kann drucken

◆ Freigeben des Betriebsmittels

- **Gabel:** Philosoph legt Gabel wieder zwischen die Teller

2 Voraussetzungen für Verklemmungen

■ Vier notwendige Bedingungen

◆ *Exklusive Belegung*

Mindestens ein Betriebsmitteltyp muss nur exklusiv belegbar sein.

◆ *Nachforderungen von Betriebsmittel möglich*

Es muss einen Prozess geben, der bereits Betriebsmittel hält, und ein neues Betriebsmittel anfordert.

◆ *Kein Entzug von Betriebsmitteln möglich*

Betriebsmittel können nicht zurückgefordert werden bis der Prozess sie wieder freigibt.

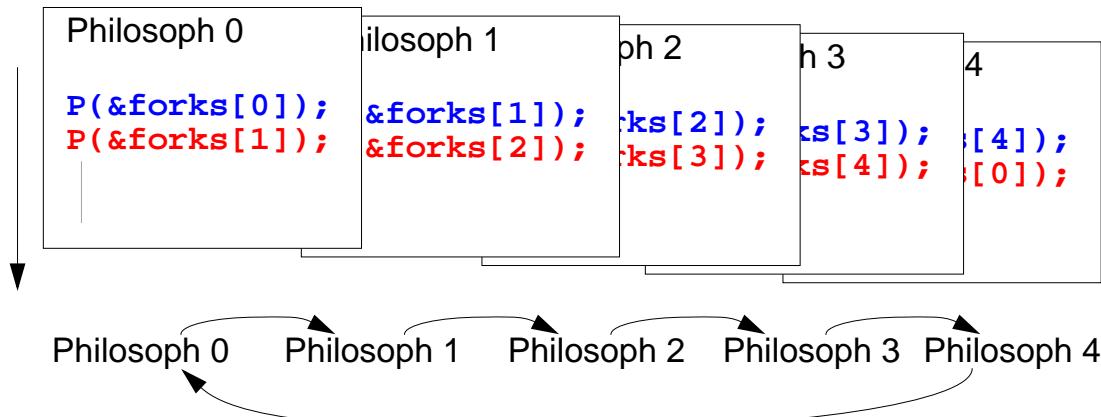
◆ *Zirkuläres Warten*

Es gibt einen Ring von Prozessen, in dem jeder auf ein Betriebsmittel wartet, das der Nachfolger im Ring besitzt.

2 Voraussetzungen für Verklemmung (2)

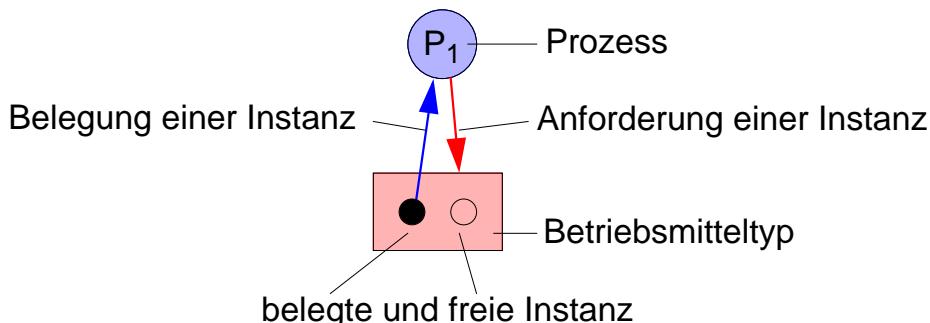
■ Beispiel: fünf Philosophen

- ◆ Exklusive Belegung: **ja**
- ◆ Nachforderungen von Betriebsmittel möglich: **ja**
- ◆ Entzug von Betriebsmitteln: **nicht vorgesehen**
- ◆ Zirkuläres Warten: **ja**



3 Betriebsmittelgraphen

■ Veranschaulichung der Belegung und Anforderung durch Graphen (nur exklusive Belegungen)

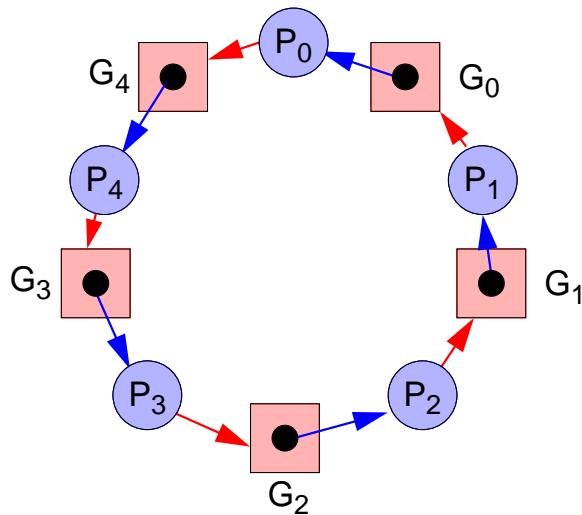


■ Regeln:

- ◆ kein Zyklus im Graph \rightarrow keine Verklemmung
- ◆ Zyklus im Graph \rightarrow Verklemmung
- ◆ nur jeweils eine Instanz pro Betriebsmitteltyp und Zyklus \rightarrow **Verklemmung**

3 Betriebsmittelgraphen (2)

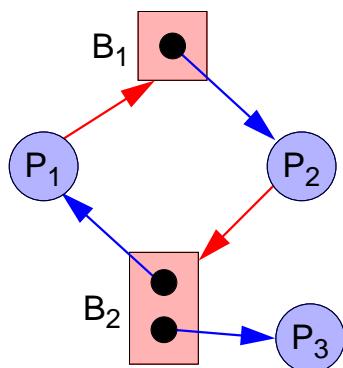
■ Beispiel: fünf Philosophen



◆ Zyklus und jeder Betriebsmitteltyp hat nur eine Instanz → **Verklemmung**

3 Betriebsmittelgraphen (3)

■ Beispiel mit Zyklus und ohne Verklemmung



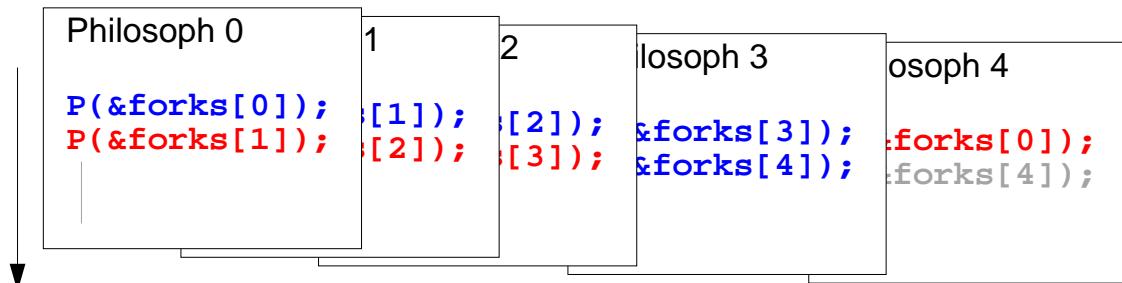
◆ Prozess 3 kann seine Instanz vom Betriebsmitteltyp B₂ wieder zurückgeben und den Zyklus damit auflösen

H.3 Vermeidung von Verklemmungen

- Ansatz: Vermeidung der notwendigen Bedingungen für Verklemmungen
 - ◆ *Exklusive Belegung*: oft nicht vermeidbar
 - ◆ *Nachforderungen von Betriebsmittel möglich*:
 - alle Betriebsmittel müssen auf einmal angefordert werden
 - ungenutzte aber belegte Betriebsmittel vorhanden
 - Aushungerung möglich: ein anderer Prozess hält immer das nötige Betriebsmittel belegt
 - ◆ *Kein Entzug von Betriebsmitteln möglich*:
 - Entzug von Betriebsmitteln erlauben
 - bei neuer Belegung werden alle gehaltenen Betriebsmittel freigegeben und mit der neuen Anforderung zusammen wieder angefordert
 - während ein Prozess wartet, werden seine bereits belegten Betriebsmittel anderen Prozessen zur Verfügung gestellt
 - möglich für CPU oder Speicher jedoch nicht für Drucker, Bandlaufwerke oder ähnliche

H.3 Vermeidung von Verklemmungen (2)

- ◆ *Zirkuläres Warten*: Vermeidung von Zyklen
 - Totale Ordnung auf Betriebmitteltypen
 - Anforderungen nur in der Ordnungsreihenfolge erlaubt

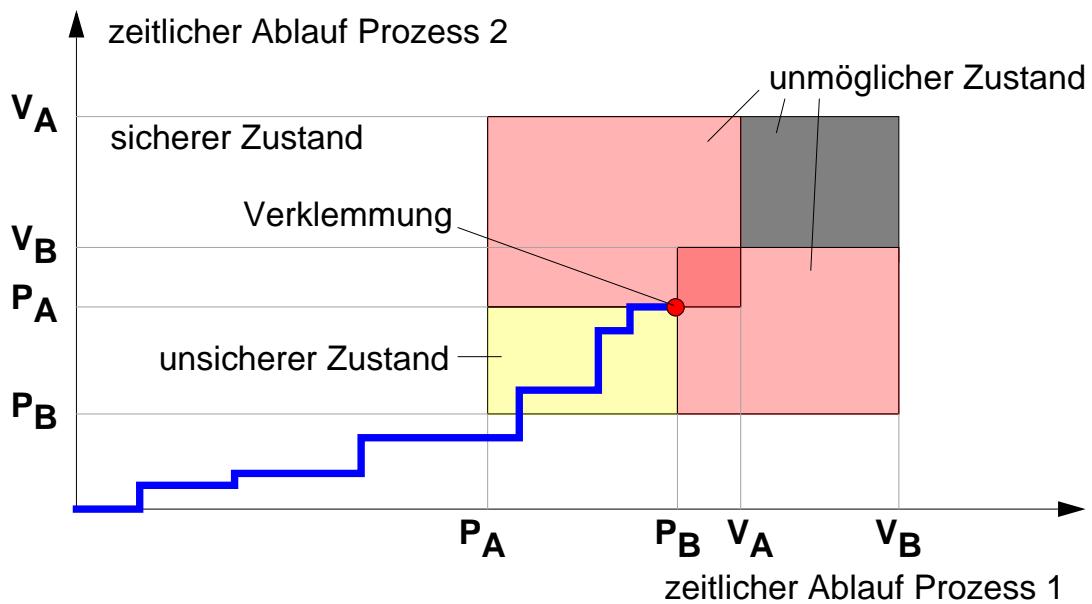


z.B. Gabeln: geordnet nach Gabelnummer

- Bei neuer Anforderung wird geprüft, ob letzte Anforderung kleiner bzgl. der totalen Ordnung war (Instanzen gleichen Typs müssen gleichzeitig angefordert werden); sonst: Abbruch mit Fehlermeldung
- Philosoph 4 bekäme eine Fehlermeldung, wenn er in der obigen Situation zuerst Gabel 4 und dann Gabel 0 anfordert: Rückgabe und neuer Versuch

H.4 Verhinderung von Verklemmungen

- Annahme: es ist bekannt, welche Betriebsmittel ein Prozess brauchen wird (hier je zwei binäre Semaphore A und B)
 - ♦ Betriebssystem überprüft System auf unsichere Zustände



1 Sichere und unsichere Zustände

- Sicherer Zustand
 - ♦ Es gibt eine Sequenz, in der die vorhandenen Prozesse abgearbeitet werden können, so dass ihre Anforderungen immer befriedigt werden können.
 - ♦ Sicherer Zustand erlaubt immer eine verklemmungsfreie Abarbeitung
- Unsicherer Zustand
 - ♦ Es gibt keine solche Sequenz.
 - ♦ Verklemmungszustand ist ein unsicherer Zustand
 - ♦ Ein unsicherer Zustand führt zwangsläufig zur Verklemmung, wenn die Prozesse ihre angenommenen Betriebsmittel wirklich anfordern bevor sie von anderen Prozessen wieder freigegeben werden.

1 Sichere und unsichere Zustände (2)

■ Beispiel:

- ◆ 12 Magnetbandlaufwerke vorhanden
- ◆ P_0 braucht (bis zu) 10 Laufwerke
- ◆ P_1 braucht (bis zu) 4 Laufwerke
- ◆ P_2 braucht (bis zu) 9 Laufwerke

- ◆ Aktuelle Situation: P_0 hat 5, P_1 hat 2 und P_2 hat 2 Laufwerke
- ◆ Zustand sicher?

- ◆ Aktuelle Situation: P_0 hat 5, P_1 hat 2 und P_2 hat 3 Laufwerke
- ◆ Zustand sicher?

1 Sichere und unsichere Zustände (3)

■ Verhinderung von Verklemmungen

- ◆ Verhinderung von unsicheren Zuständen
- ◆ Anforderungen blockieren, falls sie in einen unsicheren Zustand führen würden

■ Beispiel von Folie H.15:

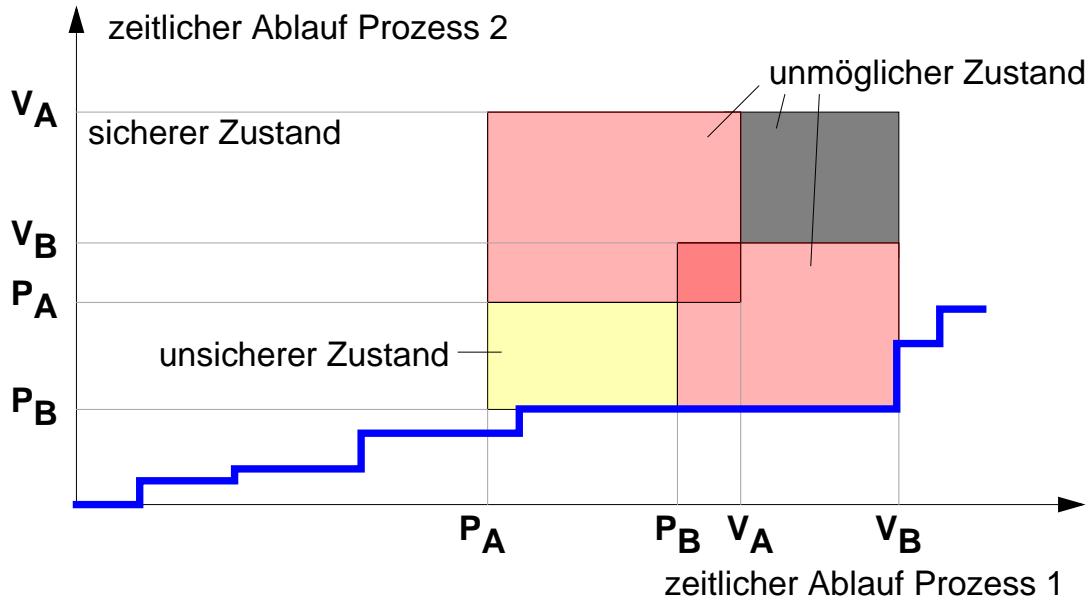
- ◆ Zustand: P_0 hat 5, P_1 hat 2 und P_2 hat 2 Laufwerke
- ◆ P_2 fordert ein zusätzliches Laufwerk an
- ◆ Belegung würde in unsicheren Zustand führen: P_2 muss warten

▲ Verhinderung von unsicheren Zuständen schränkt Nutzung von Betriebsmitteln ein

- ◆ verhindert aber Verklemmungen

1 Sichere und unsichere Zustände (4)

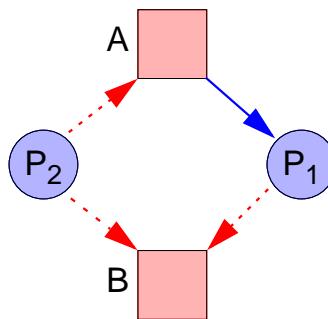
■ Beispiel von Folie H.13:



- ◆ Prozess 2 darf P_B nicht durchführen und muss warten

2 Betriebsmittelgraph

- Annahme: eine Instanz pro Betriebsmitteltyp
 - ◆ Einsatz von Betriebsmittelgraphen zur Erkennung unsicherer Zustände

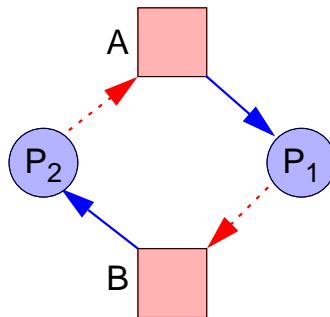


- ◆ zusätzliche Kanten zur Darstellung möglicher Anforderungen (Ansprüche, *Claims*)
- ◆ Anspruchskanten werden gestrichelt dargestellt und bei Anforderung in Anforderungskanten umgewandelt
- ◆ Anforderung und Belegung von B durch P_2 führt in einen unsicheren Zustand (siehe Beispiel von Folie H.13)

2 Betriebsmittelgraph (2)

- Erkennung des unsicheren Zustands an Zyklen im erweiterten Betriebsmittelgraph

- ◆ Anforderung und Belegung von B durch P_2 führt zu:



- ◆ Zyklenerkennung hat einen Aufwand von $O(n^2)$

- ▲ Betriebsmittelgraph nicht anwendbar bei mehreren Instanzen eines Betriebsmitteltyps
 - ◆ Banker's Algorithmus (siehe Betriebsprogrammierung II)

H.5 Erkennung von Verklemmungen

- Systeme ohne Mechanismen zur Vermeidung oder Verhinderung von Verklemmungen
 - ◆ Verklemmungen können auftreten
 - ◆ Verklemmung sollte als solche erkannt werden
 - ◆ Auflösung der Verklemmung sollte eingeleitet werden (Algorithmus nötig)

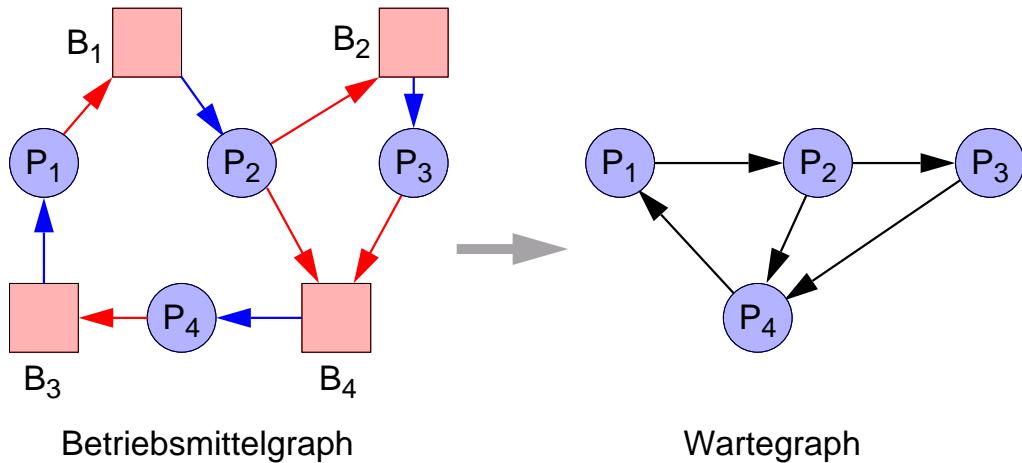
1 Wartegraphen

- Annahme: nur eine Instanz pro Betriebsmitteltyp
 - ◆ Einsatz von Wartegraphen, die aus dem Betriebsmittelgraphen gewonnen werden können

1 Wartegraphen (2)

■ Wartegraphen

- ◆ Betriebsmittel und Kanten werden aus Betriebsmittelgraph entfernt
- ◆ zwischen zwei Prozessen wird eine „wartet auf“-Kante eingeführt, wenn es Kanten vom ersten Prozess zu einem Betriebsmittel und von diesem zum zweiten Prozess gäbe



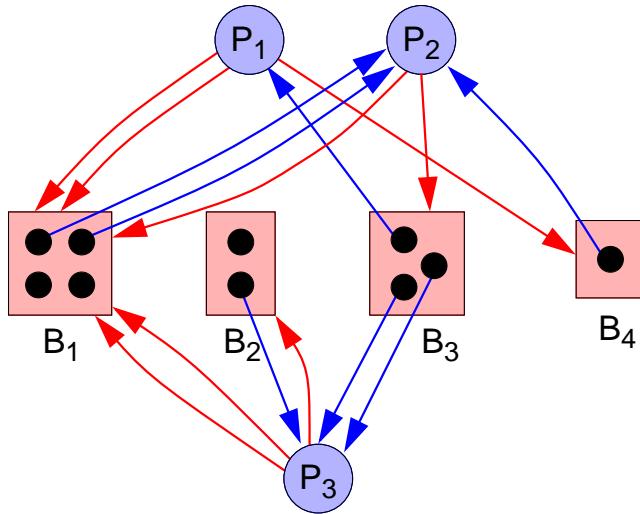
1 Wartegraphen (3)

■ Erkennung von Verklemmungen

- ◆ Wartegraph enthält Zyklen: System ist verklemmt
- ▲ Betriebsmittelgraph nicht für Systeme geeignet, die mehrere Instanzen pro Betriebsmitteltyp zulassen
 - ◆ kompliziertere Algorithmen ähnlich dem Banker's Algorithmus nötig

2 Erkennung durch graphische Reduktion

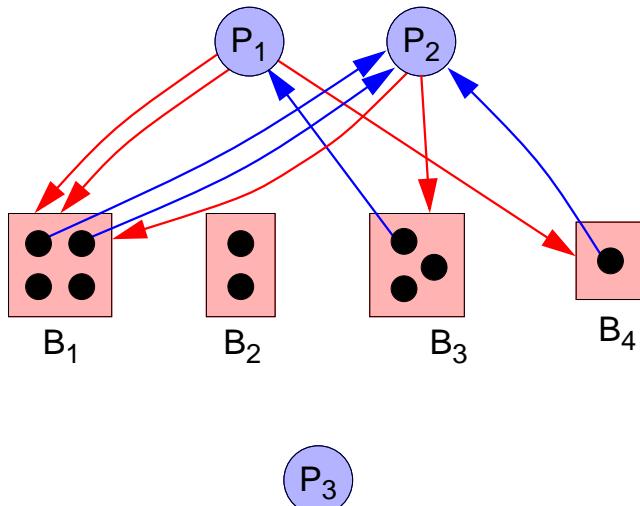
■ Betriebsmittelgraph des Beispiels



- ◆ Auswahl eines Prozesses für den Anforderungen erfüllbar: nur P_3 möglich
- ◆ Löschen aller Kanten des Prozesses

2 Erkennung durch graphische Reduktion (2)

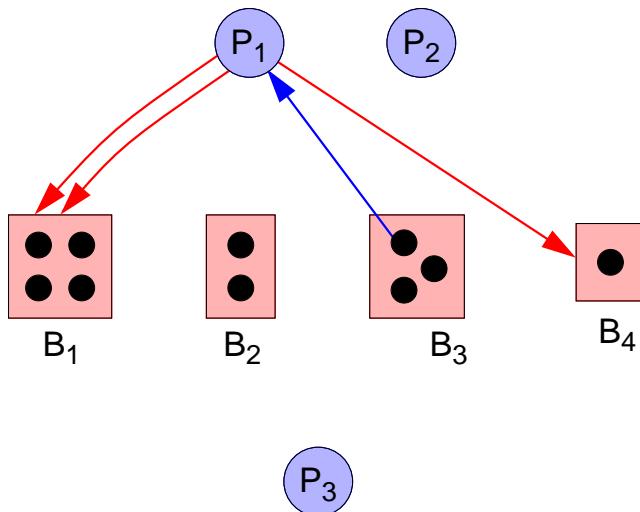
■ Betriebsmittelgraph des Beispiels (1. Reduktion)



- ◆ Auswahl eines Prozesses für den Anforderungen erfüllbar: nur P_2 möglich
- ◆ Löschen aller Kanten des Prozesses

2 Erkennung durch graphische Reduktion (3)

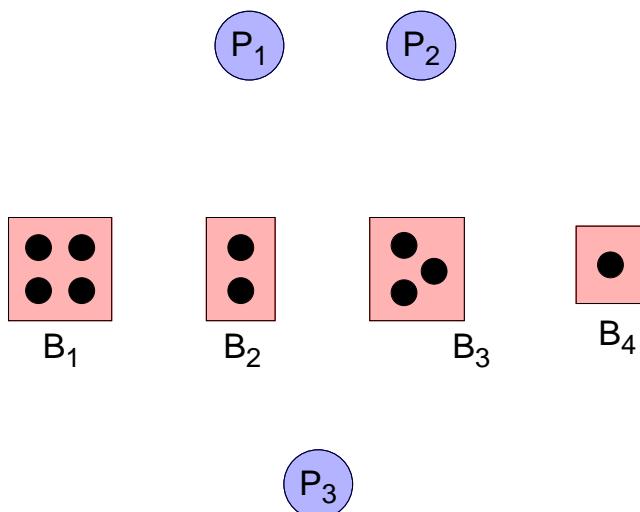
■ Betriebsmittelgraph des Beispiels (2. Reduktion)



- ◆ Auswahl eines Prozesses für den Anforderungen erfüllbar: P_1
- ◆ Löschen aller Kanten des Prozesses

2 Erkennung durch graphische Reduktion (4)

■ Betriebsmittelgraph des Beispiels (3. Reduktion)



- ◆ es bleiben keine Prozesse mit Anforderungen übrig \rightarrow keine Verklemmung
- ◆ übrig bleibende Prozesse sind verklemmt und in einem Zyklus

3 Erkennung durch Reduktionsverfahren

■ Annahmen:

- ◆ m Betriebsmitteltypen; Typ i verfügt über b_i Instanzen
- ◆ n Prozesse

■ Definitionen

- ◆ B ist der Vektor (b_1, b_2, \dots, b_m) der vorhandenen Instanzen
- ◆ R ist der Vektor (r_1, r_2, \dots, r_m) der noch verfügbaren Restinstanzen
- ◆ C_j sind die Vektoren $(c_{j,1}, c_{j,2}, \dots, c_{j,m})$ der aktuellen Belegung durch den Prozess j

■ Es gilt: $\sum_{j=1}^n c_{j,i} + r_i = b_i$ für alle $1 \leq i \leq m$

3 Erkennung durch Reduktionsverfahren (2)

■ Weitere Definitionen

- ◆ A_j sind die Vektoren $(a_{j,1}, a_{j,2}, \dots, a_{j,m})$ der aktuellen Anforderungen durch den Prozess j
- ◆ zwei Vektoren A und B stehen in der Relation $A \leq B$, falls die Elemente der Vektoren jeweils paarweise in der gleichen Relation stehen

■ Algorithmus

- (1) alle Prozesse sind zunächst unmarkiert
 - (2) wähle einen Prozess j , so dass $A_j \leq R$
(Prozess ist ohne Verklemmung ausführbar)
 - (3) falls ein solcher Prozess j existiert, addiere C_j zu R , markiere Prozess j und beginne wieder bei Punkt (2)
(Bei Terminierung wird der Prozess alle Betriebsmittel freigeben)
 - (4) falls ein solcher Prozess nicht existiert, terminiere Algorithmus
- ◆ alle nicht markierten Prozesse sind an einer Verklemmung beteiligt

3 Erkennung durch Reduktionsverfahren (3)

■ Beispiel

- ◆ $m = 4; B = (4, 2, 3, 1)$
- ◆ $n = 3; C_1 = (0, 0, 1, 0); C_2 = (2, 0, 0, 1); C_3 = (0, 1, 2, 0)$
- ◆ daraus ergibt sich $R = (2, 1, 0, 0)$
- ◆ Anforderungen der Prozesse lauten:
 $A_1 = (2, 0, 0, 1); A_2 = (1, 0, 1, 0); A_3 = (2, 1, 0, 0)$

■ Ablauf

- ◆ Auswahl eines Prozesses: Prozess 3, da $A_3 \leq R$; markiere Prozess 3
- ◆ Addiere C_3 zu R : neues $R = (2, 2, 2, 0)$
- ◆ Auswahl eines Prozesses: Prozess 2, da $A_2 \leq R$; markiere Prozess 2
- ◆ Addiere C_2 zu R : neues $R = (4, 2, 2, 1)$
- ◆ Auswahl eines Prozesses: Prozess 1, da $A_1 \leq R$; markiere Prozess 1
- ◆ kein Prozess mehr unmarkiert: keine Verklemmung

4 Einsatz der Verklemmungserkennung

■ Wann sollte Erkennung ablaufen?

- ◆ Erkennung ist aufwendig (Aufwand $O(n^2)$ bei Zyklenerkennung)
- ◆ Häufigkeit von Verklemmungen eher gering
- ◆ zu häufig: Verschwendung von Ressourcen zur Erkennung
- ◆ zu selten: Betriebsmittel werden nicht optimal genutzt, Anzahl der verklemmten Prozesse steigt

■ Möglichkeiten:

- ◆ Erkennung, falls eine Anforderung nicht sofort erfüllt werden kann
- ◆ periodische Erkennung (z.B. einmal die Stunde)
- ◆ CPU Auslastung beobachten; falls Auslastung sinkt, Erkennung starten

5 Erholung von Verklemmungen

- Verklemmung erkannt: Was tun?
 - ◆ Operateur benachrichtigen; manuelle Beseitigung
 - ◆ System erholt sich selbst
- Abbrechen von Prozessen (terminierte Prozesse geben ihre Betriebsmittel wieder frei)
 - ◆ alle verklemmten Prozesse abbrechen
(großer Schaden)
 - ◆ einen Prozess nach dem anderen abbrechen bis Verklemmung behoben
(kleiner Schaden aber rechenzeitintensiv)
 - ◆ mögliche Schäden:
 - Verlust von berechneter Information
 - Dateninkonsistenzen

5 Erholung von Verklemmungen (2)

- Entzug von Betriebsmitteln
 - ◆ Aussuchen eines „Opfer“-Prozesses
(Aussuchen nach geringstem entstehendem Schaden)
 - ◆ Entzug der Betriebsmittel und Zurückfahren des „Opfer“-Prozesses
(Prozess wird in einen Zustand zurückgefahren, der unkritisch ist; benötigt Checkpoint oder Transaktionsverarbeitung)
 - ◆ Verhinderung von Aushungerung
(es muss verhindert werden, dass immer derselbe Prozess Opfer wird und damit keinen Fortschritt mehr macht)

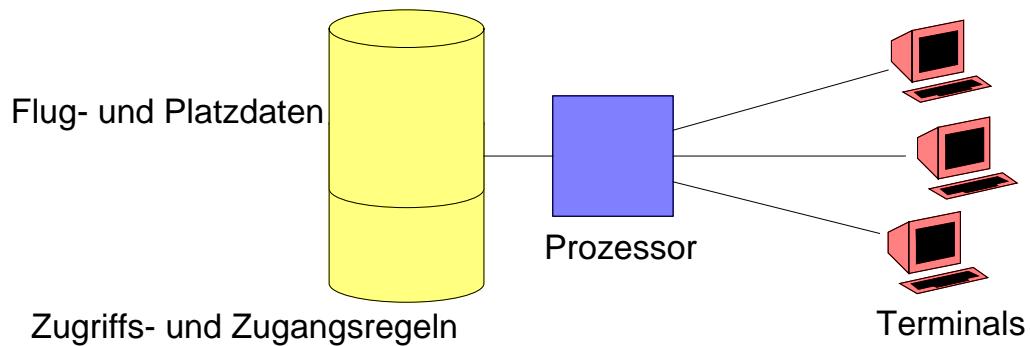
H.6 Kombination der Verfahren

- Einsatz verschiedener Verfahren für verschiedene Betriebsmittel
 - ◆ Interne Betriebsmittel:
Verhindern von Verklemmungen durch totale Ordnung der Betriebsmittel (z.B. IBM Mainframe-Systeme)
 - ◆ Hauptspeicher:
Verhindern von Verklemmungen durch Entzug des Speichers (z.B. durch Swap-Out)
 - ◆ Betriebsmittel eines Jobs:
Angabe der benötigten Betriebsmittel beim Starten; Einsatz der Vermeidungsstrategie durch Feststellen unsicherer Zustände
 - ◆ Hintergrundspeicher (Swap-Space):
Vorausbelegung des Hintergrundspeichers

I Datensicherheit und Zugriffsschutz

I.1 Problemstellung

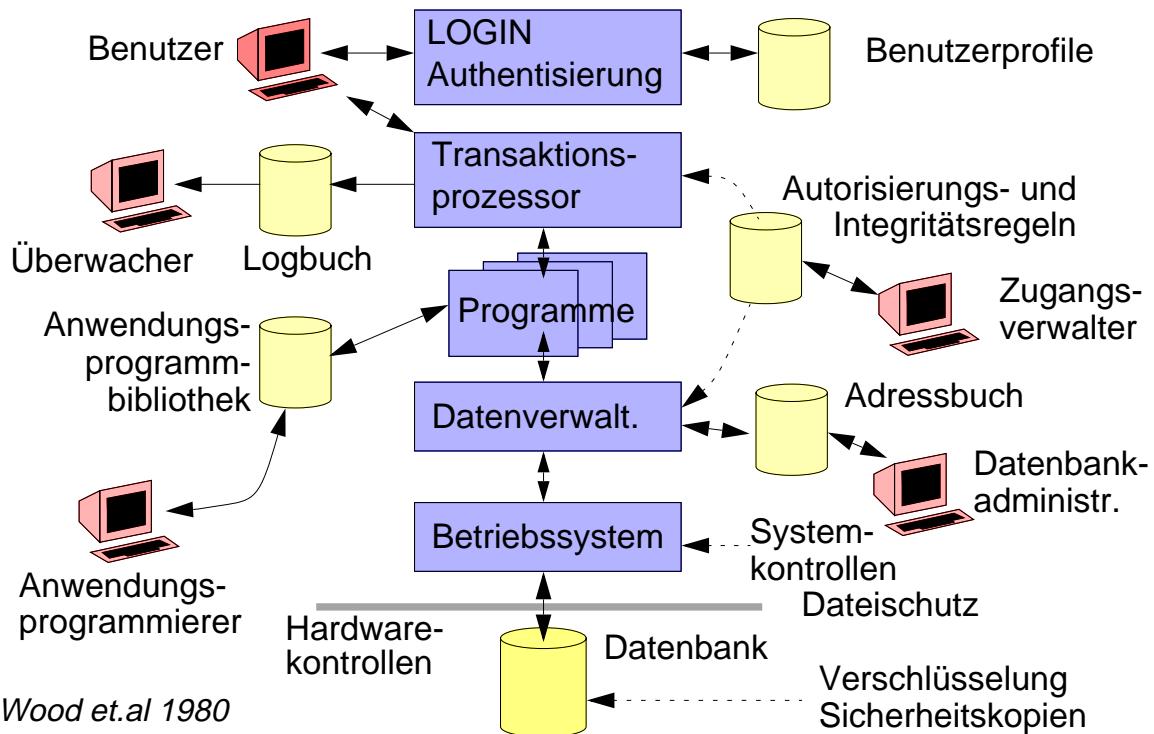
- Beispiel: Zugang zu einer Datenbank zur Flugreservierung und -buchung



- ◆ Was sind mögliche Beeinträchtigungen der Datensicherheit?

I.1 Problemstellung (2)

- ## ■ Überprüfungen beim Transaktionsbetrieb (Datenbankanwendung)



Nach Wood et.al 1980

I.1 Problemstellung (3)

- Illegaler Datenzugriff
 - ◆ Daten sind zugreifbar, die vertraulich behandelt werden sollen
 - Illegales Löschen von Daten
 - ◆ Kein Zugriff, aber Daten werden gelöscht
 - Illegales Manipulieren von Daten
 - ◆ Daten werden in böswilliger Absicht verändert
 - Zerstörung von Rechensystemen
 - ◆ physisches Zerstören von Teilen der Rechenanlage

1 Umgebung der Rechenanlage

▲ Naturkatastrophen

- ◆ Erdbeben, Vulkanausbrüche etc. können Rechenanlage und Datenbestand zerstören

▲ Unfälle

- ◆ Gasexplosion, Kühlwasserlecks in der Klimaanlage oder Ähnliches zerstören Rechner und Daten

▲ Böswillige Angriffe

- ◆ Zerstörung der Rechenanlage und des Datenbestands durch Sabotage (Bombenanschlag, Brandanschlag etc.)

▲ Unbefugter Zutritt zu den Räumen des Rechenzentrums

- ◆ Diebstahl von Datenträgern
- ◆ Zerstörung von Daten
- ◆ Zugang zu vertraulichen Daten

2 Systemsoftware

▲ Versagen der Schutzmechanismen

- ◆ System lässt Unbefugte auf Daten zugreifen oder Operationen ausführen

▲ Durchsickern von Informationen

- ◆ Anwender können anhand scheinbar unauffälligen Systemverhaltens Rückschlüsse auf vertrauliche Daten ziehen (*Covert channels*)

■ Beispiel: verschlüsselt abgespeicherte Passwörter sind zugänglich

- ◆ Entschlüsselungsversuch der Passwörter außerhalb der Rechenanlage
- ◆ "Wörterbuchattacke": Raten von Passwörtern möglich

3 Systemprogrammierer

- ▲ Umgehen oder Abschalten der Schutzmechanismen
- ▲ Installation eines unsicheren Systems
 - ◆ erlaubt dem Systemprogrammierer die Schutzmechanismen von außen zu umgehen
- ▲ Fehler beim Nutzen von Bibliotheksfunktionen innerhalb sicherheitskritischer Programme
 - ◆ S-Bit Programme unter UNIX laufen mit der Benutzerkennung des Dateibesitzers, nicht unter der des Aufrufers
 - Fehlerhafte S-Bit Programme können zur Ausführung von Code unter einer fremden Benutzerkennung gebracht werden
 - S-Bit Programme mit "root-Rechten" besonders gefährlich

3 Systemprogrammierer (2)

- ◆ Buffer-Overflow Fehler bei Funktionen
 - `gets()`, `strcpy()`, `strcat()`, `sprintf("%s", ...)`
 - Zu lange Eingaben überschreiben den Stackspeicher des Prozessors
 - Mit genauer Kenntnis ist das Ausführen beliebigen Codes erzwingbar
- ◆ Fehlerhafte Parameterprüfung beim Aufruf von Funktionen
 - `system()`
- ◆ Beispiel: Löschen einer Datei
 - Name der Datei wurde in Variable `file` eingelesen
 - Aufruf von `system` mit Parameter `strcat("rm ", file)`
 - Gibt man für den Dateinamen den String
 - `"fn ; xterm -display myhost:0 &"`ein, bekommt man ein Fenster auf der aufgebrochenen Maschine

4 Rechnerhardware

- ▲ Versagen der Schutzmechanismen
 - ◆ erlauben nicht-autorisierten Zugriff
- ▲ Fehlerhaft Befehlsausführung
 - ◆ Zerstörung von wichtigen Daten
- ▲ Abstrahlungen
 - ◆ erlaubt Ausspähen von Daten

5 Datenbasis

- ▲ Falsche Zugriffsregeln
 - ◆ erlauben nicht-autorisierten Zugriff

6 Operateur

- ▲ Kopieren vertraulicher Datenträger
- ▲ Diebstahl von Datenträgern
- ▲ Initialisierung mit unsicherem Zustand
 - ◆ Operateur schaltet beispielsweise Zugriffskontrolle ab
- ▲ Nachlässige Rechnerwartung
 - ◆ Nachbesserungen der Systemsoftware (*Patches*) werden nicht eingespielt
 - Sicherheitslücken werden nicht gestopft

7 Sicherheitsbeauftragter

- ▲ Fehlerhafte Spezifikation der Sicherheitspolitik
 - ◆ dadurch Zugang für Unbefugte zu vertraulichen Daten oder
 - ◆ Änderungen von Daten durch Unbefugte möglich
- ▲ Unterlassene Auswertung von Protokolldateien
 - ◆ Einbrüche und mögliche Sicherheitslücken werden nicht rechtzeitig entdeckt

8 Kommunikationssystem

- ▲ Abhören der Kommunikationsleitungen (*Sniffing*)
 - ◆ z.B. Telefonverbindung bei Modemnutzung oder serielle Schnittstellen
 - ◆ z.B. Netzwerkverkehr auf einem Netzwerkstrang
- ▲ Ermitteln von Passwörtern und Benutzerkennungen
 - manche Dienste übertragen Passwörter im Klartext (z.B. ftp, telnet, rlogin)
- ▲ Zugriff auf vertrauliche Daten
- ▲ unbefugte Datenveränderungen
 - Verfälschen von Daten
 - Übernehmen von bestehenden Verbindungen (*Hijacking*)
 - Vorspiegeln falscher Netzwerkadressen (*Spoofing*)

8 Kommunikationssystem (2)

- ▲ Illegale Nutzung von Diensten über das Netzwerk
 - ◆ Standardsysteme bieten eine Menge von Diensten an (z.B. ftp, telnet, rwho u.a.)
 - ◆ Sicherheitslücken von Diensten werden publik gemacht und sind auch von "dummen" Hackern nutzbar (*Exploit scripts*)
<http://www.rootshell.org>
 - ◆ Auch bei temporär am Netzwerk angeschlossenen Computern eine Gefahr
 - z.B. Linux-Maschine mit PPP-Verbindung an das Uni-Netz
 - Voreinstellungen der Standardinstallation meist unsicher

9 Terminal

- ▲ Ungeschützter Zugang zum Terminal
 - ◆ Nutzen einer fremden Benutzerkennung
 - ◆ Zugriff auf vertrauliche Daten
 - ◆ unbefugte Datenveränderungen

10 Benutzer

- ▲ Nutzen anderer Kennungen
 - ◆ erlauben nicht-autorisierten Zugriff
 - ◆ unbefugte Datenveränderungen
 - ◆ unbefugte Weitergabe von Informationen
- ▲ Einbruch von Innen
 - ◆ Benutzer hat leichter Zugang zu möglichen Sicherheitslücken (z.B. bei Diensten)

11 Anwendungsprogrammierung

- ▲ Nichteinhalten der Spezifikation
 - ◆ Umgehen der Zugriffskontrollen
- ▲ Einfügen von „bösertigen“ Befehlsfolgen
 - ◆ *Back door*: Hintertür gibt dem Programmierer im Betrieb Zugang zu vertraulichen Daten oder illegalen Operationen
 - ◆ *Trojan horse*: Unter bestimmten Bedingungen werden illegale Operationen ohne Trigger von außen angestoßen

12 "Tracker Queries"

- Beispiel: Datenbanksysteme
 - ◆ Zugriff auf Einzelinformationen ist verboten (Vertraulichkeit)
 - ◆ statistische Informationen sind erlaubt
- ▲ Grenzen möglicher Sicherheitsmaßnahmen:
Zugriff auf Einzelinformationen dennoch möglich
 - ◆ geeignete Anfragen kombinieren (*Tracker queries*)
- Beispiel: Gehaltsdatenbank

■ Tabelle der Datenbankeinträge:

Nr.	Name	Geschl.	Fach	Stellung	Gehalt	Spenden
1	Albrecht	m	Inf.	Prof.	60.000	150
2	Bergner	m	Math.	Prof.	45.000	300
3	Cäsar	w	Math.	Prof.	75.000	600
4	David	w	Inf.	Prof.	45.000	150
4	Engel	m	Stat.	Prof.	54.000	0
5	Frech	w	Stat.	Prof.	66.000	450
6	Groß	m	Inf.	Angest.	30.000	60
8	Hausner	m	Math.	Prof.	54.000	1500
9	Ibel	w	Inf.	Stud.	9.000	30
10	Jost	m	Stat.	Angest.	60.000	45
11	Knapp	w	Math.	Prof.	75.000	300
12	Ludwig	m	Inf.	Stud.	9.000	0

■ Anfragen und Antworten:

- ◆ Anzahl('w'): 5
- ◆ Anzahl('w' und (nicht 'Inf' oder nicht 'Prof.')): 4
- ◆ mittlere Spende('w'): 306
- ◆ mittlere Spende('w' und (nicht 'Inf.' oder nicht 'Prof.')): 345

■ Berechnung:

- ◆ Spende('David'): 306 * 5 - 345 * 4 =
1530 - 1380 =
150

I.2 Zugriffslisten

- Identifikation von Subjekten, Objekten und Berechtigungen
 - ◆ Subjekt: Person oder Benutzerkennung im System
(repräsentiert jemanden, der Aktionen ausführen kann)
 - ◆ Objekt: Komponente des Systems
(repräsentiert Ziel einer Aktion)
 - ◆ Berechtigung: z.B. Leseberechtigung auf einer Datei
(repräsentiert die Erlaubnis für die Ausführung einer Aktion)
- Erfassung der Berechtigungen in einer Subjekt-Objekt-Matrix:
Zugriffsliste (*Access control list, ACL*)

1 Beispiel für Zugriffslisten

- Personaldatensatz
 - ◆ besteht aus: **Name**, **Abteilung**, **Personalnummer**, **Lohn- oder Gehaltsgruppe**
- Personaldateien (Objekte)
 - ◆ D_{LA} : Personaldaten der leitenden Angestellten
 - ◆ D_{AN} : Personaldaten der sonstigen Angestellten
 - ◆ D_{AR} : Personaldaten der Arbeiter
- Prozeduren (gehören zu den Aktionen)
 - ◆ R_{LA} : Lesen von Pers.-Nr. und Lohn-/Gehaltsgr. aus D_{LA}
 - ◆ $R_{AN/AR}$: Lesen von Pers.-Nr. und Lohn-/Gehaltsgr. aus D_{AN} oder D_{AR}
 - ◆ R_{post} : Lesen von Name, Abteilung und Pers.-Nr.

1 Beispiel für Zugriffslisten (2)

■ Benutzer (Subjekte)

- ◆ S_{pers} : Leiter des Personalbüros
 - Besitzer aller Dateien und Prozeduren
 - Lese- und Schreibrecht für alle Dateien
 - Aufrufrecht für alle Prozeduren
- ◆ S_{stellv} : Sachbearb. leitende Angestellte, stellvert. Leiter Personalbüro
 - Lese- und Schreibrecht für D_{AN} und D_{AR}
 - Aufrufrecht für R_{LA}
- ◆ S_{sach} : Sachbearbeiter Angestellte u. Arbeiter
 - Aufrufrecht für $R_{AN/AR}$
- ◆ S_{post} : Poststelle
 - Aufrufrecht für R_{post} auf alle Dateien

1 Beispiel für Zugriffslisten (3)

■ Berechtigungen werden in Matrix ausgedrückt:

	D_{LA}	D_{AN}	D_{AR}	R_{LA}	$R_{AN/AR}$	R_{post}
S_{pers}	O, R, W	O, R, W	O, R, W	O, I	O, I	O, I
S_{stellv}		R, W	R, W	I		
S_{sach}					I	
S_{post}						I
R_{LA}	R					
$R_{AN/AR}$		R	R			
R_{post}	R	R	R			

- O = *Owner*; Besitzer der Datei oder Prozedur
- R = *Read*; volle Leseberechtigung
- W = *Write*; volle Schreibberechtigung
- I = *Invoke*; Aufrufberechtigung

2 Beispiel: UNIX

- Zugriffslisten für
 - ◆ Dateien und Geräte
 - ◆ Shared memory-Segmente
 - ◆ Message queues
 - ◆ Semaphore
 - ◆ etc.
- Berechtigungen:
 - ◆ Lesen (*read*), Schreiben (*write*), Ausführen (*execute*)
 - ◆ für Besitzer, Gruppe und alle anderen unterscheidbar
- Subjekte:
 - ◆ Prozesse
 - ◆ Besitzer (Benutzer) und Zugehörigkeit zu einer oder mehreren Gruppen

2 Beispiel: UNIX

- *Superuser*
 - ◆ Benutzer *root* hat automatisch alle Zugriffsrechte
- S-Bit–Programme
 - ◆ S-Bit ist ein besonderes Recht auf der Binärdatei des Programms
 - ◆ Besitzer der Datei wird bei der Ausführung auch Besitzer des Prozess (sonst wird Aufrufer Besitzer des Prozess)
- ★ Vorteil
 - ◆ Bereitstellen von Prozessen, die kontrolliert Aufrufern höhere Zugriffsberechtigungen erlauben
- ▲ Nachteil
 - ◆ Fehler im Prozess gibt Aufrufer volle Rechte des Programmbesitzers
 - ◆ fatal, falls das Programm *root* gehört