Betriebssystem

April 3, 2014

Was ist ein Betriebssystem?

1. Ein Ressourcen Verwalter (CPU -> Reichenleistung, Speicher, I/O Geräte, Daten (Sekundärspeicher: Disks, FS, etc))
2. Eine virtuelle Maschine (Schnittstelle zwischen Anwendersoftware und Hardware, abstrahiert von den Details der Hardware)
3. Anbieter von z.T standartisierten Diensten
4. Ziel: einfache und effiziente Nutzung von Rechnersystemen
5. Wichtig für BS:
   (a) Interrupts (erlaubt Unterbruch laufender Progamme, bessere CPU, Ausnutzung durch Multitasking).
   (b) System Calls (Schnitstelle zwischen User und Systembereich, schützt BS vor unerlaubtem Zugriff)
   (c) Speicherhierarchie (versteckt Zugriffszeiten auf Speicher, nicht-deterministische Zugriffszeiten, problematisch bei Echtzeitsystemen)
   (d) Boot-Vorgang (Systeminitialisierung, abhängig von Komplexität des BS)

Batch-Systems: nur ein Job aus Batch im Speicher • Jobs werden sequentiell abgearbeitet • Monitor zur Steuerung, keine Interaktion mit Anwender

Multiprogrammed-Batch-Systems: mehrere Jobs im Speicher • Scheduler notwendig • Interrupt und Speicherverwaltung notwendig

Time-Sharing-Systems: mehrere interaktive Jobs werden "gleichzeitig" abgearbeitet • Schutz des Filesystems und Arbeitsspeichers notwendig • Mutex (gegenseitiger Ausschluss) notwendig - Zugriff auf Drucker, etc.

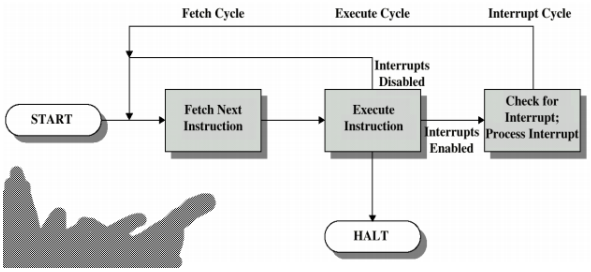
Job-Control-Language: is a scripting language used on IBM mainframe operating systems to instruct the system on how to run a batch job or start a subsystem.

1 Interrupts

Quellen für Interrupts

- Asynchrone Interrupts:
  - Timer
  - I/O Device
  - Hardwarefehler
- synchrone Interrupts:
  - Programm, z.B. div/0
  - Trap, SWI

-> Bessere CPU Nutzung
Ablauf: • Instruktion zu Ende führen • PC, PSW, Register, etc. speichern (umschalten in System Mode) • Interrupt Vektortabelle Adresse der ISR • ISR Instruktionen ausführen • PC, PSW, Register, etc. wiederherstellen (umschalten in User Mode) • Instruktionen des Anwenderprogramms
Auslöser Trap: Software, Interrupt: Hardware



System Mode - User Mode

Die meisten Prozessoren arbeiten in zwei Modi (Umschaltung über mode bits):

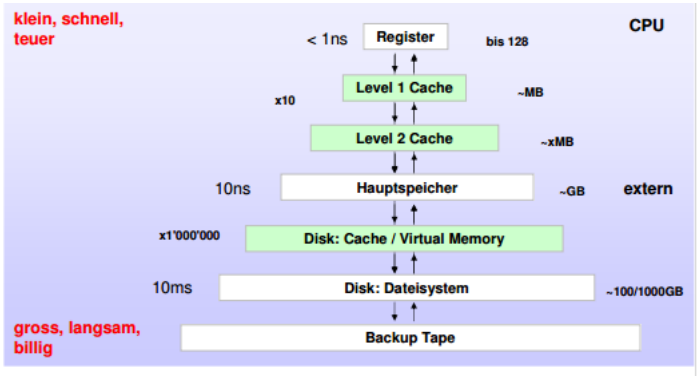
- System Mode
  - System Mode (Kernel, Supervisor, privilegierter Mode)
    - Zurgriff auf kritische Rechner-Ressourcen (Konstrollregister, I/O Instruktionen, Speicherverwaltung...)
- User Mode
  - Zugriff auf unkritische Rechner-Ressourcen
    - kein Zugriff auf Hardware (nur über BS)
    - Instruktionsmenge eingeschränkt

I/O Kommunikation

3 Techniken für Kommunikation mit I/O Geräten

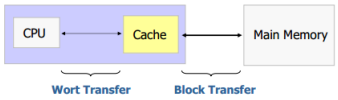
- Programmed I/O oder synchroner I/O
  - Benötigt keine Interrupts, CPU wartet auf Beendigung jeder einzelnen Operation
  - Busy wait
- Interrupt Driven I/O oder asynchroner I/O
  - CPU führt während I/O Operation Code aus, wird unterbrochen, wenn I/O Operation beendet
  - CPU kann andere Arbeiten ausführen
- Direct Memory Access
  - ein Speicherblock mit Daten wird vom/zum Speicher übertragen ohne Rechenleistung der CPU zu beanspruchen

Speicherhierarchie



DMA: Direct Memory Access. Daten gehen nicht über CPU, CPU wird somit nicht blockiert.

Cache

- Prozessor liest Wort
  - Wort im Cache –> Transfer an CPU
  - Wort nicht im Cache –> Block aus Speicher mit Wort ins Cache transferieren
  - transparent für Benutzer, Zusammenarbeit mit Memory Management Unit
  - 

Lokalitätsprinzip

Wieso funktioniert Cache?

- räumliche Lokalität (spacial locality)
  - grosse Wahrscheinlichkeit, dass nächster Speicherzugriff auf "nahe" liegende Daten stattfindet
- zeitliche Lokalität (temporal locality)
  - grosse Wahrscheinlichkeit, dass Speicherzugriff auf gleiches Datum nochmals staffindet

Hit Ratio

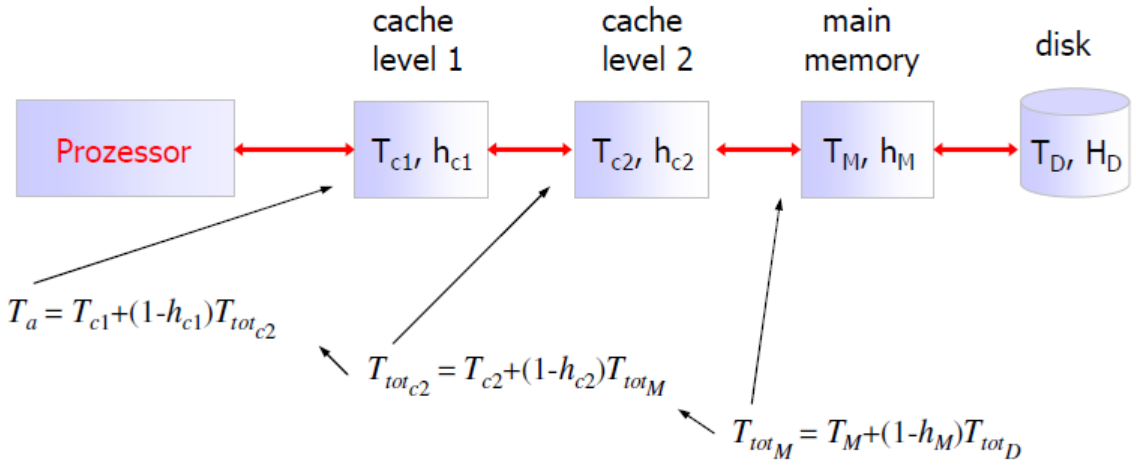
Mittlere Zugriff T_a:

h = Hitrate, 1-h = Missrate, T_M = Zugriffszeit auf Hauptspeicher

$T_a = h \cdot T_C + (1-h) \cdot (T_C + T_M) = T_C + (1-h) \cdot T_M$

$h = \frac{(T_a - (1-h) \cdot (T_C + T_M))}{T_C}$

Beispiel:



Speicher	Prozent	Zugriffszeit
Level 1 Cache	95 %	200 ps $= 2 \cdot 10^{-10}$s
Level 2 Cache	97 %	500 ps $= 5 \cdot 10^{-10}$s
Main Memory	30 %	2 ns $= 2 \cdot 10^{-9}$s
HD	100 %	1 ms $= 10^{-3}$

$T_a = T_{c1} + (1-h_{c1}) \cdot (T_{c2} + (1-h_{c2}) \cdot (T_M + (1-h_M) \cdot T_D))$

$T_a = 2 \cdot 10^{-10}s + (1-0.95) \cdot (5 \cdot 10^{-10}s + (1-0.97) \cdot (2 \cdot 10^{-9}s + (1-0.3) \cdot 10^{-3}s)) = 2 \cdot 10^{-10}s + 0.05 \cdot (5 \cdot 10^{-10}s + 0.03 \cdot (2 \cdot 10^{-9}s + 0.7 \cdot 10^{-3}s)) = 1.05023 \cdot 10^{-6}s$

Stack

Eigenschaften

- **Sequentiell Liste von Datenelementen**
- **Zugriffsverfahren LIFO (last in, first out)**
- **Zugriffpunkt top of stack**
- **Zugriff über Stackpointer auf top of stack**

Anwendungen

- **Prozeduraufruf (Speicherung der Rücksprungadresse)**
- **Temporäre Datenablage (mit Push und Pop)**
- **Speicherung des Prozessorzustandes bei Interrupts (Flag, PC, ev. Register)**
- **Reentrant Prozeduren (Stackframes), Parameter und temporäre Daten werden auf Stack abgelegt**

Systemstart

Zwei Phasen

- **Hardwareabhängige Phase**
  - **Start auf Reset-Adresse**
  - **Code aus Festwertspeicher ausführen (Hardwareüberprüfung, initialisierung Minimalzugriff auf Disk und Netzwerk, Boot Code laden)**
- **Start des Betriebssystems**
  - **Boot Code ausführen**

Prozesse

Prozessbasierte Betriebsystem

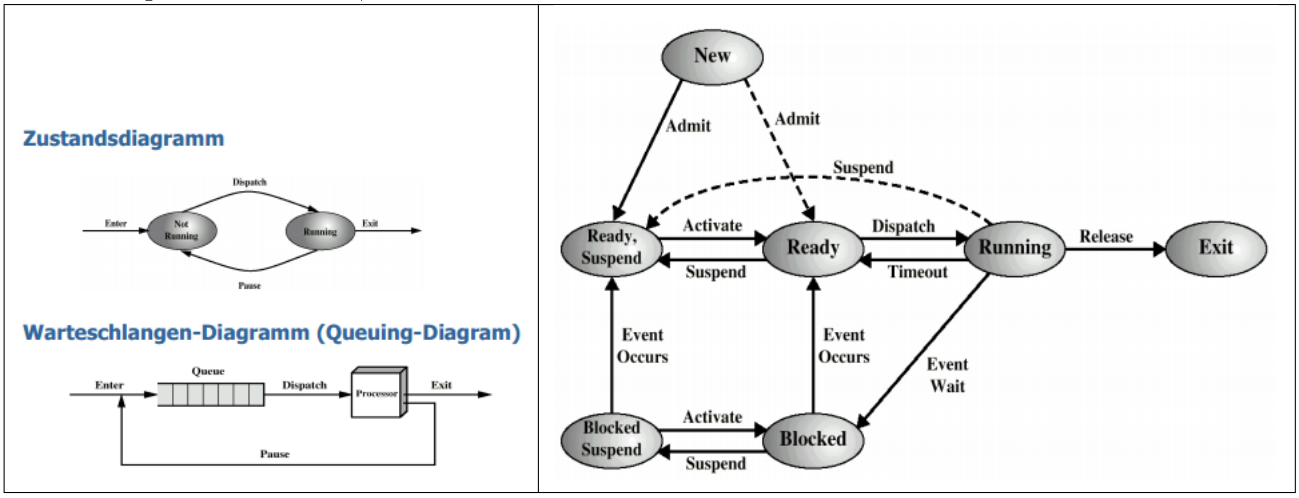
Gut für Multiprozessorsysteme, grössere Kernel Funktionen sind eigenständige Prozesse, BS ist Sammlung von Systemprozessen

Unit-of-Resource-Ownership: eine Einheit, die Ressourcen besitzt, ein virtueller Adressraum, in dem das Prozess Image steht, Kontrolle über Ressourcen (Files, I/O Geräte, etc.) hat

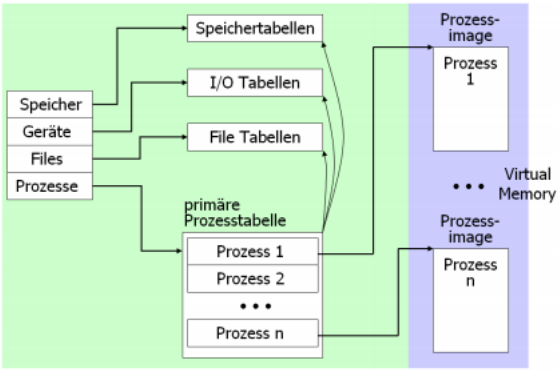
Unit-of-Scheduling: Eine Einheit, die schedulierbar ist. CPU-Scheduler weist der CPU einen Prozess zu (dispatch). zum Prozess gehören der Execution State (PC, SP, Register) und eine Ausführungspriorität

Zustands-Prozessmodell

Modellierung Prozessverhalten/7-Zustands-Modell



Prozess und Ressourcenmanangment



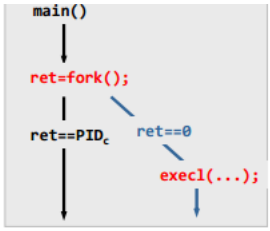
- Prozess Image besteht aus, ist im virtuellen Speicher (Adresse 0) abgelegt.
  - Benutzerprogramm (Code), Daten, Stack
  - Kontext, im Prozesskontrollblock (PCB) gespeichert
    - PCB, eine Datenstruktur mit Zustandsinformation zum Prozess

PCB Process Identification (PID, Parentproess, Benutzer), Process State Information (Inhalt der Register, Flags, SP), Process Control Information (Queues, Processprivileges,Memory Mgmt).

Prozess erzeugung

PID erzeugen -> Speicher für Prozess Image allozieren -> PCB initialisieren -> Verkettung für Queues aufsetzen -> Weitere Datenstrukturen init

Beispiel: Prozesserzeugung unter Unix / Linux




```
#include <sys/types.h>
#include <stdio.h>

int main(void) {
   pid_t ret;
   ret = fork();
   if (ret == 0) {
     printf("I am the child\n");
     if (execl("./test",NULL)<0) {
        printf("could not exec");
        exit(-1);
     }
   } else
        wait(null);
}
```

```
// make child
// child: return value = 0

// start program "test"
// should not come here


// parent: return value = PIDc
// wait for child to terminate
```

ZHAW, BSy, M. Thaler Januar 14

exec(...) überlagert das Programm und den Datenbereich mit neuem Programm und neuem Datenbereich, Prozesskontext wird von den Eltern geerbt (kann z.B. weiterhin auf geöffnete Files des Elternprozesses zugreifen)

COW fork: (Copy on Write, häufig implementiert). Kindprozess nicht vollständige Kopie der Eltern, nur wenn Kind schreibt werden Datenbereiche erzeugt. Alle lesbaren Bereiche können genutzt werden.

Threads

Threads sind billig, laufen innerhalb eines Prozesses. Kann schnell erzeugt und beendet werden. Threadumschaltung ist schnell: nur PC, SP und Register austauschen. Brauchen wenig Ressourcen, keinen neuen Adressraum oder Datenbereich oder Programmcode. Thread blockiert: User level -> ganzer Prozess blockiert. Kernel level -> nur Thread blockiert

Zombie, Orphan, Daemon

Zombie

- Terminiert ein Prozess, so ist dies ein Zombie, bis der Parent Prozess den Status des Prozesses abfragt.

Orphan (Waise)

- Terminiert der Parent Prozess vor dem Child Prozess, so wird der Child Prozess zum Waisen.
  Diesem wird der Prozess mit der PID 1 als Parent zugeordnet

Daemon

- Ein Daemon ordnet sich bewusst den Prozess mit PID 1 als Parent zu.
  Somit ist er für den Benutzer nicht mehr sichtbar und agiert als Systemprozess im Hintergrund.

Schedulers

Für Alle gilt: Fairness, Einhalten von Policies, System optimal nutzen
Long-Term-Scheduling (Admission Scheduling): degree of multiprocessing
Medium-Term-Scheduling (Memory Scheduling): auslagern auf disk / zurückholen
Short-Term-Scheduling (CPU Scheduling): zuteilung Rechenleistung, whos next?
IO-Bound: short CPU Bursts
CPU-Bound: long CPU Bursts
-> je schneller die CPU, desto eher IO bound
Schätzfunktion: $S_{n+1} = \alpha \cdot T_n + (1-\alpha) \cdot S_n$ **bei S1=0 hohe Startpriorität**
Preemption: Pausierung und zurückstellung in die ReqdyQueue
Unix-Sched: $CPU_i[j] = \frac{CPU_i[j-1]}{2}$ **und** $P_i[j] = base_i + \frac{CPU_i[j]}{2} + nice_i$ **(prio drop while high CPU) (RT,Kern,TimeShare)**

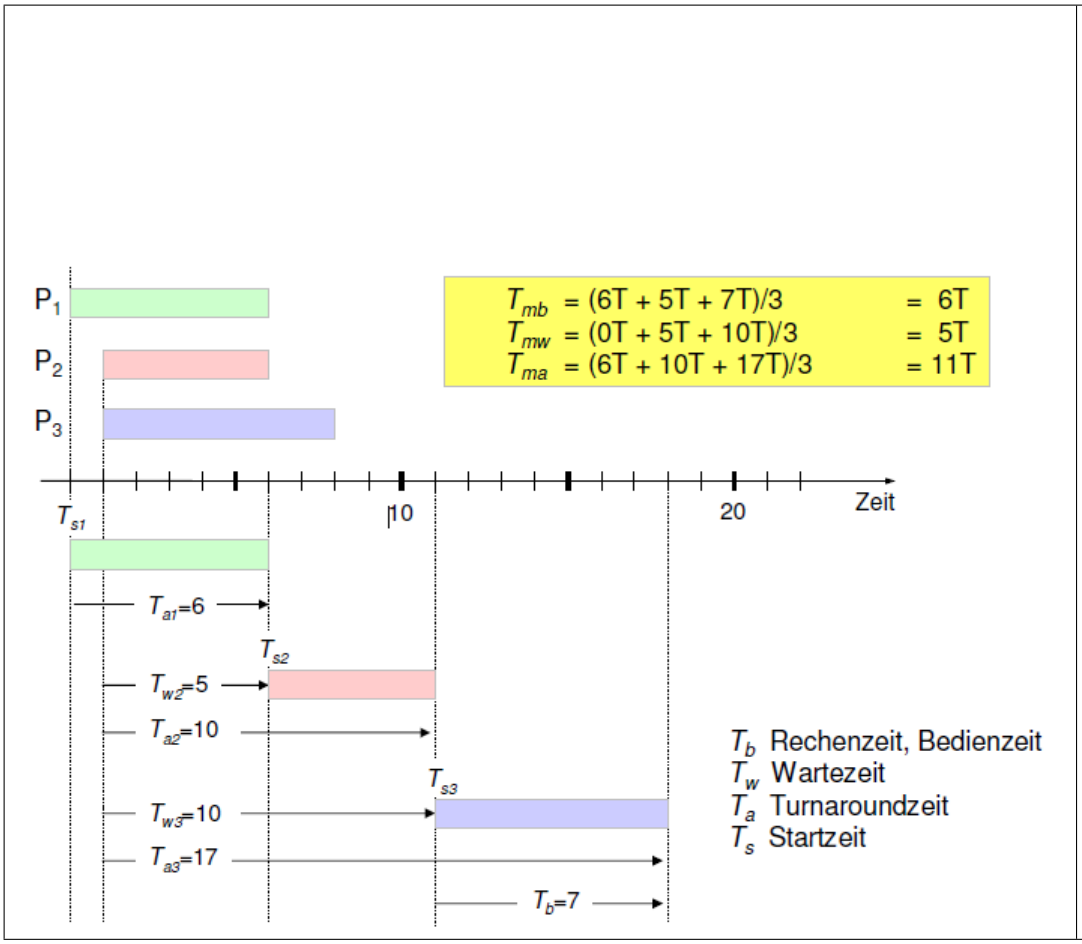
- **Preemptive (interaktiv (antwortzeit: schnelle Reaktion, erwartung: erfüllen der Anwendererwartungen): Anwender wartet, Umschalten notwendig)**
  - **FCFS, SPN(starv,notfair,oh), SJN(starv,notfair,oh), Pri. Sched.(starv)**
- **Non-Preemptive (batch (Durchsatz: #jobs/zeit, turnaround zeit: terminierung/aufgabe, cup: möglichst 100%): kein Anwenderterminal, am Stück verarbeitet)**
  - **RR(notfair:IO, nostarv,nooh,troughput: abh v timeslice), ML(starv:lowprio, rr pro queue), MLF(starv,notfair:CUP (nostarv with dyn prio)**
- **Real-Time (deadlines: kein datenverlust,):(Zur richtigen Zeit verfügbar, periodische jobs) (krit:period,sporadisch, nonkrit,notnessecary)**
  - **RateMonotonic (prio relativ zu repetitionsrate,preem,tasks müssen unabhängig sein), DeadlineS, CyclicExec(statisches sched, nonpreem)**

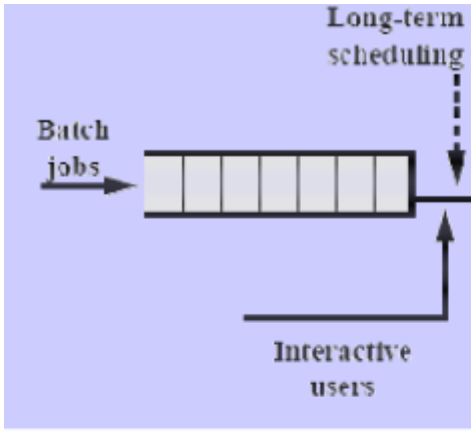
Preemptive

Round Robin: Time slices bestimmen, jeder Prozess hat max z.B. 20ms dann wieder zurück in die Ready-Queue. Mehr Context Switching –> Overhead

NonPreemptive

(Shortest Job First oder Shortest Remaining Time) -> best at minimizing average response time, First oder FCFS: nicht blockierende Prozesse werden am Stück abgearbeitet - blockierende Prozesse werden in die Blocked Queue gehängt und bleiben dort, bis sie von einem entsprechenden Event geweckt werden, dann werden sie in die Ready Queue gehängt –> lange Antwortzeit bei verschiedene Prozesslängen. Sehr einfach.
Multilevel: batch, interaktiv, rechenintensive Arten von Jobs. Jobs mit tiefen Prios können verhungern. Queue pro Prio.
Multilevel Feedback Scheduling: Jobs steigen in Prio queues auf, sinken wenn Rechenintensiv nach Prozessor runter.





Linux System

cmd	description
init.d	start/stop/reload/restart, autostart
fstab	automated mounting of partitions, ex: /dev/mapper/fedora_zecomputer-home /home ext4 defaults 1 2
mtab	wie fstab, einfach nur mit currently mounted
samba	windows support, file/printer sharing
nfs	schneller als samba, file sharing
boot	boot (loads bootloader in flopy, cd, hd etc. gives control to boatloader)
	mbr (/dev/hda(sda) excecutes grub bootloader), g
	grub (choose kernel, basic configuration, excecutes kernel and initrd image), kernel (mounts root fs)
	init (Looks at the /etc/inittab file to decide the Linux run level. replaced by systemd), runlevel
runlevels	0-6, /etc/rc*.d directory. S12syslog starts before S80sendmail. S are used during startup, K during shutdown
/etc/passwd	jorismorger:x:1000:1000:jorismorger:/home/jorismorger:/bin/zsh
/etc/shadow	encrypted passwords

2 Deadlocks

Grundsätzliche Probleme

- Starvation (Verhungern)
  - Prozess erhält keinen Zutritt zu Ressource
  - Ursache, z.B. unfaire Zuweisungspolicy: FILO (Stack)
  - Abhilfe: nur faire Policies verwenden, z.B. FIFO
- Deadlock (Verklemmung)
  - Prozesse warten gegenseitig auf Freigabe von Ressourcen
  - Die Prozesse und eventuell das gesamte System bleiben hängen

t

Voraussetzungen

- Mutual Exclusion
  - mindestens eine Ressource ist exklusiv reserviert
- Hold and wait
  - mindestens ein Task hat eine Ressource exklusiv reserviert und wartet auf weitere Ressourcen
- No preemption
  - reservierte Ressourcen können dem Task nicht entzogen werden (freiwillige Rückgabe nur, wenn Aufgabe gelöst)
- Circular wait
  - geschlossene "Kette" von Tasks existiert, in der jeder Prozess mindestens eine Ressource reserviert hat, die auch von einem Nachfolger in der Kette benötigt wird

1-3 sind Vorbedingungen, mit 4 ist es ein Deadlock.
die Circular Wait Bedingung kann nicht gelöst werden, wenn Bedingungen 1, 2 und 3 gegeben sind

Umgang mit Deadlocks

- Keine Deadlocks zulassen
  - verhindern (prevention), dafür sorgen, dass mindestens eine der 4 Bedingungen nicht auftritt
  - vermeiden (avoidance), Ressource wird nicht zugesprochen, falls Deadlockgefahr besteht
- Deadlocks zulassen
  - diesen Zustand lösen, wenn er eingetreten ist.
- Das Problem ignorieren
  - annehmen, dass kein Deadlock auftritt (die meisten Betriebsysteme verwenden diese Methode)