

# Betriebssysteme I

Hochschule München – SS 2008

Hans-Georg Eßer

## Zusammenfassung

/home/esser/Daten/Dozent/HS-Muenchen-2008/Folien/betriebssysteme-14.odp

## Gliederung

### Betriebssysteme I

2. Prozesse und Threads
3. Interrupts
4. Scheduler
5. Synchronisation und Deadlocks
6. Speicherverwaltung
7. Dateisysteme

```
Sep 19 14:20:18 amd64 sshd[20494]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61557
Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[30103]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6516]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6694]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sshd[9077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10102]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17478]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:39 amd64 sshd[32993]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[46]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[54]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 22 20:23:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[24739]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[25555]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted publickey for esser from ::ffff:87.234.201.207 port 62514
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[10998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64556
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[662]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:02 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11551]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:13 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778
```

# 2. Prozesse und Threads

**Prozess:** Programm, das in den Speicher geladen wurde und ausgeführt wird / werden soll

Mehr als nur der Programmcode:

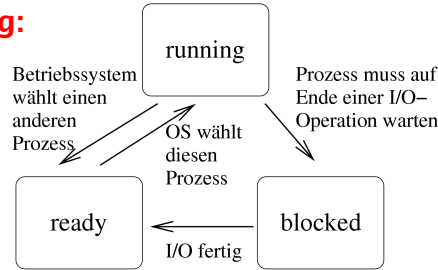
- Eigener Adresraum
- Stack, Stack-Pointer
- Programmzähler
- Umgebung
- Hardware-Register

**Process Control Block (PCB):**

- Identifizier (PID)
- Registerwerte inkl. Befehlszähler
- Speicherbereich des Prozess
- Liste offener Dateien und Sockets
- Verwaltungsinformationen

# Prozess-Zustände

- **laufend / running:** gerade aktiv
- **bereit / ready:** würde gerne laufen
- **blockiert / blocked / waiting:** wartet auf I/O



- **suspendiert:** vom Anwender unterbrochen
- **schlafend / sleeping:** wartet auf Signal (IPC)
- **ausgelagert / swapped:** Daten nicht im RAM

# Threads

- einer von mehreren Aktivitätssträngen in einem Prozess
- Gemeinsamer Zugriff auf Daten des Prozess (kein eigener Speicher)
- aber: Stack, Befehlszähler, Stack Pointer, Hardware-Register separat pro Thread
- Multi-Prozessor-System: Mehrere Threads echt gleichzeitig aktiv
- Ist ein Thread durch I/O blockiert, arbeiten die anderen weiter (nur bei Kernel Level Threads)
- Besteht Programm logisch aus parallelen Abläufen, ist die Programmierung mit Threads einfacher

# Prozess-Hierarchien / fork/exec

- Prozesse erzeugen einander
- Erzeuger heißt **Vaterprozess** (parent process), der andere **Kindprozess** (child process)
- Kinder sind selbständig (also: eigener Adressraum, etc.)
- Nach Prozess-Ende: Rückgabewert an Vaterprozess

## Prozesse unter Linux/Unix:

**fork ():** dupliziert aktiven Prozess

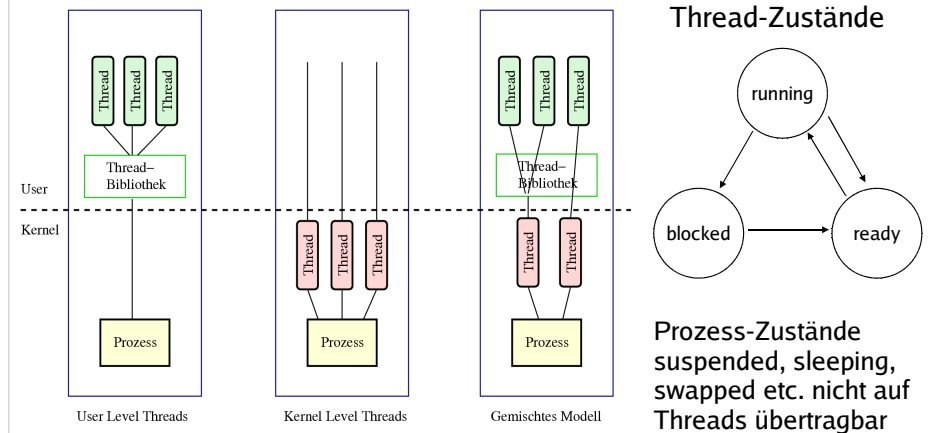
- Vater erhält bei fork() die PID des Sohnes zurück
- Sohn erhält bei fork() den Wert 0 zurück

**exec ():** fremdes Programm in laufenden Prozess laden

- überschreibt aktuellen Code
- exec () kehrt darum nie zurück

**wait ():** auf einen Sohnprozess warten

# User-Level- vs. Kernel-Level-Threads



```

Sep 19 14:27:18 amd64 sshd[4974]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62522
Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[39278]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[10103]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6316]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:44 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6941]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sshd[9077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10292]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63564
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17878]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[13088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:51:39 amd64 sshd[13265]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[14743]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[14401]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:25:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[1]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[355]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted publickey for esser from ::ffff:192.168.1.5 port 5971 ssh2
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:01 amd64 /usr/sbin/cron[562]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 25 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64283
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778

```

# 3 Interrupts

## Interrupts

- **Effizienz** (I/O-Zugriff sehr langsam → sehr lange Wartezeiten, wenn Prozesse warten, bis I/O abgeschlossen ist)
- **Programmierlogik** (Nicht immer wieder Gerätestatus abfragen, sondern abwarten, bis passender Interrupt kommt)
- **Kein Polling** (Polling: BS fragt regelmäßig bei allen Geräten nach, ob ein Ereignis stattgefunden hat)

## Interrupts

### Mehrfach-Interrupts

- Während Abarbeitung eines Interrupts alle weiteren ausschließen (DI, disable interrupts) → Interrupt-Warteschlange
- Während Abarbeitung beliebige Interrupts zulassen
- Interrupt-Prioritäten: Nur Interrupts mit höherer Priorität unterbrechen solche mit niedrigerer

### Multitasking und Interrupts

- Multitasking verbessert CPU-Nutzung:
  - I/O-lastiger Prozess wartet auf I/O-Events,
  - CPU-lastiger Prozess rechnet weiter
- Prozess stößt I/O-Operation an und legt sich schlafen (wartet auf Signal)

## Linux-Interrupt-Handler

### Für jedes Gerät:

- Interrupt Request (IRQ) Line
- Interrupt Handler (Interrupt Service Routine, ISR) → Teil des Gerätetreibers
- läuft in speziellem Context (Interrupt Context)

### top half

- Interrupt Handler startet sofort, erledigt zeitkritische Dinge
- bestätigt (der Hardware) den Erhalt des Interrupts, setzt Gerät zurück etc.

### bottom half / Tasklet

- startet später, macht die eigentliche Arbeit
- kein Prozess (struct tasklet\_struct), läuft direkt im Kernel; im Interrupt-Context
- Zwei Prioritäten: *tasklet\_hi\_schedule*, *tasklet\_schedule*

# 4. Scheduling

```

Sep 19 14:27:18 amd64 sshd[4974]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62529
Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[39278]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[31013]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6316]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:44 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6941]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62314
Sep 20 15:27:35 amd64 sshd[9077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[11265]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17051]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17878]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:51:39 amd64 sshd[31265]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[16743]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[15499]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:23:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[124719]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[124719]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6541]: Accepted pubkey for esser from ::ffff:192.168.1.5 port 59771 ssh2
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[29989]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[662]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 25 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9179]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11610]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778

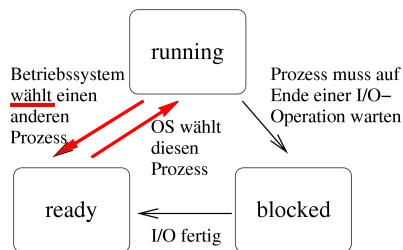
```

# Scheduler

- **Kooperatives Scheduling:**
  - Prozess rechnet bis zum nächsten I/O-Aufruf oder `exit()`
  - Scheduler wird nur bei Prozess-Blockieren oder freiwilliger CPU-Aufgabe aktiv
- **Präemptives (unterbrechendes) Scheduling:**
  - Timer aktiviert regelmäßig Scheduler, der neu entscheiden kann, „wo es weiter geht“
- **I/O-lastig:** Prozess hat zwischen I/O-Phasen nur kurze Berechnungsphasen (CPU)
- **CPU-lastig:** Prozess hat zwischen I/O-Phasen lange Berechnungsphasen

# Scheduler

- Rechenzeit (CPU) an Prozesse verteilen



- Scheduling-Prinzipien: **präemptiv** vs. **kooperativ**
- Scheduling-Verfahren:
  - FIFO, Shortest Job First
  - Shortest Remaining Time, Round Robin, Priority, Lottery

# Scheduling-Ziele

- **[A1] Ausführdauer:** Wie lange läuft der Prozess insgesamt?
- **[A2] Reaktionszeit:** Wie schnell reagiert der Prozess auf Benutzerinteraktion?
- **[A3] Deadlines** einhalten
- **[A4] Vorhersehbarkeit:** Gleichartige Prozesse sollten sich auch gleichartig verhalten, was obige Punkte angeht
- **[A5] Proportionalität:** „Einfaches“ geht schnell
- **[S1] Durchsatz:** Anzahl der Prozesse, die pro Zeit fertig werden
- **[S2] Prozessorauslastung:** Zeit, die der Prozessor aktiv war „verhungern“
- **[S3] Fairness:** Prozesse gleich behandeln, keiner darf „verhungern“
- **[S4] Prioritäten** beachten
- **[S5] Ressourcen** gleichmäßig einsetzen

# Anforderungen an Betriebssystem

## Stapelverarbeitung

- S3 Fairness
- S4 Prioritäteneinsatz
- S5 Ressourcen-Balance
- S1 Durchsatz
- A1 Ausführdauer
- S2 Prozessor-Auslastung

## Interaktives System

- S3 Fairness
- S4 Prioritäteneinsatz
- S5 Ressourcen-Balance
- A2 Reaktionszeit
- A5 Proportionalität

# Scheduler für Batch-Betrieb

## First Come, First Served (FCFS)

- Warteschlange, kooperativ (keine Unterbrechung)
- Blockierende Prozesse stellen sich wieder in Warteschlange an
- gut für lange Prozesse, schlecht für I/O-lastige Prozesse

## Shortest Job First (SJF)

- kooperativ
- nächster Burst (Rechendauer bis zum Blockieren) muss bekannt sein (woher? Burst-Prognose-Verfahren); Prozess mit dem kürzesten Burst auswählen
- minimiert durchschnittliche Laufzeit über alle Prozesse

## Shortest Remaining Job Next (SRJ)

- ähnlich SJF, aber *mit* Unterbrechungen
- Scheduler prüft Reihenfolge bei neuen Jobs
- Prozess mit kürzerer Restlaufzeit unterbricht den laufenden

# Scheduler für Batch-Betrieb

- Kein interaktiver Betrieb (kein Login etc.)
- Job-Management-Tool nimmt Jobs an
- Long term scheduler entscheidet, wann ein Job gestartet wird – evtl. basierend auf Informationen über Ressourcenverbrauch und erwartete Laufzeit des Programms

## Scheduling-Verfahren für Batch-Betrieb

- First Come, First Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time (SRT)
- Priority Scheduling

# Scheduler für interaktive Systeme

- Typisch: Interaktive und Hintergrund-Prozesse
- Desktop- und Server-PCs
- Eventuell mehrere / zahlreiche Benutzer, die sich die Rechenkapazität teilen
- Scheduler für interaktive Systeme prinzipiell auch für Batch-Systeme brauchbar (aber nicht umgekehrt)

## Scheduling-Verfahren für interaktive Systeme

- Round Robin (RR)
- Virtual Round Robin (VRR)
- Prioritäten-Scheduler
- Lotterie-Scheduler

# Scheduler für interaktive Systeme

## Round Robin / Time-Slicing

- wie FCFS, aber mit Unterbrechung
- jeder Prozess erhält Zeitscheibe (Quantum)
- läuft der Prozess bei Ablauf noch → unterbrechen
- zum Quantum:
  - groß → Verzögerungen; klein → häufige Context Switches
  - oft: etwas größer als typische Zeit, die für Interaktion nötig ist
- bevorzugt CPU-lastige Prozesse (I/O-lastige brauchen immer nur Teil ihres Quantums)

## Virtual Round Robin

- Round Robin, aber: I/O-lastigen Pr. helfen → „Restguthaben“
- hat Prozess Quantum nicht verbraucht? → merken und Prozess in Extra-Queue stecken
- Scheduler bevorzugt Prozesse in Extra-Queue und lässt sie ihr Restguthaben aufbrauchen

# Scheduler für interaktive Systeme

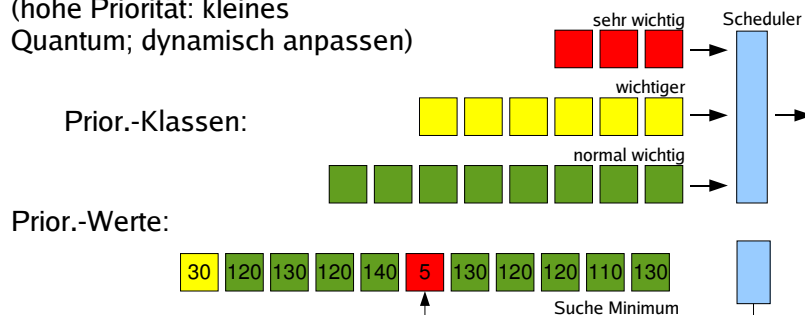
## Lotterie-Scheduler

- Prozesse besitzen Lose, Scheduler zieht ein Los
- Prioritäten: über Anzahl der Lose, die ein Prozess besitzt
- Gruppenbildung / Kooperation: Prozesse können einander Lose überlassen (etwa ein Client einem Server)

# Scheduler für interaktive Systeme

## Prioritäten-Scheduler

- Prioritätsklassen oder individuelle Prioritätswerte
- Scheduler bevorzugt Prozesse mit hoher Priorität
- Prior. statisch vergeben oder dynamisch regelmäßig neu ber.
- Vorsicht: **Prioritätsinversion** (Ausweg: **Aging**)
- Variante: Verschiedene Quantenlängen (hohe Priorität: kleines Quantum; dynamisch anpassen)



```

Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -e "severity=DBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[30103]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6541]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6694]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sshd[9077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10302]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -e "severity=DBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17878]: (root) CMD (/sbin/evlogmgr -e 'age > *30d*')
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:10 amd64 sshd[32091]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:41:00 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:41:00 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[24140]: (root) CMD (/sbin/evlogmgr -e "severity=DBUG")
Sep 22 02:00:01 amd64 /usr/sbin/cron[24140]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[24140]: (root) CMD (/sbin/evlogmgr -e "severity=DBUG")
Sep 23 01:00:01 amd64 /usr/sbin/cron[24140]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 23 02:00:01 amd64 /usr/sbin/cron[24140]: (root) CMD (/sbin/evlogmgr -e "severity=DBUG")
Sep 23 02:00:01 amd64 /usr/sbin/cron[24140]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 23 18:04:05 amd64 sshd[12319]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61316
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[12319]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61316
Sep 24 01:00:01 amd64 /usr/sbin/cron[24140]: (root) CMD (/sbin/evlogmgr -e "severity=DBUG")
Sep 24 01:00:01 amd64 /usr/sbin/cron[24140]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 24 11:35:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:35:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[19397]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_mid_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: amd_aseq: unsupported module, tainting kernel.
Sep 24 20:25:11 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[6621]: (root) CMD (/sbin/evlogmgr -e "severity=DBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:17 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:17 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:13 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63769
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778
    
```

# 5. Synchronisation und Deadlocks

## Kritische Abschnitte

- Programmteil, der auf gemeinsame Daten zugreift
- Nicht „den Code schützen“, sondern die Daten
- Formulierung: kritischen Bereich „betreten“ und „verlassen“
- Anforderung an parallele Threads:
  - maximal ein Thread gleichzeitig im kritischen Abschnitt
  - Kein Thread, der außerhalb kritischer Bereiche ist, darf einen anderen blockieren
  - Kein Thread soll ewig auf Betreten des kritischen Bereichs warten
  - Deadlocks vermeiden (z. B.: zwei Prozesse sind in verschiedenen krit. Bereichen und blockieren sich gegenseitig)

## Aktives und passives Warten

- **Aktives Warten (busy waiting):**
  - Ausführen einer Schleife, bis eine Variable einen bestimmten Wert annimmt.
  - Der Thread ist bereit und belegt die CPU.
  - Die Variable muss von einem anderen Thread gesetzt werden.
- **Passives Warten (sleep and wake):**
  - Ein Thread blockiert und wartet auf ein Ereignis, das ihn wieder in den Zustand „bereit“ versetzt.
  - Der blockierte Thread verschwendet keine CPU-Zeit.
  - Ein anderer Thread muss das Eintreten des Ereignisses bewirken. → (kleines) Problem, wenn der andere Thread endet.
  - Bei Eintreten des Ereignisses muss der blockierte Thread geweckt werden, z. B. durch anderen Thread oder durch OS.

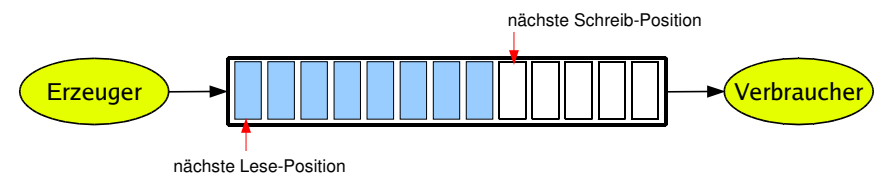
## Gegenseitiger Ausschluss

- Tritt nie mehr als ein Thread gleichzeitig in den kritischen Bereich ein, heißt das „**gegenseitiger Ausschluss**“ (englisch: **mutual exclusion**)
- Es ist Aufgabe der Programmierer, diese Bedingung zu garantieren
- Das Betriebssystem bietet Hilfsmittel („Synchronisationswerkzeuge“), mit denen gegenseitiger Ausschluss durchgesetzt werden kann, schützt aber nicht vor Programmierfehlern

## Erzeuger-Verbraucher-Problem

**Erzeuger-Verbraucher-Problem** (producer consumer problem, bounded buffer problem): zwei kooperierende Threads

- Erzeuger speichert Informationen in einem **beschränkten Puffer**.
- Verbraucher liest Informationen aus diesem Puffer.



### Synchronisation:

- Puffer nicht überfüllen
- Nicht aus leerem Puffer lesen



# Semaphore

**Semaphor:** Integer- (Zähler-) Variable

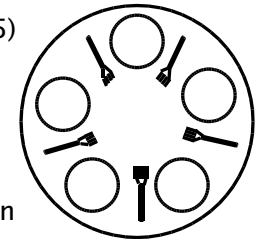
- festgelegter Anfangswert N („Anzahl der verfügbaren Ressourcen“).
- **Anfordern** eines Semaphors (**Wait-Operation**):
  - Semaphor-Wert um 1 erniedrigen, falls er positiv ist,
  - Thread blockieren und in eine Warteschlange einreihen, wenn der Semaphor-Wert 0 ist.
- **Freigabe** eines Semaphors (**Signal-Operation**):
  - einen Thread aus Warteschlange wecken, falls diese nicht leer ist,
  - Semaphor-Wert um 1 erhöhen (wenn kein Thread wartet)
- Code sieht dann immer so aus: 

```
wait (&sem);
/* Code, der die Ressource nutzt */
signal (&sem);
```

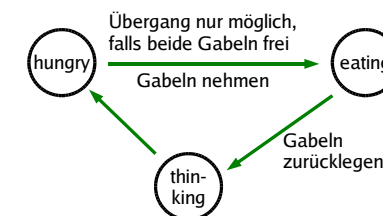
# Philosophenproblem

- Fünf Philosophen an einem Tisch (Dijkstra 1965)

- je Philosoph ein Teller Spaghetti.
- zwischen je zwei Tellern eine Gabel.
- Jeder Philosoph wechselt ab zwischen Denken und Essen.
- Zum Essen benötigt ein Philosoph die beiden Gabeln rechts und links von seinem Teller.



- Ziel: Philosophen nicht verhungern lassen und maximale Parallelität



# Mutexe

- **Mutex:** boolesche Variable (true/false), die den Zugriff auf gemeinsam genutzte Daten synchronisiert (true: erlaubt; false: verboten)
- **blockierend:** Ein Thread, der sich Zugang verschaffen will, während ein anderer Thread Zugang hat, blockiert → Warteschlange
- Bei Freigabe: – Warteschlange enthält Threads → einen wecken  
– Warteschlange leer → Mutex auf true setzen
- **Mutex (mutual exclusion) = binärer Semaphor**, also ein Semaphor, der nur die Werte 0 / 1 annehmen kann

```
wait (mutex) {
    if (mutex==1)
        mutex=0;
    else BLOCK_CALLER;
}

signal (mutex) {
    if (P in QUEUE(mutex)) {
        wakeup (P);
        remove (P, QUEUE);
    }
    else mutex=1;
}
```

- Neue Interpretation: wait → lock, signal → unlock
- Mutexe für exklusiven Zugriff auf kritische Bereiche

# Monitore (1)

## Motivation

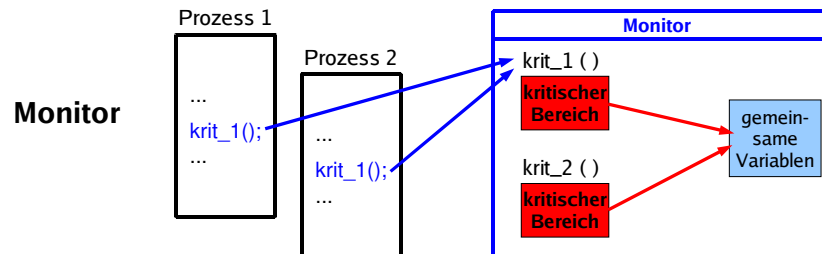
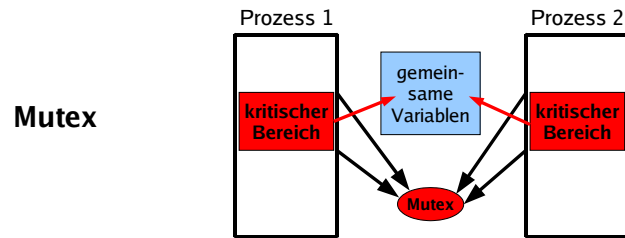
- Arbeit mit Semaphoren und Mutexen zwingt den Programmierer, vor und nach jedem kritischen Bereich wait() und signal() aufzurufen
- Wird dies ein einziges Mal vergessen, funktioniert die Synchronisation nicht mehr
- **Monitor** kapselt die kritischen Bereiche

**Monitor:** Sammlung von Prozeduren, Variablen, speziellen **Bedingungsvariablen** und Datenstrukturen:

- Prozesse können die Prozeduren des Monitors aufrufen, können aber nicht von außerhalb des Monitors auf dessen Datenstrukturen zugreifen.
- Zu jedem Zeitpunkt kann **nur ein einziger Prozess aktiv im Monitor** sein (d. h.: eine Monitor-Prozedur ausführen).
- Monitor wird durch Verlassen der Monitorprozedur frei gegeben



## Monitore (2)

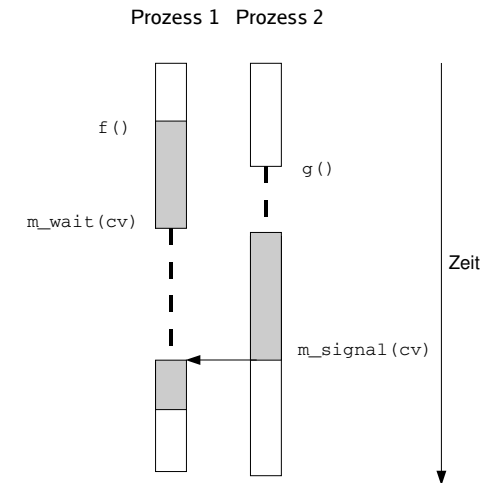
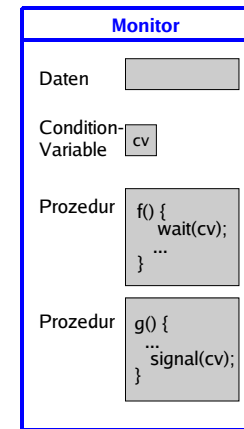


Hans-Georg Eßer, Hochschule München

Betriebssysteme I, SS 2008

Zusammenfassung – Folie 33

## Monitore (4)



Hans-Georg Eßer, Hochschule München

Betriebssysteme I, SS 2008

Zusammenfassung – Folie 35

## Monitore (3)

- Monitor-Konzept erinnert an Klassen oder Module
- Kapselung der Prozeduren und Variablen (außer über als public deklarierte Prozeduren kein Zugriff auf Monitor)
- Einfaches und übersichtliches Verfahren, um kritische Bereiche zu schützen, aber:
- Busy waiting → Schlafen/Wecken wäre besser

### Zustandsvariablen (condition variables)

Für jede Zustandsvariable Wait- und Signal-Funktionen:

- `m_wait (var)`: aufrufenden Prozess sperren (er gibt den Monitor frei)
- `m_signal (var)`: gesperrten Prozess entsperren (weckt einen Prozess, der den Monitor mit `m_wait()` verlassen hat); erfolgt unmittelbar vor Verlassen des Monitors

Hans-Georg Eßer, Hochschule München

Betriebssysteme I, SS 2008

Zusammenfassung – Folie 34

## Locking

**Locking** erweitert die Funktionalität von Mutexen, indem es verschiedene **Lock-Modi** (Zugriffsarten) unterscheidet, und deren „Verträglichkeit“ miteinander festlegt:

- **Concurrent Read**: Lesezugriff, andere Schreiber erlaubt.
- **Concurrent Write**: Schreibzugriff, andere Schreiber erlaubt.
- **Protected Read**: Lesezugriff, andere Leser erlaubt, aber keine Schreiber (share lock)
- **Protected Write**: Schreibzugriff, andere Leser erlaubt, aber kein weiterer Schreiber (update lock)
- **Exclusive**: Schreibzugriff, keine anderen Zugriffe erlaubt

Hans-Georg Eßer, Hochschule München

Betriebssysteme I, SS 2008

Zusammenfassung – Folie 36

# Synchronisation im Linux-Kernel

- Spin Locks / Reader-Writer Spin Locks
- Semaphore / Reader-Writer-Semaphore

## Spin Locks (1)

Lock mit Mutex-Funktion: Gegenseitiger Ausschluss  
Code, der ein Spin Lock anfordert und nicht erhält, läuft in Schleife weiter, bis das Lock verfügbar wird („spinning“)

Typ: `spinlock_t`

```
spinlock_t xy_lock = SPIN_LOCK_UNLOCKED;

spin_lock(&xy_lock);
/* kritischer Abschnitt */
spin_unlock(&xy_lock);
```

Da Spin Locks nicht schlafen, kann man sie in Interrupt-Handlern verwenden

# Kernel-Semaphore

- Kernel-Semaphore: „schlafende“ Locks, mit Warteschlange
- Geeignet für Sperren, die über einen längeren Zeitraum gehalten werden: keine Verschwendung von Rechenzeit
- Semaphore sind nur im Prozess-Kontext einsetzbar, nicht in Interrupt-Handlern (Interrupt-Handler dürfen nicht schlafen)
- Code, der einen Semaphor verwenden will, darf nicht bereits ein normales Spin Lock besitzen (Semaphor-Zugriff kann dazu führen, dass der Thread sich schlafen legt.)
- Verwendung mit `up()` und `down()`

- Varianten von `down()`

```
down(&sem);
/* kritischer Abschnitt */
up(&sem);

- down(&sem);
- down_interruptible(&sem);
- down_trylock(&sem);
```

## Spin Locks (2)

- Ggf. zusätzlich Interrupts sperren:

```
unsigned long flags;
spin_lock_irqsave(&xy_lock, flags);
/* kritischer Abschnitt */
spin_unlock_irqrestore(&xy_lock, flags);
```

(Interrupts sichern, dann sperren; urspr. Zustand wiederherstellen)

- Wenn zu Beginn alle Interrupts aktiviert sind, geht es auch einfacher:

```
spin_lock_irq(&xy_lock);
/* kritischer Abschnitt */
spin_unlock_irq(&xy_lock);
```

- Spin Locks sind nicht „rekursiv“, d.h.: es ist nicht möglich, das gleiche Spin Lock zweimal nacheinander anzufordern, etwa beim rekursiven Aufruf einer Funktion

- Alternative zu Spin Locks: Reader Writer Locks

## Deadlocks: kleinstes Beispiel

Prozess P

```
lock(A);
lock(B);
/* krit. Ber. */
unlock(A);
unlock(B);
```

Prozess Q

```
lock(B);
lock(A);
/* krit. Ber. */
unlock(B);
unlock(A);
```

**Problematische Reihenfolge:**

```
P: lock(A)
Q: lock(B)
P: lock(B) <- blockiert
Q: lock(A) <- blockiert
```

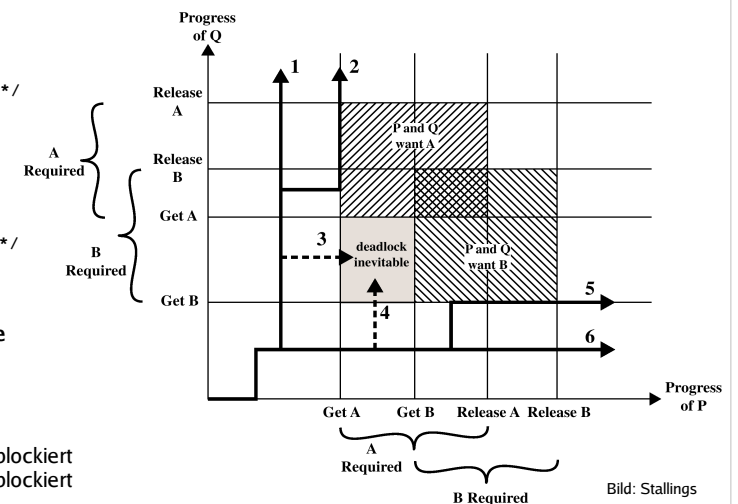
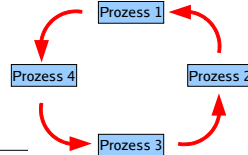


Bild: Stallings

# 4 Deadlock-Bedingungen

- Gegenseitiger Ausschluss (mutual exclusion)**
- Hold and Wait (besitzen und warten)**  
Ein Prozess ist bereits im Besitz einer oder mehrerer Ressourcen, und er kann noch weitere anfordern
- Ununterbrechbarkeit der Ressourcen**  
Ressource kann nicht durch das Betriebssystem entzogen werden
- Zyklisches Warten**  
Man kann die Prozesse in einem Kreis anordnen, in dem jeder Prozess eine Ressource benötigt, die der folgende Prozess belegt hat



- (1) bis (4) sind **notwendige und hinreichende** Bedingungen für einen Deadlock (Äquivalenz)
- (4) ist der erfolgversprechendste Ansatzpunkt, um Deadlocks aus dem Weg zu gehen

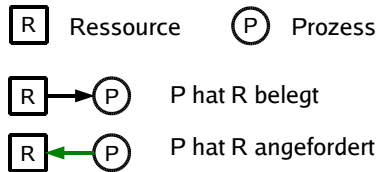
# Deadlock-Erkennung (1)

- Idee: Deadlocks zunächst zulassen
- System regelmäßig auf Vorhandensein von Deadlocks überprüfen und diese dann abstellen
- Beispiel:

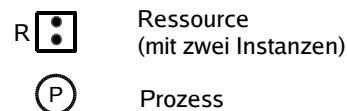
	ISDN-Karte	Streamer	Scanner	DVD-Brenner		ISDN-Karte	Streamer	Scanner	DVD-Brenner	
$E =$	1	3	2	4		$A =$	0	0	1	2
Ressourcenvektor						Ressourcenrestvektor				
$C =$	0	1	0	0		$R =$	1	0	0	2
	1	0	0	2			0	1	0	1
	0	2	1	0			0	0	1	2
Belegungsmatrix						Anforderungsmatrix				
										Prozess 1
										Prozess 2
										Prozess 3

# Ressourcen-Zuordnungs-Graph (1)

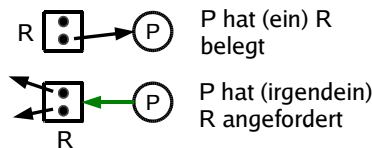
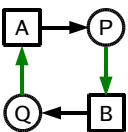
- Belegung und (noch unerfüllte) Anforderung grafisch darstellen:



- Variante für Ressourcen, die mehrfach vorkommen können



- P, Q aus Minimalbeispiel



- Deadlock = Kreis im Graph

- Kreis nur notwendig für Deadlock

# Deadlock-Erkennung (2)

## Algorithmus

- Suche einen unmarkierten Prozess  $P_i$ , dessen verbleibende Anforderungen vollständig erfüllbar sind, also  $R_{ij} \leq A_j$  für alle  $j$
- Gibt es keinen solchen Prozess, beende den Algorithmus
- Ein solcher Prozess könnte erfolgreich abgearbeitet werden. Simuliere die Rückgabe aller belegten Ressourcen:  $A := A + C_i$  ( $i$ -te Zeile von  $C$ )  
Markiere den Prozess – er ist nicht Teil eines Deadlocks
- Weiter mit Schritt 1

## Deadlock-Behebung

### Entziehen einer Ressource?

In den Fällen, die wir betrachten, unmöglich (ununterbrechbare Ressourcen)

### Abbruch eines Prozesses, der am Deadlock beteiligt ist

**Rücksetzen** eines Prozesses in einen früheren Prozesszustand, zu dem die Ressource noch nicht gehalten wurde (erfordert regelmäßiges Sichern der Prozesszustände)

# Deadlock-Vermeidung (1)

## Deadlock Avoidance (Vermeidung)

- **Idee:** BS erfüllt Ressourcenanforderung nur dann, wenn dadurch auf keinen Fall ein Deadlock entstehen kann
- Das funktioniert nur, wenn man die **Maximalforderungen aller Prozesse** kennt
  - Prozesse registrieren **beim Start** für alle denkbaren Ressourcen ihren Maximalbedarf
  - für die Praxis i. d. R. irrelevant, nur in wenigen Spezialfällen nützlich

## Sichere vs. unsichere Zustände

- Ein Zustand heißt **sicher**, wenn es eine Ausführreihenfolge der Prozesse gibt, die auch dann keinen Deadlock verursacht, wenn alle Prozesse sofort ihre maximalen Ressourcen-forderungen stellen.
- Ein Zustand heißt **unsicher**, wenn er nicht sicher ist.
- Unsicher bedeutet nicht zwangsläufig Deadlock!

# Deadlock-Verhinderung (1)

## Deadlock-Verhinderung (prevention):

Vorbeugendes Verhindern

- mache mindestens eine der vier Deadlock-Bedingungen unerfüllbar, dann sind keine Deadlocks mehr möglich (denn die vier Bedingungen sind notwendig)

### 1. Gegenseitiger Ausschluss

- Ressourcen nur exklusiv zuteilen, wenn es keine Alternative gibt
- Beispiel: Statt mehrerer konkurrierender Prozesse, die einen gemeinsamen Drucker verwenden wollen, einen Drucker-Spooler einführen
  - keine Konflikte mehr bei Zugriff auf Drucker
  - aber: Problem evtl. nur verschoben (Größe des Spool-Bereichs bei vielen Druckjobs begrenzt?)

# Deadlock-Vermeidung (2)

## Banker-Algorithmus

- Datenstrukturen wie bei Deadlock-Erkennung:
  - $n$  Prozesse  $P_1 \dots P_n$ ,  $m$  Ressourcentypen  $R_1 \dots R_m$  mit je  $E_i$  Ressourcen-Instanzen ( $i=1, \dots, m$ ) → **Ressourcenvektor**  $E = (E_1 \dots E_m)$
  - **Ressourcenrestvektor**  $A$  (wie viele sind noch frei?)
  - **Belegungsmatrix**  $C$   
 $C_{ij}$  = Anzahl Ressourcen vom Typ  $j$ , die Prozess  $i$  belegt
  - **Maximalbelegung**  $Max$ :  
 $Max_{ij}$  = max. Bedarf, den Prozess  $i$  an Ressource  $j$  hat
  - **Maximale zukünftige Anforderungen:**  $R = Max - C$ ,  
 $R_{ij}$  = # Ress. vom Typ  $j$ , die Prozess  $i$  noch maximal anfordern kann

Feststellen, ob ein Zustand sicher ist = Annehmen, dass alle Prozesse sofort ihre Maximalforderungen stellen, und dies auf Deadlocks überprüfen

# Deadlock-Verhinderung (2)

### 2. Hold and Wait

- Alle Prozesse müssen die benötigten Ressourcen gleich beim Prozessstart anfordern (und blockieren)
- hat verschiedene Nachteile:
  - Ressourcen-Bedarf entsteht oft dynamisch (ist also beim Start des Prozesses nicht bekannt)
  - verschlechtert Parallelität (Prozess hält Ressourcen über einen längeren Zeitraum)

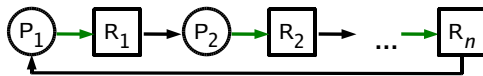
### 3. Ununterbrechbarkeit der Ressourcen

- Ressourcen entziehen?
- siehe Deadlock-Behebung (Abbruch / Rücksetzen)

# Deadlock-Verhinderung (3)

## 4. Zyklisches Warten (1)

- Ressourcen durchnummerieren
  - $ord: R = \{R_1, \dots, R_n\} \rightarrow \mathbb{N}, ord(R_i) \neq ord(R_j)$  für  $i \neq j$
- Prozess darf Ressourcen nur in der durch  $ord$  vorgegebenen Reihenfolge anfordern
- Das macht Deadlocks unmöglich. (Widerspruchsbeweis)
- Problem: Gibt es eine feste Reihenfolge der Ressourcenbelegung, die für alle Prozesse geeignet ist?
- reduzierte Parallelität (Ressourcen zu früh belegt)



```

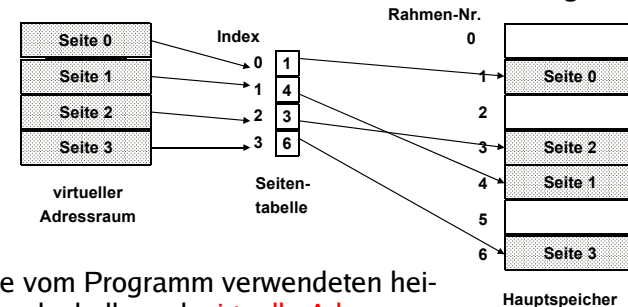
Sep 19 14:27:41 amd64 ssshd[2997]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62522
Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[30103]: (root) CMD (/sbin/evlogmgr -c "age > *30d**")
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6541]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6604]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62314
Sep 20 15:27:35 amd64 sshd[9077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10192]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63975
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17058]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17878]: (root) CMD (/sbin/evlogmgr -c "age > *30d**")
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[131088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:39 amd64 sshd[131269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[31]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[24739]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:23:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[24739]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[28555]: (root) CMD (/sbin/evlogmgr -c "age > *30d**")
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted publickey for esser from ::ffff:87.234.201.207 port 62105
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c "age > *30d**")
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[12098]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: amd_osa: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[4621]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c "age > *30d**")
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8891]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9172]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11633]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778
    
```

# Virtuelle Speicherverwaltung (Paging)

- Aufteilung des Adressraums in **Seiten (pages)** fester Größe und des Hauptspeichers in **Seitenrahmen (page frames)** gleicher Größe.
- Der lineare, zusammenhängende Adressraum eines Prozesses („virtueller Adressraum“) wird auf beliebige, nicht zusammenhängende Seitenrahmen abgebildet.
- **Eine** einzige Liste freier Seitenrahmen verwalten
- Berechnung der **physikalischen** Speicheradresse aus der vom Programm angegebenen **virtuellen** Adresse
  - zur Laufzeit des Programms, transparent für das Programm,
  - muss von der Hardware unterstützt werden.
- **Vorteile** der virtuellen Speicherverwaltung:
  - Einfache Zuteilung von Hauptspeicher.
  - Keine externe Fragmentierung, geringe interne Fragmentierung.
  - Kein Aufwand für den Programmierer.

# Virtueller Adressraum

- Beim Paging wird der Zusammenhang zwischen Programmadresse und physikalischer Hauptspeicheradresse erst zur Laufzeit mit Hilfe der Seitentabellen hergestellt.

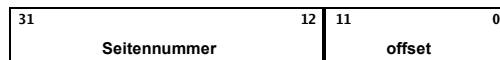


- Die vom Programm verwendeten heißen deshalb auch **virtuelle Adressen**.
- Der **virtuelle Adressraum** eines Programms ist der **lineare, zusammenhängende Adressraum**, der dem Programm zur Verfügung steht.

## Adressübersetzung beim Paging (1)

- Programmadresse in zwei Teile aufteilen:  
Seitennummer + relative Adresse (Offset) in der Seite

Beispiel: 32-bit-Adresse bei einer Seitengröße von 4 KB, also 4096 ( $=2^{12}$ ) Byte:



- Für jeden Prozess gibt es eine **Seitentabelle** (page table). Diese enthält für jede Prozess-Seite
  - eine Angabe, ob die Seite im Speicher ist,
  - die Nummer des Seitenrahmens im Hauptspeicher, der die Seite enthält.
- Ein spezielles Register enthält die Anfangsadresse der Seitentabelle für den aktuellen Prozess.
- Seitennummer als Index in die Seitentabelle verwenden.

## Aufgabenbeispiel

Paging mit folgenden Parametern:

- 32-Bit-Adressbus
- 32 KB Seitengröße
- 64 MB RAM

- a) 32 KB (Seitengröße) =  $2^5 \times 2^{10}$  Byte =  $2^{15}$  Byte  
d.h.: Offset ist 15 Bit lang



Zu berechnen:

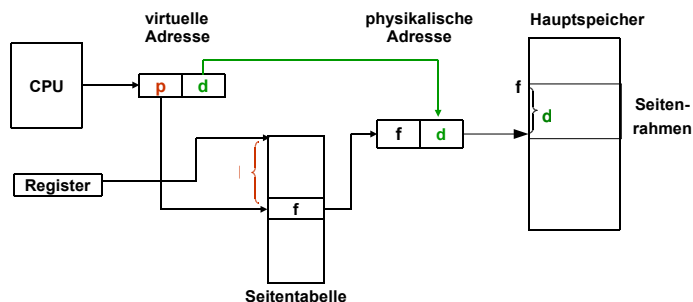
- maximale Anzahl der adressierbaren virtuellen Seiten
- Größe der erforderlichen Seitentabelle (in MB)

Also gibt es  $2^{17}$  virtuelle Seiten

- b) Zur Seitentabelle:  
In 64 MB RAM passen  $64 \text{ M} / 32 \text{ K} = 2 \text{ K} = 2048$  ( $2^{11}$ ) Seitenrahmen  
Ein Eintrag in der Seitentabelle benötigt darum 11 Bit, in der Praxis 2 Byte.

=> Platzbedarf:  
#(virt. Seiten) x Größe(Eintrag)  
 $= 2^{17} \times 2 \text{ Byte} = 2^{18} \text{ Byte} = 256 \text{ KB} = \frac{1}{4} \text{ MB}$

## Adressübersetzung beim Paging (2)



- Für jeden Hauptspeicherzugriff wird ein zusätzlicher Hauptspeicherzugriff auf die Seitentabelle benötigt. Dies muss durch Caches in der Hardware beschleunigt werden!
- Seite nicht im Speicher -> **page fault (Seitenfehler)** auslösen.

## Virtueller Speicher allgemein (1)

- Mehr Prozesse können effektiv im Speicher gehalten werden  
→ **bessere Systemauslastung**
- Ein Prozess kann viel **mehr Speicher** anfordern als **physikalisch verfügbar**
- **allgemeiner Vorgang:**
  - Nur Teile des Prozesses befinden sich im physikalischen Speicher
  - falls Zugriff auf eine Adresse, die ausgelagert ist:
    - BS setzt den Prozess auf blockiert
    - BS setzt eine Disk-I/O-Leseanfrage ab
    - Nach Laden des fehlenden Stücks (Seite oder Segment) wird ein I/O-Interrupt abgesetzt
    - BS setzt Prozess zuletzt wieder in den Bereit-Zustand

## Virtueller Speicher allgemein (2)

- „*thrashing*“ (siehe später): Prozessor verbringt die meiste Zeit mit Ein- und Auslagern von Prozessteilen statt mit der Ausführung von Prozesanweisungen
- **Lokalitätsprinzip:**
  - Zugriffe auf Daten und Programmcode häufig lokal gruppiert;  
→ Annahme gerechtfertigt, dass nur wenige Prozessstücke während einer kurzen zeitlichen Periode gleichzeitig vorgehalten werden müssen

## Invertierte Seitentabellen

- Bei großem virtuellen Speicher sehr viele Einträge in der Seitentabelle nötig, z.B.  $2^{32}$  Byte Adressraum, 4 Kbyte/Seite  
→ über 1 Mio. Seiteneinträge, also Tabelle >4 MByte (je Prozess)
- Platz sparen durch invertierte Seitentabellen:
  - normal: ein Eintrag pro (virtueller) Seite mit Verweis auf den Seitenrahmen (im Hauptspeicher)
  - invertiert: ein Eintrag pro *Seitenrahmen* mit Verweis auf Tupel (Prozess-ID, virtuelle Seite)
- Problem: Suche zu Prozess  $p$  und seiner Seite  $n$  nach dem Eintrag  $(p,n)$  in der invertierten Tabelle → langwierig
- Auch hier TLB einsetzen, um auf „meist genutzte“ Seiten schnell zugreifen zu können
- Bei TLB-Miss hilft aber nichts: Suchen...
- Andere Lösung für Problem der großen Seitentabellen: Mehrstufiges Paging

## Translation Look-Aside Buffer

- **Translation Lookaside Buffer (TLB):** schneller **Hardware-Cache**, mit den zuletzt benutzten Seitentableneinträgen
- Treffer im TLB → Speicherzugriff auf Seitentabelle unnötig
- Fehltreffer → Zugriff auf die Seitentabelle (Alten Eintrag im TLB durch neuen ersetzen)
- Trefferquote (hit ratio) beeinflusst die durch-schnittliche Zeit einer Adressübersetzung.
- **Lokalitätsprinzip** → auch bei kleinen TLBs **hohe Trefferquoten**
- Inhalt des TLB ist **prozessspezifisch!** Zwei Möglichkeiten:
  - Jeder Eintrag im TLB hat „valid bit“. Bei **Prozesswechsel** (Context Switch) gesamten TLB **invalidieren**.
  - Jeder Eintrag im TLB enthält Prozessidentifikation (**PID**), die mit der PID des zugreifenden Prozesses verglichen wird.

## Mehrstufiges Paging (1)

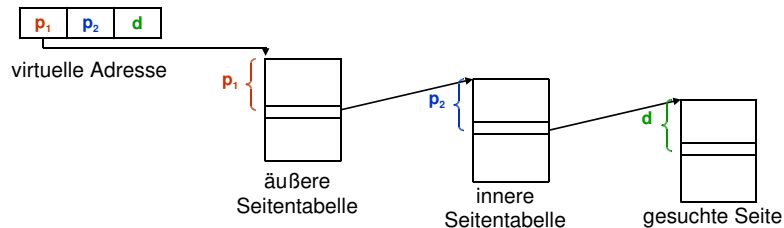
- **Zweistufiges Paging:**
  - Seitennummer noch einmal unterteilen, z. B.:  

31	22 21	12 11	0
p1	p2	offset	

←—————→  
Seitennummer
  - $p_1$ : Index in **äußere Seitentabelle**, deren Einträge jeweils auf eine **innere Seitentabelle** zeigen
  - $p_2$ : Index in eine der inneren Seitentabellen, deren Einträge auf Seitenrahmen im Speicher zeigen
  - Die **inneren Seitentabellen** müssen **nicht** alle **speicherresident** sein
- Analog dreistufiges Paging etc. implementieren

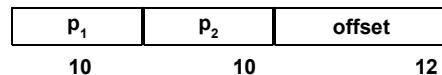


## Mehrstufiges Paging (2)



- Größe der Seitentabellen:

Beispiel:



- Äußere Tabelle: 1024 (210) Einträge, die auf (bis zu) 1024 innere Tabellen zeigen, die wieder je 1024 Einträge haben.
- 4 Byte pro Tabelleneintrag → Größe der Tabelle = 1 Seite
- nur so viele innere Seitentabellen verwenden, wie nötig.

## Seitenersetzung

- Wenn bei einem Page Fault **kein freier Seitenrahmen** zur Verfügung steht, muss das Betriebssystem einen frei machen.
- Algorithmus wählt nach einer Strategie diesen Rahmen aus.
- Falls die zu ersetzende Seite, seit sie zuletzt (aus dem Swap) in den Speicher geholt wurde, verändert wurde, muss ihr aktueller Inhalt gesichert werden:
  - Ein **modify bit (dirty bit)** vermerkt, ob Seite verändert wurde.
  - veränderte Seite auf Platte sichern (**Page-/Swap-Bereich**).
- Eine unveränderte Seite kann später - bei Bedarf - wieder von der alten Stelle auf der Platte geladen werden.
- Im Seitentableneintrag für die ersetzte Seite **valid bit** löschen und merken, von wo die Seite wieder geladen werden kann.
- Seiteneretzungsstrategien: So wenig page faults wie möglich
- Lokale vs. globale Ersetzung

## Demand Paging

- Der Adressbereich eines Prozesses muss nicht vollständig im Hauptspeicher sein.
  - **Lokalitätsprinzip**: Prozess spricht in einer Zeitspanne nur relativ wenige, nahe beieinanderliegende Adressen an.
  - Teile des Programms werden bei einem bestimmten Ablauf evtl. nicht benötigt (Spezialfälle, Fehlerbehandlung etc.).
- **Demand Paging** bedeutet
  - dass eine Seite nur dann in den Speicher geladen wird, wenn der Prozess sie anspricht,
  - Seite kann auch wieder aus dem Speicher entfernt werden
- Vorteile von Demand Paging:
  - Der Adressbereich eines Prozesses kann größer sein als der physikalische Hauptspeicher.
  - Prozesse belegen weniger Platz im Hauptspeicher, somit können mehr Prozesse gleichzeitig aktiv sein.

## Optimale Strategie / FIFO

**Optimale Strategie:** Diejenige Seite ersetzen, auf die in Zukunft am längsten nicht zugegriffen wird.

- Vorteil: Diese Strategie verursacht die kleinste Zahl an Page Faults.
- Nachteil: Diese Strategie ist nicht implementierbar.

Die optimale Strategie kann modellhaft zur Bewertung anderer Strategien benutzt werden.

**First In, First Out (FIFO):** Die Seite ersetzen, die schon am längsten im Speicher ist.

- Vorteil: Sehr einfach zu implementieren:
- Nachteil: Die ersetzte Seite kann in dauernder Benutzung sein und gleich wieder angefordert werden.

# Least Recently Used (LRU)

Die Seite ersetzen, die **am längsten nicht benutzt** worden ist.

- **Vorteil:** In der Regel weniger Page Faults als FIFO.
- **Nachteil:** Aufwändige Implementierung.

Zwei mögliche Implementierungen: mit **Zähler** oder **verketteter Liste**

```
...
Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[2978]: (root) CMD (/sbin/evlogmgr -c *severity=DBGU*)
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[3010]: (root) CMD (/sbin/evlogmgr -c *age > *30d**)
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6516]: Accepted rsa for esser from :ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd[6609]: Accepted rsa for esser from :ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6694]: Accepted rsa for esser from :ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sshd[9077]: Accepted rsa for esser from :ffff:87.234.201.207 port 64242
Sep 20 15:37:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10102]: Accepted rsa for esser from :ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10101]: Accepted rsa for esser from :ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c *severity=DBGU*)
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17078]: (root) CMD (/sbin/evlogmgr -c *age > *30d**)
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31088]: Accepted rsa for esser from :ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:39 amd64 sshd[31269]: Accepted rsa for esser from :ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c *severity=DBGU*)
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[5489]: (root) CMD (/sbin/evlogmgr -c *age > *30d**)
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:23:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[247]: (root) CMD (/sbin/evlogmgr -c *age > *30d**)
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[25]: (root) CMD (/sbin/evlogmgr -c *age > *30d**)
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[596]: Accepted publickey for esser from :ffff:192.168.1.5 port 5977; sshd
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:14 amd64 sshd[606]: Accepted rsa for esser from :ffff:87.234.201.207 port 62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c *severity=DBGU*)
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c *age > *30d**)
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from :ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[3197]: Accepted rsa for esser from :ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: md_seq_mid_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: md_seq_oss: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from :ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[662]: (root) CMD (/sbin/evlogmgr -c *severity=DBGU*)
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c *age > *30d**)
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from :ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from :ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from :ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11954]: Accepted rsa for esser from :ffff:87.234.201.207 port 62922
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from :ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from :ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from :ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from :ffff:87.234.201.207 port 62778

```

# 7. Dateisysteme

## Referenz-Bits

- Jeder Seitentableneintrag kann ein **Referenz-Bit** enthalten
  - das bei einem Zugriff auf die Seite gesetzt wird (Hardware),
  - das nach bestimmten Kriterien gelöscht wird (Software).
- Ein Referenz-Bit
  - liefert die Information, ob auf eine Seite seit dem letzten Löschen des Bits zugegriffen wurde,
  - sagt nichts über den Zeitpunkt des Zugriffs auf eine Seite aus,
  - sagt nichts über die Reihenfolge der Zugriffe auf mehrere Seiten aus.
- Mit Referenz-Bits kann man weitere Seitenersetzungsstrategien implementieren, z. B.
  - Modifikation von LRU, die weniger aufwändig ist: binärer Zähler mit Aging
  - **Second-Chance-Algorithmus**, eine Verbesserung der FIFO-Strategie.

## Dateisysteme: Cluster-Allokation (1)

- **contiguous allocation** (zusammenhängende Allokation)
  - externe Fragmentierung (verlangt Defragm.)
  - gut für sequent. und direkten Zugriff
  - Datei vergrößern schwierig
- **chained allocation** (Ketten-Allokation)
  - jeder Cluster enthält Verweis auf nächsten
  - keine externe Fragmentierung
  - nur sequentieller Zugriff möglich
  - schlechtere Performance als bei zusammenh. All.

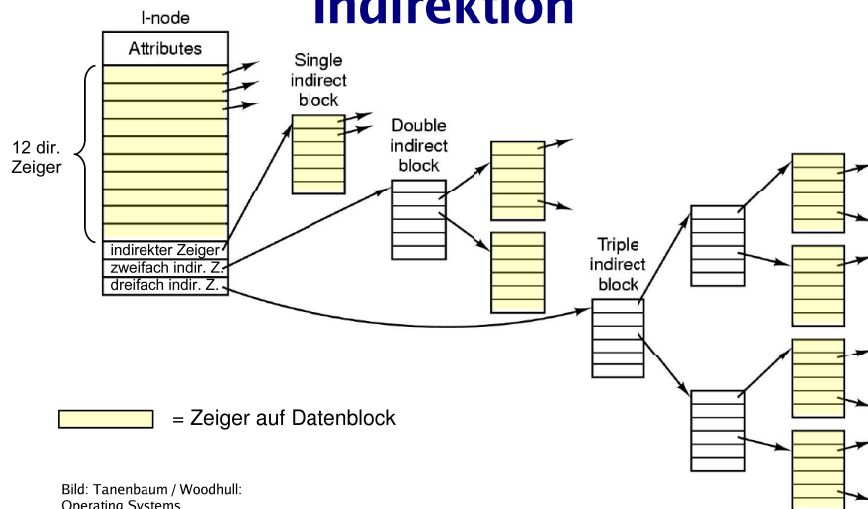
## Dateisysteme: Cluster-Allokation (2)

- **indexed allocation** (Index-Allokation)
  - Index-Cluster enthalten Adressen von Daten-Clustern
  - keine externe Fragmentierung
  - sequentieller und direkter Zugriff möglich
  - schlechtere Performance als bei zusammenh. Allok.
- **indexed allocation with variable-lengths extents** (Index-A. mit Extents beliebiger Länge)
  - Index-Block listet nicht alle Datenblöcke, sondern Anfangsblocknummer und Länge auf
  - Mischung aus zus.-hängend/nicht zus.-hängend

## Indirektion und Dateigrößen

- Beim Erstellen des Dateisystems unterschiedliche Blockgrößen möglich:
  - $\times 2$  1024-Byte-Blöcke -> max. Größe 16 GByte
  - $\times 2$  2048-Byte-Blöcke -> max. Größe 256 GByte
  - $\times 2$  4096-Byte-Blöcke -> max. Größe 4096 GByte (4 TB)
- Bei großen Dateien: Speichern der Blocknummern in Indirektionsblöcken (bis zu 3 Stufen):  
 Verdoppeln der Blockgröße = Ver-16-fachen ( $2^4$ ) der Blocknummern: Faktor  $2^3$  aus Indirektion, Faktor 2 aus Blockgröße

## Inodes und Blockadressen: Indirektion



## Aufgabenbeispiel

Dateisystem mit folgenden Parametern:

- Dateisystemgröße: **16 GB**

- Blockgröße: **32 KB**

- I-Node enthält:

- **9 direkte** Verweise
- **4 1-fach indirekte** Verweise
- **2 2-fach indirekte** Verweise
- **1 3-fach indirekter** Verweis

Zu berechnen:

a) Größe der Blockadresse

b) # Adressen pro Block

c) Maximale Dateigröße, die ein Inode zulässt

a) Dateisystem: 16 GB, Block: 32 KB  
 -> Es gibt  $16 \text{ G} / 32 \text{ K} = 0,5 \text{ M} = 512 \text{ K} = 2^{19}$  Blöcke

Blockadressen sind also mind. 19 Bit lang, in der Praxis: 4 Byte (kleinste 2er Potenz, in die 19 Bit passen)

b) Blockgröße / Adressgröße =  $32 \text{ KB} / 4 \text{ Byte} = 8 \text{ K} = 8192 = 2^{13}$

c) # (adressierb. Blöcke) x Blockgröße =  
 $(9 + 4 \times 8192 + 2 \times 8192^2 + 1 \times 8192^3) \times 32 \text{ KB}$   
 $= 17.596.482.060.576 \text{ KB} \approx 16.781.313 \text{ GB}$   
 $\approx 16.388 \text{ Terabyte}$  (sehr groß)

## Disk-Scheduler

- Ziel: Seek-Time und rotational delay reduzieren
- **FIFO**-Disk-Scheduler
  - Anfragen in Reihenfolge des Eintreffens
  - fair; brauchbar bei wenig parallelen Anfragen
- **SSTF** (Shortest Service Time First) Disk-Scheduler
  - Anfrage auswählen, die die geringste Bewegung der Schreib-/Leseköpfe verursacht
  - minimiert Seek-Time
  - Anfrage muss ggf. lange auf Bearbeitung warten
- **SCAN**-Disk-Scheduler
  - Köpfe in eine Richtung bewegen; alle Anfragen „auf dem Weg“ erfüllen; dann Richtungswechsel
  - **C-SCAN**: nur eine Richtung

## Und das war's auch schon...

- Wir sehen uns zur Prüfung ... und bei der Klausureinsicht
- Viel Erfolg beim Lernen!
- Ergebnisse der Evaluation: <http://hm.hgesser.de/>
- Wer mag: Ich habe auch ein MeinProf-Profil
  
- **Nicht vergessen:** Fragen bis zum Klausurtermin jederzeit
  - per Mail: [hans-georg.esser@hm.edu](mailto:hans-georg.esser@hm.edu)
  - Antworten kommen per Mail
  - ... und landen im Prüfungs-Blog: <http://hm.hgesser.de/blog>

## Journaling

- Schreibzugriffe auf Dateisystem führen immer auch zu Änderungen an den Metadaten
- fehlerhafte Metadaten = Inkonsistenz des Dateisystems
- Journaling: Metadaten-Änderungen vor der Änderung protokollieren → erlaubt spätere schnelle Recovery
- verwandt mit Transaktionskonzept in Datenbanken
- es gibt auch vollständiges Journaling (neben Änderungen an Metadaten auch solche an Daten); ist aber langsam