

```

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[6102]: 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[6102]: 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[10102]: 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[54993]: (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 18:43:26 amd64 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 /usr/sbin/cron[54993]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[14844]: (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 /usr/sbin/cron[14844]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 23 01:00:01 amd64 /usr/sbin/cron[13253]: (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[13253]: (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 59771 sshd
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[20988]: 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[19371]: 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 kernel: snd_seq_seq: 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[14844]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 25 02:00:02 amd64 /usr/sbin/cron[662]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
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[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

```

5. Synchronisation (3)

5. Synchronisation

5.1 Einführung

5.2 Kritische Abschnitte

5.3 Synchr.-Methoden

5.4 Deadlocks

```

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[6102]: 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[6102]: 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[10102]: 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[54993]: (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 18:43:26 amd64 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 /usr/sbin/cron[54993]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 22 02:00:01 amd64 /usr/sbin/cron[14844]: (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 /usr/sbin/cron[14844]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 23 01:00:01 amd64 /usr/sbin/cron[13253]: (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[13253]: (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 59771 sshd
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[20988]: 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[19371]: 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 kernel: snd_seq_seq: 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[14844]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 25 02:00:02 amd64 /usr/sbin/cron[662]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
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[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

```

Linux. Synchronisation im Kernel (Fortsetzung)

Reader Writer Locks (1)

- Alternative zu normalen Locks, die mehrere Lesezugriffe zulässt – bei schreibendem Zugriff aber exklusiv (wie ein normales Lock) ist

```

rwlock_t xy_rwlock = RW_LOCK_UNLOCKED;

```

Lesender Code	Schreibender Code
<pre> read_lock (&xy_rwlock)) { /* kritischer Abschnitt, read-only */ read_unlock (&xy_rwlock); </pre>	<pre> write_lock (&xy_rwlock)) { /* kritischer Abschnitt, read & write */ write_unlock (&xy_rwlock); </pre>

- Nur bei klarer Trennung zwischen lesenden / schreibenden Programmteilen!

Hans-Georg Eßer, Hochschule München Betriebsysteme I, SS 2008 5. Synchronisation (2) – Folie 3

Reader Writer Locks (2)

- | | Es gibt schon einen Leser | Es gibt schon einen Schreiber | Noch keine Sperre |
|-----------------------------|---------------------------|-------------------------------|-------------------|
| read_lock(&lck) | erfolgreich | schlägt fehl | erfolgreich |
| write_lock(&lck) | schlägt fehl | schlägt fehl | erfolgreich |
- Auch hier Varianten für Interrupt-Behandlung:
 - read_lock_irq** **read_unlock_irq**
 - read_lock_irqsave** **read_unlock_irqrestore**
 - write_lock_irq** **write_unlock_irq**
 - write_lock_irqsave** **write_unlock_irqrestore**

Hans-Georg Eßer, Hochschule München Betriebsysteme I, SS 2008 5. Synchronisation (2) – Folie 4

Semaphore (1)

- Kernel-Semaphore sind „schlafende“ Locks
- Ist ein Semaphor schon gelockt, werden weitere Interessenten in eine Warteschlange eingereiht.
- Bei Freigabe eines Semaphors wird der erste wartende Thread in der Warteschlange geweckt
- Semaphore eignen sich für Sperren, die über einen längeren Zeitraum gehalten werden
 - keine Verschwendung von Rechenzeit

Semaphore (3)

Typ: *semaphore*

Statische Deklaration

```
static DECLARE_SEMAPHORE_GENERIC (name, count);  
static DECLARE_MUTEX (name);          /* count=1 */
```

Dynamische Semaphor-Erzeugung

```
sema_init (&sem, count);  
init_MUTEX (&sem);                    /* count=1 */
```

- Verwendung mit *up()* und *down()*

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

Semaphore (2)

- Semaphore sind nur im Prozess-Kontext einsetzbar, nicht in Interrupt-Handlern (Interrupt-Handler werden nicht vom Scheduler behandelt)
- Code, der einen Semaphor verwenden will, darf nicht bereits ein normales Lock besitzen (Semaphor-Zugriff kann dazu führen, dass der Thread sich schlafen legt.)
- Semaphore können auch mehr als einen Thread auf die Ressource zugreifen lassen

Semaphore (4)

- Varianten von *down()*
 - *down (&sem);*
nicht unterbrechbarer Schlaf, falls Semaphor nicht verfügbar
 - *down_interruptible (&sem);*
unterbrechbarer Schlaf, falls Sem. nicht verfügbar
 - *down_trylock (&sem);*
versucht, den Semaphor zu erhalten – falls das nicht gelingt, kehrt die Funktion sofort mit False-Wert zurück

Semaphore (5)

- Beispiel für `down_trylock()`

```
/* Auszug aus /usr/src/linux/kernel/printk.c */

if (!down_trylock(&console_sem)) {
    console_locked = 1;
    /*
     * We own the drivers. We can drop the spinlock and let
     * release_console_sem() print the text
     */
    spin_unlock_irqrestore(&logbuf_lock, flags);
    console_may_schedule = 0;
    release_console_sem();
    /* Funktion release_console_sem() führt up(&console_sem); aus */
} else {
    /*
     * Someone else owns the drivers. We drop the spinlock, which
     * allows the semaphore holder to proceed and to call the
     * console drivers with the output which we just produced.
     */
    spin_unlock_irqrestore(&logbuf_lock, flags);
}
```

Reader-Writer-Semaphore (2)

```
static DECLARE_RWSEM (xy_rwsem);
```

Lesender Code

```
down_read (&xy_rwsem) ) {
    /* kritischer Abschnitt,
     read-only */
    up_read (&xy_rwsem);
```

Schreibender Code

```
down_write (&xy_rwsem) ) {
    /* kritischer Abschnitt,
     lesen und schreiben */
    up_write (&xy_rwsem);
```

Genau wie bei Reader Writer Locks:

	Es gibt schon einen Leser	Es gibt schon einen Schreiber	Noch keine Sperre
<code>down_read(&sem)</code>	erfolgreich	schlägt fehl	erfolgreich
<code>down_write(&sem)</code>	schlägt fehl	schlägt fehl	erfolgreich

Reader-Writer-Semaphore (1)

- Analog zu Reader Writer Locks: Typ `rw_semaphore`, der spezielle Up- und Down-Operationen für Lese- und Schreibzugriff erlaubt
- Alle Reader-Writer-Semaphore sind Mutexe (Zähler ist bei Initialisierung immer 1)

Statische Deklaration

```
static DECLARE_RWSEM (name);
```

Dynamische Semaphore-Erzeugung

```
init_rwsem (&sem);
```

```
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[6691]: 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[10120]: 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[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[10188]: 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[11269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64397
Sep 21 19:43:26 amd64 sshd[10188]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64397
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=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[5499]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 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 "severity=DEBUG")
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6564]: Accepted publickey for esser from ::ffff:192.168.1.9 port 59771 ssh
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[12461]: (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[120998]: 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[12321]: 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_oss: 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[12939]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62556
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 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[889]: 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 11:00:02 amd64 sshd[9170]: 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 14:07:37 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:03 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
```

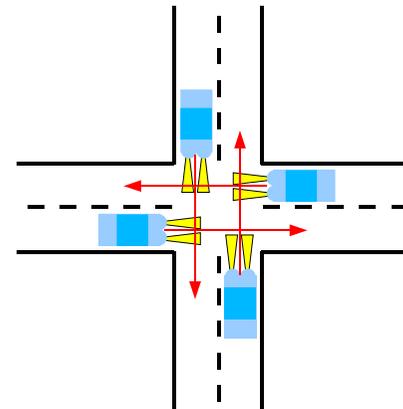
Deadlocks

Deadlocks – Gliederung

- Einführung
- Ressourcen-Typen
- Hinreichende und notwendige Deadlock-Bedingungen
- Deadlock-Erkennung und -Behebung
- Deadlock-Vermeidung (avoidance): Banker-Algorithmus
- Deadlock-Verhinderung (prevention)

Deadlock: Rechts vor Links (1)

- Der Klassiker: Rechts-vor-Links-Kreuzung

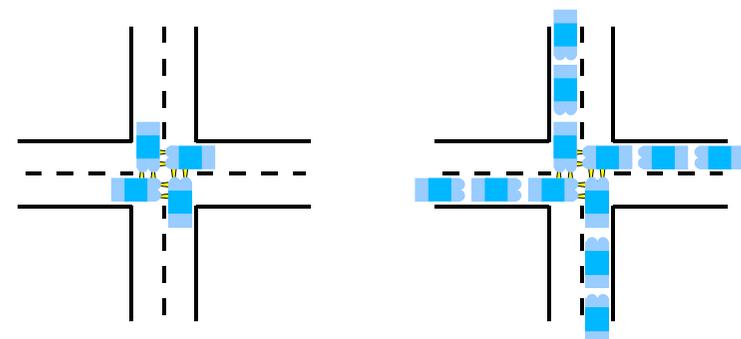


Wer darf fahren?
Potenzieller Deadlock

Was ist ein Deadlock?

- Eine Menge von Prozessen befindet sich in einer **Deadlock-Situation**, wenn:
 - jeder Prozess auf eine Ressource wartet, die von einem anderen Prozess blockiert wird
 - keine der Ressourcen freigegeben werden kann, weil der haltende Prozess (indem er selbst wartet) blockiert ist
- In einer Deadlock-Situation werden also die Prozesse dauerhaft verharren
- Deadlocks sind unbedingt zu vermeiden

Deadlock: Rechts vor Links (2)

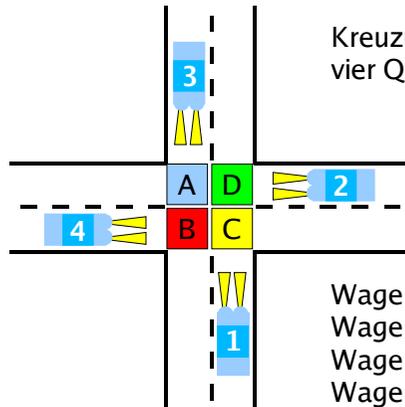


Deadlock, aber behebbar:
eines oder mehrere Autos
können zurücksetzen

Deadlock, nicht behebbar:
beteiligte Autos können
nicht zurücksetzen

Deadlock: Rechts vor Links (3)

Analyse:



Kreuzungsbereich besteht aus vier Quadranten A, B, C, D

Wagen 1 benötigt C, D
 Wagen 2 benötigt D, A
 Wagen 3 benötigt A, B
 Wagen 4 benötigt B, C

Deadlock: kleinstes Beispiel (1)

- Zwei Locks A und B
 - z. B. A = Scanner, B = Drucker, Prozesse P, Q wollen beide eine Kopie erstellen
- Locking in verschiedenen Reihenfolgen

Prozess P	Prozess Q	Problematische Reihenfolge:
<code>lock (A);</code>	<code>lock (B);</code>	P: <code>lock(A)</code>
<code>lock (B);</code>	<code>lock (A);</code>	Q: <code>lock(B)</code>
<i>/* krit. Bereich */</i>	<i>/* krit. Bereich */</i>	P: <code>lock(B)</code> <- blockiert
<code>unlock (A);</code>	<code>unlock (B);</code>	Q: <code>lock(A)</code> <- blockiert
<code>unlock (B);</code>	<code>unlock (A);</code>	

Deadlock: Rechts vor Links (4)

```

wagen_3 () {
    lock(A);
    lock(B);
    go();
    unlock(A);
    unlock(B);
}

wagen_2 () {
    lock(D);
    lock(A);
    go();
    unlock(D);
    unlock(A);
}

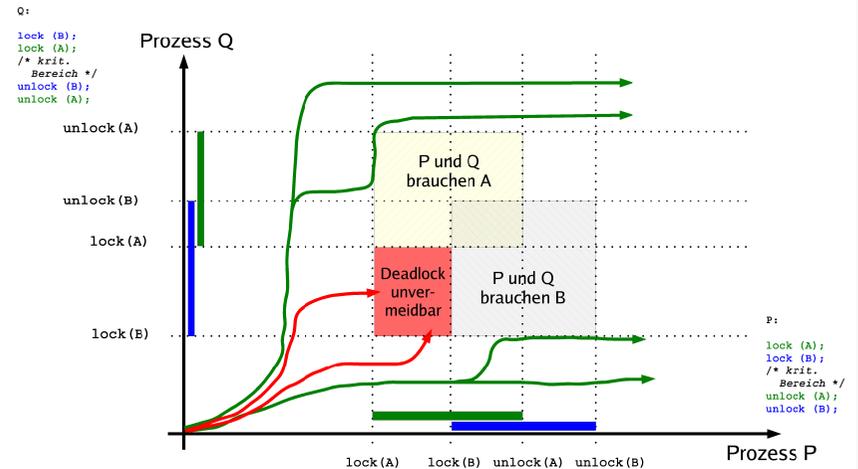
wagen_4 () {
    lock(B);
    lock(C);
    go();
    unlock(B);
    unlock(C);
}

wagen_1 () {
    lock(C);
    lock(D);
    go();
    unlock(C);
    unlock(D);
}
    
```

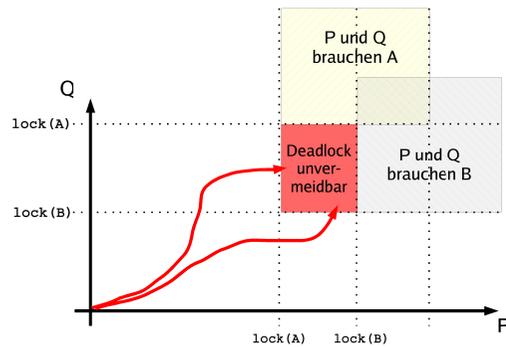
Problematische Reihenfolge:

- w1: `lock(C)`
- w2: `lock(D)`
- w3: `lock(A)`
- w4: `lock(B)`
- w1: `lock(D)` <- blockiert
- w2: `lock(A)` <- blockiert
- w3: `lock(B)` <- blockiert
- w4: `lock(C)` <- blockiert

Deadlock: kleinstes Beispiel (2)



Deadlock: kleinstes Beispiel (3)



Programmverzahnungen, die zwangsläufig in den Deadlock führen:

oberer roter Weg:

Q: lock (B)

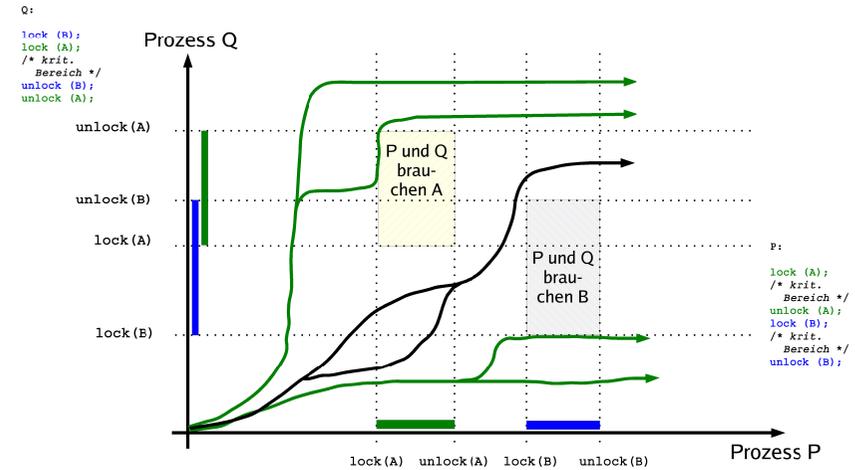
P: lock (A)

unterer roter Weg:

P: lock (A)

Q: lock (B)

Deadlock: kleinstes Beispiel (5)



Deadlock: kleinstes Beispiel (4)

- Problem beheben:
P benötigt die Locks nicht gleichzeitig

Prozess P

```
lock (A);
/* krit. Bereich */
unlock (A);
```

```
lock (B);
/* krit. Bereich */
unlock (B);
```

Prozess Q

```
lock (B);
lock (A);
```

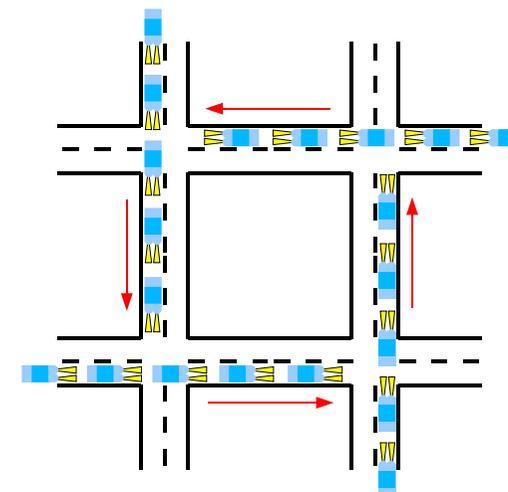
```
/* krit. Bereich */
```

```
unlock (B);
unlock (A);
```

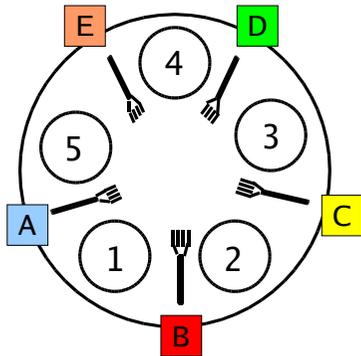
Jetzt kann kein Deadlock mehr auftreten

- Andere Lösung: P und Q fordern A, B in gleicher Reihenfolge an

Grid Lock



Fünf-Philosophen-Problem



Philosoph 1 braucht Gabeln A, B
Philosoph 2 braucht Gabeln B, C
Philosoph 3 braucht Gabeln C, D
Philosoph 4 braucht Gabeln D, E
Philosoph 5 braucht Gabeln E, A

Problematische Reihenfolge:

p1: lock (B)
p2: lock (C)
p3: lock (D)
p4: lock (E)
p5: lock (A)
p1: lock (A) <- blockiert
p2: lock (B) <- blockiert
p3: lock (C) <- blockiert
p4: lock (D) <- blockiert
p5: lock (E) <- blockiert

Ressourcen-Typen (2)

- nicht unterbrechbare Ressourcen
 - Betriebssystem kann Ressource nicht (ohne fehlerhaften Abbruch) entziehen – Prozess muss diese freiwillig zurückgeben
 - Beispiele:
 - DVD-Brenner (Entzug → zerstörter Rohling)
 - Tape-Streamer (Entzug → sinnlose Daten auf Band oder Abbruch der Bandsicherung wg. Timeout)
- Nur die *nicht* unterbrechbaren sind interessant, weil sie Deadlocks verursachen können

Ressourcen-Typen (1)

Zwei Kategorien von Ressourcen: unterbrechbar / nicht unterbrechbar

- unterbrechbare Ressourcen
 - Betriebssystem kann einem Prozess solche Ressourcen wieder entziehen
 - Beispiele:
 - CPU (Scheduler),
 - Hauptspeicher (Speicherverwaltung)
 - das kann Deadlocks vermeiden

Ressourcen-Typen (3)

- wiederverwendbare vs. konsumierbare Ressourcen
 - **wiederverwendbar**: Zugriff auf Ressource zwar exklusiv, aber nach Freigabe wieder durch anderen Prozess nutzbar (Platte, RAM, CPU, ...)
 - **konsumierbar**: von einem Prozess erzeugt und von einem anderen Prozess konsumiert (Nachrichten, Interrupts, Signale, ...)

Deadlock-Bedingungen (1)

1. Gegenseitiger Ausschluss (mutual exclusion)

- Ressource ist exklusiv: Es kann stets nur ein Prozess darauf zugreifen

2. Hold and Wait (besitzen und warten)

- Ein Prozess ist bereits im Besitz einer oder mehrerer Ressourcen, und
- er kann noch weitere anfordern

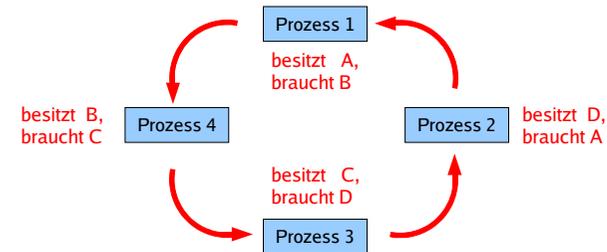
3. Ununterbrechbarkeit der Ressourcen

- Die Ressource kann nicht durch das Betriebssystem entzogen werden

Deadlock-Bedingungen (3)

4. Zyklisches Warten

- Man kann die Prozesse in einem Kreis anordnen, in dem jeder Prozess eine Ressource benötigt, die der folgende Prozess im Kreis belegt hat



Deadlock-Bedingungen (2)

- (1) bis (3) sind **notwendige** Bedingungen für einen Deadlock
- (1) bis (3) sind aber auch „wünschenswerte“ Eigenschaften eines Betriebssystems, denn:
 - gegenseitiger Ausschluss ist nötig für korrekte Synchronisation
 - Hold & Wait ist nötig, wenn Prozesse exklusiven Zugriff auf mehrere Ressourcen benötigen
 - Bei manchen Betriebsmitteln ist eine Präemption prinzipiell nicht sinnvoll (z. B. DVD-Brenner, Streamer)

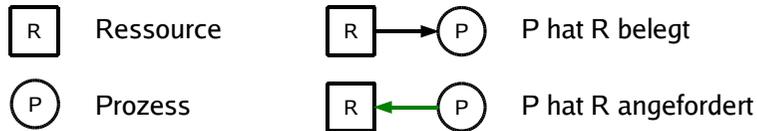
Deadlock-Bedingungen (4)

1. Gegenseitiger Ausschluss
2. Hold and Wait
3. Ununterbrechbarkeit der Ressourcen
4. Zyklisches Warten

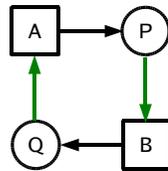
- (1) bis (4) sind **notwendige und hinreichende** Bedingungen für einen Deadlock
- Das zyklische Warten (4) (und dessen Unauflösbarkeit) sind Konsequenzen aus (1) bis (3)
- (4) ist der erfolgversprechendste Ansatzpunkt, um Deadlocks aus dem Weg zu gehen

Ressourcen-Zuordnungs-Graph (1)

- Belegung und (noch unerfüllte) Anforderung grafisch darstellen:

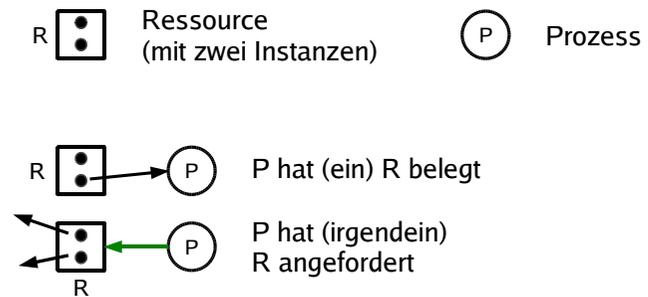


- P, Q aus Minimalbeispiel:
- Deadlock = Kreis im Graph



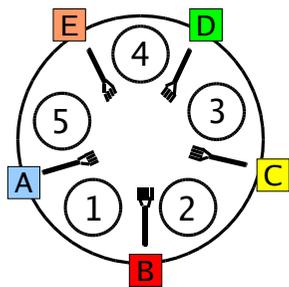
Ressourcen-Zuordnungs-Graph (3)

- Variante für Ressourcen, die mehrfach vorkommen können

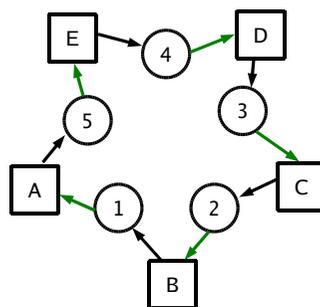


Ressourcen-Zuordnungs-Graph (2)

Philosophen-Beispiel

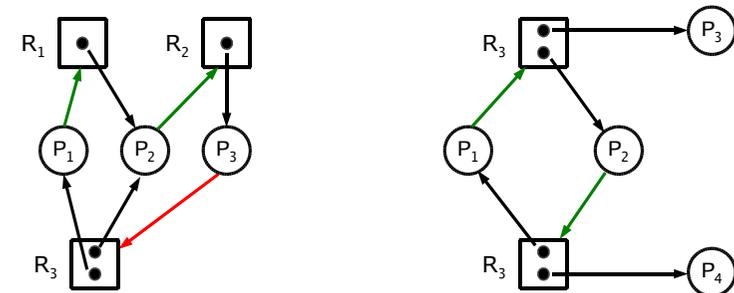


Situation, nachdem alle Philosophen ihre rechte Gabel aufgenommen haben



Ressourcen-Zuordnungs-Graph (4)

- Beispiele mit mehreren Instanzen



Mit roter Kante ($P_3 \rightarrow R_2$) gibt es einen Deadlock (ohne nicht)

Kreis, aber kein Deadlock – Bedingung ist nur **notwendig**, nicht hinreichend!