

```

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[6694]: 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[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 17:53:39 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[14674]: (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[1484]: (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[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 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[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 20:25:31 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[862]: (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[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

```

5 Synchronisation (5)

- 5. Synchronisation
- 5.4 Linux
- Prozesse
- Linux-Kernel

Benannte POSIX-Semaphore (1)

Benutzung wie bei Threads, aber durch Vergabe eines (system-einheitlichen) Namens auf Prozesse ausdehnbar

```
#include <semaphore.h>
posix_sem = sem_open("/MeinSemaphor", O_CREAT,
                    0644, POSIX_UNLOCKED);
```

→ erzeugt Eintrag in */dev/shm*:

```
$ ls -l /dev/shm/
-rw-r----- 1 esser users 16 2006-12-05 15:46 sem.MeinSemaphor
```

```

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[6693]: 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[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 17:53:39 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[14674]: (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[1484]: (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[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 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[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 20:25:31 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[862]: (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[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

```

Linux: Prozess-Synchronisation

Benannte POSIX-Semaphore (2)

Zum Kennenlernen: 4 einfache Testprogramme:

- *named-sem-init.c* Semaphore initialisieren benutzt *sem_open*, *sem_init*
- *named-sem-query.c* Semaphore-Wert abfragen *sem_open*, *sem_getvalue*
- *named-sem-wait.c* Semaphore erniedrigen (Wait-Operation), *sem_open*, *sem_wait*
- *named-sem-signal.c* Semaphore erhöhen (Signal-Operation), *sem_open*, *sem_post*

Benannte POSIX-Semaphore (3)

```
/* named-sem-init.c */
#include <semaphore.h>
#include <asm/fcntl.h>
#define POSIX_LOCKED 0
#define POSIX_UNLOCKED 1

sem_t *posix_sem;

main () {
    posix_sem = sem_open("/MeinSemaphor", O_CREAT, 0644, POSIX_UNLOCKED);
    sem_init(posix_sem, 0, 5); /* init: 5 */
}

$ ./named-sem-init
$ ls -l /dev/shm/
-rw-r----- 1 esser users 16 2006-12-05 15:46 sem.MeinSemaphor
$ hexdump /dev/shm/sem.MeinSemaphor
00000000 0005 0000 0000 0000 0000 0000 0000 0000
00000010
```

Benannte POSIX-Semaphore (5)

Signal-Operation

```
/* named-sem-signal.c */
#include <semaphore.h>
#include <asm/fcntl.h>

#define POSIX_LOCKED 0
#define POSIX_UNLOCKED 1

sem_t *posix_sem;

main () {
    posix_sem = sem_open("/MeinSemaphor", O_CREAT, 0644, POSIX_UNLOCKED);
    sem_post(posix_sem);
}

$ ./named-sem-query
Semaphor-Wert 5
$ ./named-sem-signal
$ ./named-sem-query
Semaphor-Wert 6
```

Benannte POSIX-Semaphore (4)

```
/* named-sem-query.c */
#include <semaphore.h>
#include <asm/fcntl.h>
#define POSIX_LOCKED 0
#define POSIX_UNLOCKED 1

sem_t *posix_sem;

main () {
    int ret;
    posix_sem = sem_open("/MeinSemaphor", O_CREAT, 0644, POSIX_UNLOCKED);
    sem_getvalue(posix_sem, &ret);
    printf("Semaphor-Wert %d \n", ret);
}

$ ./named-sem-query
Semaphor-Wert 5
```

Benannte POSIX-Semaphore (6)

Wait-Operation

```
/* named-sem-wait.c */
#include <semaphore.h>
#include <asm/fcntl.h>

#define POSIX_LOCKED 0
#define POSIX_UNLOCKED 1

sem_t *posix_sem;

main () {
    posix_sem = sem_open("/MeinSemaphor", O_CREAT, 0644, POSIX_UNLOCKED);
    sem_wait(posix_sem);
}

$ ./named-sem-query
Semaphor-Wert 2
$ ./named-sem-wait
$ ./named-sem-query
Semaphor-Wert 1
$ ./named-sem-wait
```

hier blockiert der Prozess, bis der Semaphor erhöht wird

Mutex für Prozesse

- benannten POSIX-Semaphor verwenden (Erinnerung: binärer Semaphor = Mutex)

- also:

```
posix_sem = sem_open("/mutex", O_CREAT,  
                    0644, POSIX_UNLOCKED);  
sem_init(&posix_sem, 0, 1);    /* 1: Mutex */
```

System-V-IPC-Semaphore (2)

- Semaphor erzeugen: `semget()`
- Semaphor initialisieren: `semctl()`
- Semaphor verwenden: `semop()`

– erlaubt u. a. Signal- und Wait-Operationen:

```
struct sembuf          /* in <sys/sem.h> */  
{  
    unsigned short int sem_num; /* semaphore number */  
    short int sem_op;          /* semaphore operation */  
    short int sem_flg;        /* operation flag */  
};
```

`sem_op`: -1 = wait, 1 = signal

`sem_flg`: IPC_NOWAIT → Fehler statt Warten

System-V-IPC-Semaphore (1)

- Alternative zu benannten Posix-Semaphoren: System-V-IPC-Semaphore
- System-V-IPC: Methoden für die Inter-Prozess-Kommunikation (IPC) (mehr dazu: Kap. 6, IPC)
- Etwas komplexer:
 - Semaphor-Set (kann mehrere Semaphore enthalten)
 - private Semaphore (nur für Prozess und Kinder)
 - Public-Semaphore (mit Namen)

System-V-IPC-Semaphore (3)

Producer-Consumer-Problem mit SysV-Semaphoren (1)

```
/*  
 * sem-producer-consumer.c  
 */  
  
#include <stdio.h>          /* standard I/O routines.          */  
#include <stdlib.h>        /* rand() and srand() functions    */  
#include <unistd.h>        /* fork(), etc.                    */  
#include <time.h>          /* nanosleep(), etc.              */  
#include <sys/types.h>     /* various type definitions.       */  
#include <sys/ipc.h>       /* general SysV IPC structures     */  
#include <sys/sem.h>       /* semaphore functions and structs. */  
  
#define NUM_LOOPS 20      /* number of loops to perform.     */  
  
union semun { int val; struct semid_ds *buf; unsigned short *array; };  
  
int main(int argc, char* argv[]) {  
    int sem_set_id;        /* ID of the semaphore set.        */  
    union semun sem_val;  /* semaphore value, for semctl().  */  
    int child_pid;        /* PID of our child process.       */  
    int i;                 /* counter for loop operation.     */  
    struct sembuf sem_op; /* structure for semaphore ops.    */  
    int rc;                /* return value of system calls.   */  
    struct timespec delay; /* used for wasting time.          */
```

System-V-IPC-Semaphore (4)

Producer-Consumer-Problem mit SysV-Semaphoren (2)

```
/* create private sem. set with one sem. in it, access only to the owner. */
sem_set_id = semget(IPC_PRIVATE, 1, 0600);
if (sem_set_id == -1) { perror("main: semget"); exit(1); }
printf("semaphore set created, semaphore set id '%d'.\n", sem_set_id);

/* initialize the first (and single) semaphore in our set to '0'. */
sem_val.val = 0;
rc = semctl(sem_set_id, 0, SETVAL, sem_val);

/* fork-off a child process, and start a producer/consumer job. */
child_pid = fork();
switch (child_pid) {
    case -1: perror("fork"); exit(1);
    case 0: /* child process: consumer */
        for (i=0; i<NUM_LOOPS; i++) {
            /* block on the semaphore, unless its value is non-negative. */
            sem_op.sem_num = 0;
            sem_op.sem_op = -1; /* <- -1: count down */
            sem_op.sem_flg = 0;
            semop(sem_set_id, &sem_op, 1); /* wait (semaphore) */
            printf("consumer: '%d'\n", i); fflush(stdout);
        }
        break;
}
```

System-V-IPC-Semaphore (6)

Variante mit zwei getrennten Programmen:

- gemeinsamer Key erlaubt Zugriff auf (gleichen) Semaphor
- Key erzeugen mit `ftok()` („filename to key“):

```
key_t semkey = ftok("/tmp", 'a');
```

- `semget()-Aufruf anpassen: Aus`

```
/* privates Semaphor-Set erzeugen */
sem_set_id = semget(IPC_PRIVATE, 1, 0600);

wird

/* öffentliches Semaphor-Set erzeugen */
semkey = ftok("/tmp", 'a');
sem_set_id = semget(semkey, 1, 0);
```

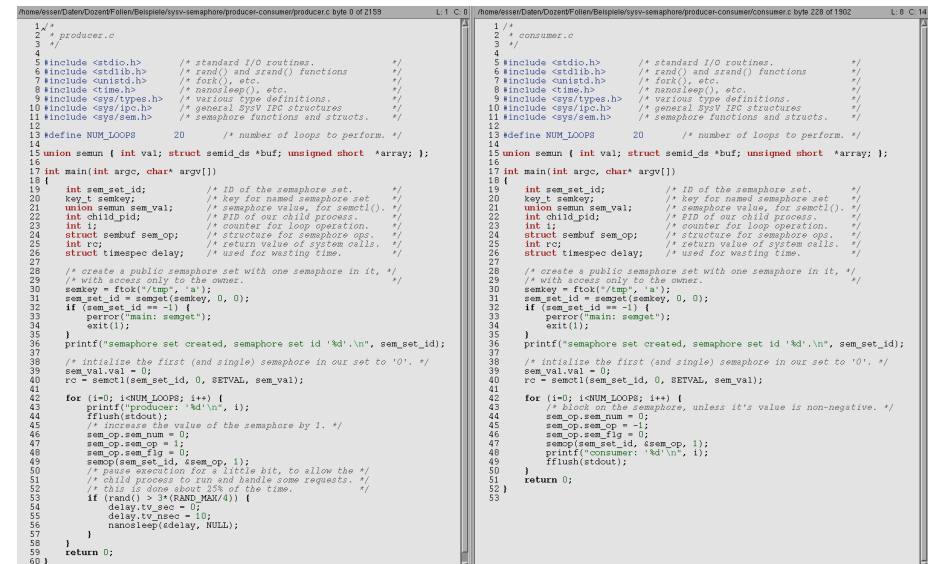
System-V-IPC-Semaphore (5)

Producer-Consumer-Problem mit SysV-Semaphoren (3)

```
default: /* parent process: producer */
    for (i=0; i<NUM_LOOPS; i++) {
        printf("producer: '%d'\n", i); fflush(stdout);
        /* increase the value of the semaphore by 1. */
        sem_op.sem_num = 0;
        sem_op.sem_op = 1; /* <- +1: count up */
        sem_op.sem_flg = 0;
        semop(sem_set_id, &sem_op, 1); /* signal (semaphore) */
        /* pause execution for a bit, to allow the child process to run */
        /* and handle some requests. this is done about 25% of the time. */
        if (rand() > 3*(RAND_MAX/4)) {
            delay.tv_sec = 0;
            delay.tv_nsec = 10;
            nanosleep(&delay, NULL);
        }
        break;
    }
return 0;
}
```

Quelle: <http://users.actcom.co.il/~choo/lupg/tutorials/multi-process/multi-process.html#semaphores>

System-V-IPC-Semaphore (7)



```
1 /*
2  * producer.c
3  */
4
5 #include <stdio.h> /* standard I/O routines. */
6 #include <stdlib.h> /* rand() and srand() functions. */
7 #include <unistd.h> /* fork(), etc. */
8 #include <time.h> /* nanosleep(), etc. */
9 #include <sys/types.h> /* various type definitions. */
10 #include <sys/ipc.h> /* general SysV IPC structures. */
11 #include <sys/sem.h> /* semaphore functions and structs. */
12
13 #define NUM_LOOPS 20 /* number of loops to perform. */
14
15 union semun { int val; struct semid_ds *buf; unsigned short *array; };
16
17 int main(int argc, char* argv[])
18 {
19     int sem_set_id; /* ID of the semaphore set. */
20     key_t semkey; /* key for named semaphore set. */
21     union semun sem_val; /* semaphore value, for semctl(). */
22     int child_pid; /* PID of our child process. */
23     int i; /* counter for loop operation. */
24     struct sembuf sem_op; /* structure for semaphore ops. */
25     int rc; /* return value of system calls. */
26     struct timespec delay; /* used for wasting time. */
27
28     /* create a public semaphore set with one semaphore in it,
29     /* with access only to the owner. */
30     semkey = ftok("/tmp", 'a');
31     sem_set_id = semget(semkey, 0, 0);
32     if (sem_set_id == -1) {
33         perror("main: semget");
34         exit(1);
35     }
36     printf("semaphore set created, semaphore set id '%d'.\n", sem_set_id);
37
38     /* initialize the first (and single) semaphore in our set to '0'. */
39     sem_val.val = 0;
40     rc = semctl(sem_set_id, 0, SETVAL, sem_val);
41
42     for (i=0; i<NUM_LOOPS; i++) {
43         printf("producer: '%d'\n", i);
44         fflush(stdout);
45         /* increase the value of the semaphore by 1. */
46         sem_op.sem_num = 0;
47         sem_op.sem_op = 1;
48         sem_op.sem_flg = 0;
49         semop(sem_set_id, &sem_op, 1);
50         /* pause execution for a little bit, to allow the
51         /* child process to run and handle some requests. */
52         /* this is done about 25% of the time. */
53         if (rand() > 3*(RAND_MAX/4)) {
54             delay.tv_sec = 0;
55             delay.tv_nsec = 10;
56             nanosleep(&delay, NULL);
57         }
58     }
59     return 0;
60 }
```

System-V-IPC-Semaphore (8)

Terminal 1

```
$ gcc -o consumer consumer.c
$ gcc -o producer producer.c
$ ./consumer
semaphore set created,
semaphore set id '374964228'.
consumer: '0'
consumer: '1'
consumer: '2'
consumer: '3'
consumer: '4'
consumer: '5'
consumer: '6'
consumer: '7'
consumer: '8'
consumer: '9'
consumer: '10'
consumer: '11'
consumer: '12'
consumer: '13'
consumer: '14'
consumer: '15'
consumer: '16'
consumer: '17'
consumer: '18'
consumer: '19'
$ _
```

Terminal 2

```
$ ./producer
semaphore set created,
semaphore set id '374964228'.
producer: '0'
producer: '1'
producer: '2'
producer: '3'
producer: '4'
producer: '5'
producer: '6'
producer: '7'
producer: '8'
producer: '9'
producer: '10'
producer: '11'
producer: '12'
producer: '13'
producer: '14'
producer: '15'
producer: '16'
producer: '17'
producer: '18'
producer: '19'
$ _
```

consumer.c und producer.c auf der Vorlesungs-Web-Seite verfügbar

Synchronisation im Linux-Kernel

- Atomare Operationen
 - auf Integer-Variablen (atomic_set, atomic_add, atomic_inc, ...)
 - Bit-Operationen auf Bitvektoren (set_bit, clear_bit, test_and_set, ...)
- Spin Locks / Reader-Writer Spin Locks
- Semaphore / Reader-Writer-Semaphore
- „Big Kernel Lock“

```
Sep 19 14:20:18 amd64 sbhd[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[30333]: (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 sbhd[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 12:54:41 amd64 sbhd[6694]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sbhd[6694]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sbhd[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 sbhd[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 sbhd[10141]: 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 sbhd[11088]: 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 sbhd[11089]: 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 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c 'severity=DEBUG')
Sep 22 01:00:01 amd64 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c 'severity=DEBUG')
Sep 22 02:00:01 amd64 /usr/sbin/cron[5194]: (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[2311]: (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[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 sbhd[5541]: Accepted rsa for esser from ::ffff:109.168.201.59771 ssh2
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sbhd[6606]: Accepted rsa for esser from ::ffff:109.168.201.62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[3121]: (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 sbhd[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 sbhd[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 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sbhd[28399]: 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[1621]: (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[14884]: (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 sbhd[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 sbhd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sbhd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sbhd[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 sbhd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sbhd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sbhd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sbhd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778
```

Linux: Synchronisation im Kernel

Atomare Integer-Operationen (1)

- Neuer Typ *atomic_t* (24 Bit Integer)
- Initialisierung: *atomic_t var = ATOMIC_INIT(0);*
- Wert setzen: *atomic_set(&var, wert);*
- Addieren: *atomic_add(wert, &var);*
- ++: *atomic_inc(&var);*
- Subtrahieren: *atomic_sub(wert, &var);*
- --: *atomic_dec(&var);*
- Auslesen: *int i = atomic_read(&var);*

Atomare Integer-Operationen (2)

- ***res = atomic_sub_and_test (i, &var);***
zieht atomar *i* von *var* ab.
 - Rückgabewert true, falls Ergebnis 0 ist;
 - Rückgabewert false, falls Ergebnis nicht 0
- ***res = atomic_dec_and_test (&var);***
res = atomic_inc_and_test (&var);
führt atomar *var--*; bzw. *var++*; aus.
 - Rückgabewert true, falls Ergebnis 0 ist;
 - Rückgabewert false, falls Ergebnis nicht 0

Atomare Bit-Operationen (1)

- Einzelne Bits in Bitvektoren setzen
- Datentyp: beliebig, z. B. *unsigned long bitvektor = 0;*
 - nur über Pointer ansprechen
 - Anzahl der setz-/testbaren Bits hängt von Größe des verwendeten Datentyps ab
- ***set_bit (i, &bitvektor);*** *i*-tes Bit setzen
- ***clear_bit (i, &bitvektor);*** *i*-tes Bit löschen
- ***change_bit (i, &bitvektor);*** *i*-tes Bit kippen

Atomare Integer-Operationen (3)

- ***res = atomic_add_negative (i, &var);***
addiert atomar *i* zu *var*.
 - Rückgabewert true, falls Ergebnis negativ ist;
 - Rückgabewert false, falls Ergebnis ≥ 0 ist

Atomare Bit-Operationen (2)

- Test-and-Set-Operationen geben zusätzlich den vorherigen Wert des jeweiligen Bits zurück
 - ***b = test_and_set_bit (i, &bitvektor);***
 - ***b = test_and_clear_bit (i, &bitvektor);***
 - ***b = test_and_change_bit (i, &bitvektor);***
- Einzelne Bits auslesen
 - ***b = test_bit (i, &bitvektor);***
- Suchfunktionen
 - ***pos = find_first_bit (&bitvektor, laenge);***
 - ***pos = find_first_zero_bit (&bitvektor, laenge);***

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);
```

Spin Locks (3)

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

```
spinlock_t xy_lock = SPIN_LOCK_UNLOCKED

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

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

Spin Locks (2)

- Da Spin Locks nicht schlafen, kann man sie in Interrupt-Handlern verwenden
- In dem Fall: zusätzlich Interrupts sperren:

```
spinlock_t xy_lock = SPIN_LOCK_UNLOCKED
unsigned long flags;

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

(aktuelle Interrupts in *flags* sichern, dann sperren bzw. ursprünglichen Zustand wiederherstellen)

Spin Locks (4)

- Um Blockieren zu vermeiden, ist Lock-Abfrage mit *spin_is_locked (&xy_lock);* möglich
- Locking-Versuch mit *spin_try_lock;*

```
if ( spin_try_lock (&xy_lock) ) {
    /* kritischer Abschnitt */
    spin_unlock (&xy_lock);
} else {
    /* durfte nicht in den kritischen Abschnitt */
}
```
- Beide Funktionen sollte man nicht verwenden: Entweder braucht man das Lock (und muss dann ggf. warten), oder man braucht es nicht...

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

```
read_lock (&xy_rwlock) {
    /* kritischer Abschnitt,
       read-only */
    read_unlock (&xy_rwlock);
```

Schreibender Code

```
write_lock (&xy_rwlock) {
    /* kritischer Abschnitt,
       read & write */
    write_unlock (&xy_rwlock);
```

- Nur bei klarer Trennung zwischen lesenden / schreibenden Programmteilen!

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

Reader Writer Locks (2)

- | | Es gibt schon einen Leser | Es gibt schon einen Schreiber | Noch keine Sperre |
|-----------------------------------|---------------------------|-------------------------------|-------------------|
| <code>read_lock(&lck)</code> | erfolgreich | schlägt fehl | erfolgreich |
| <code>write_lock(&lck)</code> | 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_unpock_irqrestore`

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 (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 (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);
}
```

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

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 Semaphor-Erzeugung

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

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

„Big Kernel Lock“ (BKL) (2)

- BKL nur im Prozess-Kontext benutzbar (nicht in Interrupt-Routinen)
- Prozess, der das BKL hält, darf schlafen
 - Beim Schlafenlegen wird das BKL automatisch aufgegeben
 - Beim Aufwecken wird es wieder erworben
- BKL ist rekursiv: Prozess, der bereits das BKL hält, darf also erneut `lock_kernel()` ausführen
- Nicht benutzen!

„Big Kernel Lock“ (BKL) (1)

- Überbleibsel aus älteren Kernel-Versionen
- Globales Lock für den gesamten Kernel (das alle Code-Teile betrifft, die damit Datenzugriff schützen)

```
lock_kernel ();
/* kritischer Abschnitt */
unlock_kernel ();
```

```
if ( kernel_locked() ) {
    ...
}
```

Vorschau

Nächstes Mal:

Synchronisation von Prozessen und Threads unter Windows

(inkl. „Crashkurs“ zu Prozessen und Threads)