



# 5. Synchronization (5)

## 5. Synchronization

### 5.4 Linux

- Processes
- Linux Kernel

**Linux: Pro**  
**Synchroniz**

```
Sep 19 14:27:41 amd64 syslogd[ng7653]: STAT2: dropped 0
Sep 20 00:01:00 amd64 syslogd[ng7653]: /sbin/cron[29278]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 20 01:00:01 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 20 02:00:01 amd64 syslogd[ng7653]: /sbin/cron[30103]: (root) CMD ('/sbin/evlogmgr -c "age > 30d"')
Sep 20 03:00:01 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 20 12:46:44 amd64 sshd[16151]: Accepted rsh for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 20 12:48:41 amd64 sshd[16694]: Accepted rsh for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[16694]: Accepted rsh for esser from ::ffff:87.234.201.207 port 62514
Sep 20 13:00:01 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 20 15:27:35 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 20 16:37:11 amd64 sshd[10120]: Accepted rsh for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:57:11 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 21 00:01:00 amd64 syslogd[ng7653]: /sbin/cron[10551]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 21 01:00:01 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 21 02:00:01 amd64 syslogd[ng7653]: /sbin/cron[17978]: (root) CMD ('/sbin/evlogmgr -c "age > 30d"')
Sep 21 02:00:01 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 21 03:00:01 amd64 syslogd[ng7653]: /sbin/cron[18000]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 21 03:00:01 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 21 17:43:26 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 21 17:53:39 amd64 sshd[13269]: Accepted rsh for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 22 00:01:00 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 22 01:00:01 amd64 syslogd[ng7653]: /sbin/cron[4674]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 22 02:00:01 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 22 02:00:01 amd64 syslogd[ng7653]: /sbin/cron[4599]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 22 03:00:01 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 23 01:00:01 amd64 syslogd[ng7653]: /sbin/cron[247391]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 23 01:00:01 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 23 02:00:01 amd64 syslogd[ng7653]: /sbin/cron[255551]: (root) CMD ('/sbin/evlogmgr -c "age > 30d"')
Sep 23 02:00:01 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 23 18:04:05 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 24 01:00:01 amd64 syslogd[ng7653]: /sbin/cron[12436]: (root) CMD ('/sbin/evlogmgr -c "severity=DEBUG"')
Sep 24 02:00:01 amd64 syslogd[ng7653]: /sbin/cron[13253]: (root) CMD ('/sbin/evlogmgr -c "age > 30d"')
Sep 24 02:00:01 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 24 11:15:48 amd64 sshd[19989]: Accepted rsh for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 24 13:49:08 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 24 15:42:07 amd64 kernel: smem_semi_mmu_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 24 20:31:31 amd64 sshd[22991]: Accepted rsh for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslogd[ng7653]: STAT3: 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 syslogd[ng7653]: STAT3: dropped 0
Sep 25 02:00:02 amd64 syslogd[ng7653]: /sbin/cron[662]: (root) CMD ('/sbin/evlogmgr -c "age > 30d"')
Sep 25 10:59:25 amd64 sshd[1889]: Accepted rsh for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 25 10:59:47 amd64 sshd[18921]: Accepted rsh for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:59:55 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsh for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslogd[ng7653]: STAT3: dropped 0
Sep 25 14:07:17 amd64 sshd[11690]: Accepted rsh for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:08:33 amd64 sshd[11690]: Accepted rsh for esser from ::ffff:87.234.201.207 port 63707
Sep 25 15:25:33 amd64 sshd[11690]: Accepted rsh for esser from ::ffff:87.234.201.207 port 62778
```

# Linux: Process Synchronization

# Named POSIX Semaphores (1)

can be used like the variant for threads, but extends to processes by using a semaphore name that is unique on the system

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

→ creates entry in `/dev/shm`

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

Hans-Georg Eßer, FH München Operating Systems I, WS 2006/07 – 2006-12-06 5. Synchronization (5) – Slide 3

## Named POSIX Semaphores (2)

## as introduction: 4 simple test programs

- *named-sem-init.c* initialize semaphore  
uses *sem\_open*, *sem\_init*
  - *named-sem-query.c* query semaphore value  
*sem\_open*, *sem\_getvalue*
  - *named-sem-wait.c* decrement semaphore  
(Wait operation),  
*sem\_open*, *sem\_wait*
  - *named-sem-signal.c* increment semaphore  
(Signal operation),  
*sem\_open*, *sem\_post*

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 4

## Named POSIX Semaphores (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( "/MySemaphore", O_CREAT, 0644, POSIX_UNLOCKED );
    sem_init(posix_sem, 0, 5);           /* init: 5 */
}
}

$ ./named-sem-init
$ ls -l /dev/shm/
-rw-r----- 1 lesser users 16 2006-12-05 15:46 sem.MySemaphore
$ hexdump /dev/shm/sem.MySemaphore
00000000 0005 0000 0000 0000 0000 0000 0000
0000010
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 5

## Named POSIX Semaphores (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("/MySemaphore", O_CREAT, 0644, POSIX_UNLOCKED);
    sem_getvalue(posix_sem, &ret);
    printf ("Semaphore value %d \n", ret);
}

$ ./named-sem-query
Semaphore value 5
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 6

## Named POSIX Semaphores (5)

## Signal operation

## 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("/MySemaphore", O_CREAT, 0644, POSIX_UNLOCKED);
    sem_post(posix_sem);
}

$ ./named-sem-query
Semaphore value 5
$ ./named-sem-signal
$ ./named-sem-query
Semaphore value 6
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 7

## Named POSIX Semaphores (6)

## Wait operation

```
$ ./named-sem-query  
Semaphore value 0  
$ ./named-sem-wait
```

here the process blocks until the semaphore is incremented

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 6

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 8

## Mutex for Processes

- use a named POSIX semaphore  
(reminder: binary semaphore = mutex)

- SO:

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

## System V IPC Semaphores (2)

- create semaphore: *semget()*
- initialize semaphore: *semctl()*
- use semaphore: *semop()*
  - includes **signal** and **wait** operations:

```
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 → error instead of waiting

## System V IPC Semaphores (1)

- Alternative to named Posix semaphores:  
**System V IPC Semaphores**
- System V IPC**: Methods for Inter Process Communication (IPC)  
(more about this: Ch. 6, IPC)
- A little more complex:
  - semaphore sets (can contain several semaphores)
  - private semaphores (only for process and children)
  - public semaphores (with identifiers)

## System V IPC Semaphores (3)

Producer Consumer Problem with SysV semaphores (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 Semaphores (4)

## Producer Consumer Problem with SysV semaphores (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;
}
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 13

# System V IPC Semaphores (5)

## Producer Consumer Problem with SysV semaphores (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;
}
```

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

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 14

# System V IPC Semaphores (6)

## Variant with two separate programs:

- common key allows accessing the (same) semaphore
- create key with *ftok()* („filename to key“):

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

- change *semget()* call:

```
/* create private semaphore set */
sem_set_id = semget(IPC_PRIVATE, 1, 0600);
```

becomes

```
/* create public semaphore set */
semkey = ftok("/tmp", 'a');
sem_set_id = semget(semkey, 1, 0);
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 15

# System V IPC Semaphores (7)

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

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 16

# System V IPC Semaphores (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'
$ _
```

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

Hans-Georg Eßer, FH München

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'
$ _
```

## Linux: Synchronization in the Kernel

```
Sep 19 14:20:18 and64 sshd[20494]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61557
Sep 19 14:27:43 and64 syslog-ng[7653]: STAT5: dropped 0
Sep 19 14:27:43 and64 syslog-ng[7653]: (root) CMD (./sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 and64 syslog-ng[7653]: STAT5: dropped 0
Sep 20 02:00:01 and64 syslog-ng[7653]: (root) CMD (/bin/evlogmgr -c 'age > *30d')
Sep 20 02:46:44 and64 sshd[6516]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:48:43 and64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 and64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:39 and64 sshd[9077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:30:43 and64 sshd[10102]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 and64 sshd[10102]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:38:10 and64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 and64 syslog-ng[7653]: (root) CMD (./sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 and64 syslog-ng[7653]: (root) CMD (./sbin/evlogmgr -c "severity=INFO")
Sep 21 02:00:01 and64 /usr/sbin/cron[17878]: (root) CMD (/bin/evlogmgr -c 'age > *30d')
Sep 21 02:43:26 and64 sshd[31086]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:53:39 and64 sshd[11023]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 and64 syslog-ng[7653]: STAT5: dropped 0
Sep 21 19:43:26 and64 sshd[46741]: (root) CMD (/bin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 and64 /usr/sbin/cron[46741]: (root) CMD (/bin/evlogmgr -c 'age > *30d')
Sep 22 02:00:01 and64 syslog-ng[7653]: (root) CMD (/bin/evlogmgr -c "severity=DEBUG")
Sep 22 20:23:23 and64 syslog-ng[7653]: STAT5: dropped 0
Sep 23 01:00:01 and64 /usr/sbin/cron[12523]: (root) CMD (/bin/evlogmgr -c "severity=DEBUG")
Sep 23 01:00:01 and64 syslog-ng[7653]: (root) CMD (./sbin/evlogmgr -c "age > *30d")
Sep 23 02:00:01 and64 /usr/sbin/cron[25555]: (root) CMD (./sbin/evlogmgr -c "age > *30d")
Sep 23 18:04:09 and64 sshd[6554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 23 18:04:14 and64 sshd[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64457
Sep 23 18:04:34 and64 sshd[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64458
Sep 24 01:00:01 and64 /usr/sbin/cron[54887]: (root) CMD (/bin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 and64 syslog-ng[7653]: STAT5: dropped 0
Sep 24 02:00:01 and64 /usr/sbin/cron[12523]: (root) CMD (/bin/evlogmgr -c "age > *30d")
Sep 24 11:15:48 and64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 13:49:08 and64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 15:42:07 and64 kernel: snd_soc_midi_sw: unsupported module, tainting kernel.
Sep 24 15:42:07 and64 syslog-ng[7653]: STAT5: dropped 0
Sep 24 20:25:31 and64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 25 01:00:01 and64 /usr/sbin/cron[12523]: (root) CMD (/bin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 and64 syslog-ng[7653]: STAT5: dropped 0
Sep 25 02:00:01 and64 /usr/sbin/cron[14841]: (root) CMD (/bin/evlogmgr -c "age > *30d")
Sep 25 10:15:28 and64 syslog-ng[7653]: STAT5: dropped 0
Sep 25 10:59:25 and64 syslog-ng[7653]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:47 and64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 and64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:30:33 and64 syslog-ng[7653]: STAT5: dropped 0
Sep 25 14:05:37 and64 syslog-ng[7653]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:06:10 and64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 and64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 and64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:31 and64 sshd[12303]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778
```

# Synchronization in the Linux Kernel

- atomic operations
  - on Integer variables  
(atomic\_set, atomic\_add, atomic\_inc, ...)
  - bit operations on bit vectors  
(set\_bit, clear\_bit, test\_and\_set, ...)
- Spin Locks / Reader Writer Spin Locks
- Semaphores / Reader Writer Semaphores
- „Big Kernel Lock“

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 19

## Atomic Integer Operations (1)

- new type **atomic\_t** (24 bit integer)
- initialization: **atomic\_t var = ATOMIC\_INIT(0);**
- set value: **atomic\_set (&var, wert);**
- add: **atomic\_add (wert, &var);**
- increment: **atomic\_inc (&var);**
- subtract: **atomic\_sub (wert, &var);**
- decrement: **atomic\_dec (&var);**
- read: **int i = atomic\_read (&var);**

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-06

5. Synchronization (5) – Slide 20

## Atomic Integer Operations (2)

- `res = atomic_sub_and_test (i, &var);`  
atomically subtracts *i* from *var*
  - return value true, if result is 0;
  - return value false, if result is not 0
- `res = atomic_dec_and_test (&var);`  
`res = atomic_inc_and_test (&var);`  
atomically decrements or increments *var*
  - return value true, if result is 0;
  - return value false, if result is not 0

## Atomic Integer Operations (3)

- `res = atomic_add_negative (i, &var);`  
atomically adds *i* to *var*.
  - return value true, if result is negative;
  - return value false, if result is  $\geq 0$

## Atomic Bit Operations (1)

- set and clear single bits in bit vector
- datatype: arbitrary, e.g.  
`unsigned long bitvector = 0;`
  - access through pointer
  - number of settable / clearable bits depends on size of the used datatype
- `set_bit (i, &bitvector);` set *i*-th bit
- `clear_bit (i, &bitvector);` clear *i*-th bit
- `change_bit (i, &bitvector);` change *i*-th bit

## Atomic Bit Operations (2)

- Test-and-Set operations return the previous value of the bit
  - `b = test_and_set_bit (i, &bitvector);`
  - `b = test_and_clear_bit (i, &bitvector);`
  - `b = test_and_change_bit (i, &bitvector);`
- read single bit
  - `b = test_bit (i, &bitvector);`
- search functions
  - `pos = find_first_bit (&bitvector, length);`
  - `pos = find_first_zero_bit (&bitvector, length);`

## Spin Locks (1)

- Lock with Mutex functionality: mutual exclusion
- Code which requests a Spin Lock but is not successful, will spin in a loop until the lock becomes available
- Type: *spinlock\_t*

```
spinlock_t xy_lock = SPIN_LOCK_UNLOCKED  
  
spin_lock (&xy_lock);  
/* critical region */  
spin_unlock (&xy_lock);
```

## Spin Locks (3)

- when all interrupts are on before aquiring the spin lock, there's a simpler method:

```
spinlock_t xy_lock = SPIN_LOCK_UNLOCKED  
  
spin_lock_irq (&xy_lock);  
/* critical section */  
spin_unlock_irq (&xy_lock);
```

disables / reenables all interrupts

- spin locks are not „recursive“, i.e.: it's impossible to aquire the same spin lock twice, e.g. when calling a function recursively

## Spin Locks (2)

- since Spin Locks don't sleep they can be used inside interrupt handlers
- In that case: also disable interrupts:

```
spinlock_t xy_lock = SPIN_LOCK_UNLOCKED  
unsigned long flags;  
  
spin_lock_irqsave (&xy_lock, flags);  
/* critical region */  
spin_unlock_irqrestore (&xy_lock, flags);
```

(save current interrupt settings in *flags*, then disable; restore original state)

## Spin Locks (4)

- to avoid blocking, it is possible to query the spin lock state with *spin\_is\_locked* (*&xy\_lock*);
- locking attempt with *spin\_try\_lock*:

```
if ( spin_try_lock (&xy_lock) ) {  
    /* critical region */  
    spin_unlock (&xy_lock);  
} else {  
    /* was not allowed to enter the critical region */  
}
```

- both functions should not be used: either you need the lock (and in that case will possibly have to wait), or you don't need it ...

## Reader Writer Locks (1)

- alternative to normal locks, allowing several concurrent read accesses – but exclusive for write access (like a standard lock):

```
rwlock_t xy_rwlock = RW_LOCK_UNLOCKED;  
  
reading code           writing code  
read_lock (&xy_rwlock) {          write_lock (&xy_rwlock) {  
    /* critical region,        /* critical region,  
     read-only */           read & write */  
    read_unlock (&xy_rwlock);   write_unlock (&xy_rwlock);
```

- only use in case of strict separation of reading and writing code parts

## Reader Writer Locks (2)

	there is already a reader	there is already a writer	no locks yet
read_lock(&lck)	successful	fails	successful
write_lock(&lck)	fails	fails	successful

- there are variants for disabling interrupts, too:
  - read\_lock\_irq
  - read\_lock\_irqsave
  - write\_lock\_irq
  - write\_lock\_irqsave
  - read\_unlock\_irq
  - read\_unlock\_irqrestore
  - write\_unlock\_irq
  - write\_unlock\_irqrestore

## Semaphores (1)

- Kernel semaphores are „sleeping“ locks
- if a semaphore is already locked, new requesters are put into a queue.
- when freeing a semaphore, the first waiting thread of the queue will be awoken
- semaphores are useful for locks that shall be kept over a longer period of time
  - no waste of CPU time

## Semaphores (2)

- semaphores can only be used in process context, not in interrupt handlers (the scheduler does not deal with interrupt handlers)
- code which wants to use a semaphore, must not hold a normal lock (semaphore access can cause the thread to be put to sleep)
- semaphores can let more than one thread access the resource

## Semaphores (3)

Type: *semaphore*

Static declaration

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

dynamic semaphore creation

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

- use with *up()* and *down()*

```
down (&sem);
/* critical section */
up (&sem);
```

## Semaphores (4)

- Variants of *down()*
  - *down (&sem);*  
non-interruptible sleep, if semaphore is not available
  - *down\_interruptible (&sem);*  
interruptible sleep, if semaphore is not available
  - *down\_trylock (&sem);*  
tries to acquire the semaphore – if that fails, this function will return with a false value

## Semaphores (5)

- example for *down\_trylock()*

```
/* taken from /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();
    /* function release_console_sem() calls up(&console_sem); */
} 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 Semaphores (1)

- similar to Reader Writer Locks:  
Type *rw\_semaphore*, allowing special Up and Down operations for read and write access
- all Reader Writer Semaphores are mutexes  
(counter is always 1 at initialization)

static declaration

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

dynamical semaphore creation

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

## Reader Writer Semaphores (2)

```
static DECLARE_RWSEM (xy_rwsem);  
  
reading code           writing code  
down_read (&xy_rwsem) {  
    /* critical region,  
       read-only */  
    up_read (&xy_rwsem);  
  
    down_write (&xy_rwsem) {  
        /* critical region,  
           read & write */  
        up_write (&xy_rwsem);
```

just like Reader Writer Locks:

	there is already a reader	there is already a writer	no locks yet
down_read(&sem)	successful	fails	successful
down_write(&sem)	fails	fails	successful

## „Big Kernel Lock“ (BKL) (1)

- relic from older kernel versions
- global lock for the whole kernel (which affects all kernel regions that protect data access with it)

```
lock_kernel ();  
/* critical region */  
unlock_kernel ();  
  
if ( kernel_locked() ) {  
    ...  
}
```

## „Big Kernel Lock“ (BKL) (2)

- BKL can only be used in process context (not in interrupt routines)
- a process holding the BKL may sleep
  - when going to sleep, the BKL is automatically released
  - on wake-up it will be re-acquired
- BKL is recursive: a process which already holds the BKL may call *lock\_kernel()* again
- don't use it!