Operating systems

Semaphores (IPC)

Samuele Geminiani

samuele.germiniani@univr.it

University of Verona
Department of Engineering for Innovation Medicine

2023/2024

Table of Contents

1. Semaphores
   - Creating and Opening
   - Control Operations
   - Other Operations

Semaphores

Semaphores

Creating and Opening

Creating/Opening a Semaphore Set

The `semget` system call creates a new **semaphore set** or obtains the identifier of an existing set.

```
#include <sys/sem.h>

// Returns semaphore set identifier on success, or -1 error
int semget(key_t key, int nsems, int semflg);
```

The key arguments are: an IPC `key`, `nsems` specifies the number of semaphores in that set, and must be greater than 0. `semflg` is a bit mask specifying the permissions (see `open(...)` system call, `mode` argument) to be places on a new semaphore set or checked against an existing set.
In additions, the following flags can be ORed (|) in `semflg`:

- `IPC_CREAT`: If no semaphore set with the specified `key` exists, create a new set.
- `IPC_EXCL`: in conjunction with `IPC_CREAT`, it makes `semget` fail if a semaphore set exists with the specified `key`.

Creating/Opening a Semaphore Set

Example showing how to create a semaphore set having 10 semaphores

```c
int semid;
ket_t key = //... (generate a key in some way, i.e. with ftok)

// A) delegate the problem of finding a unique key to the kernel
semid = semget(IPC_PRIVATE, 10, S_IRUSR | S_IWUSR);

// B) create a semaphore set with identifier key, if it doesn’t already exist
semid = semget(key, 10, IPC_CREAT | S_IRUSR | S_IWUSR);

//C) create a semaphore set with identifier key, but fail if it exists already
semid = semget(key, 10, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
```

Semaphore Control Operations

The `semctl` system call performs a variety of control operations on a semaphore set or on an individual semaphore within a set.

```c
#include <sys/sem.h>

// Returns nonnegative integer on success, or -1 error
int semctl(int semid, int semnum, int cmd, ... /* union semun arg */);
```

The `semid` argument is the identifier of the semaphore set on which the operation is to be performed.
Certain control operations (`cmd`) require a third/fourth argument. Before presenting the available control operations on a semaphore set, the union `semun` is introduced.

Semaphore Control Operations - union *semun*

The union `semun` must be **explicitly defined by the programmer** before calling the `semctl` system call.

```
#ifndef SEMUN_H
#define SEMUN_H
#include <sys/sem.h>
// definition of the union semun
union semun {
    int val;
    struct semid_ds * buf;
    unsigned short * array;
};
#endif
```

Semaphores

Control Operations

Semaphore Control Operations

Generic control operations

```
int semctl(semid, 0 /*ignored*/, cmd, arg);
```

- IPC_RMID: Immediately remove the semaphore set. Any processes blocked is awakened (error set to EIDRM). The arg argument is not required.
- IPC_STAT: Place a copy of the semid_ds data structure associated with this semaphore set in the buffer pointed to by arg.buf.
- ICP_SET: Update selected fields of the semid_ds data structure associated with this semaphore set using values in the buffer pointed to by arg.buf.

Semaphore Control Operations

Generic control operations

```
struct semid_ds {
    struct ipc_perm sem_perm; /* Ownership and permissions */
    time_t sem_otime;         /* Time of last semop() */
    time_t sem_ctime;         /* Time of last change */
    unsigned long sem_nsems;  /* Number of semaphores in set */
};
```

Only the subfields *uid*, *gid*, and *mode* of the substructure *sem_perm* can be updated via `IPC_SET`.

Semaphore Control Operations

Generic control operations (Example)

Example showing how to **change the permissions** of a semaphore set

```
ket_t key = //... (generate a key in some way, i.e. with ftok)
// get, or create, the semaphore set
int semid = semget(key, 10, IPC_CREAT | S_IRUSR | S_IWUSR);
// instantiate a semid_ds struct
struct semid_ds ds;
// instantiate a semun union (defined manually somewhere)
union semun arg;
arg.buf = &ds;
// get a copy of semid_ds structure belonging to the kernel
if (semctl(semid, 0 /*ignored*/, IPC_STAT, arg) == -1)
    errExit("semctl IPC_STAT failed");
// update permissions to guarantee read access to the group
arg.buf->sem_perms.mode |= S_IRGRP;
// update the semid_ds structure of the kernel
if (semctl(semid, 0 /*ignored*/, IPC_SET, arg) == -1)
    errExit("semctl IPC_SET failed");
```

Semaphore Control Operations

Generic control operations (Example)

Example showing how to **remove** semaphore set

```
if (semctl(semid, 0/*ignored*/, IPC_RMID, 0/*ignored*/) == -1)
    errExit("semctl failed");
else
    printf("semaphore set removed successfully\n");
```

Semaphore Control Operations

Retrieving and initializing semaphore values

```
int semctl(semid, semnum, cmd, arg);
```

- SETVAL: the value of the *semnum-th* semaphore in the set referred to by semid is initialized to the value specified in arg.val.
- GETVAL: as its function result, semctl returns the value of the *semnum-th* semaphore in the semaphore set specified by semid. The arg argument is not required.

```
int semctl(semid, 0 /*ignored*/, cmd, arg);
```

- SETALL: initialize all semaphores in the set referred to by *semid*, using the values supplied in the array pointed to by *arg.array*.
- GETALL: retrieve the values of all of the semaphores in the set referred to by semid, placing them in the array pointed to by arg.array.

Semaphore Control Operations

Retrieving and initializing semaphore values (Example)

Example showing how to **initialize a specific semaphore** in a semaphore set

```c
ket_t key = //... (generate a key in some way, i.e. with ftok)
// get, or create, the semaphore set
int semid = semget(key, 10, IPC_CREAT | S_IRUSR | S_IWUSR);
// set the semaphore value to 0
union semun arg;
arg.val = 0;
// initialize the 5-th semaphore to 0
if (semctl(semid, 5, SETVAL, arg) == -1)
    errExit("semctl SETVAL");
```

A semaphore set must be always initialized before using it!

Semaphore Control Operations

Retrieving and initializing semaphore values (Example)

Example showing how to **get the current state** of a specific semaphore in a semaphore set.

```
ket_t key = //... (generate a key in some way, i.e. with ftok)
// get, or create, the semaphore set
int semid = semget(key, 10, IPC_CREAT | S_IRUSR | S_IWUSR);

// get the current state of the 5-th semaphore
int value = semctl(semid, 5, GETVAL, 0/*ignored*/);
if (value == -1)
   errExit("semctl GETVAL");
```

Once returned, the semaphore may already have changed state!

Semaphore Control Operations

Retrieving and initializing semaphore values (Example)

Example showing how to **initialize a semaphore** set having 10 semaphores

```c
ket_t key = //... (generate a key in some way, i.e. with ftok)
// get, or create, the semaphore set
int semid = semget(key, 10, IPC_CREAT | S_IRUSR | S_IWUSR);
// set the first 5 semaphores to 1, and the remaining to 0
int values[] = {1,1,1,1,1,0,0,0,0,0};
union semun arg;
arg.array = values;
// initialize the semaphore set
if (semctl(semid, 0/*ignored*/, SETALL, arg) == -1)
    errExit("semctl SETALL");
```

A semaphore set must be always initialized before using it!

Semaphore Control Operations

Retrieving and initializing semaphore values (Example)

Example showing how to **get the current state** of a semaphore set having 10 semaphores

```
ket_t key = //... (generate a key in some way, i.e. with ftok)
// get, or create, the semaphore set
int semid = semget(key, 10, IPC_CREAT | S_IRUSR | S_IWUSR);
// declare an array big enougth to store the semaphores' value
int values[10];
union semun arg;
arg.array = values;
// get the current state of a semaphore set
if (semctl(semid, 0/*ignored*/, GETALL, arg) == -1)
    errExit("semctl GETALL");
```

Once returned, a semaphore may already have changed state!

Semaphore Control Operations

Retrieving per-semaphore information

```
int semctl(semid, semnum, cmd, 0);
```

- `GETPID`: return the process ID of the last process to perform a `semop` on the *semnum-th* semaphore
- `GETNCNT`: return the number of processes currently waiting for the value of the *semnum-th* semaphore to increase
- `GETZCNT`: return the number of processes currently waiting for the value of the *semnum-th* semaphore to become 0;

Semaphore Control Operations

Retrieving per-semaphore information (Example)

Example showing how to **get information about a semaphore** of the semaphore set

```
ket_t key = //... (generate a key in some way, i.e. with ftok)
// get, or create, the semaphore set
int semid = semget(key, 10, IPC_CREAT | S_IRUSR | S_IWUSR);
// ...
// get information about the first semaphore of the semaphore set
printf("Sem:%d getpid:%d getncnt:%d getzcnt:%d\n",
semid,
semctl(semid, 0, GETPID, NULL),
semctl(semid, 0, GETNCNT, NULL),
semctl(semid, 0, GETZCNT, NULL));
```

Once returned, the semaphore may already have changed state!

Semaphores

Other Operations

Semaphore Operations

The semop system call performs one or more operations (wait (P) and signal (V)) on semaphores.

```
#include <sys/sem.h>

// Returns 0 on success, or -1 on error
int semop(int semid, struct sembuf *sops, unsigned int nsops);
```

The sops argument is a pointer to an array that contains a sorted sequence of operations to be performed atomically, and nsops (> 0) gives the size of this array. The elements of the sops array are structures of the following form:

```
struct sembuf {
    unsigned short sem_num; /* Semaphore number */
    short sem_op;           /* Operation to be performed */
    short sem_flg;          /* Operation flags */
};
```

Semaphore Operations

The `sem_num` field identifies the semaphore within the set upon which the operation is to be performed. The `sem_op` field specifies the operation to be performed:

- `sem_op` > 0: value of `sem_op` is added to the value of the `sem_num-th` semaphore.
- `sem_op = 0`: the value of the `sem_num-th` semaphore is checked to see whether it currently equals 0. If it doesn't, the calling process is blocked until the semaphore is 0.
- `sem_op` < 0: decrease the value of the `sem_num-th` semaphore by the amount specified in `sem_op`. it blocks the calling process until the semaphore value has been increased to a level that permits the operation to be performed without resulting in a negative value.

Semaphore Operations

When a `semop(...)` call blocks, the process remains blocked until on of the following occurs:

- Another process modifies the value of the semaphore such that the requested operation can proceed.
- A signal interrupts the `semop(...)` call. In this case, the error EINTR results.
- Another process deletes the semaphore referred to by `semid`. In this case, `semop(...)` fails with the error `EIDRM`.

We can prevent `semop(...)` from blocking when performing an operation on a particular semaphore by specifying the `IPC_NOWAIT` flag in the corresponding `sem_flg` field. In this case, if `semop(...)` would have blocked, it instead fails with the error `EAGAIN`.

Semaphore Operations

Example showing how to initialize an array of `sembuf` operations

```c
struct sembuf sops[3];

sops[0].sem_num = 0;
sops[0].sem_op = -1; // subtract 1 from semaphore 0
sops[0].sem_flg = 0;

sops[1].sem_num = 1;
sops[1].sem_op = 2; // add 2 to semaphore 1
sops[1].sem_flg = 0;

sops[2].sem_num = 2;
sops[2].sem_op = 0; // wait for semaphore 2 to equal 0
sops[2].sem_flg = IPC_NOWAIT; // but don't block if operation cannot be
    performed immediately
```

Semaphore Operations

Example showing how to perform operations on a semaphore set

```c
struct sembuf sops[3];

// .. see the previous slide to initilize sembuf

if (semop(semid, sops, 3) == -1) {
    if (errno == EAGAIN) // Semaphore 2 would have blocked
        printf("Operation would have blocked\n");
    else
        errExit("semop"); // Some other error
}
```