Operating systems

Interprocess communication (IPC)
Part 1 of 3: System V and Semaphores

Created by
Enrico Fraccaroli
enrico.fraccaroli@gmail.com

Table of Contents

Introduction to System V IPC

Introduction to System V IPC

Unix System V (aka "System Five")

Unix System V is one of the first commercial versions of the Unix operating system. It was originally developed by AT&T and first released in 1983. Four major versions of System V were released, numbered 1, 2, 3, and 4. System V is sometimes abbreviated to SysV.

Interprocess communication (IPC)

Interprocess communication (IPC) refers to mechanisms that coordinate activities among cooperating processes. A common example of this need is managing access to a given system resource.

Introduction to System V IPC

System V IPCs refers to three different mechanisms for interprocess communication:

- *Semaphores* let processes to synchronize their actions. A semaphore is a kernel-maintained value, which is appropriately modified by system's processes before performing some critical actions
- *Message queues* can be used to pass messages among processes.
- *Shared memory* enables multiple processes to share a their region of memory.

Other IPC

- **Signals**
- **Pipes**
- **FIFOs**

Introduction to System V IPC

Creating and Opening

Creating and opening a System V IPC object

Each System V IPC mechanism has an associated *get* system call (`msgget`, `semget`, or `shmget`), which is analogous to the `open` system call.

Given an integer *key* (analogous to a filename), the *get* system call can either first create a new IPC, and then returns its unique identifier, or returns the identifier of an existing IPC.

An IPC *identifier* is analogous to a *file descriptor*. It is used in all subsequent system calls to refer to the IPC object.

Creating and opening a System V IPC object

Example showing how to create a semaphore (overview)

```
// PERM: rw-------
id = semget(key, 10 ,IPC_CREAT | S_IRUSR | S_IWUSR);
if (id == -1)
    errExit(semget);
```

As with all of the *get* calls, the *key* is the first argument. It is a value sensible for the application using the IPC object. The returned IPC *identifier* is a unique code identifying the IPC object in the system.

Mapping with the *open* system call:

key ->filename
id ->file descriptor

System V IPC keys

System V IPC keys are integer values represented using the data type `key_t`. The IPC get calls translate a key into the corresponding integer IPC identifier.

So, how do we provide a unique key that guarantees we do not accidentally obtain the identifier of an existing IPC object used by some other application?

System V IPC keys - IPC_PRIVATE flag

When creating a new IPC object, the *key* may be specified as `IPC_PRIVATE`. In this way, we delegate the problem of finding a unique key to the kernel.

Example of the usage of `IPC_PRIVATE`:

```
id = semget(IPC_PRIVATE, 10, S_IRUSR | S_IWUSR);
```

This technique is especially useful in *multiprocess applications* where the parent process creates the IPC object prior to performing a `fork()`, with the result that the child inherits the identifier of the IPC object.

System V IPC keys - `ftok()`

The `ftok` (file to key) function converts a `pathname` and a `proj_id` (*i.e.*, project identifier) to a System V IPC key.

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

// Returns integer key on succcess, or -1 on error (check errno)
key_t ftok(char *pathname, int proj_id);
```

The provided `pathname` has to refer to an existing, accessible file. The last 8 bits of `proj_id` are actually used, and they have to be a nonzero value).

Typically, `pathname` refers to one of the files, or directories, created by the application.

System V IPC keys - `ftok()`

Example shows a typical usage of the function `ftok`

```
key_t key = ftok("/mydir/myfile", 'a');
if (key == -1)
    errExit("ftok failed");

int id = semget(key, 10, S_IRUSR | S_IWUSR);
if (id == -1)
    errExit("semget failed");
```

Introduction to System V IPC

Data Structures

Associated Data Structure - ipc_perm

The kernel maintains an associated data structure (msqid_ds, semid_ds, shmid_ds) for each instance of a System V IPC object. As well as data specific to the type of IPC object, each associated data structure **includes** the substructure ipc_perm holding the granted permissions.

```
struct ipc_perm {
    key_t __key;            /* Key, as supplied to 'get' call */
    uid_t uid;              /* Owner's user ID */
    gid_t gid;              /* Owner's group ID */
    uid_t cuid;             /* Creator's user ID */
    gid_t cgid;             /* Creator's group ID */
    unsigned short mode;    /* Permissions */
    unsigned short __seq; /* Sequence number */
};
```

Associated Data Structure - ipc_perm

- The `uid` and `gid` fields specify the ownership of the IPC object.
- The `cuid` and `cgid` fields hold the user and group IDs of the process that created the object.
- The `mode` field holds the permissions mask for the IPC object, which are initialized using the lower 9 bits of the flags specified in the *get* system call used to create the object.

Some important notes about `ipc_perm`:

1. The `cuid` and `cgid` fields are immutable.
2. Only read and write permissions are meaningful for IPC objects. Execute permission is meaningless, and it is ignored.

Associated Data Structure - ipc_perm - Example

Example shows a typical usage of the `semctl` to change the owner of a semaphore.

```
struct semid_ds semq;
// get the data structure of a semaphore from the kernel
if (semctl(semid, 0, IPC_STAT, &semq) == -1)
    errExit("semctl get failed");
// change the owner of the semaphore
semq.sem_perm.uid = newuid;
// update the kernel copy of the data structure
if (semctl(semid, IPC_SET, &semq) == -1)
    errExit("semctl set failed");
```

Similarly, the `shmctl` and `msgctl` system calls are applied to update the kernel data structure of a *shared memory* and *message queue*.

IPCs Commands

IPCs Commands

ipcs

The `ipcs` command

Using `ipcs`, we can obtain information about IPC objects on the system. By default, `ipcs` displays all objects, as in the following example:

```
user@localhost[~]$ ipcs
------ Message Queues --------
key    msqid      owner     perms     used-bytes  messages
0x1235 26         student   620       12          20

------ Shared Memory Segments --------
key    shmid      owner     perms     bytes     nattch    status
0x1234 0          professor 600       8192      2

------ Semaphore Arrays --------
key    semid      owner     perms     nsems
0x1111 102        professor 330       20
```

IPCs Commands

ipcrm

The `ipcrm` command

Using `ipcrm`, we can remove IPC objects from the system.
Remove a message queue:

```
ipcrm -Q 0x1235    ( 0x1235 is the key of a queue )
ipcrm -q 26        ( 26 is the identifier of a queue )
```

Remove a shared memory segment

```
ipcrm -M 0x1234    ( 0x1234 is the key of a shared memory seg. )
ipcrm -m 0         ( 0 is the identifier of a shared memory seg. )
```

Remove a semaphore array

```
ipcrm -S 0x1111    ( 0x1111 is the key of a semaphore array )
ipcrm -s 102       ( 102 is the identifier of a semaphore array )
```

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

```
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.

```
#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

```
Usage template: 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");
```

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

```
Usage template: 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.

```
Usage template: 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

```
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

```
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

```
Usage template: 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 `semnum-th` semaphore.
- `sem_op = 0`: the value of the `semnum-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 `semnum-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

```
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
// but don't block if operation cannot be performed immediately
sops[2].sem_flg = IPC_NOWAIT;
```

Semaphore Operations

Example showing how to perform operations on a semaphore set

```
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
}
```

Next Lectures

Next Lectures

- Lecture 2 of 3: System V IPC:
  - Message queues
  - Shared memory
- Lecture 3 of 3: IPC:
  - Signal
  - Pipe
  - Fifo