Operating systems

Introduction to IPC

Samuele Germiniani

samuele.germiniani@univr.it

University of Verona
Department of Engineering for Innovation Medicine

2023/2024

Table of Contents

1. Introduction to System V IPC
2. Creating and Opening
3. Data Structures
4. IPCs Commands

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

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");
```

Example: Character ”a”

- ASCII = 097
- Binary = 01100001

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.

```c
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

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
```

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 )
```