IPC en Unix

System V
MENSAJES

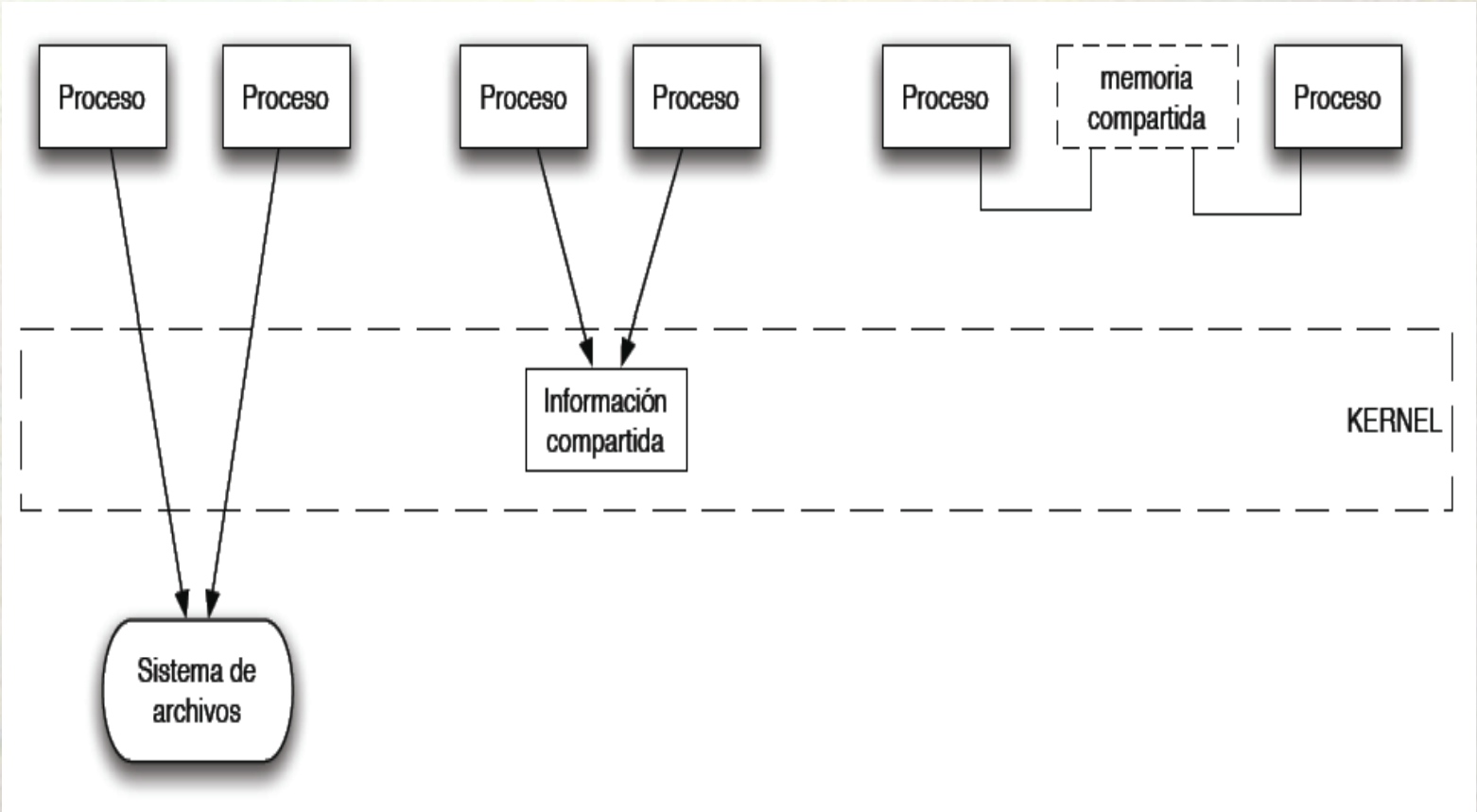
Francisco Javier Montero Vega
Francisco García Rodríguez

DSO 2004-05

Inter-Proces Comunicaction

- Paso de mensages(colas)
- Sincronización (semáforos, cerrojos)
- Memoria compartida
- Funciones remotas (RPC)
- Sockets

Inter-Proces Comunicaction



System V

- Núcleos System V ,creados en 1980.
- Redefinición Posix en 1993 .
- Cada recurso es identificado por un mecanismo de claves.

Implementa

- Cola de mensajes
- Semáforos
- Memoria compartida

System V

- El sistema operativo BSD, utiliza sockets para la IPC.
- Linux sin embargo utiliza sockets y SYSTEM V.

Identificadores IPC

- Cada objeto IPC tiene asociado un único identificador. Cada “objeto de IPC” es un mensaje simple de una cola, de un semáforo o de un segmento de memoria compartida.
- El hecho de que cada “objeto” en cuestión posea un ÍD único, identificará exactamente la cola a la que pertenece y a el recurso asociado.

Gestión de claves de IPC

- Primero pasos para la creación de una aplicación cliente/servidor:

1. -Para obtener un único ID, una clave debe de ser utilizada.
2. -La clave debe ser aceptada mutuamente por cada proceso cliente y servidor.

Gestión de claves de IPC

Ejemplo:

Supongamos por un momento que deseamos realizar una llamada telefónica. Necesitamos saber el número el cual deseamos llamar.Marcamos.

La compañía de teléfono debe de saber como redireccionar la salida de la llama hacia el destino.

Cuando el destino responda, se crea la conexión.

Gestión de claves de IPC

•En el caso de SYSTEM V IPC facilita los "teléfono" correlativos directamente con el tipo de objetos que están siendo utilizados. El "Teléfono de la compañía", o el método de enrutamiento, puede ser asociado directamente con una clave de IPC.

• Si el programador quiere, la clave puede tener el mismo valor asignándole un valor directamente en el código.

Gestión de claves de IPC

- Este método tiene la desventaja de que la clave posiblemente este en uso.
- Para solucionar dicho problema, La función ftok() se utiliza para generar el valor de clave tanto para el cliente como para el servidor.

Gestión de claves de IPC

LIBRARY FUNCTION: ftok();

PROTOTYPE:

key_t ftok (char *pathname, char proj);

RETURNS:

Un nuevo valor de clave IPC, si es correcto devuelve. –1 si es incorrecto, y guarda el motivo del error en “errno” .

Gestión de claves de IPC

- El valor de clave que devuelve ftok() es generado por la combinación del número de inode y el número del dispositivo menor desde el fichero,de un argumento, con el primer identificador del carácter del proyecto como segundo argumento.
- Esto no garantiza la unicidad de la clave, pero una aplicación puede chequear y comprobar posibles colisiones y reintentar generando una nueva clave.

Gestión de claves de IPC

Un ejemplo de lo visto es :

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

El algoritmo de generación de la clave es utilizado para poder dar total discreción en las aplicaciones del programador.

Comandos IPCs

- Los comandos ipcs pueden ser utilizados para obtener el estado de todos los objetos SYSTEM V IPC.
- **Los comandos son:**

ipcs -q: Sólo muestra mensajes de la cola.

ipcs -s: Sólo muestra semáforos.

ipcs -m: Sólo muestra memoria compartida.

ipcs --help: Argumentos Adicionales.

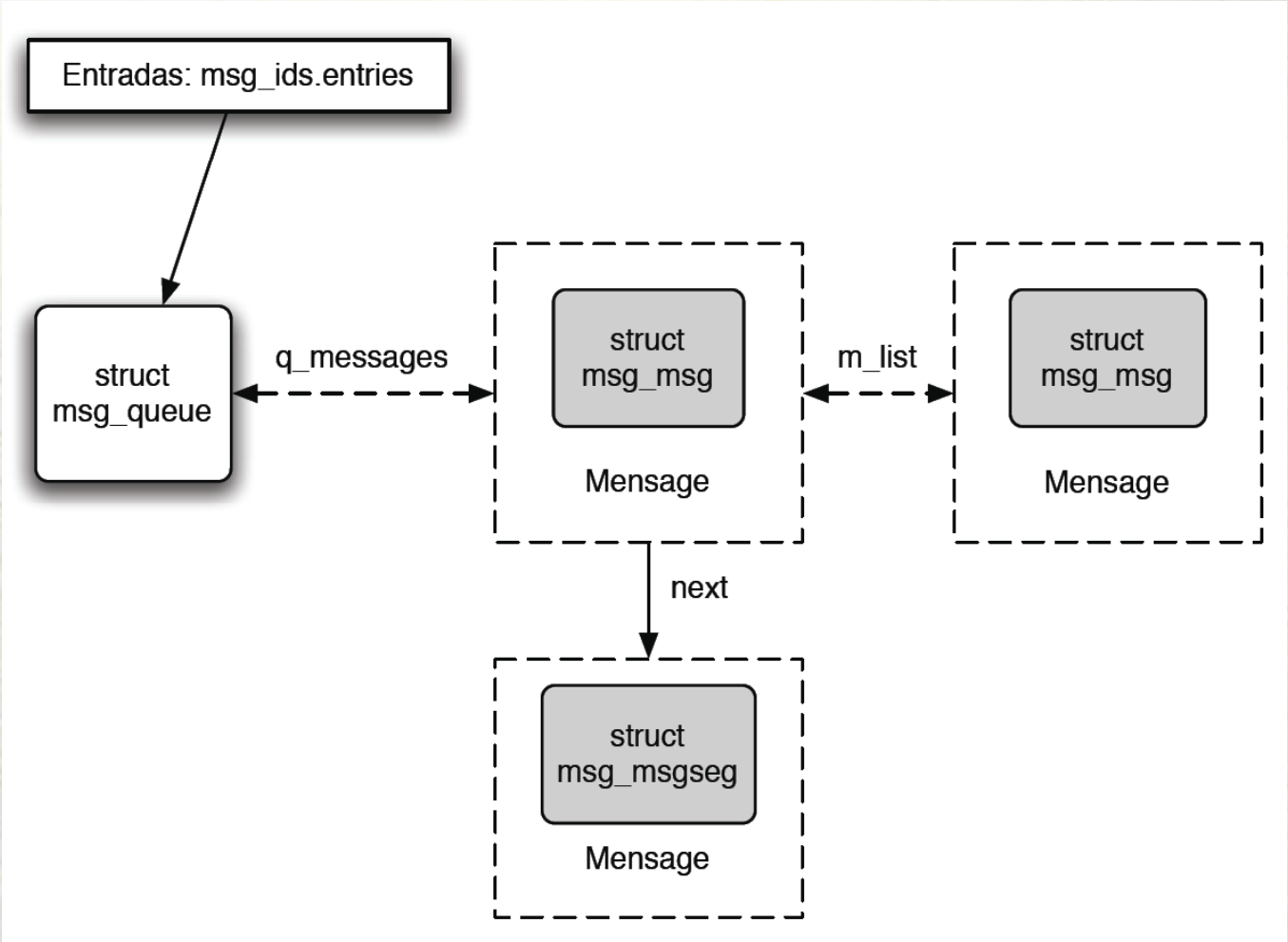
Comandos IPCs

Por defecto, las tres categorías de objetos son mostrados. A continuación un ejemplo de salida de IPCs:

```
------ Shared Memory Segments --------
shmid owner perms bytes nattch status
------ Semaphore Arrays --------
semid owner perms nsems status
------ Message Queues --------
msqid owner perms used-bytes messages
0          root     660          5            1
```

Colas de mensajes

Colas de mensajes



Colas de mensajes

- Las colas de mensajes, son listas enlazadas dentro del espacio de direccionamiento del kernel.
- Los mensajes pueden ser retirados de la cola en cualquier orden, cada uno con un único ID

Estructuras básicas:

- Buffer de mensajes
- Estructura msg.
- Estructura msqid_ds
- Estructura ipc_perm

Buffer de mensajes

No es una estructura del kernel, sino del programador. Esta es una plantilla de su aspecto.

El tamaño máximo posible para toda la estructura está definido en la macro MSGMAX, actualmente de 4Kbytes.

```
/* buffer de mensajes para las llamadas a msgsnd y msgrcv */
struct msgbuf {
        long mtype;
        char mtext[];
};
```

Estructura msg

El kernel almacena cada mensaje de la cola en una lista simplemente enlazada donde cada nodo es una estructura msg.

```
struct msg {
 struct msg *msg_next; /* Siguiente mensaje de la cola */
 long msg_type;
 char *msg_spot; /* puntero al mensaje */
 short msg_ts; /* tamaño del mensaje */
};
```

Estructura msqid_ds

Cada uno de los tres tipos IPC tienen una estructura de datos interna. Para las colas de mensajes es la estructura “msgid_ds” definidia en “/linux/msg.h” ,posee los siguientes campos:

```
struct msqid_ds {
        struct ipc_perm msg_perm; /* estructura de permisos*/
        struct msg *msg_first; /* primer mensaje de la cola no utilizado*/
        struct msg *msg_last; /* ultimo mensaje no utilizado en la cola */
        __kernel_time_t msg_stime; /* ult.imo mensaje enviado */
        __kernel_time_t msg_rtime; /* ultimo mensaje recibido */
        __kernel_time_t msg_ctime; /* ultimo cambio */
        unsigned long msg_lcbytes; /* reuso de un campo para  32 bit */
        unsigned long msg_lqbytes; /* evitar la repeticion */
        unsigned short msg_cbytes; /* numero de bytes en la cola actual */
        unsigned short msg_qnum; /* numero de mensajes en la cola */
        unsigned short msg_qbytes; /* numero maximo de bytes en la cola */
        __kernel_ipc_pid_t msg_lspid; /* PID del ultimo mensaje */
        __kernel_ipc_pid_t msg_lrpid; /* Ultimo PID recivido */};
```

Estructura ipc_perm

Esta es la estructura que guarda los permisos de acceso a cada mensaje de la cola:

```
struct ipc_perm{
key_t  key;  ushort uid;   /* Propietario de euid y egid */
ushort gid;  ushort cuid;  /*Creador de euid y egid */
ushort cgid;  ushort mode;  /* modo de acceso */
ushort seq;   /* Slot con la secuencia de numeros que se utiliza*/
};
```

Llamadas al sistema

Llamadas al sistema

	Cola de mensajes	Semáforos	Memoria compartida
Cabecera	sys/msg.h	sys/sem.h	sys/shm.h
Función para crear o abrir	msgget	semget	shmget
Función para controlar operaciones	msgctl	semctl	shmctl
Función para operaciones IPC	msgsnd msgrcv	semop	shmat shmdt

msgget()

Se utiliza para crear un nuevo mensaje en la cola o para acceder a uno ya existente

El parámetro de flags es siempre IPC_CREAT, con ello añade un mensaje si no lo encuentra en la cola o devuelve su identificador si lo encuentra. Si queremos que la función falle cuando el mensaje ya existe añadimos IPC_EXCL.

Msgget()

SYSTEM CALL:

msgget();

PROTOTYPE:

int msgget (key_t key, int msgflg);

RETURNS:

message queue identifier on success -1 on error:

errno = EACCESS (permission denied)

EEXIST (Queue exists, cannot create)

EIDRM (Queue is marked for deletion)

ENOENT (Queue does not exist)

ENOMEM (Not enough memory to create queue)

ENOSPC (Maximum queue limit exceeded)

Msgsnd()

Es la función para enviar mensajes a la cola, cuando conocemos el identificador de ésta.

Tiene dos flags. Con IPC_NOWAIT, cuando la cola está llena no se escribe, con lo que no bloquea el programa.

Cuando el tamaño de un mensaje es más grande que “msgz” se produce un error salvo si se utiliza el flag MSG_NOERROR, en cuyo caso se trunca.

Msgsnd()

SYSTEM CALL:

msgsnd();

PROTOTYPE:

int msgsnd (int msqid, struct msgbuf *msgp, int msgsz, int msgflg);

RETURNS:

0 si fue exitoso -1 en caso de error:

errno = EAGAIN (queue is full, and IPC_NOWAIT was asserted)

EACCES (permission denied, no write permission)

EFAULT (msgp address isn't accessable - invalid)

EIDRM (The message queue has been removed)

EINTR (Received a signal while waiting to write)

EINVAL (Invalid message queue identifier, nonpositive message type, or invalid message size)

ENOMEM (Not enough memory to copy message buffer)

Msgctl()

Realiza operaciones de control sobre mensajes de la cola, los comandos son

IPC_STAT: Coge una copia de la estructura msqid_ds.

IPC_SET: Establece los permisos de acceso a una cola.

IPC_RMID:Borra una cola.

msgctl()

SYSTEM CALL:

msgctl();

PROTOTYPE:

int msgctl (int msgqid, int cmd, struct msqid_ds *buf);

RETURNS:

0 on success -1 on error:

errno = EACCES (No read permission and cmd is IPC_STAT)

EFAULT (Address pointed to by buf is invalid with IPC_SET and IPC_STAT commands)

EIDRM (Queue was removed during retrieval)

EINVAL (msgqid invalid, or msgsz less than 0)

EPERM (IPC_SET or IPC_RMID command was issued, but calling process does not have write (alter) access to the queue)

Creación y busqueda de colas - I

•La llamada al sistema msgget() cumple dos funciones:

1. Creación de una nueva cola de mensajes

2. Búsqueda de una cola de mensajes existentes (creada por la otra aplicación, por ejemplo) mediante su clave

•En ambos casos, sólo se puede usar si se posee una clave.

Creación y busqueda de colas - II

•El prototipo de esta llamada al sistema es el siguiente:

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
int msgget (key_t clave, int opcion);
```

•Como parámetros de entrada posee la clave de la cola que ya existe o de la que se desea crear y el tipo de opción

•crear. Si se pasa como clave el valor IPC_PRIVATE, se crea una cola.

Creación y busqueda de colas - III

- Si se pasa una clave diferente, se presenta dos posibilidades:
    1. La clave no es utilizada por otra cola de mensajes, en este caso es necesario pasar el valor IPC_PRIVATE, creándose la cola con la clave pasada.
    2. La clave se utiliza en otra cola de mensajesLa clave se utiliza en una cola de mensajes. Es necesario que se pase IPC_CREAT o bien IPC_EXCL como parámetro.A continuación se puede leer o escribir en la cola de mensajes, si los derechos lo permiten.
- Si todo ocurre correctamente, **msgget** devuelve el identificador de la cola de mensajes

Creación y busqueda de colas - IV

```
asmlinkage int sys_msgget (key_t key, int msgflg)
{
        int id, ret = -EPERM;
        struct msqid_ds *msq;
        lock_kernel();                  // Bloquea el nucleo del kernel
        if (key == IPC_PRIVATE)         //Compara el valor de clave
                ret = newque(key, msgflg);
        else if ((id = findkey (key)) == -1)
        { /* clave no usada */
        if (!(msgflg & IPC_CREAT))
                ret = -ENOENT; // Si esta creado es error
        else
                ret = newque(key, msgflg); //Sino creamos la cola
        } else
.
.
.
```

Creación y busqueda de colas - V

```
if (msgflg & IPC_CREAT && msgflg & IPC_EXCL) {
ret = -EEXIST; //
}
else {
        msq = msgque[id];
        if (msq == IPC_UNUSED || msq == IPC_NOID)
        ret = -EIDRM;
        else if (ipcperms(&msq->msg_perm, msgflg))
        ret = -EACCES;
        else
        ret = (unsigned int) msq->msg_perm.seq * MSGMNI + id;
    }
unlock_kernel();
return ret;
}
```

Emisión de mensajes - I

La llamada al sistema **msgsnd** permite enviar un mensaje a una cola de mensajes.

Su prototipo es el siguiente:

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

#include<sys/ipc.h>

#include<sys/msg.h>

int msgsnd (int msqid, struct msgbuf *msgp, int msgsize, int msgopt);
```

Emisión de mensajes - II

- Para que esta operación se desarrolle normalmente, es necesario haber obtenido el identificador de la cola de mensajes
- Poseer los derechos necesarios para escribir en la cola de mensajes.
- El segundo parámetro corresponde a los datos que se quiere enviar a la cola de mensajes.
- El parámetro **msgsize** corresponde al tamaño del objeto que se envía a la cola.

Emisión de mensajes - III

La llamada es la siguiente:

```
lock_kernel();
ret = real_msgsnd(msqid, msgp, msgsz, msgflg);
unlock_kernel();
```

Emisión de mensajes - IV

```
static int real_msgsnd (int msqid, struct msgbuf *msgp, size_t msgsz,
int msgflg)
{
int id;
struct msqid_ds *msq; //Estructuras principales
struct ipc_perm *ipcp;
struct msg *msgh;
long mtype; //Comprobación tamaño máximo msg
if (msgsz > MSGMAX || (long) msgsz < 0 || msqid < 0)
return -EINVAL;
...
```

Emisión de mensajes - V

```
/* Situar la cabecera del mensaje y el espacio de texto*/
msgh = (struct msg *) kmalloc (sizeof(*msgh) + msgsz,
GFP_KERNEL);
if (!msgh)
return -ENOMEM;
msgh->msg_spot = (char *) (msgh + 1);
//Realizar una copia de msgp
if (copy_from_user(msgh->msg_spot, msgp->mtext, msgsz)) {
        kfree(msgh);   //En caso de error libera la memoria
        return -EFAULT;
}
…
```

Emisión de mensajes - VI

```
if (msgque[id] == IPC_UNUSED || msgque[id] == IPC_NOID
|| msq->msg_perm.seq != (unsigned int) msqid / MSGMNI)
{ //Error por permisos, IPC no usado
kfree(msgh);
return -EIDRM;
}
msgh->msg_next = NULL;  //Continua inicializando msgh (local)
msgh->msg_ts = msgsz;   //valores pasados por parámetro.
msgh->msg_type = mtype;
msgh->msg_stime = CURRENT_TIME;
...
```

Emisión de mensajes - V

```
if (!msq->msg_first) //Se toma el siguiente mensaje
            msq->msg_first = msq->msg_last = msgh;
else {      //Si ya existia uno avanza.
            msq->msg_last->msg_next = msgh;
            msq->msg_last = msgh;}
msq->msg_cbytes += msgsz;
msgbytes += msgsz;
msghdrs++;
msq->msg_qnum++;
msq->msg_lspid = current->pid;
msq->msg_stime = CURRENT_TIME;
wake_up (&msq->rwait);
return 0;
}
```

Control de las colas de mensajes - I

•Tras haber creado una cola de mensajes, es posible manipularla modificando por ejemplo los permisos de acceso a la cola.

•Los IPC se gestionan en el núcleo por la tabla de colas de mensajes.

• La llamada al sistema **msgctl** permite acceder y modificar ciertos campos de esta tabla para las colas a las que se tiene acceso.

Control de las colas de mensajes - II

```
asmlinkage int sys_msgctl (int msqid, int cmd, struct msqid_ds *buf)
{
int id, err = -EINVAL;
struct msqid_ds *msq;
struct msqid_ds tbuf;
struct ipc_perm *ipcp;
lock_kernel();

switch (cmd) {
...
 id = (unsigned int) msqid % MSGMNI;
msq = msgque [id];
err = -EINVAL;
```

Control de las colas de mensajes - III

```
ipcp = &msq->msg_perm;
switch (cmd) {
        case IPC_STAT://Coge una copia de la estructura msqid_ds.
         err = -EACCES;   //Si tiene permiso
        if (ipcperms (ipcp, S_IRUGO)) goto out;
        tbuf.msg_perm = msq->msg_perm; //estructura msqid
        tbuf.msg_stime = msq->msg_stime;//inicializada
        tbuf.msg_rtime = msq->msg_rtime;
        tbuf.msg_ctime = msq->msg_ctime;
        tbuf.msg_cbytes = msq->msg_cbytes;
        tbuf.msg_qnum = msq->msg_qnum;
        tbuf.msg_qbytes = msq->msg_qbytes;
        tbuf.msg_lspid = msq->msg_lspid;
        tbuf.msg_lrpid = msq->msg_lrpid;
...
```

Control de las colas de mensajes - IV

```
case IPC_SET: //Establece los permisos de acceso a la cola.
err = -EPERM;
if (current->euid != ipcp->cuid && current->euid != ipcp->uid &&
                !capable(CAP_SYS_ADMIN))  goto out;
if (tbuf.msg_qbytes > MSGMNB &&
            !capable(CAP_SYS_RESOURCE)) goto out;
msq->msg_qbytes = tbuf.msg_qbytes;
ipcp->uid = tbuf.msg_perm.uid;  //permisos
ipcp->gid = tbuf.msg_perm.gid;
ipcp->mode = (ipcp->mode & ~S_IRWXUGO) |
(S_IRWXUGO & tbuf.msg_perm.mode);
msq->msg_ctime = CURRENT_TIME;
err = 0;
goto out;
```

Control de las colas de mensajes - V

```
case IPC_RMID: //Permiso para borrar la cola
err = -EPERM;    //comprobación pertinente.
if (current->euid != ipcp->cuid && current->euid != ipcp->uid &&
            !capable(CAP_SYS_ADMIN))  goto out;
freeque (id);
err = 0;
goto out;
default:
err = -EINVAL;
goto out;
}
out:
unlock_kernel();
return err;
}
```

FIN