Memoria Compartida

C Prácticas

La forma más rápida de comunicar dos procesos es hacer que compartan una zona de memoria. En esta entrada explicamos cuál es el mecanismo en linux para poder compartir memoria entre varios procesos. En otra entrada pondremos un ejemplo del uso de la memoria compartida entre procesos padre e hijos.

La forma más rápida de comunicar dos procesos es hacer que compartan una zona de memoria. Para enviar datos de un proceso a otro, sólo hay que escribir en memoria y automáticamente esos datos están disponibles para que los lea cualquier otro proceso.

La memoria convencional direccionada por un un proceso es local al mismo y cualquier intento de direccionar esa memoria desde otro proceso provoca una violación de segmento.

Tenemos una serie de llamadas al sistema para poder gestionar memoria compratida entre varios procesos. Las librerías relacionadas son:

```
<sys/types.h>
<sys/ipc.h>
<sys/shm.h>
```

Las llamadas al sistema son las siguientes:

- shmget, para crear una zona de memoria compartida o habilitar el acceso a una ya creada.
- shmctl, para acceder y modificar la información administrativa y de control que el nucleo asocia a cada zona de memoria compartida.
- shmat, para vincular una zona de memoria compartida a un proceso.
- shmdt, para desvincular una zona de memoria previamente vinculada.

Para usar memoria compartida en Linux es necesario seguir una serie de pasos usando las llamadas al sistema que hemos presentado.

Para trabajar con memoria compartida tenemos que:

- Pedir la clave de la zona de memoria, que es un identificador de recurso compartido.
- Crear la el segmento de memoria compartida usando la clave. El proceso que la crea ya está vinculado a ella y por tanto puede usarla
- Otro proceso se vincula con la misma clave al segmento de memoria compartida y entonces puede usarla y desvincularse si quiere.
- El proceso que creó la zona de memoria la debe destruir.

Petición de la clave de la memoria compartida

ftok es una función que permite obtener un identificador para un recurso compartido, en este caso memoria compartida, pero también se usa para colas de mensajes y semáforos.

```
key_t ftok(char *, int)
```

a la que se suministra como primer parámetro el nombre y path de un fichero cualquiera que exista y al que se tenga acceso y como segundo un entero cualquiera.

Importante: Todos los procesos que quieran compartir la memoria, deben suministrar el mismo fichero y el mismo entero.

Creación de la memoria compartida

Con shmget vamos a obtener un identificador con el que podamos realizar futuras llamadas al sistema para controlar una zona de memoria compartida. Su declaración es:

```
int shmget(key_t key, int size, int shmflg)
```

- key es una clave que tiene el mismo significado que vimos en Semáforos. Utilizaremos ftok para crear la clave.
- size es el tamaño en bytes de la zona que queremos crear
- shmflg es una máscara de bits que tiene el mismo significado que vimos con Semáforos.

Si la llamada se ejecuta con éxito devuelve la clave asociada a la zona de memoria, si falla devolverá -1 y en errno estará el código de error. **La clave asociada es heredada por los procesos descendientes**.

El siguiente código muestra cómo crear una zona de memoria de 4096 bytes donde sólo el usuario va a a tener permisos de lectura y escritura

```
int shmid;
...
if ((shmid = shmget ( IPC_PRIVATE, 4096, IPC_CREAT | 0600)) == -1 ){
   /*Error en la creación de memoria compartida*/
   }
....
```

Control de la zona de memoria compartida

Con shmctl podremos realizar operaciones de control sobre la memoria compartida creada previamente con shmget e identificada con la clave obtenida.

```
int shmctl( int shmid, int cmd, struct shmid_ds *buf);
```

Donde:

- shmid, es la clave obtenida con shmget
- cmd indica el tipo de operación de control que queremos realizar:
  - IPC_STAT: Lee el estado de la estructura de control de la memoria y lo devuelve a través de la zona de memoria apuntada por buf. Para ver información sobre los campos de buf consultar shmctl(2)
  - IPC_SET: Inicializa algunos de los campos de la estructura de control de la memoira compartida. El valor de estos campos los toma de la estructura apuntada por buf.
  - IPC_RMID: Elimina del sistema la zona de memoria compartida identificada por shmid. Si el segmento de memoria está vinculada a varios procesos, el borrado no se hace efectivo hasta que todos los procesos liberen (se desvinculen) de la memoria.
  - SHM_LOCK: Bloquea en memoria el segmento identificado por shmid. Esto quiere decir que no se realizará Intercambio (pasar a memoria secundaria) sobre el. Sólo root va a poder hacer esta operación.
  - SHM_UNLOCK: Desbloquea el segmento de memoria compartida, con lo que los mecanismos de Intercambio van a poder trasladarlo a memoria secundaria y viceversa. Sólo root va a poder hacer esta operación.

La estructura shmid_ds se define como sigue:

```
struct shmid_ds {
    struct          ipc_perm     shm_perm; /* Ownership and permissions */
    size_t          shm_segsz; /* Size of segment (bytes) */
    time_t       shm_atime;  * Last attach time */
    time_t       shm_dtime; /* Last detach time */
    time_t       shm_ctime; /* Last change time */
    pid_t         shm_cpid; /* PID of creator */
    pid_t         shm_lpid; /* PID of last shmat(2)/shmdt(2) */
    shmatt_t  shm_nattch; /* No. of current attaches */
    ...
    };
```

La siguiente línea muestra cómo borrar del sistema una zona de memoria compartida

```
shmctl( shmid, IPC_RMID, 0)
```

Operaciones con la memoria compartida

Podemos vincularnos con shmat, desvincularnos con shmdt y finalmente eliminar el segmento de memoria compartida con shmctl.

Atención!!!, el segmento queda asignado en memoria aunque el proceso termine, cuando ejecutemos de nuevo nuestro proceso para utilizar un segmento del mismo tamaño y con la misma key nos asignará el que ya tiene asignado, podremos cambiar su contenido. Si ejecutamos con la misma key pero queremos cambiar el tamaño nos dará error la función shmget. Para eliminar el segmento de memoria compartida, al terminar el programa utilizar *shmctl(shmid, IPC_RMID, 0)*

Antes de poder usar una zona de memoria compartida tenemos que asignarla al espacio de direcciones virtuales de nuestro proceso, esto es lo que se conoce como vincularse o unirse (attach) a la zona de memoria compartida.

Una vez que dejamos de usar la zona de memoria compartida tendremos que desvincularnos o desatarnos de ésta. Cuando nos desvinculamos, el segmento de memoria compartida no se destruye, simplemente ya no está accesible desde nuestro proceso.

Las llamadas son las siguientes:

```
char *shmat (int shmid, char *shmaddr, int shmflg);
int shmdt(char *shmaddr);
```

donde:

- shmid es el identificador de una zona de memoria creada con shmget
- shmaddr es la dirección virtual donde queremos que empiece la zona de memoria compartida, normalmente dejaremos al núcleo que gestione esto con lo que le pasaremos un 0. Si la función termina con éxito nos devuelve un puntero a la zona de memoria virtual de nuestro proceso donde ha vinculado la zona compartida. Lo que haremos es igualar este valor a un puntero de nuestro proceso para poder acceder a la zona de memoria a partir de dicho puntero. En el caso de la función shmdt, shmaddr es la dirección del segmento de memoria compartida que queremos desvincular, es decir, el puntero que asignamos en shmat.
- shmflg es una máscara de bits que indica la forma de acceso a memoria, entre otros, puede ser:
  - SHM_EXEC: Lo que permite al proceso que se vincula ejecutar el contenido del segmento de memoria. Tiene que tener permisos de ejecución.
  - SHM_RDONLY: Únicamente permitirá el acceso en lectura al segmento vinculado. El proceso debe tener permiso de lectura. No se entiende un acceso en sólo escritura a un segmento de memoria compartida. Si no se indica este flag, se

otorga permisos de lectura y escritura al segmento de memoria compartida. El proceso tiene que tener ambos permisos.

Una vez que el proceso está vinculado al segmento de memoria compartida, entonces, a través del puntero al que hemos igualado la llamada a shmat podremos acceder a la memoria como con cualquier otro puntero.

Cuando otro proceso escriba en la memoria, el proceso podrá leer aquello escrito vía el puntero. Hay que tener en cuenta la sincronización de procesos para poder leer y escribir datos consistentemente, lo normal es utilizar semáforos para la sincronización de los procesos que comparten memoria.

← Entrada anterior

Entrada siguiente →