6 Memoria compartida

Introducción

Como se ha visto, la creación de procesos permite elaborar programas poderosos siempre y cuando pueda existir comunicación entre dichos procesos. En principio podemos pensar en el uso de un archivo de lectura y escritura como un rudimentario mecanismo para comunicar dos o más procesos. Sin embargo, ésta no es una solución óptima si consideramos que los tiempos para acceder a un archivo en dispositivos mecánicos son muy lentos comparados con los tiempos de acceso a la memoria. Por ello se diseñó en UNIX® System V un mecanismo que permite a dos o más procesos obtener un apuntador a una dirección de memoria a partir de la cual se han reservado *n* bytes contiguos. Dichos procesos podrán acceder a esa memoria común para leer o escribir, lo cual permite la comunicación entre los mismos. Cabe aclarar que al ser esta memoria un recurso compartido, pueden presentarse condiciones de competencia.

Dado que un sistema UNIX® es multiusuario se presupone que pueden existir varios proyectos en el mismo sistema de cómputo. Es evidente que la comunicación entre los procesos implicados en cada proyecto requiere independencia respecto a mecanismos de comunicación de otros procesos. Por esta razón se incluyó el concepto de llave, el cual podemos entender como un número clave bajo el cual se van a agrupar uno o varios mecanismos de comunicación. Si dos procesos quieren compartir una región de memoria deberán conocer ambos dicha clave, de otro modo será imposible que se comuniquen.

Ejemplo práctico

Compile y ejecute el código del programa 6-1 en dos terminales distintas y después en tres o más. No escriba los números al inicio de las líneas, sólo son referencias para posteriores explicaciones.

Programa 6-1 Comparte un área de memoria entre varios procesos

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main(int argc, char *argv[])
{
   int shmid, *variable;
   key_t llave;

1  llave = ftok(argv[0], 'K');

2  if((shmid = shmget(llave, sizeof(int), IPC_CREAT | 0600)) ==
-1)
   {
      perror("Error en shmget");
      exit(-1);
   }
   /* Nos atamos a la memoria compartida */
3  if((variable = (int *)shmat(shmid, NULL,  0)) == (int *)(-1))
   {
      perror("Fallo shmat");
      exit(-1);
   }

   while(1)
   {
      printf("\nIntroduzca m para modificar el valor de la
variable, v para visualizarla y t para terminar:\n ");

      switch(leer_car()) {
      case 't':
         /* Libera la memoria compartida */
         shmctl(shmid, IPC_RMID, 0);
         exit(0);
      case 'v':
         /* Visualiza la variable */
         printf("variable = %d\n", *variable);
         break;
      case 'm':
         printf("Nuevo valor de la variable en memoria
compartida: \n");
         scanf("%d", variable);
         break;
      default:
         printf("Se introdujo una letra incorrecta = %d\n");
         break;
      }
   }
}

int leer_car()
{
   char letra;
   char almacen[80];

   scanf("%s", &almacen);
   sscanf(almacen, "%c", &letra);
   return letra;
}
```

Descripción del funcionamiento

Línea 1 Dado que dos o más procesos se pueden comunicar mediante un mecanismo IPC (*inter process comunications*), se requiere un identificador único o llave de IPC que pueda servir de referencia común a los procesos que quieran hacer uso de dicho IPC. La función `ftok` devuelve una llave basada en un nombre de un archivo y una letra. Para un mismo nombre de archivo y letra siempre se devuelve la misma llave.

Pregunta 6.1

Imprima el valor de la llave y observe que es el mismo en cada ejecución. Cambie la letra y compare la llave devuelta con la anterior. Introduzca el nombre de cualquier otro archivo que exista en la misma carpeta (puede ser en carpeta distinta pero se requiere la ruta absoluta), como primer parámetro de `ftok`. Por último ponga el nombre de un archivo que no exista en el sistema. Escriba los valores de las llaves obtenidas en los casos siguientes:

Llave devuelta por `ftok(argv[0], 'K');`
Llave devuelta por `ftok(argv[0], ' ') ;`
Llave devuelta por `ftok("      ", 'K');`
Llave devuelta por `ftok("./chetito", 'K') ;`

Línea 2 Al usar `shmget` obtendremos un identificador, con el cual podremos realizar llamadas al sistema que controlan la zona de memoria compartida. Su primer parámetro es la llave devuelta por `ftok`, el segundo es el tamaño en *bytes* de la zona de memoria que se desea crear. En el tercer parámetro, `IPC_CREAT` indica que la memoria compartida se creará siempre y cuando otro proceso no la haya creado antes, 0666 proporciona permisos de lectura y escritura en la memoria compartida (véase el manual del comando `chmod`). El id devuelto por `shmget` es heredado por los procesos descendientes del actual.

Línea 3 Del mismo modo que una variable declarada en lenguaje C nos permite acceder a una dirección de memoria, para acceder a la memoria compartida se requiere asociarle un nombre y poder hacer referencia a ella. Esto se realiza mediante `shmat`, la cual devuelve un apuntador (dirección de memoria) a una cadena de caracteres. Dicha dirección es la de inicio de la memoria compartida. El primer parámetro es el identificador devuelto por `shmget` el segundo y tercer parámetro con los valores `NULL` y 0 respectivamente, permiten dejar la decisión al sistema operativo sobre la dirección de inicio de la memoria compartida.

Ahora queremos comprobar que si se crea un nuevo proceso inmediatamente después de ejecutar `shmat`, ambos procesos (padre e hijo) tendrán los mismos valores de llave `shmid`, y de dirección de memoria devuelta por `shmat`.

Pregunta 6.2 Modifique el programa original para imprimir los tres valores mencionados en el párrafo anterior tanto por el padre como por el hijo. No olvide imprimir la dirección de memoria en formato hexadecimal (`"%x"`).

Pregunta 6.3 ¿Qué puede ocurrir si no se imprime la dirección de memoria en formato hexadecimal?

Ejercicio 6.1 Modifique el programa original para simular una pila de 10 elementos de tipo carácter. Un proceso debe funcionar como productor e introducir elementos en la pila, y otro proceso debe funcionar como consumidor sacando elementos de esta. Debe existir un tercer proceso que imprima el contenido completo de la pila en la pantalla cada vez que ésta sea modificada.

Pregunta 6.4 `Scanf` en cualquier compilador es vulnerable a los datos adicionales que están al fin de la línea (como lo es la tecla "fin de línea") y que lee el siguiente `scanf`, por lo que se recomienda utilizarlo junto con `sscanf`. Modifique el programa para tratar de utilizar sólo una línea `scanf` en la función `leer_car`, ¿qué sucede?