Sistemas Operativos Multimedia

Práctica 1.2

Comunicación entre procesos: memoria compartida

MEMORIA COMPARTIDA

Tal y como hemos estudiados, cuando ejecutamos la llamada `fork` se crea un nuevo proceso, idéntico al padre, en el que se duplican las variables, funciones, etc. Si el hijo actualiza una de sus variables, el proceso padre no tiene tal actualización ya que son variables distintas (son copias). Para que el padre y el hijo tengan la misma variable y se puedan comunicar es necesario emplear las llamadas al sistema que permiten gestionar la memoria compartida.

Para utilizar la memoria compartida se siguen 3 pasos:

- Obtención de un segmento de memoria compartida con su identificador en el sistema (un número entero). Se hace con la llamada `shmget`.
- Vincular el segmento de memoria compartida a través de un puntero a un determinado tipo de datos. Se usa la llamada `shmat`.
- Acceder a la memoria compartida por medio del puntero.
- Desvincular el puntero del segmento de la memoria compartida. Se usa la llamada `shmdt`.
- Eliminación del segmento de memoria compartida. Se usa la llamada `shmctl`.

A continuación se describen brevemente estas llamadas.

LLAMADA SHMGET

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

La llamada `shmget` devuelve el identificador de memoria compartida asociado al segmento de memoria. Tiene como parámetros:

- La clave que identifica al segmento de memoria compartida. Puede ser IPC_PRIVATE (el proceso y sus descendientes pueden acceder al segmento) u otra clave obtenida mediante la función `ftok` (varios procesos sin parentesco pueden acceder al segmento).
- El tamaño del segmento.
- Indicadores de permisos para acceder al recurso compartido.

Si la ejecución se realiza con éxito, entonces devolverá un valor no negativo denominado identificador de segmento compartido. En caso contrario, retornará -1 y la variable global `errno` tomará en código del error producido.

LLAMADA SHMAT

```
char *shmat( int shmid, void *shmaddr, int shmflg )
```

Esta llamada asocia o vincula el segmento de memoria compartida especificado por `shmid` (el identificador devuelto por `shmget`) al segmento de datos del proceso invocador. Presenta tres parámetros:

- El identificador obtenido mediante `shmget`,

- La dirección donde se desea que se incluya (en la práctica será NULL (0), lo cuál permite que se incluya en cualquier zona libre del espacio de direcciones del proceso).
- Una serie de identificadores de permisos (en la práctica también será NULL).

Si la llamada se ejecuta con éxito, entonces devolverá la dirección de comienzo del segmento compartido, si ocurre un error devolverá -1 y la variable global `errno` tomará el código del error producido.

LLAMADA SHMDT

```
int shmdt ( char *shmaddr)
```

Desasocia o desvincula del segmento de datos del proceso invocador el segmento de memoria compartida ubicado en la localización de memoria especificada por `shmaddr`. Si la función se ejecuta sin error, entonces devolverá 0, en caso contrario retornará -1 y `errno` tomará el código del error producido.

LLAMADA SHMCTL

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

La llamada `shmctl` permite realizar un conjunto de operaciones de control sobre una zona de memoria compartida identificada por `shmid`. El argumento `cmd` se usa para codificar la operación solicitada. Los valores más usados para este parámetro son:

- `IPC_STAT`: lee la estructura de control asociada a `shmid` y la deposita en la estructura apuntada por `buff`.
- `IPC_RMID`: elimina el identificador de memoria compartida especificado por `shmid` del sistema, destruyendo el segmento de memoria compartida y las estructuras de control asociadas.
- `SHM_LOCK`: bloquea la zona de memoria compartida especificada por `shmid`. Este comando sólo puede ejecutado por procesos con privilegios de acceso apropiados.
- `SHM_UNLOCK`: desbloquea la región de memoria compartida especificada por `shmid`. Esta operación, al igual que la anterior, sólo la podrán ejecutar aquellos procesos con privilegios de acceso apropiados.

La función `shmctl` retornará el valor 0 si se ejecuta con éxito, o -1 si se produce un error, tomando además la variable global `errno` el valor del código del error producido.

EJEMPLOS DE USO

Como primer ejemplo vamos a ver cómo se comparte la variable entera `numero`. Para ello, tendremos como variables globales la variable `numero` y el identificador de la memoria compartida `shmid`.

```
int shmid;
int *numero=NULL;
```

Para las llamadas anteriores hay que incluir los archivos de cabecera:

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

La función `creaComp` crea el identificador de la memoria compartida y la variable que contiene.

```
int creaComp(void)
{
      if((shmid=shmget(IPC_PRIVATE,sizeof(int),IPC_CREAT|0666))==-1)
      {
            perror("Error al crear memoria compartida: ");
            return 1;
      }

      /* vinculamos el segmento de memoria compartida al proceso */
      numero=(int *) shmat (shmid,0,0);

      /*iniciamos las variables */
      *numero=0;

   return 0;
}
```

La función `borraComp` separa la variable `numero` del proceso y a continuación elimina la memoria compartida.

```
void borraComp(void)
{
   char error[100];
   if (numero!=NULL)
   {
      /* desvinculamos del proceso la memoria compartida */
      if (shmdt((char *)numero)<0)
      {
          sprintf(error,"Pid %d: Error al desligar la memoria compartida:
",getpid());
         perror(error);
         exit(3);
      }
```

```
      /* borramos la memoria compartida */
      if (shmctl(shmid,IPC_RMID,0)<0)
      {
            sprintf(error,"Pid %d: Error al borrar memoria compartida:
",getpid());
         perror(error);
         exit(4);
      }
      numero=NULL;
      }
}
```

Como segundo ejemplo vamos a ver cómo todos y cada uno de los procesos de un árbol escriben sus pid en la memoria compartida, y el proceso principal los lee y muestra dicha información.

```
include <stdio.h>
include <unistd.h>
include <stdlib.h>
include <fcntl.h>

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

int main()
 {
  pid_t* mispids;
   // se crea la memoria compartida
   int shmid = shmget(IPC_PRIVATE,sizeof(pid_t),IPC_CREAT|0666);

   // se vincula la memoria
   mispids = (int *)shmat(shmid,0,0);

   pid_t padre = getpid();

   // el proceso principal guarda su pid en memoria
   *(mispids) = getpid();
   printf("Soy el proceso principal, mi pid es %d, y he guardado en memoria
%d\n",getpid(),(int)(*(mispids)));

   // generamos los hijos en horizontal que escriben en memoria
   int j;

   for(j=0;j<2;j++)
    {
     pid_t pidy = fork();

     if(pidy==0) // código del hijo
      {
        mispids = (int *)shmat(shmid,0,0); // se vincula la memoria en los
procesos hijo
       *(mispids+j+1) = getpid();          // se escribe en memoria
       printf("Soy el hijo con pid %d, la j es %d y he guardado en memoria
%d\n",getpid(),j,(int)(*(mispids+j+1)));
       break;
      }
```

```
    else // código del padre
     {
     }
   }
   // una vez creado el árbol de procesos
   // incorporamos un tiempo de espera para
   // verificarlo mediante el comando pstree
   sleep(15);

   // proceso de lectura de memoria desde el proceso principal

   if (getpid() == padre)
    {
      sleep(2);

     printf("Soy el proceso principal con pid(%d) y mis hijos son: ",(int)
(*(mispids)));

      // bucle de impresión de los valores
      // almacenados por los hijos
      int cont;

      for(cont=1;cont<=2,cont++)
       {
        printf("pid(%d) ",(int)(*(mispids+cont)));
       }
      printf("\n");

      // se desvincula la memoria y se borra
      shmdt((pid_t *)mispids);
      shmctl(shmid,IPC_RMID,0);
    }
   else
    {
     // se desvincula la memoria desde los procesos hijo
     shmdt((pid_t *)mispids);
    }
   return 0;
 }
```

REFERENCIAS

BIBLIOGRAFÍA

- F. M. Márquez: UNIX. Programación avanzada, Rama, 2004.
- K. A. Robbins y S. Robbins: UNIX. Programación práctica, Prentice may, 2000.