



INSTITUTO POLITÉCNICO NACIONAL
UNIDAD PROFESIONAL INTERDISCIPLINARIA
INGENIERÍA Y TECNOLOGÍAS AVANZADAS



SEÑALES Y MEMORIA COMPARTIDA

PROFESORA:

Susana Sanchez Najera

INTEGRANTES DEL EQUIPO:

Vargas Sánchez Andrea Liliana
Cisneros Martínez Daphne Liliana
Escala Acosta Andres Rafael

GRUPO: 2TV3

Fecha de entrega: 24 de noviembre del 2021

SEÑALES

Una señal es un mensaje que puede dar un proceso a otro, siempre que tengan relación de parentesco (hijo, hermano, papá, etc). Estas señales software se envían en un instante determinado pero el proceso no sabe cuándo le llega, ni quien se la envía.

Dependiendo de la versión de Unix existen diferentes interrupciones o señales, pero en general pueden dividirse en 5 categorías:

- Condiciones hardware: Son enviadas automáticamente por el sistema cuando se produce un error durante la ejecución de un proceso.

SIGSEV : Será enviada a un proceso cuando acceda a direcciones que están fuera de su espacio de direccionamiento o zonas de memoria p g rote idas.

SIGBUS : Cuando se intenta realizar un acceso a un hardware sobre el que no tengo privilegio.

SIGILL : Ejecución de una instrucción ilegal.

- Condiciones software: Señales generadas cuando el usuario tiene un requerimiento específico.

SIGINT : Generada cuando se pulsa ctrl-c (es enmascarable).

SIGQUIT : Generada cuando se pulsa ctrl-esc

SIGTERM : La envía un usuario a un proceso con la orden kill. El proceso morirá a menos que esté en modo sistema, si lo está se esperará a que termine.

SIGHUP : Indica un corte en la línea de comunicaciones, usualmente apagar la terminal.

SIGKILL : Indica muerte imperativa.

SIGUSR1 : Definibles por el usuario.

SIGALRM : Es enviada a un proceso cuando alguno de sus temporizadores descendientes llega a cero.

- Notificaciones de E/S

SIGPOLL : La revive un proceso cuando se está produciendo una e/s por un stream que está siendo gestionado por una interrupción poll.

- Control de Procesos

SIGCONT : Lleva al proceso a background, mientras continúa la ejecución del proceso.

SIGCLD : El hijo le envía esta señal al padre cuando cambia de estado.

- Control de recursos

SIGXCPU : Exceso de consumo de recursos por parte del proceso.

SIGXFSZ : El proceso ha sobrepasado el tamaño máximo para un archivo.

El problema al comunicar señales con procesos es que no se puede enviar mucha información solo un número.

MEMORIA COMPARTIDA

La memoria compartida quiere decir que dos procesos van a compartir la misma área de memoria y pueden acceder al contenido del área. Para poder hacer uso de la memoria compartida, un proceso debe crear un espacio de memoria dedicado a la comunicación, los demás procesos van a asignar a ese espacio como su espacio de direcciones, entonces cuando se lea y se escriba, el área de memoria compartida correspondiente en su espacio de direcciones se va a comunicar con los otros procesos.

Es el método más rápido de comunicación IPC, cada procesos tiene punteros al área de memoria abierta, cuando se accede se comienza a operar directamente como un control de memoria en el proceso.

Para una memoria, la memoria compartida es una parte del proceso que manipula directamente ésta área a través de un puntero, eliminando el proceso de captura de kernel, este proceso va a operar directamente el área de memoria compartida a través del puntero y así elimina la copia de datos entre el modo de usuario y el modo del kernel.

DESVENTAJAS

Como la memoria compartida es la comunicación más rápida entre procesos y la cantidad de información es grande, su administración se vuelve complicada y los procesos deben de estar en la misma máquina físicamente para poder comunicarse.

Su seguridad es baja, se pueden presentar problemas cuando alguno de los procesos contenga algún virus, ya que se va a transmitir infectando al otro proceso.

Las variables globales se deben de tomar en cuenta para comunicarse entre subprocesos ya que en un mismo proceso no se le va a llamar memoria compartida. (Los subprocesos son una secuencia de ejecución dentro del proceso).

Es un recurso crítico, por lo que se debe realizar el control de sincronización entre procesos, y el control de sincronización debe lograrse mejor usando un semáforo.

FUNCIONES

Se debe de comenzar con las librerías:

`<sys/types.h>`

`<sys/ipc.h>`

- Función `shmget`: devuelve un identificador de memoria compartida que es utilizado por otras funciones.

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

donde:

- Primer parámetro: es un valor clave proporcionado por el programa para nombrar la memoria compartida.
 - Segundo parámetro: especifica la cantidad de memoria que se debe compartir en bytes.
 - Tercer parámetro: si no existe la memoria compartida se crea con `IPC_CREAT`, se crea una bandera con permiso de `Onnn`, que va a permitir que el proceso que crea la memoria compartida pueda escribir datos, establezca el proceso creado por otros usuarios en solo lectura.
 - Valor retorno: si se tiene éxito va a devolver un número entero no negativo, es un identificador de memoria compartida.
- Función `shmat`: se conecta el segmento de memoria compartida al espacio de direcciones del proceso.

*`void *shmat(int shm_id, const *shm_addr, int shmflg);`*

donde:

- Primer parámetro: es el identificador de memoria compartida devuelto por la función `shmget`.
- Segundo parámetro: la memoria compartida se conectará a la ubicación de dirección en el proceso actual. Normalmente es un `NULL`, entonces el sistema puede seleccionar la dirección donde aparece la memoria compartida.
- Tercer parámetro: es el conjunto de banderas de bits, su valor es `SHM_RND` y `shm_addr` se usan en combinación para controlar la dirección continua del área de memoria compartida. `SHM_RDONLY` va a hacer que la memoria continua sea solo lectura.
- Valor retorno: si se tiene éxito, devuelve un puntero al primer byte de la memoria compartida.

- Función `shmdt`: desconecta pero no elimina la memoria compartida.

*int shmdt(void *addr);*

donde:

- `addr`: es el valor retorno de la llamada anterior a la función `shmat`, cuando se tiene éxito, `shmdt` disminuirá el valor del contador `shm_nattch` en la estructura `shmid_ds` relevante en 1.
- Valor retorno: va a devolver 0 en caso de tener éxito.

- Función `shmctl`: elimina el objeto del kernel.

*int shmctl(int shm_id, int command, struct shmid_ds *buf);*

Estructura de `shmid_ds`:

```
struct shmid_ds{
    uid_t shm_perm.uid;
    uid_t shm_perm.gid;
    mode_t shm_perm.mode;
}
```

donde:

- Primer parámetro: identificador de memoria compartida devuelto por `shmget`.
- Segundo parámetro: acción a realizar tomando en cuenta lo siguiente:
 - `IPC_STAT`: establece datos en la estructura `shmid_ds` en el valor asociado actual de la memoria compartida.
 - `IPC_SET`: si el proceso tiene permisos suficientes, establece el valor asociado actual de la memoria compartida al valor de la estructura `shmid_ds`.
 - `IPC_RMID`: borra el segmento de memoria compartida.
- Tercer parámetro: `buf` es un puntero que va a apuntar a la estructura que contiene el modo de memoria compartida y los permisos de acceso.
- Valor retorno: va a devolver 0 en caso de tener éxito.

EJEMPLOS

Ejemplo 1: Código obtenido desde [2]

```
//
// Programa para demostración de memoria compartida.
// Javier Abellán, 10 Mayo 2002.
//
#include <sys/shm.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
void main()
{
    key_t Clave;
    int Id_Memoria;
    int *Memoria = NULL;
    int i,j;

    //
    // Conseguimos una clave para la memoria compartida. Todos los
    // procesos que quieran compartir la memoria, deben obtener la misma
    // clave. Esta se puede conseguir por medio de la función ftok.
    // A esta función se le pasa un fichero cualquiera que exista y esté
    // accesible (todos los procesos deben pasar el mismo fichero) y un
    // entero cualquiera (todos los procesos el mismo entero).
    //
    Clave = ftok ("/bin/ls", 33);
    if (Clave == -1)
    {
        printf("No consigo clave para memoria compartida");
        exit(0);
    }

    //
    // Creamos la memoria con la clave recién conseguida. Para ello
    // llamamos
    // a la función shmget pasándole la clave, el tamaño de memoria que
    // queremos reservar (100 enteros en nuestro caso) y unos flags.
    // Los flags son los permisos de lectura/escritura/ejecucion
    // para propietario, grupo y otros (es el 777 en octal) y el
    // flag IPC_CREAT para indicar que cree la memoria.
    // La función nos devuelve un identificador para la memoria recién
    // creada.
```

```

//
Id_Memoria = shmget (Clave, sizeof(int)*100, 0777 | IPC_CREAT);
if (Id_Memoria == -1)
{
    printf("No consigo Id para memoria compartida");
    exit (0);
}

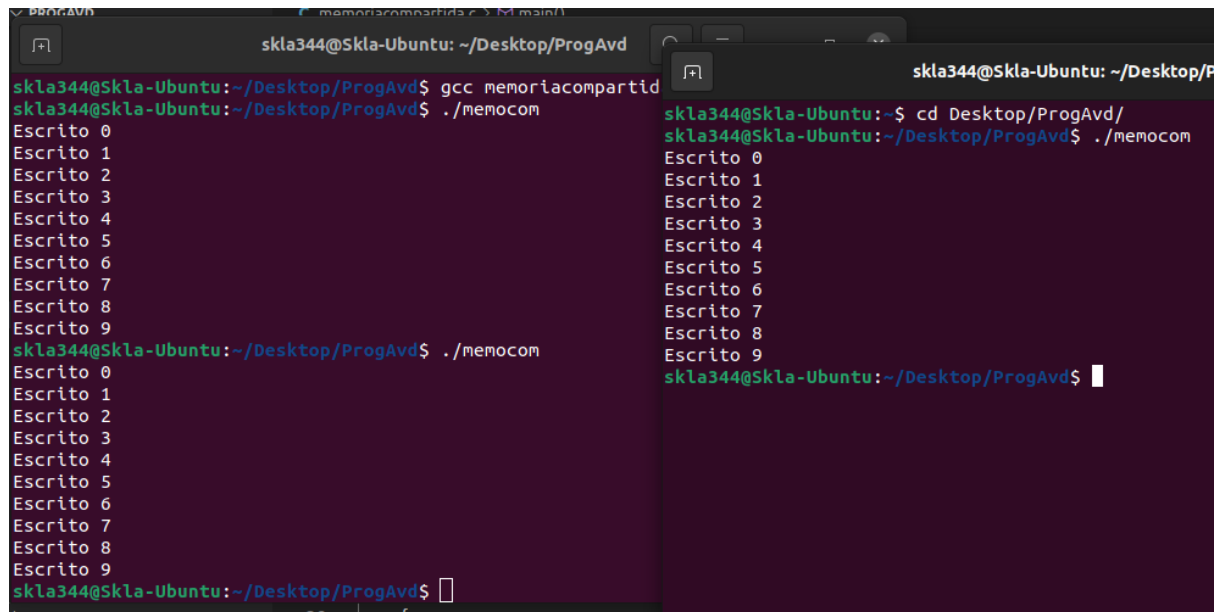
//
// Una vez creada la memoria, hacemos que uno de nuestros punteros
// apunte a la zona de memoria recién creada. Para ello llamamos a
// shmat, pasándole el identificador obtenido anteriormente y un
// par de parámetros extraños, que con ceros vale.
//
Memoria = (int *)shmat (Id_Memoria, (char *)0, 0);
if (Memoria == NULL)
{
    printf("No consigo memoria compartida");
    exit (0);
}

//
// Ya podemos utilizar la memoria.
// Escribimos cosas en la memoria. Los números de 1 a 10 esperando
// un segundo entre ellos. Estos datos serán los que lea el otro
// proceso.
//
for (i=0; i<10; i++)
{
    for (j=0; j<100; j++)
    {
        Memoria[j] = i;
    }
    printf("Escrito %i\n",i);

    sleep (1);
}

//
// Terminada de usar la memoria compartida, la liberamos.
//
shmdt ((char *)Memoria);
shmctl (Id_Memoria, IPC_RMID, (struct shmid_ds *)NULL);
}

```



```
skla344@Skla-Ubuntu: ~/Desktop/ProgAvd
skla344@Skla-Ubuntu:~/Desktop/ProgAvd$ gcc memoria compartida.c -o memocom
skla344@Skla-Ubuntu:~/Desktop/ProgAvd$ ./memocom
Escrito 0
Escrito 1
Escrito 2
Escrito 3
Escrito 4
Escrito 5
Escrito 6
Escrito 7
Escrito 8
Escrito 9
skla344@Skla-Ubuntu:~/Desktop/ProgAvd$ ./memocom
Escrito 0
Escrito 1
Escrito 2
Escrito 3
Escrito 4
Escrito 5
Escrito 6
Escrito 7
Escrito 8
Escrito 9
skla344@Skla-Ubuntu:~/Desktop/ProgAvd$

skla344@Skla-Ubuntu: ~$ cd Desktop/ProgAvd/
skla344@Skla-Ubuntu:~/Desktop/ProgAvd$ ./memocom
Escrito 0
Escrito 1
Escrito 2
Escrito 3
Escrito 4
Escrito 5
Escrito 6
Escrito 7
Escrito 8
Escrito 9
skla344@Skla-Ubuntu:~/Desktop/ProgAvd$
```

Ejemplo 2: Código obtenido desde [1]

```
void sem_get()
{
    // Obtener el conjunto de semáforos
    semid = semget((key_t)1234,1,0666);
    if (semid == -1)
    {
        // Completar la creación de un conjunto de semáforos
        semid = semget((key_t)1234, 1, 0666 | IPC_CREAT);
        assert(semid != -1);
        if (semid == -1)
        {
            printf("error\n");
            exit(0);
        }
        // Inicialización completa
        union sembuff data;
        data.val = 1;
        semctl(semid,0,SETVAL,data);
    }
}
```

//ARCHIVO shma.c

//Se compila con el comando gcc -o shma.c sem.c


```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <assert.h>
#include <sys/shm.h>
#include "sem.h"

void main()
{
    int shmid = shmget((key_t)1234,128,IPC_CREAT | 0666);
    assert(shmid != -1);
    char *ptr = shmat(shmid,NULL,0);
    assert(ptr != NULL);
    sem_get();
    while(1)
    {
        sem_p();
        printf("please input: ");
        fflush(stdout);
        fgets(ptr,128,stdin);
        if (strncmp(ptr,"end",3) == 0)
        {
            break;
        }
        ptr[strlen(ptr)-1] = 0;
        sem_v();
    }
}

```

//ARCHIVO shmb.c
//Se compila con el comando gcc -o shmb.c sem.c

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <assert.h>
#include <sys/shm.h>
#include "sem.h"

```

```

void main()
{

```

```

int shmid = shmget((key_t)1234,128,IPC_CREAT | 0666);
assert(shmid != -1);
char *ptr = shmat(shmid,NULL,0);
assert(ptr != NULL);
sem_get();
while(1)
{
    sem_p();
    if (strncmp(ptr,"end",3) == 0)
    {
        break;
    }
    sleep(5);
    printf("size : %d\n",strlen(ptr));
    sem_v();
}
}

```

Al correr el archivo shma.c, al ser el primer proceso, este crea el espacio de memoria compartido y escribe los datos que le serán compartidos al segundo proceso (una cadena de caracteres)

Al correr el archivo shmb.c, el segundo proceso lee el espacio de memoria e imprime el tamaño de las cadenas de caracteres que envió el primer proceso

Nota: Al parecer los códigos del ejemplo 2 tienen problemas para ser ejecutados en nuestras respectivas máquinas, por lo que no tenemos capturas de pantalla de la corrida

CONCLUSIONES

Al momento de comparar las señales y la memoria compartida observamos que ambas cuentan con ciertas ventajas o facilidades y ciertas desventajas. Por el lado de las señales, estas se permiten siempre que los procesos tengan relación de parentesco, se envían en un instante determinado pero el proceso no sabe cuándo le llega, ni quien se la envía; y cuenta con el inconveniente de que, al comunicar señales con procesos, no se puede enviar mucha información, solo un número.

Por otro lado, la memoria compartida se refiere a que dos procesos van a compartir la misma área de memoria y pueden acceder al contenido del área, similar a trabajar con un puntero; y, con cada cambio, el área de memoria compartida correspondiente se va a comunicar con los otros procesos. Pero, a pesar de ser un método de comunicación muy rápido, cuenta con varias desventajas como: que la cantidad de información es grande, por lo que su administración se vuelve complicada y los procesos deben de estar en la misma máquina físicamente para poder comunicarse; su seguridad es baja, por lo que se puede transmitir un virus entre los procesos; y, al ser un recurso crítico, se debe realizar el control de sincronización entre procesos, y el control de sincronización debe lograrse mejor usando un semáforo.

Independientemente de cual método se desee usar, es importante tomar en consideración lo que se quiere lograr, con tal de implementar el método que permita un mejor desempeño y un mejor beneficio.

REFERENCIAS

- [1] Memoria compartida para comunicación entre procesos. Recuperado el 23 de noviembre de 2021 desde <https://programmerclick.com/article/16011051102/>
- [2] Recena, M. *Memoria compartida en C para Linux*. Recuperado 23 de noviembre de 2021, de http://www.chuidiang.org/clinix/ipcs/mem_comp.php
- [3] Sosa, V. (2006). *Inter-process Communication(IPC): Repaso*. Cinvestav, Tamaulipas. PDF. Recuperado 23 de noviembre de 2021, de https://www.tamps.cinvestav.mx/~vjsosa/clases/sd/IPC_Repaso_p1.pdf