



Instituto Politécnico Nacional  
Escuela Superior de Cómputo  
Sistemas Operativos

Práctica 3:  
Comunicación entre procesos/hilos  
Integrantes:

No. Lista	Nombre
23	<i>Angelo Mihaelle Ojeda Gómez</i>
18	<i>Hernández Hernndez Julio Cesar</i>
20	<i>Martínez Robledo Luis Antonio</i>
11	<i>Estrada Corona Alexis</i>
9	<i>Espinoza Garcia Hilario Sebastian</i>
15	<i>García Silva Jonathan</i>

Grupo: 4CV3

<b>Objetivo</b>	<b>2</b>
<b>Descripción</b>	<b>3</b>
Comunicación entre procesos	3
Memoria compartida	3
<b>Creación de un objeto de memoria compartido POSIX</b>	<b>4</b>
Código fuente:	4
Prueba de pantalla	4
<b>Escritura en un objeto de memoria compartido</b>	<b>5</b>
Código fuente	5
Prueba de pantalla	5
<b>Lectura de un objeto de memoria compartido</b>	<b>6</b>
Código fuente	6
Prueba de pantalla	6
<b>Conclusiones</b>	<b>6</b>
<b>Referencias</b>	<b>8</b>

# Objetivo

Utilizar el mecanismo de memoria compartida para comunicar dos o más procesos en un sistema operativo LINUX/UNIX.

## Descripción

### Comunicación entre procesos

La comunicación entre procesos es una función básica de los sistemas operativos. Los procesos pueden comunicarse entre sí a través de compartir espacios de memoria, ya sean variables compartidas o buffers, o a través de las herramientas provistas por las rutinas de IPC.

Los procesos en los que se descompone una aplicación pueden ejecutarse en un mismo ordenador o en máquinas diferentes. En este último caso, la comunicación entre procesos involucra el uso de redes de ordenadores.

Existen 4 tipos de comunicaciones entre procesos:

- Sincrónica: El emisor se bloquea a la espera de respuesta del receptor
- Asincrónica: El emisor continúa ejecutándose mientras se comunica con el receptor
- Persistente: El receptor no tiene que estar operativo durante la comunicación, los mensajes se almacenan hasta que se reciban
- Momentánea: Los mensajes se descarta si el receptor no está operativo

### Memoria compartida

En informática, la memoria compartida es aquel tipo de memoria que puede ser accedida por múltiples programas, ya sea para comunicarse entre ellos o para evitar copias redundantes. La memoria compartida es un modo eficaz de pasar datos entre aplicaciones.

Al ser el objetivo de este tipo de comunicación la transferencia de datos entre varios procesos, los programas que utilizan memoria compartida deben normalmente establecer algún tipo de protocolo para el bloqueo. Este protocolo puede ser la utilización de semáforos, que es a su vez otro tipo de comunicación (sincronización) entre procesos.

La memoria que maneja un proceso, y también la compartida, va a ser virtual, por lo que su

La dirección física puede variar con el tiempo. Esto no va a plantear ningún problema, ya que los procesos no generan direcciones físicas, sino virtuales, y es el núcleo (con su gestor de memoria) el encargado de traducir unas a otras.

Para la práctica se recurrió a las normas del POSIX, en específica a POSIX. 1b extensiones para tiempo real; memoria compartida (Shared Memory). Y los comando utilizados fueron:

- `shm_open(3)`: Crea y abre un nuevo objeto, o abre uno ya existente
- `mmap(2)`: Asigna el objeto de memoria compartida al espacio de direcciones virtuales del proceso de llamada.
- `close(2)`: Cierre el descriptor de archivo asignado por `shm_open(3)` cuando ya no sea necesario.
- `fstat(2)`: Obtenga un `stat` structure que describa el objeto de memoria compartida. Entre la información devuelta por esta llamada se encuentran el tamaño del objeto (`st_size`), los permisos (`st_mode`), el propietario (`st_uid`) y el grupo (`st_gid`).

Entre otras funciones incluidas en las librerías `<stdio.h>` y `<stdlib.h>`

## Creación de un objeto de memoria compartido POSIX

### Código fuente:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/mman.h>
4 #include <sys/stat.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7 #include <sys/types.h>
8
9 #define SIZEOF_SMOBJ 300
10 #define SMOBJ_NAME "/ObjetoMemoriaC"
11
12 int main(void)
13 {
14     int fd;
15     fd = shm_open (SMOBJ_NAME, O_CREAT | O_RDWR , 00500);
16     if(fd == -1)
17     {
18         printf("Error file descriptor \n");
19         exit(1);
20     }
21     if(-1 == ftruncate(fd, SIZEOF_SMOBJ))
22     {
23         printf("Error, la memoria no puede ser redimesnionada \n");
24         exit(1);
25     }
26     printf("Memoria creada. Ok\n");
27     close(fd);
28
29     return 0;
30 }
```

Para crear un objeto de memoria recurrimos a la función `shm`, esta cuenta funciones de `open`, `write`, entre otros.

Para crear el objeto ocupamos la función `shm_open`, ya que sirve para crear y leer este tipo de elementos, se le asigna un nombre en el primer argumento, el segundo argumento son las instrucciones para crearlo, y el tercer argumento son los permisos que va a tener este objeto (lectura, escritura y ejecución)

## Prueba de pantalla

```
sebas@sebas-VirtualBox: /dev/shm x sebas@sebas-VirtualBox: ~/Escritorio x
sebas@sebas-VirtualBox:/dev/shm$ ls -l
total 0
-rwx----- 1 sebas sebas 200 sep 19 17:00 mObjetoMemoria
-rw----- 1 sebas sebas 300 sep 19 17:07 ObjetoMemoriaC
-r-x----- 1 sebas sebas 300 sep 19 17:08 ObjetoMemoriaC02
sebas@sebas-VirtualBox:/dev/shm$
```

Podemos ver que el objeto de memoria compartida se creó dentro de la dirección /dev/shm (esto en linux), y tiene los permisos que le pusimos, también el nombre y el tamaño

## Escritura en un objeto de memoria compartido

### Código fuente

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/mman.h>
4 #include <sys/stat.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7 #include <sys/types.h>
8 #include <errno.h>
9 #include <string.h>
10
11 #define SMOBJ_NAME "/ObjetoMemoriaC"
12
13 int main(void)
14 {
15     int fd;
16     char bufer[] = "Hola mundo, esta es una prueba de MC\n";
17     char *ptr;
18
19     fd = shm_open(SMOBJ_NAME, O_RDWR, 00200);
20     if(fd == -1)
21     {
22         printf("Error file descriptor %s\n", strerror(errno));
23         exit(1);
24     }
25
26     ptr = mmap(NULL, sizeof(bufer), PROT_WRITE, MAP_SHARED, fd, 0);
27     if(ptr == MAP_FAILED)
28     {
29         printf("El mapeado fallo: %s\n", strerror(errno));
30         exit(1);
31     }
32
33     memcpy(ptr, bufer, sizeof(bufer));
34     printf("Tamaño del bufer <%d> \n", (int)sizeof(bufer));
35     close(fd);
36
37     return 0;
38 }
```

El código sirve para que se pueda escribir en el objeto previamente creado.

La función mmap nos ayudará a mapear el espacio de memoria para poder trabajar en él. Los parámetros que recibe son una dirección de memoria, el espacio que ocupará, las protecciones o permisos, la dirección de la memoria compartida y desde donde se quiere mapear.

## Prueba de pantalla

```
sebas@sebas-VirtualBox: /dev/shm x sebas@sebas-VirtualBox: ~/Escritorio x
sebas@sebas-VirtualBox:/dev/shm$ ls -l
total 4
-rwx----- 1 sebas sebas 200 sep 19 17:00 mObjetoMemoria
-rw----- 1 sebas sebas 300 sep 19 18:03 ObjetoMemoriaC
-r-x----- 1 sebas sebas 300 sep 19 17:08 ObjetoMemoriaC02
sebas@sebas-VirtualBox:/dev/shm$ cat ObjetoMemoriaC
Hola mundo, esta es una prueba de MC
sebas@sebas-VirtualBox:/dev/shm$
```

Podemos ver que se pudo escribir en ese objeto de memoria por medio de otro proceso, eso nos indica que en efecto ambos procesos comparten la misma memoria.

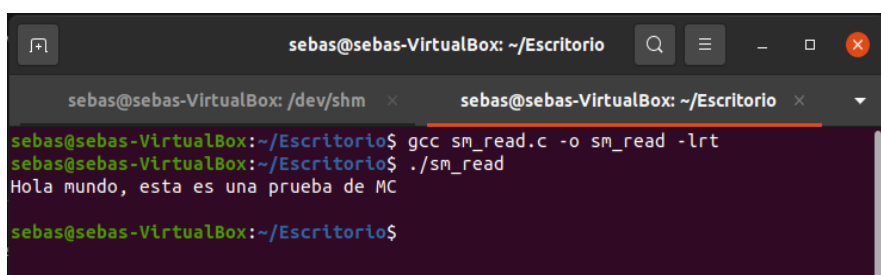
# Lectura de un objeto de memoria compartido

## Código fuente

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/mman.h>
4 #include <sys/stat.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7 #include <sys/types.h>
8 #include <errno.h>
9 #include <string.h>
10
11 #define SMOBJ_NAME "/ObjetoMemoriaC"
12
13 int main(void)
14 {
15     int fd;
16     char *ptr;
17     struct stat shmobj_st;
18
19     fd = shm_open(SMOBJ_NAME, O_RDONLY, 00400);
20     if(fd == -1)
21     {
22         printf("Error file descriptor %s\n", strerror(errno));
23         exit(1);
24     }
25
26     if(fstat(fd, &shmobj_st) == -1)
27     {
28         printf(" error fstat \n");
29         exit(1);
30     }
31     ptr = mmap(NULL, shmobj_st.st_size, PROT_READ, MAP_SHARED, fd, 0);
32     if(ptr == MAP_FAILED)
33     {
34         printf("El mapeado fallo al leer: %s\n", strerror(errno));
35         exit(1);
36     }
37
38     printf("%s \n", ptr);
39     close(fd);
40
41     return 0;
42 }
```

El código nos muestra como se puede leer la memoria de un objeto previamente escrito a través de otro proceso

## Prueba de pantalla



```
sebas@sebas-VirtualBox: ~/Escritorio
sebas@sebas-VirtualBox: /dev/shm x sebas@sebas-VirtualBox: ~/Escritorio x
sebas@sebas-VirtualBox:~/Escritorio$ gcc sm_read.c -o sm_read -lrt
sebas@sebas-VirtualBox:~/Escritorio$ ./sm_read
Hola mundo, esta es una prueba de MC
sebas@sebas-VirtualBox:~/Escritorio$
```

Podemos observar que al ejecutar el programa en efecto puede leer lo que se escribió en otro proceso distinto

# Conclusiones

Se puede concluir que la comunicación entre procesos con el uso de memoria compartida es de vit

Al realizar dicha práctica, se nota la eficiencia en cuanto a la utilización de este método, ya que permitió realizar un programa o código que permitiera a dos procesos no relacionados acceder a la misma memoria lógica. Es de gran ayuda cuando los procesos pueden conectar la misma pieza de memoria física a su propio espacio de direcciones y así poder lograr que todos ellos logren acceder a dicha memoria compartida; esto porque si un proceso escribe datos en la memoria compartida, los cambios afectarán de manera inmediata a cualquier otro proceso que pueda acceder a la misma memoria.

Gracias a lo analizado anteriormente, se determina que el uso de la memoria compartida es la manera más factible para la comunicación y desarrollo de buenos sistemas de trabajo.

# Referencias

- Silberschatz, Abraham; Galvin, Peter Baer (1999). *Sistemas operativos* (5ª edición). Addison Wesley Longman
- Luis R Castellanos. (2015). Comunicacion entre procesos. 18-09-21, de Sitio web: <https://lcsistemasoperativos.wordpress.com/2015/02/03/03-02/>
- Stallings, William (2005). *Sistemas operativos: aspectos internos y principios de diseño* (5ª edición). Pearson Prentice Hall
- Tanenbaum, Andrew (2009). «Sistemas Operativos Modernos». 3 edición.
- Daniel P. Bovet, Marco Cesati (2005), "Understanding The Linux Kernel". 3rd Edition. O'Reilly Media, Inc., ISBN: 0-596-00565-2