

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA

ESCUELA DE COMPUTACIÓN



“Guía N° 10 – MEMORIA COMPARTIDA”

Presentado por:

Escrích Cornejo, Mónica Gabriela

EC130669

Valencia Nájera, Juan Carlos

VN121171

Catedrático: Walter Ovidio Sánchez Campos

Materia: Sistemas Operativos

Grupo: 03L

Miércoles 21 de abril de 2016

El Salvador, Centro América

Desarrollo de Práctica

Ejemplo 1: a continuación el programa productor creará números aleatorios en memoria compartida y el consumidor los leerá.

```
usuario@usuario-virtualbox:~$ gcc -o productor productor.c
usuario@usuario-virtualbox:~$ gcc -o consumidor consumidor.c
usuario@usuario-virtualbox:~$ ./productor
39515142715435147397896932830996839389654107062929519885316554568687396022
538671133417936568134795969317361264423260572980
usuario@usuario-virtualbox:~$ ./consumidor
39515142715435147397896932830996839389654107062929519885316554568687396022
538671133417936568134795969317361264423260572980
```

Incluimos tanto en el proceso productor como en el proceso consumidor la librería `<sys/shm.h>`, esta librería es la que describe las estructuras que son utilizadas por las subrutinas que realizan operaciones de memoria compartida.

También definimos tamaño y clave, la clave es de tipo `key_t` y será la llave asociada al identificador de la memoria compartida que creemos en ambos procesos.

Para crear la memoria compartida nos auxiliamos del comando `shmget` este comando obtiene un segmento de memoria compartida y devuelve en un identificador de memoria compartida que está asociada a la llave. Este valor es asignado a la variable `shmid`.

`Shmat` asocia el segmento de memoria compartida especificado por `shmid` al segmento de datos del proceso invocador. Enviamos como segundo parámetro `0` ya que el segmento de memoria compartida aún no ha sido asociado al proceso invocador entonces el segmento se asocia a una posición en memoria seleccionado por el sistema operativo.

Utilizamos `srand` para generar números aleatorios, esta función recibe como parámetro un valor semilla que en este caso es el id del proceso. Cada valor aleatorio generado es asignado en una posición de memoria compartida.

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

#define TAMANIO 64
#define CLAVE (key_t) 1000

main(){
    int shmid;
    int *adr;
    int i;

    if((shmid=shmget(CLAVE, TAMANIO*sizeof(int), IPC_CREAT|0666))==-1){
        perror("shmget");
        exit(2);
    }

    if((adr=shmat(shmid,0,0))==(int *) -1) {
        perror("shmat");
        exit(2);
    }

    srand(getpid());
    for(i=0; i<TAMANIO; i++)
        printf("%d", adr[i]=rand()%100);
    putchar('\n');
}
```

En el proceso consumidor hacemos uso de *shmget* para obtener la asignación de memoria compartida, el valor de *clave* es el mismo utilizado en el proceso productor.c, posteriormente obtenemos con *shmat* la posición de memoria compartida que asignó el sistema operativo.

A diferencia del proceso productor.c en el proceso consumidor.c no se generan números aleatorios, en este proceso solo se lee de memoria compartida lo que había escrito anteriormente el proceso productor.c

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

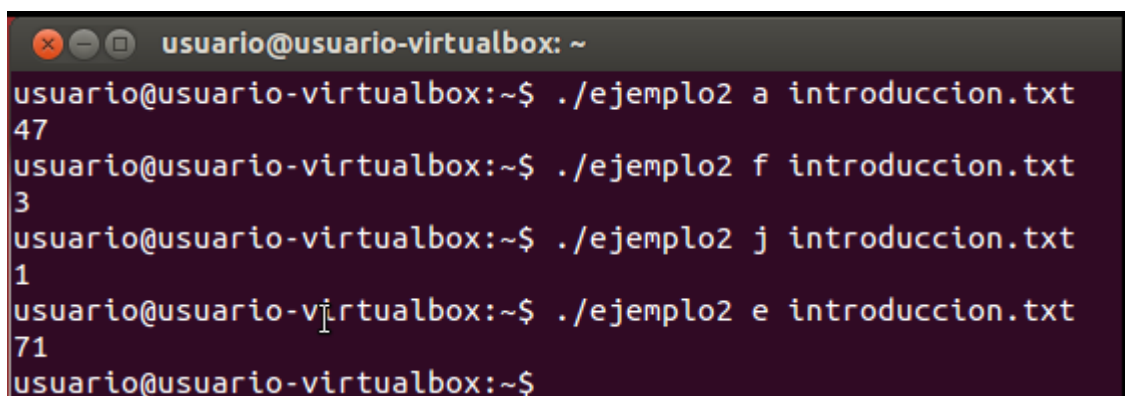
#define TAMANIO 64
#define CLAVE (key_t) 1000

main(){
    int shmid;
    int *adr;
    int i;

    if((shmid=shmget(CLAVE, TAMANIO*sizeof(int),IPC_CREAT|0666))==-1){
        perror("shmget");
        exit(2);
    }
    if((adr=shmat(shmid,0,0))==(int *)-1){
        perror("shmat");
        exit(2);
    }

    srand(getpid());
    for(i=0;i<TAMANIO;i++){
        printf("%d",adr[i]);
        putchar('\n');
    }
}
```

Ejemplo 2: El siguiente programa hace uso de proyección de archivos, al programa se le pasa el carácter a buscar y el archivo donde se realizará la búsqueda.



```
usuario@usuario-virtualbox: ~
usuario@usuario-virtualbox:~$ ./ejemplo2 a introduccion.txt
47
usuario@usuario-virtualbox:~$ ./ejemplo2 f introduccion.txt
3
usuario@usuario-virtualbox:~$ ./ejemplo2 j introduccion.txt
1
usuario@usuario-virtualbox:~$ ./ejemplo2 e introduccion.txt
71
usuario@usuario-virtualbox:~$
```

Incluimos la librería *mman.h* utilizada para la gestión de memoria. Este programa recibirá 2 parámetros, el primero será el carácter a buscar y el tercero el nombre del archivo en el que se desea buscar.

Empezamos declarando las variables que utilizaremos a lo largo del programa, posteriormente verificamos que se hayan recibido 3 argumentos (nombre del programa, carácter a buscar, archivo en el que se buscara el carácter).

Asignamos el segundo parámetro a la variable *carácter*.

Abrimos el archivo que pasamos como 3er parámetros, lo abrimos con permisos de escritura únicamente y asignamos el valor devuelto a la variable *fd* que es nuestro descriptor de archivo.

Verificamos que el archivo se encuentre en buen estado con *fstat*.

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>

int main(int argc, char **argv) {
    int i, fd, contador = 0;
    char caracter;
    char *org, *p;
    struct stat bstat;

    if (argc != 3) {
        fprintf(stderr, "Uso: %s caracter archivo\n", argv[0]);
        return (1);
    }

    caracter=argv[1][0];
    if ((fd=open(argv[2], O_RDONLY)) < 0) {
        perror("No puede abrirse el archivo");
        return(1);
    }

    if (fstat(fd, &bstat) < 0) {
        perror("Error en fstat del archivo");
        close(fd);
        return(1);
    }
```

Hacemos la proyección del archivo con *mmap*. El primer parámetro que recibe esta función es la dirección de inicio de la nueva proyección, en este caso, pasamos *caddr_t* que es un byte alineado de direcciones de memoria virtual, como segundo parámetro pasamos el tamaño de la proyección, el tercer parámetro es la protección que queremos darle al archivo proyectado, Le damos permiso de lectura.

```
if ((org = mmap((caddr_t)0, bstat.st_size, PROT_READ, MAP_SHARED, fd, 0)) == MAP_FAILED) {
    perror("Error en la proyeccion del archivo");
    close(fd);
    return(1);
}

close(fd);
```

El cuarto parámetro que recibe *mmap* es una bandera que determina si las actualizaciones archivo proyectado son visibles por otros procesos, *MAP_SHARED* permite que otros procesos puedan ver las actualizaciones de este archivo.

El quinto parámetro es el descriptor de archivo a proyectar, y el último es un valor de desplazamientos.

Con la ayuda de *for* se recorre todo el archivo en búsqueda del carácter pasado en el segundo parámetro de la llamada al programa y se almacena el número de veces que se repite en un *contador*.

Finalmente utilizamos *munmap* para borrar la proyección del archivo e imprimimos la variable *contador*.

```
p = org;
for (i = 0; i < bstat.st_size; i++)
    if (*p++ == caracter) contador++;

munmap(org, bstat.st_size);
printf("%d\n", contador);
return (0);
}
```