**Sistemas Operativos: Práctica 3**

**Curso 2020/2021**

Grupo 2292\_02

Javier López de la Nieta

Daniel Varela Sánchez

20/04/2021

**Ejercicio 1: Creación de Memoria Compartida.**

1. *Explicar en qué consiste este código, y qué sentido tiene utilizarlo para abrir un objeto de memoria compartida.*

Este código ejecuta el comando shm\_open para abrir en formato lectura y escritura un segmento de memoria compartida con el nombre dado SHM\_NAME. Al hacerlo con los parámetros O\_CREAT y O\_EXCL, el programa intentará crear un nuevo segmento con dicho nombre, y en caso de que exista, reportará un error, y por último, están los parámetros S\_IRUSR y S\_IWUSR los cuales definirán los permisos del usuario en esta zona de memoria compartida. Si al intentar abrirlo hemos obtenido un error del tipo EEXIST, abriremos el segmento sin el parámetro O\_CREAT, para intentar abrir un segmento existente con el nombre dado.

El sentido de este programa es evitar la duplicación de segmentos ya que si ya existe un segmento con el nombre SHM\_NAME, no se creará otro sinó que se abrirá el ya existente y en caso de que no exista se creará uno con ese nombre y se abrirá.

1. *En un momento dado se desearía forzar (en la próxima ejecución del programa) la inicialización del objeto de memoria compartida SHM\_NAME. Explicar posibles soluciones (en código C o fuera de él) para forzar dicha inicialización.*

Para forzar la inicialización del objeto de memoria compartida usaremos previamente a shm\_open la función ftruncate con el parámetro O\_TRUNC lo cual nos reducirá el tamaño del segmento a 0 en caso de que este ya exista forzando así la inicialización.

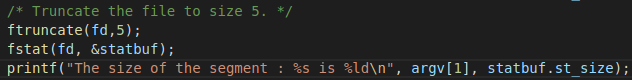
**Ejercicio 2: Tamaño de Ficheros.**

1. *Completar el código anterior para obtener el tamaño del fichero abierto*





1. *Completar el código anterior para truncar el tamaño del fichero a 5 B. ¿Qué contiene el fichero resultante?*



Podemos ver en el archivo que se genera que en vez de guardar en el segmento “Test message”, se guarda solo “Test “ al ser ahora la size de 5, no de 12.

**Ejercicio 3: Mapeado de Ficheros.**

1. *¿Qué sucede cuando se ejecuta varias veces el programa anterior? ¿Por qué?*

Al ejecutar el programa, cada vez aumenta en uno el valor del counter que se muestra por pantalla, esto es debido a que al ser el primer parámetro de mmap NULL y en cada ejecución enlazar una zona de memoria compartida nueva, el sistema operativo asigna una dirección de memoria nueva a partir de las que no ha usado aún, así que irá incrementando el número en 1 tras cada ejecución.

1. *¿Se puede leer el contenido del fichero “test\_file.dat” con un editor de texto? ¿Por qué?*

No ya que es un fichero de texto binario .dat el cual no es soportado por los editores de texto tradicionales.

**Ejercicio 4: Memoria Compartida.**

1. *¿Tendría sentido incluir shm\_unlink en el lector? ¿Por qué?*

No, ya que es una función que solo se tiene que llamar una vez y ya está haciéndolo el escritor, por lo tanto no tendría sentido volver a llamarla.

1. *¿Tendría sentido incluir ftruncate en el lector? ¿Por qué?*

Al igual que la función shm\_unlink, esta solo se tiene que ejecutar una vez y no es necesario volver a llamarla. Ya que el escritor está realizando la llamada a ftruncate, no tendría ningún sentido volver a hacer la llamada.

1. *¿Cuál es la diferencia entre shm\_open y mmap? ¿Qué sentido tiene que existan dos funciones diferentes?*

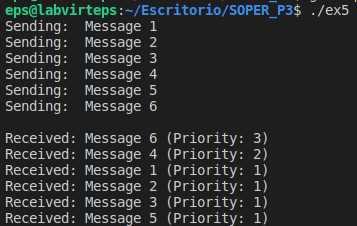
La diferencia es que shm\_open crea o abre un segmento de memoria el cual se puede usar como un fichero normal ya que nos devuelve un descriptor, en cambio mmap nos permite usar un fichero de segmento de memoria como si fuera una parte de la memoria del proceso, eso nos facilita localizar las variables de memoria en la memoria compartida. Por lo tanto al tener funcionalidades distintas ambas son necesarias.

1. *¿Se podría haber usado la memoria compartida sin enlazarla con mmap? Si es así, explicar cómo.*

Si, se podría ya que shm\_open nos devuelve un descriptor de fichero el cual podríamos editar del mismo modo en el que usamos un fichero normal con las instrucciones read o write.

**Ejercicio 5: Envío y Recepción de Mensajes en Colas.**

1. *¿En qué orden se envían los mensajes y en qué orden se reciben? ¿Por qué?*

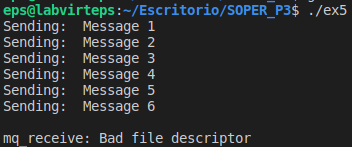


Al ejecutarlo podemos ver que se envían del 1 al 6 en orden creciente y a la hora de recibirlos, el programa los recibe por orden de preferencia de mayor a menor y en caso que la preferencia sea igual, en orden de llegada.

1. *¿Qué sucede si se cambia O\_RDWR por O\_RDONLY? ¿Y si se cambia por O\_WRONLY?*



Al cambiar O\_RDWR por O\_RDONLY, la manera de acceder al segmento de memoria pasa de read and write a solo read, asi que no podemos añadir nada al fichero por lo tanto al hacer mq\_send, se llama al error que se muestra por pantalla.



Si se cambiara por O\_WRONLY, el proceso de envío de mensajes funcionaria correctamente pero a la hora de recibirlos daría un error como el que se muestra por pantalla.

**Ejercicio 6: Colas de Mensajes.**

1. *Ejecutar el código del emisor, y después el del receptor. ¿Qué sucede? ¿Por qué?*

Los envíos de los mensajes son por defecto bloqueantes, lo que quiere decir que si un proceso realiza un send a una cola de mensajes inicializado por defecto, este proceso permanecerá en estado bloqueado hasta que otro proceso ejecute un receive y extraiga el mensaje de la cola.

1. *Ejecutar el código del receptor, y después el del emisor. ¿Qué sucede? ¿Por qué?*

Al ser la recepción de mensajes también bloqueante, sucede lo mismo que en el primer apartado haciendo así que el código funcione de manera esperada en ambos.

1. *Repetir las pruebas anteriores creando la cola de mensajes como no bloqueante.*

*¿Qué sucede ahora?*

En el caso en que la cola de mensajes no fuera bloqueante, el primer caso funcionaria bien ya que ya están enviados los mensajes y después se ejecutaría el receptor, pero en el segundo caso el receptor al ser ejecutado vería que no hay mensajes y no recibiría nada.

1. *Si hubiera más de un receptor en el sistema, ¿sería adecuado sincronizar los accesos a la cola usando semáforos? ¿Por qué?*

No es necesario sincronizar los procesos mediante semáforos, ya que es el sistema operativo el que controla el funcionamiento de las colas de mensajes, y este garantiza la atomicidad de las operaciones realizadas sobre las mismas y de gestionar las llamadas bloqueantes.

*Análisis del problema*

El objetivo de este programa es desarrollar un programa el cual cree una memoria compartida que actúa como servidor de mensajes entre dos procesos hijos creados por el proceso principal. De modo que en esa zona de memoria compartida haya un buffer que según un proceso escribe en él, el otro proceso leerá la información, todo ello carácter a carácter y coordinado mediante mensajes y semáforos.

*Algoritmos utilizados*

Para este programa hemos necesitado usar la función fork para lanzar dos procesos hijos diferentes, los cuales cada uno de ellos utilizando la función execl, lanza otros dos programas previamente compilados.

*Módulos*

Para este ejercicio hemos necesitado dividir el programa en tres módulos diferentes: el módulo principal en el cual se inicializa la zona de memoria compartida,los semáforos, y se crean los procesos que lanzarán los otros módulos; otro módulo emisor, el cual se encargará de escribir en la memoria compartida, y el módulo receptor, encargado de leer en la zona de memoria compartida.

*Estructuras de datos*

Para nuestro programa hemos necesitado definir la estructura de la memoria compartida, compuesta por un buffer en el cual escribiremos, y los semáforos que utilizaremos. De este modo podremos acceder desde todos los módulos a todos los elementos de la misma con un mapeado de esta memoria.

*Pruebas realizadas*

El principal problema ha sido conseguir que ambos procesos leyesen o escribieran en el buffer, entonces hemos realizado varias pruebas con el funcionamiento del buffer para ver la mejor opción para trabajar en él. Por ello, trabajamos principalmente con el proceso emisor, y el otro era un simple return. Una vez conseguimos que este primer proceso funcionase, simplemente tuvimos que realizar el mismo procedimiento de forma análoga.