Práctica 3: Memoria compartida y cola de mensajes.

Autores: Marcos Aarón Bernuy, Kevin de la Coba Malam. Pareja 04 del grupo 2922.

**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.*

El código trata de abrir memoria compartida y después hace un control de errores sobre esta llamada. La llamada puede devolver 2 tipos de errores:

1. La función ha dado un error por una razón X.
2. La función ha dado un error porque ya existe la memoria compartida, por lo que volvemos a intentar abrir esta memoria, pero sin los argumentos O\_CREAT y O\_EXCL, ya que estos son los que juntos devuelven un error si ya existe la zona de memoria compartida
3. *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 ´el) para forzar dicha inicialización.*

Añadir un shm\_unlink en caso de que ya exista, haciendo después otro shm\_open con los mismos argumentos.

**Ejercicio 2: Tamaño de Ficheros.**

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

struct stat fileStat;

fstat(fd, &fileStat);

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

ftruncate(fd, 5\*sizeof(char));

Antes de cambiar el tamaño el fichero tiene "Test mesagge". Tras cambiar el tamaño el fichero contiene "Test".

**Ejercicio 3: Mapeado de Ficheros.**

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

Cada vez que se ejecuta el programa el contador aumenta su valor en 1. Esto es debido a que el valor se guarda en el archivo "test\_file.dat".

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

No, es un archivo binario. Porque el valor se guarda en el archivo de forma binaria.

**Ejercicio 4: Memoria Compartida.**

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

No tendría sentido ya que el escritor ya hace shm\_unlink.

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

No tendría sentido ya que en el escritor se define el tamaño de la estructura a usar, si modificamos el tamaño en el lector y tratamos de acceder a la memoria, puede que tengamos un error ya que, o bien hemos aumentado la memoria y accedemos a un lugar que no nos corresponde, o bien hemos disminuido la memoria y a lo que antes accedíamos ahora ya no podemos.

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

shm\_open devuelve un descriptor de fichero a la memoria compartida, mientras que mmap mapea la memoria compartida en una variable.

shm\_open carga en RAM la memoria, pero mmap la añade al espacio de direcciones del proceso, por lo que si en algún caso no queremos tener memoria compartida en el espacio de direcciones del proceso pues solo haríamos shm\_open.

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

Accediendo a /dev/shm, manipulando los ficheros.

**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?*

Los mensajes se envían en orden ascendente, 1, 2, 3, 4, 5, 6. Se reciben en este orden, 6, 4, 1, 2, 3, 4.

Los mensajes se envían en ese orden ya que es el orden en el que lo hacemos en el código.

Los mensajes se reciben en ese orden ya que dependen de la prioridad, a mayor prioridad, antes se reciben. Al ser una cola, si tenemos varios mensajes con la misma prioridad, se reciben en el orden de llegada (FIFO).

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

Si cambiamos el argumento por O\_RDONLY ocurre un error, ya que el descriptor recibido solo puede usarse para leer, por lo tanto, no se puede enviar ningún mensaje.

Si lo cambiamos por O\_WRONLY, podemos enviar mensajes, pero a la hora de leerlos ocurre un error.

**Ejercicio 6: Colas de Mensajes.**

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

El receptor recibe el mensaje sin problemas.

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

Sucede lo mismo pero el receptor se bloquea hasta que recibe el mensaje. Una vez lo recibe continua su ejecución.

1. Repetir las pruebas anteriores creando la cola de mensajes como no bloqueante. ¿Qué sucede ahora?

Si ejecutamos primero el receptor, la ejecución sería errónea ya que al llamar a receive, está función devuelve un error y se cierra el programa.

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

La cola de mensajes es manejada por el sistema operativo, por lo que no sería necesario sincronizar procesos, a no ser que se quiera un orden especifico de ejecución, por ejemplo, si tenemos 3 procesos receptores, y queremos que primero lea el P1, luego P3 y por último P2, si queremos ese orden siempre, sí podríamos usar un sistema se sincronización.

**Ejercicio 7: Streaming (Codificación).**

1. *Memoria compartida.*
2. *Semáforos.*
3. *Colas de instrucciones.*