**PRÁCTICA 3**

**Ejercicio 1**

1. No tiene sentido incluirlo ya que shm\_writer esperará hasta que le introduzcamos un carácter por teclado para cerrar el segmento de memoria compartida y finalizar el proceso. Hasta que finalicemos el proceso la memoria compartida continuará abierta para poder ejecutar shm\_reader y leer la memoria compartida.
2. shm\_open: Crea o abre un segmento de memoria compartida.

shm\_map: mapea la región de un objeto, descrito por su offset, y lo mete en la región de memoria compartida.

Una sirve para crear o abrir un segmento de memoria y la otra sirve para unir los diferentes segmentos que tu decidas.

1. Se podría utilizar shmget para crear el segmento de memoria compartida, shmat para unir los datos que tenemos al segmento de memoria compartida.

Tras esto se debería llamar a shmdt, que desasocia los datos de la memoria compartida y por último, llamar a shmctl, que hace que se destruya el segmento de memoria compartida.

**Ejercicio 2**

1. El código intenta crear un segmento de memoria compartida si no está ya creado (incluyendo los flags O\_CREAT | O\_EXCL). Si se recibe el error de que la memoria ya existe, simplemente abre la memoria compartida. Con este código no sería necesario saber el orden en el que se ejecutan los procesos que utilicen esta memoria compartida ya que se crearía el segmento en caso de no existir, y se abriría en el caso de existir.
2. Añadiremos la instrucción *ftruncate(fd\_shm, 1000)*, la cual añadiremos en el siguiente lugar:

}else {

ftruncate(fd\_shm, 1000)

printf (" Shared memory segment created \n") ;

}

Para que en futuras ejecuciones no se vuelva a inicializar o destruir lo ya inicializado, no haría falta hacer nada, ya que O\_EXCL, junto con O\_CREAT hace que reporte un error si el elemento ya existía.

1. Se puede ejecutar la función shm\_unlink hacia el segmento “/shared” antes de ejecutar shm\_open. Fuera del código C, se puede acceder a las secciones de memoria compartida en la carpeta /dev/shm y se puede eliminar la carpeta */shared* antes de ejecutar el programa.

**Ejercicio 3**

1. Archivo adjunto. (shm\_concurrence.c)
2. El planteamiento falla en que se crean condiciones de carrera al no proteger las zonas de lectura escritura debidamente con semáforos. Adicionalmente, en el caso de que escriban y lean correctamente, no se imprimirán todos los resultados de los hijos.
3. Archivo adjunto. (shm\_concurrence\_solved.c)

**Ejercicio 4**

1. Archivos adjuntos. (shm\_producer.c, shm\_consumer.c & queue.h)

**Ejercicio 5**

1. El emisor manda el mensaje correctamente y espera a la entrada de una tecla, mientras espera ejecutamos el receptor, el cual recibe correctamente el mensaje y espera a la entrada de una tecla. Se sigue el procedimiento normal y solo se bloquean los procesos cuando esperan la entrada de una tecla.
2. El receptor se bloquea al ejecutarlo dado que la llamada a *mq\_recieve* es bloqueante. Ejecutamos ahora el emisor el cual manda el mensaje correctamente y desbloquea el receptor, el cual recibe el mensaje correctamente. Ambos espera la entrada de una tecla.
3. Añadimos la flag O\_NONBLOCK cuando creamos la cola. Si abrimos primero el emisor ocurre lo mismo que antes. De la otra manera el receptor imprime un error al recibir el mensaje y finaliza. El emisor lo ejecutamos después y no cambia su comportamiento.

**Ejercicio 6**

Archivos adjuntos. (mq\_injector.c & mq\_workers\_pool.c)