Práctica 5 (3 ptos)

Memoria Compartida y Semáforos R/W

1.− Objetivo

En esta práctica se pretende crear una librería que dé la funcionalidad de un semáforo para resolver problemas con múltiples lectores y escritores (read/write lock) y probarla con unos pequeños programas que simulen de forma simplificada el funcionamiento de una base de datos en memoria compartida.

2.− Semáforos para lectores y escritores

Este tipo de semáforo permite resolver de forma simple situaciones con múltiples lectores y escritores, es decir, entornos en los que se puede permitir que varios procesos accedan a la sección crítica si su objetivo es leer información pero que solo pueda haber a la vez un proceso en la sección crítica que pretenda escribir, no pudiendo coincidir ni con lectores ni con otros escritores.

3.− Librería `semafororw.a`

El fichero header `semafororw.h` para la librería es el siguiente:

```
/* Comienzo de semafororw.h */
typedef struct semafororw semafororw;
semafororw *semafororw_open(unsigned char clave);
int semafororw_readlock(semafororw *s);
int semafororw_writelock(semafororw *s);
int semafororw_readunlock(semafororw *s);
int semafororw_writeunlock(semafororw *s);
int semafororw_close(semafororw *s);
/* Final de semafororw.h */
```

La página del manual que podría servir para describir estas funciones es la siguiente:

```
SEMAFORORW(3)                                    SEMAFORORW(3)
```

NAME

semafororw_open, semafororw_readlock, semafororw_writelock, semafororw_readunlock, semafororw_writeunlock, semafororw_close − operaciones sobre semáforos de lectura/escritura

SYNOPSIS

```
#include "semafororw.h"

semafororw *semafororw_open(unsigned char clave);
int semafororw_readlock(semafororw *s);
int semafororw_writelock(semafororw *s);
int semafororw_readunlock(semafororw *s);
int semafororw_writeunlock(semafororw *s);
int semafororw_close(semafororw *s);
```

DESCRIPTION

`semafororw_open` crea y devuelve un nuevo semáforo r/w empleando el byte que se le pasa para generar la información necesaria para distinguir a este semáforo de otros. Si `clave` es `0`, el semáforo que se crea sólo puede ser accedido por otros procesos mediante herencia, es decir, procesos derivados mediante la llamada al sistema `fork()` (es un semáforo privado). Dos procesos independientes entre sí pueden acceder al mismo semáforo indicando la misma clave.

`semafororw_readlock` permite que el proceso reserve el semáforo para lectura. Si al ejecutarse la llamada existe algún proceso que tiene reservado el semáforo para escritura el futuro lector se suspende hasta que el escritor libere el semáforo.

`semafororw_writelock` permite que el proceso reserve el semáforo para escritura. Si al ejecutarse la llamada existe algún proceso que tiene reservado el semáforo para lectura o para escritura el futuro escritor se suspende hasta que el semaforo sea liberado por todos los lectores y escritores.

Puede haber varios procesos que tengan el mismo semáforo reservado para lectura, en tal caso no podrá ser reservado para escritura hasta que sea liberado por todos los procesos lectores.

`semafororw_readunlock libera` el semáforo tras haber sido reservado para lectura.

`semafororw_writeunlock` libera el semáforo tras haber sido reservado para escritura.

`semafororw_close` destruye un semáforo en el ámbito del proceso que ejecuta la función.

RETURN VALUES

`semafororw_open` devuelve un `semafororw*` correspondiente al nuevo semáforo, valiendo `NULL` en caso de error. El resto de funciones devuelven 0 si se ejecutan con éxito y −1 en caso contrario.

4.− Programas de prueba

Para probar el funcionamiento de los semáforos de lectura/escritura se van a crear tres programas para gestionar una base de datos en memoria compartida. El primero de ellos, `gestor`, se limitará a crear la zona de memoria donde se encontrará la base de datos y el semáforo de lectura/escritura que controlará su acceso. Otro programa, `ver`, servirá para ver el contenido de un registro cualquiera de la base de datos. El tercer programa, `modif`, permitirá modificar el contenido de un registro especificado de la base de datos.

La base de datos es un array de `NUM_RECS` registros, donde cada uno es un `registro_bd`.

```
  /* Header para los registro de la base de datos
registro_bd.h */
  #define NUM_RECS  10 /* Numero de registros en la base de
datos */

  typedef struct registro_bd
  {
     int  campo1, campo2;
  } registro_bd;
  /* Final de registro_bd.h */
```

```
  GESTOR(1)                                          GESTOR(1)

  NAME
      gestor − crea una nueva base de datos

  SYNOPSIS
     gestor id_bd

  DESCRIPTION
      Crea una base de datos de NUM_RECS registros de tipo registro_bd (ver
registro_bd.h) con el identificador id_bd  (byte), así como un semáforo de
lectura/escritura para controlar el acceso a esta base de datos partiendo del mismo
identificador. Al recibir la señal SIGINT destruye la base de datos y el semáforo.
```

```
  MODIF(1)                                            MODIF(1)

  NAME
      modif − modifica un registro de una base de datos

  SYNOPSIS
     modif id_bd rec

  DESCRIPTION
      Permite modificar el registro de tipo registro_bd de número rec, un entero
en [0, NUM_RECS), de la base de datos identificada con id_bd (byte), mediante un
semáforo r/w con el mismo identificador. Pide por teclado los nuevos campos para ese
```

registro y los introduce en la base de datos.

```
VER(1)                                                    VER(1)

NAME
    ver − muestra el contenido de un registro de una base de datos

SYNOPSIS
    ver id_bd rec

DESCRIPTION
    Muestra por pantalla el contenido del registro de tipo registro_bd de
número rec, un entero en [0, NUM_RECS), de la base de datos identificada con id_bd
(byte), mediante un semáforo r/w con el mismo identificador.
```

5.− Ejecuciones

Se recomienda comprobar el correcto funcionamiento de la base de datos para las diferentes situaciones de intento de acceso simultáneo a ella. Para ello se pueden crear los programas `ver` y `modif` de forma que mantengan el bloqueo del semáforo durante cierto tiempo para poder controlar los accesos simultáneos.

Ha de tenerse en cuenta que las IPC de System V (semáforos, memoria compartida y colas de mensajes) siguen existiendo, si no se destruyen, aunque el programa que las creó termine. Por ello comprueben al terminar su sesión de prácticas, mediante la utilidad `ipcs`, que no han dejado ninguna en el sistema y de ser así destrúyanla con la utilidad `ipcrm` o completando sus programas.

6.− Ficheros

En el directorio `$(HOME)/solucion/prac5` debe encontrarse un `Makefile` así como todos los ficheros `.c` y `.h` necesarios para crear los módulos y programas mencionados en esta práctica. La acción por defecto del `Makefile` (la cual debe funcionar con solo hacer `make` en ese directorio) debe ser crear `semafororw.a`, `gestor`, `modif` y `ver`, igualmente debe responder correctamente a `make semafororw.a`, `make gestor`, `make modif` y `make ver`.

En el directorio `$(HOME)/../ejemplos/prac5` se pueden encontrar los ficheros `semafororw.h` y `registro_bd.h` así como unos ejemplos de `semafororw.a`, `gestor`, `modif` y `ver` para comprobar cuál es el funcionamiento deseado. Igualmente se dejan disponibles `gestor_main.o`, `modif_main.o` y `ver_main.o` que son los compilados de las funciones principales de los programas mencionados para poder ser linkados con otra librería `semafororw.a`.

Para la corrección de la práctica se borrarán todos los ejecutables, se hará un touch a todos los ficheros fuente y se recompilará mediante el `Makefile`.

7.− Aplicaciones, funciones y llamadas al sistema útiles

`ipcs(8), ipcrm(8), semget(2), semop(2), semctl(2), shmget(2), shmat(2), shmctl(2).`