Sesión 4: Memoria compartida

Objetivo: Crear/destruir/usar regiones de memoria compartida. Usar mecanismos de sincronización: mutex y semáforos.

Ejercicio 1:

Vamos a partir del ejercicio 3 de la sesión de threads. Escribe un programa, llamado *ej1.c*, en el que declares un vector global de 10 posiciones donde cada posición es un long. Crea un thread nuevo. Tanto el thread principal como el nuevo thread, tienen que recorrer todas las posiciones del vector incrementando en 1 el valor de cada posición. Este proceso lo tienen que realizar, cada uno, 10000 veces. Para acabar, el thread principal, una vez haya acabado el nuevo thread, tiene que mostrar el contenido de cada posición del vector. Usa los mecanismos de exclusión mutua (*pthread_mutex_t*) para garantizar que cada posición del vector se ha incrementado 20000 veces.

Ejercicio 2: La cena de los filósofos

Implementa un programa *ej2.c* que simule la cena de los filósofos. Hay una mesa con 5 tenedores y 5 platos para alimentar a 5 filósofos. Un filósofo necesita 2 tenedores para poder comer. Puedes usar un thread para cada filósofo y un vector de posiciones para simular los tenedores. Usa primero solo las llamadas *pthread_mutex_lock/_unlock*. ¿Pueden cenar todos los filósofos? ¿Y si cambias el *_lock* por un *_trylock*? ¿Cuántos intentos de coger el tenedor no llegan a satisfacerse por cada filósofo? NOTA: El objetivo aquí es probar la interficie *lock/unlock/trylock.*

Ejercicio 3:

Crea un programa *ej3.c* que cree 10 threads cada uno con un identificador del 1 al 10. Cada thread debe mostrar este identificador cuando se ejecute, pero en un orden específico. Primero se tienen que mostrar los identificadores pares en orden creciente, y cuando se imprima el último, se mostrarán los impares en orden decreciente. Usa la interfaz de semáforos para conseguir esta sincronización.

Ejercicio 4:

Crea 2 programas *ej4-a.c* y *ej4-b.c*. Queremos comunicar estos dos programas mediante una región de memoria compartida. En concreto el programa *ej4-a* quiere enviar números al programa *ej4-b* y para ello definimos el siguiente protocolo, además del número a enviar, la región contendrá 2 variables más: *ready* y *valid*. Cuando *ej4-a* quiera escribir un número, debe esperar a que ready esté a 1 que significa que el otro programa ha leído el valor anterior y está preparado para leer un valor nuevo. En ese momento, escribe el valor y escribe un valor 1 en la variable *valid*, que indica que ya se puede leer el valor. El programa *ej4-b*, por su parte, espera que la variable *valid* esté a 1, momento en que escribe un 0 en *ready*, lee el valor y escribe un 1 en *ready*.
¿Hay algún caso en que este programa pueda fallar?

[Ejercicio 5: Happy ending for La cena de los filósofos]

Implementa una solución al problema. Es necesario explicar al tutor el por qué de vuestra solución.