ELVI MIHAI SABAU SABAU - 51254875L - 27/11/2022

PRACTICA 5

Practicas de Programación Concurrente

Lectores / Escritores con prioridad de lectura

- Probar el problema de los lectores/escritores con prioridad en la lectura usando hilos y cerrojos (semáforos binarios) POSIX.
- Crea 10 hilos lectores (que lean el recurso 10 veces cada uno) y 5 hilos escritores (que modifiquen el recurso 5 veces cada uno). El recurso puede ser, por ejemplo, una variable entera con un valor inicial -1 y al que cada escritor le asigna un valor igual a su identificador (entre 0 y 4).
- Puedes añadir a cada hilo un pequeño retardo aleatorio para observar mejor la ejecución del programa. El código del programa buffer-circular-hilos.c visto en el tema 4 de teoría te puede servir de apoyo (observa que para este ejercicio no son necesarios los semáforos POSIX).

1. Características del problema:
   - Lectores : desean leer un recurso, dos o más pueden acceder simultáneamente al recurso
   - Escritores : actualizan la información del recurso, sólo uno puede acceder al recurso: acceso exclusivo al recurso
2. Variables y semáforos utilizados:
   - mx : controla el acceso en exclusión mutua a la variable compartida readers y sirve de barrera para los escritores.
   - writer : funciona como un semáforo de exclusión mutua para los escritores, también lo utiliza el primer/último lector para entrar/salir de la sección crítica, pero no será utilizada mientras haya otros lectores o escritores en la sección crítica
   - readers : número de lectores en la sección crítica

```c
// P5 - Practicas de Programación Concurrente
// Lectores/Escritores con prioridad de lectura
// Elvi Mihai Sabau Sabau - 51254875L
// Compilación : gcc -o p5 p5.c -lpthread

#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <errno.h>
#include <pthread.h>
```

```c
#include <stdio.h>

sem_t mx, writer;
int  LECTORES = 10, ESCRITORES = 5, readers = 0, recurso = -1;

void reader_lock() {
    sem_wait(&mx);
    readers++;

    if (readers == 1) sem_wait(&writer);
    sem_post(&mx);
}

void reader_unlock() {
    sem_wait(&mx);
    readers--;

    if (readers == 0) sem_post(&writer);
    sem_post(&mx);
}


void *lector(void *id) {

    for(int i = 0; i < LECTORES; i++) {
        reader_lock();
        printf("El lector %d ha leido un valor de %d\n", *(int*) id, recurso);
        reader_unlock();
        // Añadimos un retardo para simular lecturas de distinto tiempo.
        usleep(rand()%5);
    }

    pthread_exit(id);
}

void *escritor(void *id) {

    for(int i = 0; i < ESCRITORES; i++) {
        sem_wait(&writer);
        printf("El escritor %d ha actualizado el recurso\n", *(int*) id);
        recurso = *(int*) id;
        sem_post(&writer);
        // Añadimos un retardo para simular escrituras de distinto tiempo.
        usleep(rand()%5);
    }
    pthread_exit(id);
```

```
}

void crear_hilos(pthread_t hilos[], int id[], int num, void *(*func) (void*)) {

    for (int i = 0; i < num; i++) {
        id[i] = i;
        int error = pthread_create(&hilos[i], NULL, func, &id[i]);
        if (error) {
            fprintf(stderr, "Error: %d: %s\n", error, strerror(error));
            exit(-1);
        }
    }
}

void terminar_hilos(pthread_t hilos[], int num) {

    int *salida;

    for (int i = 0; i < num; i++) {
        int error = pthread_join(hilos[i], (void **) &salida);
        if (error) fprintf(stderr, "Error: %d: %s\n", error, strerror(error));
    }
}

int main() {
    pthread_t tidlector[LECTORES], tidescritor[ESCRITORES];
    int id_lec[LECTORES], id_esc[ESCRITORES];

    sem_init(&mx, 0, 1);
    sem_init(&writer, 0, 1);

    crear_hilos(tidescritor, id_esc, ESCRITORES, escritor);
    crear_hilos(tidlector, id_lec, LECTORES, lector);

    terminar_hilos(tidlector, LECTORES);
    terminar_hilos(tidescritor, ESCRITORES);

    sem_destroy(&mx);
    sem_destroy(&writer);

    return 0;
}
```