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

PRACTICA 6

Practicas de Programación Concurrente

Problema de los Caníbales

- Una tribu cena en comunidad una gran olla que contiene M misioneros cocinados.
- Cuando un miembro de la tribu quiere comer, él mismo se sirve de la olla un misionero, a menos que esté vacía. Los miembros de la tribu se sirven de la olla de uno en uno.
- Si la olla está vacía, el que va a cenar despierta al cocinero y se espera a que esté llena la olla.
- Desarrollar el código de las acciones de los miembros de la tribu y el cocinero usando semáforos.

Usa las siguientes variables:

- `M`: número de misioneros.
- `olla`: Entero que indica el número de misioneros en la olla. Estará inicializada a M.
- `mutex`: Semáforo para proteger la exclusión mutua sobre la variable `olla`. Inicializado a 1.
- `espera`: Semáforo utilizado para hacer que el que va a cenar se detenga hasta que el cocinero llene la `olla` cuando está vacía. Inicializado a 0.
- `coci`: Semáforo inicializado a 0 y usado para que el cocinero no haga nada cuando la `olla` no está vacía.

```c
// P6 - Practicas de Programación Concurrente
// Problema de los Caníbales
// Elvi Mihai Sabau Sabau - 51254875L
// Compilación : gcc -o canibales canibales.c -lpthread

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

// Definimos los canibales, los misioneros y el número de misioneros en la olla.
```

```c
int CANIBALES = 20, M = 5, olla = 5;
// Exclusión mutua sobre la variable olla.
sem_t mutex;
// Espera a que haya comida en la olla.
sem_t espera;
// Espera a que la olla esté vacia.
sem_t coci;

/**
1. La primera función es la que ejecutarán los hilos canibal. El caníbal esperará a que la
2. El mutex se usa para controlar el acceso a la olla. Solo un caníbal puede tomar la comid
3. El caníbal comerá la comida de la olla, y luego la olla quedará vacía. El caníbal le ind
4. La segunda función es la que ejecutará el hilo cocinero. El cocinero se quedará dormido
5. El cocinero llenará la olla y luego le indicará al caníbal que la olla está llena, despu
*/

void *cocinero () {

    printf("El Cocinero se ha despetado.\n");
    // Espera a que un canibla le indique que la olla está vacía.
    while (1) {
        // Cocinero espera.
        sem_wait(&coci);

        // Llena la olla.
        olla = M;
        printf("El Cocinero acaba de llenar la olla.\n");

        // Olla llena, avisa al canibal.
        sem_post(&espera);
    }
}

void *canibal (void *id) {

    sem_wait(&mutex);

    // Si no hay misioneros en la olla, despierta al cocinero, y espera a que la olla esté
    if (olla == 0) {
        sem_post(&coci);
        sem_wait(&espera);
    }

    // Come un misionero.
    olla--;
    printf("Canibal %d comiendo, misioneros en la olla: %d\n", *(int*) id, olla);
```

```
    sem_post(&mutex);
}

int main() {

    pthread_t hilos_canibales[CANIBALES], hilo_cocinero;
    int id_canibales[CANIBALES], error, *salida;


    // Estado inicial de los semaforos.
    sem_init(&mutex, 0, 1); // mutex hay que inicializarlo a 1, porque la primera vez que se
    sem_init(&espera, 0, 0);
    sem_init(&coci,0, 0);

    // Genera los hilos canibales.
    for (int hilo = 0; hilo < CANIBALES; hilo++) {
        id_canibales[hilo] = hilo;
        error = pthread_create(&hilos_canibales[hilo], NULL, canibal, &id_canibales[hilo]);
        if (error) {
            fprintf(stderr, "Error: %d: %s\n", error, strerror(error));
            exit(-1);
        } else printf("El canibal %d se ha despetado.\n", hilo);
    }

    // Genera el hilo cocinero.
    pthread_create(&hilo_cocinero, NULL, cocinero, NULL);

    // Los hilos canibales se unen al hilo principal.
    for (int hilo = 0; hilo < CANIBALES; hilo++) {
        error = pthread_join(hilos_canibales[hilo], (void **)&salida);
        if (error) {
            fprintf(stderr, "Error: %d: %s\n", error, strerror(error));
            exit(-1);
        }
        else printf("El canibal %d se ha dormido.\n", hilo);

    }

    printf("Todos los canibales se han saciado :)\n");

    // Cerramos.
    sem_destroy(&mutex);
    sem_destroy(&espera);
    sem_destroy(&coci);

    return 0;
```

```
}
```