UA

ProgramaciónConcurrente

DanielAsensiRochDNI : **48776120C**

27 de noviembre de 2022

Índice

1. Problema de los Caníbales **2**
1.1. Descripción del problema . . . 2
1.2. Resolución del problema . . . 2
1.2.1. Inicialización de variables y semáforos . . . 2
1.2.2. Bloqueo, desbloqueo de canibales y cocinero . . . 2
1.2.3. Main . . . 3

1. Problema de los Caníbales

1.1. Descripción del problema

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.

Características del problema:

- **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.

Variables y semáforos a utilizar:

- **mutex** : controla el acceso en exclusión mutua a la variable compartida olla y sirve de barrera para los canibales.
- **espera** : funciona como un semáforo de exclusión mutua para los escritores, también lo utiliza el primer/último canibal para entrar/salir de la sección crítica, pero no será utilizada mientras este el cocinero o otros canibales en la sección crítica
- **coci** : Despierta al cocinero para llenar la olla.

1.2. Resolución del problema

1.2.1. Inicialización de variables y semáforos

Lo primero que deberemos realizar será inicializar nuestros semáforos y variables para definir la cantidad de lectores, escritores, así como los semaforos y el recurso

```
// Variables globales para el número de canibales y misioneros
#define M 5
#define C 20
int olla = M;

sem_t mutex;
sem_t espera;
sem_t coci;
```

1.2.2. Bloqueo, desbloqueo de canibales y cocinero

Logica de bloqueo de la sección critica para bloquear la sección critica para escritores

```
void *canibal(void *id)
{
    sem_wait(&mutex);
    // Si no nos quedan misioneros entonces
    if (olla == 0)
    {
        // Despertamos al cocinero
        sem_post(&coci);
```

```
        // Los canibales se esperan
        sem_wait(&espera);
    }
    // Nos comemos un misionero
    olla--;
    printf("Canibal %d comiendo, misioneros restantes: %d\n", *(int *)id, olla);
    sem_post(&mutex);
}

void *cocinero()
{

    while (1)
    {
        // Esperamos a que los canibales nos despierten
        sem_wait(&coci);
        olla = M;
        printf("Cocinero cocinando, misioneros restantes: %d\n", olla);
        // Despertamos a los canibales
        sem_post(&espera);
        printf("Canibales a cenar!!\n");
    }
}
```

1.2.3. Main

Nuestro main se encargará de la creación de los hilos y de terminar los mismos para no delegar en funciones externas, además de destruir los semáforos.

```
// Funcion principal
int main()
{

    pthread_t tidcanibal[C];
    pthread_t thCocinero;
    int id_canibal[C];

    int h;
    int error;
    int *salida;

    // Inicializacion de semaforos
    sem_init(&mutex, 0, 1);
    sem_init(&espera, 0, 0);
    sem_init(&coci, 0, 0);

    // crear canibales
    for (h = 0; h < C; h++)
    {
        id_canibal[h] = h;
        error = pthread_create(&tidcanibal[h], NULL, canibal, &id_canibal[h]);
        if (error)
        {
            fprintf(stderr, "Error: %d: %s\n", error, strerror(error));
            exit(-1);
```

```
        }
    }

    // Cocinero
    pthread_create(&thCocinero, NULL, cocinero, NULL);

    // terminar canibales
    for (h = 0; h < C; h++)
    {
        error = pthread_join(tidcanibal[h], (void **)&salida);
        if (error)
        {
            fprintf(stderr, "Error: %d: %s\n", error, strerror(error));
        }
    }

    printf("Todos los miembros de la tribu han cenado\n");

    // Cerrar todos los semaforos
    sem_destroy(&mutex);
    sem_destroy(&espera);
    sem_destroy(&coci);

    return 0;
}
```