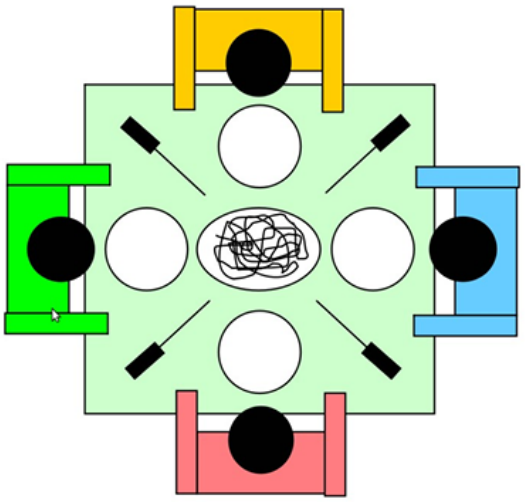
Mariana Osorio Rojas - 464679
Fabio Andrés Guzman Figueroa
Universidad Pontifica Bolivariana - Facultad de Ingeniería
Ingeniería de Sistemas e Informática - Sistemas Operativos

PRÁCTICA N6 Y TALLER N1 - FILÓSOFOS COMENSALES

En sistemas operativos, el problema de los filósofos comensales es un problema clásico utilizado para ilustrar desafíos de sincronización y exclusión mutua en entornos multitarea. Este problema modela una situación donde varios filósofos pasan tiempo pensando y comiendo alrededor de una mesa circular. Cada filósofo necesita dos tenedores para comer, uno a cada lado, pero solo puede tomar un tenedor a la vez. El desafío radica en diseñar un algoritmo que permita a los filósofos compartir los tenedores de manera segura, evitando bloqueos y asegurando que todos puedan comer sin problemas.

|Descripción del Problema|

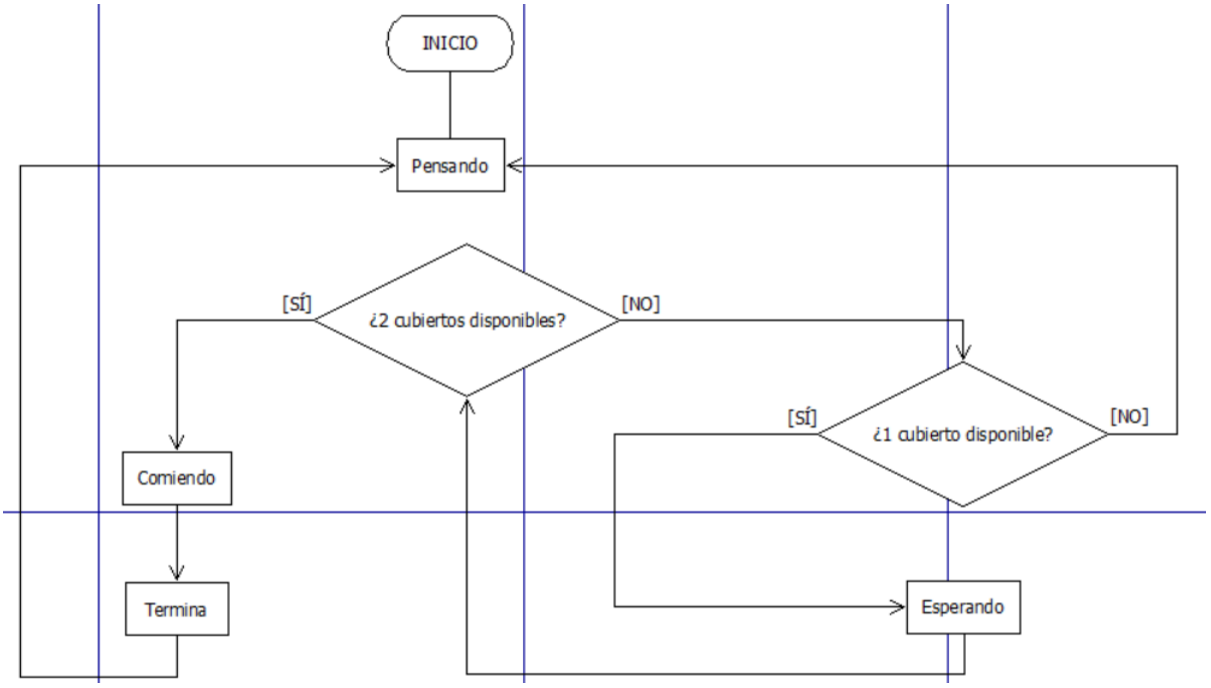
En el problema de los filósofos comensales, se modela una situación donde un número impar de filósofos se sientan alrededor de una mesa circular con un tenedor entre cada par de filósofos. Cada filósofo tiene cuatro acciones posibles: pensar, esperar, comer o terminar. Para comer, un filósofo necesita tomar los dos tenedores adyacentes a él. Sin embargo, dado que los tenedores son compartidos, surge el problema de cómo coordinar el acceso a los tenedores para evitar interbloqueos y condiciones de carrera.



- Número impar de filósofos.
- Número de tenedores igual al de filósofos.
- Todos los filósofos deben comer y pensar.
- Solo se puede comer con 2 tenedores.
- Existen 4 estados posibles:

1. Pensando.
2. Esperando.
3. Comiendo.
4. Terminó.

|Diagrama de Flujo|



|Pseudocódigo|

```
int n = 5; // Número de filósofos y tenedores
semaphore mutex = 1; // Control de acceso a la mesa
semaphore[] forks = [1, 1, 1, 1, 1]; // Un semáforo por cada tenedor

void philosopher(int i) {
    while (true) {
        think();
        wait(mutex);
        wait(forks[i]);
        wait(forks[(i + 1) % n]);
        signal(mutex);
        eat();
        signal(forks[i]);
        signal(forks[(i + 1) % n]);
    }
}
```

|Solución Propuesta|

Definición de Estructuras y Variables

- Se define una enumeración status para los estados de los filósofos: PENSANDO, HAMBRIENTO y COMIENDO.
- Se declaran los arreglos estado[] para almacenar el estado de cada filósofo y s[] para los semáforos de cada filósofo.
- Se utiliza un semáforo mutex para garantizar la exclusión mutua al acceder a las variables compartidas.

Inicialización

- Se inicializan los semáforos y se establece el estado inicial de todos los filósofos como PENSANDO.

Funciones de los filósofos

- La función Filosofo representa el ciclo de vida de un filósofo, que incluye pensar, intentar coger tenedores, comer y soltar tenedores. Este ciclo se repite dos veces por filósofo.
- Durante el pensamiento, el filósofo actualiza su estado y se imprime la escena.
- Al intentar coger tenedores, el filósofo actualiza su estado a HAMBRIENTO, verifica si puede empezar a comer y se bloquea si no puede.
- Al comer, el filósofo actualiza su estado, imprime la escena y espera un tiempo aleatorio.
- Al soltar los tenedores, el filósofo actualiza su estado, imprime la escena y libera los semáforos de los tenedores.

Verificación de disponibilidad de tenedores

- La función Comprobar verifica si un filósofo puede empezar a comer según el estado de sus vecinos.

Impresión de la escena

- La función print_scene genera una representación visual de la escena actual, mostrando el estado de cada filósofo en una línea.

Creación de hilos

- Se crean los hilos correspondientes a cada filósofo, que ejecutan la función Filosofo.

Espera de finalización

- Se espera a que todos los hilos terminen su ejecución antes de finalizar el programa.

|Código Fuente|

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define N 5 //Número de filósofos
#define VIDA 10
#define IZQUIERDA(x)    ((x) !=0 ? (x)-1 : (N-1)) //Calcula el filósofo de la izquierda
#define DERECHA(x)      (((x)+1)%N) //Calcula el filósofo de la derecha

typedef enum{
   PENSANDO, HAMBRIENTO, COMIENDO
} status;

status estado[N]; //Estado en el que está cada filósofo en cada momento
(pensando, hambriento o comiendo)
sem_t mutex, s[N]; //Mutex: Solo un filósofo puede coger o soltar tenedores; s[N]:
Controlamos si he podido coger tenedores o no, si no he podido, me bloqueo
hasta que nos despierte el filósofo de la izquierda o derecha

void pensar(int id);
void comer(int id);
void coger_tenedores(int id);
void Comprobar(int id);
void soltar_tenedores(int id);
```

```
void print_scene();

int main()
{
   extern status estado[N]; //No es necesario
   extern sem_t mutex, s[N]; //No es necesario

   int v[N]/*para decirle a cada hilo quien es*/, value, i;
   sem_init(&mutex, 0, 1);
   pthread_t filosofo[N];
   void *Filosofo(void *);

   for (i = 0; i < N; i++)              //NO SE PUEDEN LANZAR LOS HILOS HASTA
QUE LOS SEMÁFOROS NO ESTÉN INICIALIZADOS
   {
      sem_init(&s[i], 0, 0);
      estado[i] = PENSANDO;

   }

   for (i = 0; i < N; i++)
   {
      v[i] = i;
      if (value = pthread_create(&filosofo[i], NULL, Filosofo, (void *)&v[i]))
         exit(value);

   }

   for (i = 0; i < N; i++)
   {
      pthread_join(filosofo[i], NULL);
   }

   return 0;
}

void *Filosofo(void *i)
{
   int id, j;
   id = *(int *)i;

   for (j = 0; j < 2; j++)
   {
      pensar(id);
      coger_tenedores(id);
      comer(id);
      soltar_tenedores(id);
   }
```

```
    printf("Filosofo[%d] terminó de comer\n", id);
    pthread_exit(NULL);
}

void pensar(int id)
{
    estado[id] = PENSANDO;
    print_scene();
    sleep(random() % 10);
}

void comer(int id)
{
    estado[id] = COMIENDO;
    print_scene();
    sleep(random() % 10);
}

void coger_tenedores(int id)
{
    sem_wait(&mutex); //Si un filosofo está cogiendo tenedores, no podré coger y
me quedaré bloqueado. Si no hay nadie cogiendo tenedores, cogeré yo
    estado[id] = HAMBRIENTO;
    Comprobar(id);
    sem_post(&mutex); //Permito que otro filosofo coja tenedores
    sem_wait(&s[id]); //Si no hace el post de la función Comprobar(), se quedaría
pillado aquí
    printf("Filosofo[%d] cogió los tenedores %d y %d\n", id, id, DERECHA(id));
    print_scene();
}

void Comprobar(int id)
{
    if (estado[id] == HAMBRIENTO && estado[IZQUIERDA(id)] != COMIENDO &&
estado[DERECHA(id)] != COMIENDO)
    {
        estado[id] = COMIENDO;
        sem_post(&s[id]); //Sirve para liberar al filosofo para que no se quede pillado
más adelante, puesto que ha podido coger tenedores
    }
}

void soltar_tenedores(int id)
{
    sem_wait(&mutex); //Si hay alguien cogiendo tenedores, nadie más puede
coger. Si estoy cogiendo tenedores, bloqueo para que nadie más pueda hacerlo
    estado[id] = PENSANDO;
    printf("Filosofo[%d] soltó los tenedores %d y %d\n", id, id, DERECHA(id));
    Comprobar(IZQUIERDA(id));
```

```
    Comprobar(DERECHA(id));
    sem_post(&mutex);
    print_scene();
}

void print_scene() {
    char philosopher_state[N][10]; // Array para almacenar el estado de cada
filósofo
    char thinking[] = "PENSANDO";
    char eating[] = "COMIENDO";
    char waiting[] = "HAMBRIENTO";

    // Llenar el array con los estados
    for (int i = 0; i < N; i++) {
        if (estado[i] == PENSANDO) {
            snprintf(philosopher_state[i], 10, "%s", thinking);
        } else if (estado[i] == COMIENDO) {
            snprintf(philosopher_state[i], 10, "%s", eating);
        } else {
            snprintf(philosopher_state[i], 10, "%s", waiting);
        }
    }

    // Imprimir la escena
    printf("Escena: | %s | %s | %s | %s | %s |\n", philosopher_state[0],
philosopher_state[1], philosopher_state[2], philosopher_state[3],
philosopher_state[4]);
}
```

|Video Explicativo|

https://youtu.be/nXlZyqeScmc

|Captura de Ejecución|