Programación Concurrente, Marzo 2023

Práctica 4

Martínez Buenrostro Jorge Rafael.
Universidad Autónoma Metropolitana
Unidad Iztapalapa, México
molap96@gmail.com

Actividad B

Para poder resolver esta actividad fue necesario tomar como base el ejemplo dado del problema Productor Consumidor. Se le agregan las siguientes modificaciones:

1. Se crean dos variables, una para el número de consumidores y número de productores. Se solicita una al usuario y se asigna el valor a ambas; ya que si el número es igual no se podrá garantizar el funcionamiento del programa.

```
int numConsumidores, numProductores;

printf("Indique el número de consumidores y productores...");
scanf("%d",&numConsumidores);
numProductores = numConsumidores;
```

2. Se crean e inicializan dos arreglos de hilos: productores y consumidores, del tamaño de las variables del paso anterior.

```
pthread_t hiloProds[numProductores], hiloCons[numConsumidores];
/* Se crea el hilo productor */
for(i=0;i<numProductores;i++){
    idProductores[i]=i+1;
    pthread_create( &hiloProds[i], NULL, (void *) &Produce, (void *)(intptr_t)idProductores[i]);
}
/* Se crea el hilo consumidor */
for(i=0;i<numConsumidores;i++){
    idConsumidores[i]=i+1;
    pthread_create(&hiloCons[i], NULL, (void *) &Consume, (void *)(intptr_t)idConsumidores[i]);
}
```

3. Ya que habrá N productores y consumidores, se crean dos semáforos para garantizar la EM con el buffer. Ambos se inicializan 1 ya que están listos para usarse

```
sem_init(&sem_productor,0,1);
sem_init(&sem_consumidor,0,1);
```

Todo lo demás de la base queda prácticamente intacto.

Enlace GDB - https://onlinegdb.com/GoHKEoH_e

Código Fuente

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

#define TAM_BUFFER 5
#define MAX_PROD 10

void Produce( void *ptr );
void Consume( void *ptr );

sem_t sem_consumidor, sem_productor, espacios, productos;
int buffer[TAM_BUFFER];
int a_vaciar=0, a_llenar=0;

int main()
{
   int i, n, m;
   int numConsumidores, numProductores;

   printf("Indique el número de consumidores y productores...");
   scanf("%d",&numConsumidores);
   numProductores = numConsumidores;
   int idConsumidores[numConsumidores];
   int idProductores[numProductores];

   pthread_t hiloProds[numProductores], hiloCons[numConsumidores];

    /* inicializa el la semilla aleatoria */
   srand ( time(NULL) );

   sem_init(&sem_productor,0,1);
   sem_init(&sem_consumidor,0,1);
```

```
    sem_init(&espacios,0,TAM_BUFFER);
    sem_init(&productos,0,0);

    /* Se crea el hilo productor */
    for(i=0;i<numProductores;i++){
        idProductores[i]=i+1;
        pthread_create( &hiloProds[i], NULL, (void *) &Produce, (void
*)(intptr_t)idProductores[i]);
    }

    /* Se crea el hilo consumidor */
    for(i=0;i<numConsumidores;i++){
        idConsumidores[i]=i+1;
        pthread_create(&hiloCons[i], NULL, (void *) &Consume, (void
*)(intptr_t)idConsumidores[i]);
    }


    /* El hilo padre espera a que terminen los hilos hijos */
    for(i=0;i<numConsumidores;i++)
        pthread_join(hiloCons[i], NULL);

    for(i=0;i<numProductores;i++)
        pthread_join(hiloProds[i], NULL);

    /* Se destruyen los semaforos */
    sem_destroy(&sem_productor);
    sem_destroy(&sem_consumidor);
    sem_destroy(&espacios);
    sem_destroy(&productos);

    return 0;

}

void Produce( void *ptr )
{
```

```
   int  i, prod, indice, numHilo = (intptr_t) ptr;

   for (i = 0; i < MAX_PROD ; i++){
       /* Se genera el producto */
       prod = rand() % 5;
       /* Esperamos un espacio para colocarlo en el buffer */
       sem_wait(&espacios);
       /* Pedimos el derecho de acceso exclusivo al buffer */
       sem_wait(&sem_productor);
       /* Modificamos el buffer */
       buffer[a_llenar] = prod;
       indice = a_llenar;
       a_llenar = (a_llenar+1) % TAM_BUFFER;
       /* Se libera el derecho de acceso exclusivo al buffer*/
       sem_post(&sem_productor);
       /* Se avisa que hay un producto disponible */
       sem_post(&productos);
       printf("Productor(%d): produje Buffer[%d]:%d\n",numHilo,indice,prod);
   }
   pthread_exit(0);
}

void Consume( void *ptr ){
   int  i, consumible, indice, numHilo = (intptr_t) ptr;
   for (i = 0; i < MAX_PROD ; i++){
       /* Esperamos un producto para consumirlo */
       sem_wait(&productos);
       /* Pedimos el derecho de acceso exclusivo al buffer */
       sem_wait(&sem_consumidor);
       /* Modificamos el buffer */
       consumible = buffer[a_vaciar];
       indice = a_vaciar;
       a_vaciar = (a_vaciar+1) % TAM_BUFFER;
       /* Se libera el derecho de acceso exclusivo al buffer*/
       sem_post(&sem_consumidor);
       /* Se avisa que hay un espacio disponible  */
       sem_post(&espacios);
```

```
        /* Se realiza el consumo */
        printf("Consumidor (%d): Buffer[%d]:%d\n",numHilo,indice,consumible);
        sleep(consumible);
    }
    pthread_exit(0);
}
```

Actividad C

En esta actividad se puede ver que se tiene un productor, el “servidor” que solicita los viajes y N hilos que son el pool de taxis listos para atender los viajes. Se hicieron las siguientes modificaciones:

1. Se crea una estructura en la que almacenaremos los datos de un viaje, y crearemos un buffer de estas estructuras.

```
struct Viaje{
    int id;
    int distancia;
};
struct Viaje buffer[TAM_BUFFER];
```

2. Dentro de la función “Produce” se agrega un ciclo que, al terminar de las peticiones de viajes se agregan tantos viajes como taxis con el id igual a -1.

```
for(i = 0;i < NUM_TAXIS;i++){
    struct Viaje terminar;
    terminar.id = -1;
    terminar.distancia = -1;
    sem_wait(&espacios);
    buffer[a_llenar] = terminar;
    a_llenar = (a_llenar + 1) % TAM_BUFFER;
    sem_post(&viajes);
}
```

3. Dentro de la función “Consume” se agrega una condición en la que se revisa si el id del viaje es -1; en caso que lo sea se realiza el corte del taxi, lo que significa que se registra en la memoria compartida la cantidad de viajes y ganancias que hizo.

```
    if(viaje.id == -1)
        break;
```

```
registroViajes[numHilo][0] = viajesAtendidos;
registroViajes[numHilo][1] = ganancias;
```

4. Al final se agrega lo necesario en el hilo principal para saber que hilo fue el que más viajes hizo y cuál fue el que más ganancias generó.

```
printf("\nEn total se atendieron %d viajes", MAX_PROD);
for(i = 0;i < NUM_TAXIS;i++){
    if(registroViajes[i][0] > mayorViajes){
        mayorViajes = registroViajes[i][0];
        idMasViajes = i;
    }
    if(registroViajes[i][1] > mayorIngreso){
        mayorIngreso = registroViajes[i][1];
        idMasIngresos = i;
    }
}
```

Enlace GDB - https://onlinegdb.com/npPAOerr5

Código Fuente

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

#define TAM_BUFFER 5
#define NUM_TAXIS 4
#define MAX_PROD 10

void Produce(void * ptr);
void Consume(void * ptr);

struct Viaje{
```

```
    int id;
    int distancia;
};
struct Viaje buffer[TAM_BUFFER];

sem_t sem_taxis, espacios, viajes;
int a_vaciar = 0, a_llenar = 0;
int registroViajes[NUM_TAXIS][2];

int main(){
    int i, idMasViajes, mayorViajes = 0, idMasIngresos, mayorIngreso = 0;
    int idTaxis[NUM_TAXIS];
    pthread_t servidor, hiloTaxis[NUM_TAXIS];
    srand(time(NULL));

    sem_init(&sem_taxis,0,1);
    sem_init(&espacios,0,TAM_BUFFER);
    sem_init(&viajes,0,0);

    pthread_create(&servidor, NULL, (void*)&Produce,NULL);

    for(i = 0;i < NUM_TAXIS;i++){
        idTaxis[i]=i;

pthread_create(&hiloTaxis[i],NULL,(void*)&Consume,(void*)(intptr_t)idTaxis[i]);
    }

    pthread_join(servidor,NULL);
    for(i = 0;i < NUM_TAXIS;i++)
        pthread_join(hiloTaxis[i],NULL);

    printf("\nEn total se atendieron %d viajes", MAX_PROD);
    for(i = 0;i < NUM_TAXIS;i++){
        if(registroViajes[i][0] > mayorViajes){
            mayorViajes = registroViajes[i][0];
            idMasViajes = i;
        }
```

```
        if(registroViajes[i][1] > mayorIngreso){
            mayorIngreso = registroViajes[i][1];
            idMasIngresos = i;
        }
    }

    printf("\nEl taxi que más viajes atendió fue %d con %d viajes",idMasViajes,
mayorViajes);
    printf("\nEl taxi que más ingresos generó fue %d con %d pesos\n",idMasIngresos,
mayorIngreso);

    sem_destroy(&sem_taxis);
    sem_destroy(&espacios);
    sem_destroy(&viajes);

    return 0;
}

void Produce(void * ptr){
    int i;

    for(i = 0;i < MAX_PROD;i++){
        struct Viaje producto;
        producto.id = i + 1;
        producto.distancia = rand() % 100 + 1;
        sem_wait(&espacios);
        buffer[a_llenar] = producto;
        a_llenar = (a_llenar + 1) % TAM_BUFFER;
        sem_post(&viajes);
    }
    for(i = 0;i < NUM_TAXIS;i++){
        struct Viaje terminar;
        terminar.id = -1;
        terminar.distancia = -1;
        sem_wait(&espacios);
        buffer[a_llenar] = terminar;
        a_llenar = (a_llenar + 1) % TAM_BUFFER;
```

```
        sem_post(&viajes);
    }
    pthread_exit(0);
}

void Consume(void * ptr){
    int viajesAtendidos = 0, ganancias = 0, numHilo = (intptr_t) ptr, distancia = 0,
esFinal = 0;
    struct Viaje viaje;

    do{
        sem_wait(&viajes);
        sem_wait(&sem_taxis);
        viaje = buffer[a_vaciar];
        a_vaciar = (a_vaciar + 1) % TAM_BUFFER;
        sem_post(&sem_taxis);
        sem_post(&espacios);

        if(viaje.id == -1)
            break;

        esFinal = viaje.id;
        distancia = viaje.distancia;
        printf("Soy el taxi %d, voy a dormir\n",numHilo);
        sleep(distancia/10);
        ganancias += distancia * 10;
        viajesAtendidos += 1;
    }while(esFinal > 0);

    registroViajes[numHilo][0] = viajesAtendidos;
    registroViajes[numHilo][1] = ganancias;
    pthread_exit(0);
}
```

Actividad D

En esta actividad tendremos tres hilos con arquitectura Pipeline. El primer hilo será el encargado de leer los número del usuario y ponerlos en el primer buffer. El segundo hilo tomará los números del primer buffer y les calculará su secuencia de Collatz; se crea una estructura llamada **Par** en la que se guardará el número y la longitud de su secuencia de Collatz. Este **Par** lo guarda dentro del segundo buffer; el tercer hilo toma los pares y compara la longitud del **Par** para saber si es el mayor que se ha calculado.

Cuando el usuario ya no quiera ingresar números ingresará -1; el primer hilo guardará este número en el primer buffer, el segundo hilo lo leerá y agrega en el segundo buffer el par (-1,-1), el tercer buffer al leer este par imprimirá en pantalla el **Par** con la longitud más grande.

Enlace GDB - https://onlinegdb.com/dZJqZhw7km

Código Fuente

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

#define TAM_BUFFER 5
#define MAX_PROD 10

void ProduceNumeros(void * ptr);
void ConsumeNumeros(void * ptr);
void ConsumePares(void * ptr);
int Collatz(int);

struct ParCollatz{
   int numero;
   int longitud;
};

int bufferNumeros[TAM_BUFFER];
struct ParCollatz bufferPares[TAM_BUFFER];
sem_t espaciosNumeros, numeros,espaciosPares, pares;
int sacarNumero = 0, ponerNumero = 0;
int sacarPar = 0, ponerPar = 0;
```

```c
int main(){

   pthread_t generadorNumeros,analizadorNumeros,analizadorPares;

   sem_init(&espaciosNumeros, 0, TAM_BUFFER);
   sem_init(&espaciosPares, 0 , TAM_BUFFER);
   sem_init(&numeros, 0, 0);
   sem_init(&pares, 0, 0);

   pthread_create(&generadorNumeros,NULL,(void*)&ProduceNumeros,NULL);
   pthread_create(&analizadorNumeros,NULL,(void*)&ConsumeNumeros,NULL);
   pthread_create(&analizadorPares,NULL,(void*)&ConsumePares,NULL);

   pthread_join(generadorNumeros,NULL);
   pthread_join(analizadorNumeros,NULL);
   pthread_join(analizadorPares,NULL);
   return 0;
}

void ProduceNumeros(void * ptr){
   int numero;
   do{
       printf("\nAnote el número a analizar, para terminar ingrese -1...");
       scanf("%d",&numero);
       sem_wait(&espaciosNumeros);
       bufferNumeros[ponerNumero] = numero;
       ponerNumero = (ponerNumero + 1) % TAM_BUFFER;
       sem_post(&numeros);
   }while(numero > 0);
   pthread_exit(0);
}

void ConsumeNumeros(void * ptr){
   int longitud, numero;
   struct ParCollatz nuevoPar;

   do{
```

```
        sem_wait(&numeros);
        numero = bufferNumeros[sacarNumero];
        sacarNumero = (sacarNumero + 1) % TAM_BUFFER;
        sem_post(&espaciosNumeros);

        if(numero == -1){
            nuevoPar.numero = -1;
            nuevoPar.longitud = -1;
        }else{
            longitud = Collatz(numero);
            nuevoPar.numero = numero;
            nuevoPar.longitud = longitud;
        }

        sem_wait(&espaciosPares);
        bufferPares[ponerPar] = nuevoPar;
        ponerPar = (ponerPar + 1) % TAM_BUFFER;
        sem_post(&pares);

    }while(numero >0);
    pthread_exit(0);
}

void ConsumePares(void * ptr){
    struct ParCollatz maximo, terminar, aux;
    maximo.longitud = 0;
    maximo.numero = 0;
    do{
        sem_wait(&pares);
        aux = bufferPares[sacarPar];
        sacarPar = (sacarPar + 1) % TAM_BUFFER;
        sem_post(&espaciosPares);

        terminar.numero = aux.numero;
        terminar.longitud = aux.longitud;

        if(aux.longitud > maximo.longitud){
```

```
            maximo.numero = aux.numero;
            maximo.longitud = aux.longitud;
        }
    }while(terminar.numero == -1 && terminar.longitud == -1);

    printf("\nLa secuencia de Collatz más larga fue %d con el número
%d\n",maximo.longitud,maximo.numero);
    pthread_exit(0);
}

int Collatz(int num){
    int longitud = 1;

    while(num != 1){
        longitud += 1;

        if (num % 2 == 0) {
            num = num / 2;
        } else {
            num = 3 * num + 1;
        }
    }
    return longitud;
}
```