INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE CÓMPUTO

INGENIERÍA EN SISTEMAS COMPUTACIONALES

SISTEMAS OPERATIVOS

David Díaz Araujo

Práctica 4

Problemas de sincronización entre procesos / hilos

- Alumnos:
  1) Arcos Hermida Aldo Alejandro (N° lista 5)
  2) Chávez Becerra Bianca Cristina (N° lista 9)
  3) Islas Osorio Enrique (N° lista 20)
  4) Juárez Cabrera Jessica (N° lista 21)
  5) Palmerin García Diego (N° lista 28)
- Grupo: 4CM1
- Materia: Sistemas operativos
- Profesor: Araujo Díaz David

Objetivo

Utilizar el mecanismo de semáforos para sincronizar dos o más procesos en un sistema operativo LINUX / UNIX

Descripción

Los sistemas operativos tienen la habilidad de interrumpir la ejecución de un proceso e iniciar otro, así logra brindar soporte a múltiples procesos. En casos en los que se requiera la cooperación entre procesos, es posible sincronizarlos y coordinar sus actividades. Esto se puede lograr mediante el mecanismo de semáforos, una herramienta de sincronización que permite que dos o más procesos puedan cooperar por medio de señales.

Los semáforos son un mecanismo de sincronización de procesos inventados por Edsger Dijkstra en 1965. Los semáforos permiten al programador asistir al planificador del sistema operativo en su toma de decisiones de manera que permiten sincronizar la ejecución de dos o más procesos.

Los semáforos son un tipo de datos que están compuestos por dos atributos:

- Un contador, que siempre vale >= 0.
- Una cola de procesos inicialmente vacía.

Y disponen de dos operaciones básicas:

```
down(semáforo s)
{
	si s.contador == 0:
		añade proceso a s.cola_procesos
		proceso a estado bloqueado
	sino:
		s.contador--
}
```

```
up(semáforo s)
{
	si hay procesos en s.cola_procesos:
		retira proceso de s.cola_procesos
		proceso a estado preparado
	sino:
		s.contador++
}
```

Por ejemplo, se tiene la siguiente secuencia de ejecución: A, B, A, B, ... y así sucesivamente. El siguiente código resolvería el problema:

```
semaforo a = semaforo.create(1);
semaforo b = semaforo.create(0);

/* código del hilo A */
while (1) {
        a.down();
        printf("hilo A\n");
        b.up();
}

/* código del hilo B */
while (1) {
        b.down();
        printf("hilo B\n");
        a.up();
}
```

Hay semáforos contadores y semáforos binarios. Los semáforos contadores apoyan a controlar el acceso a un determinado recurso con múltiples instancias. Los semáforos binarios utilizan únicamente valores 0 y 1, se utiliza para implementar la solución de problemas de sección crítica.

Existe una librería en POSIX que permite el uso de semáforos, para utilizarla se requiere:

- #include <semaphore.h>
- El código se compila con -lpthread -lrt

```
// operación de espera sobre un semáforo
int sem_wait(sem_t *sem);

// operación abre / crea sobre un semáforo
int sem_post(sem_t *sem);

// Inicializa un semáforo sin nombre
sem_init(sem_t *sem, int shared, int val);

// Destruye un semáforo sin nombre
int sem_destroy(sem_t *sem);

// Crea un semáforo con nombre
sem_t *sem_open(char *name, int flag, mode_t mode, int val);
```

```
// cierra un semaforo con nombre
int sem_close(sem_t *sem);

// borra un semáforo con nombre
int sem_unlink(char *name);
```

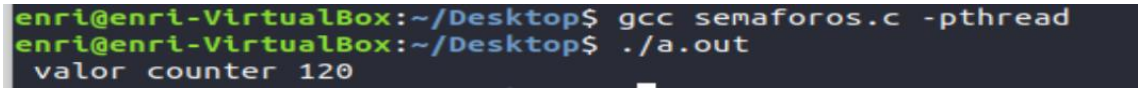
Código fuente

```
Sin semaforos
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
# define NR_LOOP 100000
static void * thread_1_function(void* arg);
static void * thread_2_function(void* arg);
static int counter =0;

int main (void)
{
   pthread_t thread_1,thread_2;
   pthread_create(&thread_1,NULL, *thread_1_function,NULL);
   pthread_create(&thread_2,NULL, *thread_2_function,NULL);
   pthread_join(thread_1,NULL);
   pthread_join(thread_2,NULL);
   printf(" valor counter %d \n",counter);
   return 0;
}

static void * thread_1_function(void* arg)
{
   for(int i=0;i< NR_LOOP;i++)
   {

      counter+=1;
   }

}

static void * thread_2_function(void* arg)
{
   for(int i=0;i< NR_LOOP;i++)
   {
      counter-=1;
   }

}
```

```
Uso de semaforos
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
# define NR_LOOP 100000
static void * thread_1_function(void* arg);
static void * thread_2_function(void* arg);
static int counter =0;
sem_t sem1;

int main (void)
{
   pthread_t thread_1,thread_2;
   sem_init(&sem1,0,1);
   pthread_create(&thread_1,NULL, *thread_1_function,NULL);
   pthread_create(&thread_2,NULL, *thread_2_function,NULL);
   pthread_join(thread_1,NULL);
   pthread_join(thread_2,NULL);
   printf(" valor counter %d \n",counter);
   return 0;
}

static void * thread_1_function(void* arg)
{
   for(int i=0;i< NR_LOOP;i++)
   {
      sem_wait(&sem1);
      counter+=1;
      sem_post(&sem1);
   }

}

static void * thread_2_function(void* arg)
{
   for(int i=0;i< NR_LOOP;i++)
   {
      sem_wait(&sem1);
      counter-=1;
      sem_post(&sem1);
   }

}
```

Prueba de pantalla

Nos debería de dar 0 pero como los hilos están accediendo a la misma variable se producen errores



Para solucionar esto hicimos uso de semáforos

Conclusiones

Los semáforos son un mecanismo de sincronización y nos ayudan a proteger el acceso a la sección critica que es donde se accede a un recurso que se comparte por procesos o hilos, en esta practica se hizo uso de una misma variable en dos hilos, para solucionar el error hicimos uso de las funciones sem_wait() y sem_post().Para el objetivo planteado los semáforos resultaron ser una herramienta consistente y eficiente. Ampliamente recomendable en administración de transacciones de ámbito local. En general se cuentan con diversos problemas para la sincronización de procesos como por ejemplo los filósofos que cenan, el barbero dormilón, lectores y escritores entre otros, en los cuales todos pueden contar con una solución para cumplir sus condiciones y es a partir de un contador o más específicamente en este tema a partir de semáforos.

Referencias

- GeeksforGeeks. (2020, 11 diciembre). *How to use POSIX semaphores in C language*. Recuperado 25 de febrero de 2022, de https://www.geeksforgeeks.org/use-posix-semaphores-c/

- Stallings, W. (2005). *Sistemas operativos: Aspectos internos y principios de diseño*. 5th ed. Madrid: Pearson Prentice Hall.

- Daniel P. Bovet, Marco Cesati, “Understanding The Linux Kernel”. 3rd Edition. O'Reilly Media, Inc., 2005. ISBN: 0-596-00565-2

- Tanenbaum, A. (2009). *Sistemas operativos modernos*. 3rd ed. México: Pearson Educación de México, SA de CV.