Dibujo de una persona

Descripción generada automáticamente con confianza bajaImagen que contiene Icono

Descripción generada automáticamente**INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE CÓMPUTO**

**INGENIERÍA EN SISTEMAS COMPUTACIONALES**

**SISTEMAS OPERATIVOS**

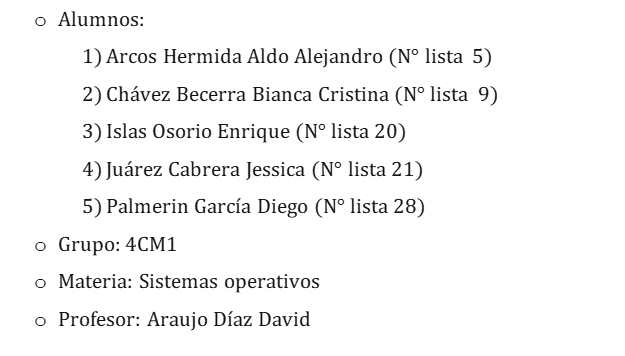
**David Díaz Araujo**

Imagen que contiene Icono

Descripción generada automáticamente

**Práctica 4**

**Problemas de sincronización entre procesos / hilos**



**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**);**

**Prueba de escritorio**

**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;

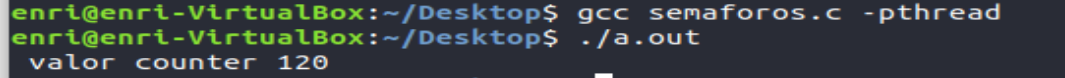
      sem\_post(&sem1);

   }

}

**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

**Texto

Descripción generada automáticamente**

**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()

**Referencias**

* GeeksforGeeks. (2020, 11 diciembre). *How to use POSIX semaphores in C language*. Recuperado 27 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.
* Silberschatz, A., Galvin, P. and Gagne, G. (2006). *Fundamentos de sistemas operativos*. 7th ed. Madrid: McGraw Hill.
* Tanenbaum, A. (2009). *Sistemas operativos modernos*. 3rd ed. México: Pearson Educación de México, SA de CV.