Instituto Politécnico Nacional

Escuela Superior de Cómputo

Sistemas Operativos

Práctica 4:

Problemas de sincronización entre procesos/hilos

Integrantes:

No. Lista	Nombre
23	*Angelo Mihaelle Ojeda Gómez*
18	*Hernández Hernndez Julio Cesar*
20	*Martínez Robledo Luis Antonio*
11	*Estrada Corona Alexis*
9	*Espinoza Garcia Hilario Sebastian*
15	*García Silva Jonathan*

Grupo: 4CV3

Objetivo

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

Descripción

Sincronización entre procesos

En muchos casos, los procesos se reúnen para realizar tareas en conjunto, a este tipo de relación se le llama procesos cooperativos. Para lograr la comunicación, los procesos deben sincronizarse, de no ser así pueden ocurrir problemas no deseados. La sincronización es la transmisión y recepción de señales que tiene por objeto llevar a cabo el trabajo de un grupo de procesos cooperativos.
Es la coordinación y cooperación de un conjunto de procesos para asegurar la comparación de recursos de cómputo. La sincronización entre procesos es necesaria para prevenir y/o corregir errores de sincronización debidos al acceso concurrente a recursos compartidos, tales como estructuras de datos o dispositivos de E/S, de procesos contendientes. La sincronización entre procesos también permite intercambiar señales de tiempo (ARRANQUE/PARADA) entre procesos cooperantes para garantizar las relaciones específicas de procedencia impuestas por el problema que se resuelve.
Para que los procesos puedan sincronizarse es necesario disponer de servicios que permitan bloquear o suspender bajo determinadas circunstancias la ejecución de un proceso. Los principales mecanismos de sincronización que ofrecen los sistemas operativos son:

- Señales
- Tuberías
- Semáforos
- Mutex y variables condicionales
- Paso de mensajes

En lo qué a esta práctica refiere utilizaremos semáforos para la sincronización de procesos, esto resolviendo un problema haciendo uso de estos.

Semáforos

Los semáforos son un mecanismo de sincronización de procesos inventado 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.
Como otros muchos mecanismos, las primitivas sobre semáforos han ido evolucionando en UNIX/Linux a través de las diferentes versiones. A partir de la versión del kernel 2.6, Linux soporta semáforos POSIX como se describe a continuación.

Operaciones con semáforos

- Los semáforos sólo pueden ser manipulados usando las siguientes operaciones (éste es el código con espera activa)
    - ```
      Inicia(Semáforo s, Entero v){
        s = v;
      }
      ```
- En el que se iniciará la variable semáforo s a un valor entero v.
    - ```
      P(Semáforo s){
        if(s>0)
            s = s-1;
        else
            wait();
      }
      ```
- La cual mantendrá en espera activa al regido por el semáforo si este tiene un valor inferior o igual al nulo.
    - ```
      V(Semáforo s){
         if(!procesos_bloqueados)
              s = s+1;
         else
              signal();
      }
      ```
      ```

Problemas de sincronización de procesos

En sistemas de multiprogramación con recursos compartidos es necesario proporcionar mecanismos de exclusión mutua, que garanticen la sincronización de los procesos al asignarles cierto recurso y la coordinación en la resolución de una tarea encomendada. En los sistemas operativos multiprogramados los procesos compiten por el acceso a los recursos compartidos o cooperan dentro de una misma aplicación para comunicar información. Ambas situaciones son tratadas por el sistema operativo mediante mecanismos de sincronización que permiten el acceso exclusivo de forma coordinada a los recursos y a los elementos de comunicación compartidos.Una solución consiste en garantizar que una sección crítica de código solo sea utilizada por un proceso. Existen diferentes algoritmos para resolver el problema de la sección crítica. Clásicamente se han definido diversos problemas de sincronización, que son importantes principalmente como ejemplos de una amplia clase de problemas de control de concurrencia, en nuestro caso, y como ejemplo, utilizaremos estos problemas para comprobar la sincronización entre procesos haciendo el uso de semáforos. En este caso los problemas serán:

- **Problema del productor-consumidor**
    - Es un ejemplo clásico de problema de sincronización de multiprocesos. El programa describe dos procesos, productor y consumidor, ambos comparten un buffer de tamaño finito. La tarea del productor es generar un producto, almacenarlo y comenzar nuevamente; mientras que el consumidor toma (simultáneamente) productos uno a uno. El problema consiste en que el productor no añade más productos que la capacidad del buffer y que el consumidor no intente tomar un producto si el buffer está vacío.

Problema productor-consumidor

Código Fuente

```
#include <stadio.h>
#include <pthread.h>
#include <semaphore.h>

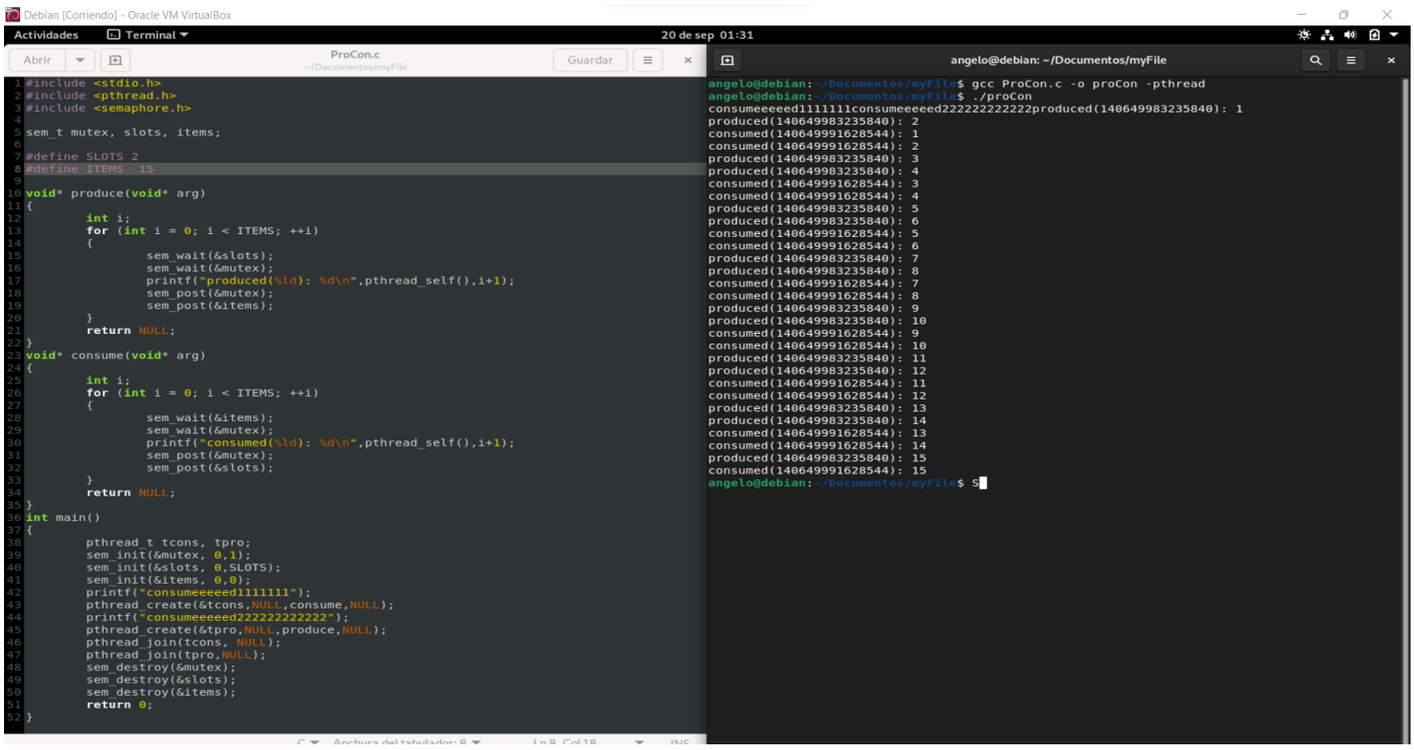
sem_t mutex, slots, items;

#define SLOTS 2
#define ITEMS 10

void* produce(void* arg)
{
	int i;
	for (int i = 0; i < ITEMS; ++i)
	{
		sem_wait(&slots);
		sem_wait(&mutex);
		printf("produced(%ld): %d\n",pthread_self(),i+1);
		sem_post(&mutex);
		sem_post(&items)
	}
	return NULL;
}
void* consume(void* arg)
{
	int i;
	for (int i = 0; i < ITEMS; ++i)
	{
		sem_wait(&items);
		sem_wait(&mutex);
		printf("consumed(%ld): %d\n",pthread_self(),i+1);
		sem_post(&mutex);
		sem_post(&slots)
	}
	return NULL;
}
int main()
{
```

```
        pthread_t tcons, tpro;
        sem_init(&mutex, 0,1);
        sem_init(&slots, 0,SLOTS);
        sem_init(&items, 0,0);
        pthread_create(&tcons,Null,consume,NULL);
        pthread_create(&tpro,NULL,produce,NULL);
        pthread_join(tcons, NULL);
        pthread_join(tpro,NULL);
        sem_destroy(&mutex);
        sem_destroy(&slots);
        sem_destroy(&items);
        return 0;
}
```

Prueba de pantalla



Explicacion del codigo

En esta prueba se puede ver la solución del problema antes mencionado haciendo el uso de semáforos para sincronizar los procesos/hilos a la hora de ejecutar el código, en concreto se muestra para 2 slots y 15 items, osea 15 artículos qué se ofrecen y un buffer con capacidad de 2, aunque ambos de estos valores se pueden cambiar arbitrariamente. los métodos usados de los semáforos fueron los siguientes, pero hay qué aclarar qué existen más:

- Inicializa el semáforo a value
    - int sem_init(sem_t sem, * int pshared, unsigned int value)
- Destruye el semáforo
    - int sem_destroy(sem_t sem)
- Espera a bloquear, (es llamada bloqueante)
    - int sem_wait(sem_t sem); *
- Incrementa el semáforo, desbloquea un hilo en espera
    - int sem_post(sem_t sem);

Son operaciones algo distintas a las descritas previamente, pero esencialmente iguales. Para concluir el uso concreto de los semáforos fue:

- Usar semáforos para conocer qué lugares están disponibles en un buffer compartido
- Usar semáforos para seguirle la pista a ítems en un buffer compartido
- Usar un semáforo/mutex para sincronizar las operaciones sobre un buffer

Conclusiones

Se sabe ahora que la sincronización entre procesos es necesaria para así poder evitar errores de temporización, esto porque se tiene o se necesita el acceso concurrente a recursos compartidos, haciendo referencia a estructuras a datos o dispositivos de E/S.
En un ámbito general, se sabe que la sincronización entre procesos también permite el intercambio de señales de temporización entre procesos cooperativos con el fin de preservar las relaciones especificadas de precedencia impuesta por el problema que se deba resolver.
Podemos concluir que los semáforos son una herramienta de gran poder en cuanto a la sincronización y de la misma manera como una variable protegida, dicha variable se puede modificar por la rutina inicialización. Al paso del tiempo y la realización de dicha práctica, se pudieron notar aspectos importantes. La principal característica que se puede dar a notar es que los programadores ahora cuentan con una herramienta que asegura una exclusión entre los procesos concurrentes a emplear y así, de esta manera, poder acceder a un recurso compartido que facilite más el trabajo.

Referencias

- Abimael hernandez Seguir Working en Celular Milenium. (n.d.). Problemas de sincronizacion de procesos. Retrieved from https://es.slideshare.net/sped1009/problemas-de-sincronizacion-de-procesos
- Durán, J. (2017, December 08). Programacion paralela en C : Semaforos. Retrieved from https://www.somosbinarios.es/programacion-paralela-en-c-semaforos/
- McHoes, A. M., Flynn, I. M., & Velásquez, H. V. (2011). Sistemas operativos. Cengage Learning.
- Problemas clásicos de sincronización. (2020, December 12). Retrieved from https://ricardogeek.com/problemas-clasicos-de-sincronizacion/
- Semáforo (informática). (2021, February 04). Retrieved from https://es.wikipedia.org/wiki/Semáforo_(informática)
- Sistemas operativos. (1986). CEDED.
- Carretero Pérez, J., De Miguel Anasagasti, P., García Carballeira, F., & Pérez Costoya, F. (2001). Sistemas operativos. Una visión aplicada. Mac Graw Hill.
- Tanenbaum, A. S., & Woodhull, A. S. (1997). Operating systems: design and implementation (Vol. 68).
- Wolf, G. (2015). Fundamentos de sistemas operativos. Lulu.com