I OBJETIVOS

Se pretende que en esta práctica el alumno:

- Probará una de las soluciones por hardware para el problema de la sección crítica con más de dos procesos
- Describirá la diferencia para lograr la comunicación entre hilos dentro de un mismo proceso y comunicación entre procesos.
- Describirá el problema de espera ocupada y encontrará una solución a este problema.
- El alumno realizará la implantación de semáforos tal como lo hace el sistema operativo en un ambiente donde hay procesos concurrentes.
- Conocerá y usará algunos mecanismos de comunicación entre procesos que provee el sistema operativo tales como la memoria compartida.
- Utilizará los semáforos como una herramienta para la sincronización de procesos concurrentes.

II BIBLIOGRAFÍA

- William Stallings, “SISTEMAS OPERATIVOS”,Prentice Hall, 4ª Ed.
- Silberschatz, Galvin, Gagne, “SISTEMAS OPERATIVOS”, Limusa Wiley, 6ª Ed.
- Neil Matthew & Richard Stones, “BEGINNING LINUX PROGRAMMING”, Wrox, 2ª Ed.

III RECURSOS

- Una estación de trabajo con Linux con su ambiente gráfico.
- Un editor de texto.
- El compilador de C GNU.

IV ACTIVIDADES

1 Descripción del problema

En los países sudamericanos Perú, Bolivia y Colombia existe una red de narcotraficantes que utilizando ferrocarriles transportan cocaína a sus clientes que están en el otro lado de la cordillera de los Andes.

En la cordillera de los Andes existe un tramo muy angosto en el cual las vías de los tres ferrocarriles se unen en una y sólo es posible que pase un ferrocarril a la vez (ver Figura 1). Para

esto, los narcos probaron sincronizarse a través de soluciones por hardware y decidieron implantar semáforos para lograr la exclusión mutua en esta zona y evitar colisiones entre los ferrocarriles.

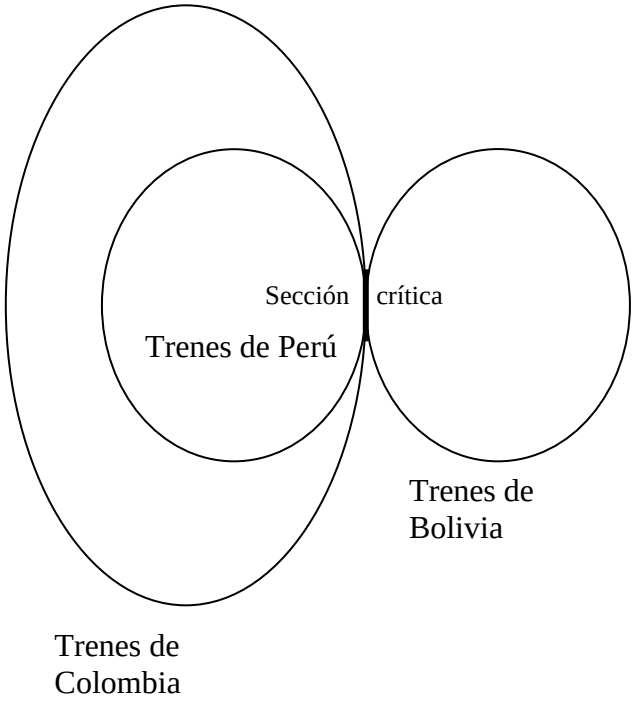

Figura 1.- Ilustración de la sección crítica en el problema de los trenes.

2 Soluciones ejemplo con soluciones por Hardware.

Una de los recursos que provee el hardware y pueden usarse para lograr la exclusión mutua de *n* procesos o hilos en una sección crítica es la instrucción `xchg` del CPU. Para esto utilizamos una variable global *g* y variables locales *l* (una por cada hilo o proceso). De esta manera inicializamos la variable global *g* en 0 y las variables locales *l* de cada proceso o hilo. Cuando un proceso o hilo desea entrar a la sección crítica intercambia los valores de su variable local *l* con la variable global *g* hasta que la variable local *l* sea igual a 0. Esto significa que ningún hilo o proceso ha hecho el intercambio y por lo tanto se le permite ingresar a la sección crítica.

2.1 Una solución con hilos

En el Ejemplo 1 mostramos el problema de los ferrocarriles de los narcos que necesitan sincronizarse para entrar a la sección crítica. En este ejemplo cada ferrocarril es un hilo y estos se sincronizan con la instrucción `XCHG`. Ya que se tratan de hilos dentro del mismo proceso, para compartir una variable es suficiente con que esta sea definida como variable global.

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

// Macro que incluye el código de la instrucción máquina xchg
#define atomic_xchg(A,B)      __asm__ __volatile__(   \
                                    "   lock xchg %1,%0 ;\n"      \
                                    : "=ir"  (A)                  \
                                    : "m"  (B), "ir" (A)          \
                                    );


#define CICLOS 10

char *pais[3]={"Peru","Bolvia","Colombia"};

int g=0;

void *hilo1(void *arg)
{
      int *mynum=(int *) arg;
      int i=*mynum;

      int k;
      int l;

      for(k=0;k<CICLOS;k++)
      {
            l=1;
            do { atomic_xchg(l,g); } while(l!=0);

            // Inicia sección Crítica
            printf("Entra %s",pais[i]);
            fflush(stdout);
            sleep(rand()%3);
            printf("- %s Sale\n",pais[i]);
            // Termina sección Crítica

            g=0;
            l=1;

            // Espera aleatoria fuera de la sección crítica
            sleep(rand()%3);
      }
}

int main()
{
      pthread_t tid[3];
      int res;
      int args[3];
      int i;
      void *thread_result;
```

```
	srand(getpid());

	// Crea los hilos
	for(i=0;i<3;i++)
	{
		args[i]=i;
		res = pthread_create(&tid[i], NULL, hilo1, (void *) &args[i]);
	}

	// Espera que terminen los hilos
	for(i=0;i<3;i++)
		res = pthread_join(tid[i], &thread_result);

}
```

Ejemplo 1. El problema de los narcos usando hilos sincronizados a través de la instrucción xchg

2.2 Una solución con procesos

En el Ejemplo 2 mostramos el mismo problema de los ferrocarriles de los narcos, pero ahora estos son representados por procesos. A diferencia del Ejemplo 1 para que estos puedan compartir la variable global *g* es necesario solicitar al sistema operativo un área de memoria compartida y definir la variable global *g* como una variable apuntador que apuntará al área de memoria compartida que nos proveerá el sistema operativo.

Para solicitar memoria compartida al sistema operativo podemos hacerlo con llamadas que este nos provee, tal como `shmget` para obtener un segmento de memoria compartida, `shmat` para conectar este segmento de memoria compartida y obtener un apuntador a él, y `shmdt` para desconectar la memoria compartida al final de la ejecución.

Cuando el sistema operativo nos provee memoria compartida, esto puede ser visible a través del comando de UNIX `ipcs`. Este comando nos proveerá información de mecanismos de comunicación entre procesos provistos por el sistema operativo como pueden ser: memoria compartida; semáforos y colas de mensajes. También podemos usar el comando `ipcrm` para eliminar aquellos que no utilizamos y se quedaron en el sistema por error en la ejecución de nuestros programas que hagan uso de estos mecanismos.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define atomic_xchg(A,B)	__asm__ __volatile__(   \
                                "   lock xchg %1,%0 ;\n"      \
                                : "=ir"  (A)                  \
                                : "m"  (B), "ir" (A)          \
                                );
```

```
#define CICLOS 10

char *pais[3]={"Peru","Bolvia","Colombia"};

int *g;

void proceso(int i)
{
	int k;
	int l;

	for(k=0;k<CICLOS;k++)
	{
		l=1;
		do { atomic_xchg(l,*g); } while(l!=0);

		printf("Entra %s",pais[i]);
		fflush(stdout);
		sleep(rand()%3);
		printf("- %s Sale\n",pais[i]);

		l=1;
		*g=0;

		// Espera aleatoria fuera de la sección crítica
		sleep(rand()%3);
	}
	exit(0);	// Termina el proceso
}

int main()
{
	int pid;
	int status;
	int shmid;
	int args[3];
	int i;
	void *thread_result;

	// Solicitar memoria compartida
	shmid=shmget(0x1234,sizeof(g),0666|IPC_CREAT);
	if(shmid==-1)
	{
		perror("Error en la memoria compartida\n");
		exit(1);
	}

	// Conectar la variable a la memoria compartida
	g=shmat(shmid,NULL,0);

	if(g==NULL)
	{
		perror("Error en el shmat\n");
		exit(2);
```

```
	}

	*g=0;

	srand(getpid());

	for(i=0;i<3;i++)
	{
		// Crea un nuevo proceso hijo que ejecuta la función proceso()
		pid=fork();
		if(pid==0)
			proceso(i);
	}

	for(i=0;i<3;i++)
		pid = wait(&status);

	// Eliminar la memoria compartida
	shmdt(g);
}
```

Ejemplo 2. El problema de los narcos usando procesos sincronizados a través de la instrucción xchg con una variable en memoria compartida.

3 *Implantación de semáforos utilizando los mecanismos de comunicación entre procesos ya conocidos.*

Utilizando mecanismos básicos de comunicación entre procesos como son señales[1] y memoria compartida[2], desarrolle una implementación de semáforos enteros.

Para esta implementación es necesario:

- Definir la estructura de datos semáforo para definir las variables semáforo.
- Las variables de tipo semáforo deben estar en memoria compartida.
- Definir las primitivas `waitsem` y `signalsem`. Ambas primitivas reciben como argumento un semáforo.
- Definir una primitiva para la inicialización de las variables semáforo `initsem`.
- Garantizar la atomicidad[3] en las llamadas `waitsem`, `signalsem` e `initsem`, puede utilizar la solución por hardware aquí mostrada.
- En el Ejemplo 3 puede verse la forma de cómo un programa deberá llamar estas primitivas y los argumentos que reciben.

[1] Señales es un mecanismo de comunicación entre procesos, utilizando señales podemos suspender y reanudar procesos, ver la llamada al sistema `kill`.
[2] Memoria compartida es un mecanismo de comunicación entre procesos que consta de que dos o más procesos pueden acceder el mismo segmento de memoria, ver llamadas al sistema `shmget shmat` y `shmctl`.
[3] Considere que n procesos pueden estar intentando ejecutar `waitsem` y `signalsem` y la ejecución de estas llamadas deben ser mutuamente exclusivas.

Tips:

- Un proceso puede bloquearse enviándosele una señal `SIGSTOP` y puede ser reanudado con una señal `SIGCONT`

3.1 Prueba de la solución

Modifica el programa ejemplo de manera que muestre el problema de los narcos que necesitan transportar cocaína, **no utilice los semáforos de UNIX**, utilice los semáforos definidos previamente para preservar la exclusión mutua en la sección crítica. En el Ejemplo 3 se muestra como podría quedar esta modificación que utilizará nuestras primitivas previamente definidas.

```
      …
      …

void proceso(int i)
{
      int k;

      for(k=0;k<CICLOS;k++)
      {
            // Llamada waitsem implementada en la parte 3
            waitsem(sem);

            printf("Entra %s ",pais[i]);
            fflush(stdout);
            sleep(rand()%3);
            printf("- %s Sale\n",pais[i]);

            // Llamada waitsignal implementada en la parte 3
            signalsem(sem);

            // Espera aleatoria fuera de la sección crítica
            sleep(rand()%3);
      }
      exit(0);    // Termina el proceso
}

int main()
{
      …
      …
      // Incializar el contador del semáforo en 1 una vez que esté
      // en memoria compartida, de manera que solo a un proceso se le
      // permitirá entrar a la sección crítica
```

```
        initsem(sem,1);
        …
        …
        …
}
```

Ejemplo 3. Utilización de las llamadas waitsem y signalsem en el problema de los narcos con procesos.

4 Experimentos y preguntas

1. Explica por qué con la solución con procesos, a diferencia de la solución con hilos es necesario solicitar memoria compartida al sistema operativo.

Ejecuta el monitor del sistema de Linux o cualquier utilería que te permita monitorear el uso del CPU en Linux.

2. ¿Cuál es la utilización del CPU durante la ejecución del Ejemplo 1?

3. ¿Cuál es la utilización del CPU durante la ejecución del Ejemplo 2?

4. ¿Cuál es la utilización del CPU durante la ejecución del Ejemplo 3 el cuál se sincroniza utilizando nuestra implementación de semáforos?

5. Existe diferencia entre la utilización del CPU en la ejecución de las soluciones anteriores, ¿cuál de todas es mejor? explica por qué.

V ENTREGA Y EVALUACIÓN

No incluya líneas de código en sus programas de las cuales desconozca su funcionamiento. El código no conocido será anulado en el funcionamiento de la práctica.

Aunque en semestres anteriores se han realizado prácticas similares a esta, hay aspectos que hacen que esta sea diferente. Cualquier evidencia que muestre el intento de entregar una práctica de semestre anterior será calificada como plagio.

1 Entrega y Revisión

Entregar en el apartado correspondiente de Moodle un archivo .ZIP que contenga:

- Los programas fuentes que se piden en la parte 3.1 que muestren que se preserva la exclusión mutua en una sección crítica utilizando las funciones que se solicitan implementar en la parte 3
- El archivo `Makefile` que genere el programa ejecutable.
- Un documento en formato .PDF con las respuestas a las preguntas de la parte 4.

La fecha límite de entrega es el Domingo 14 de Octubre a las 23:55 hrs. La revisión se realizará a partir de la semana 13

2 Equipos

Esta práctica se hará en equipos (máximo 2 integrantes), es necesario que en la revisión esté el equipo completo ya que el integrante que no se presente no tendrá calificación en la práctica.

Importante: Al indicarse que el trabajo debe ser desarrollado por equipos, se entiende que no se permite colaboración entre equipos, cualquier evidencia de esto será considerada plagio.

3 Evaluación

Puntualidad en las revisiones	El equipo estuvo completo y puntual en todas las sesiones de revisión.		Si hubo dos o más sesiones con el equipo, el equipo estuvo completo y puntual en casi todas las sesiones de revisión		Si solo hubo una sesión de revisión, el equipo no estuvo completo o no fue puntual. Si fueron dos o más sesiones de revisión, en más de una sesión el equipo no estuvo completo o fue puntual
	+5		+2.5		0
Especificaciones de entrega	La entrega del producto cumple con todas las especificaciones indicadas en el documento de la práctica, por ejemplo, los archivos se entregan de acuerdo a las formas indicadas en el documento de la práctica.				La entrega del producto no cumple con al menos una de las especificaciones indicadas en el documento de la práctica
	+5				0
Funcionamiento	El producto cumple con todas las especificaciones indicadas en el documento y no tiene fallas	El producto muestra una falla no esperada.	El producto está incompleto (falta máximo aprox 50%), pero lo demás puede funcionar bien	El producto está incompleto (falta máximo aprox 50%) y además muestra fallas o el producto está incompleto (falta máximo aprox 66%)	El producto no funciona o no está incompleto (más del 66%).
	+80	+60	+40	+20	0
Interfaz con el usuario	El producto funciona y pudo ser utilizado sin necesidad de recibir indicaciones por el desarrollador, tiene instrucciones claras para ser utilizado.		El producto funciona, pero hubo necesidad de recibir alguna indicación para su uso por parte del desarrollador del producto		El producto carece de instrucciones claras para ser utilizado y requiere que alguno de los desarrolladores esté presente para su utilización o no puede utilizarse debido a que no está completo
	+5		+3.5		0
Claridad en el código	El código es claro, usa nombres de variables adecuadas, está debidamente comentado e indentado. Puede ser entendido por cualquier otra persona que no intervino en su desarrollo.		El código carece de claridad, puede ser entendido por cualquier persona ajena a su desarrollo pero con cierta dificultad.		El código carece de comentarios, está mal indentado, usa nombres de variables no adecuadas.
	+5		+3.5		0
Defensa del producto	Todos los que presentan la práctica son capaces de explicar cualquier parte del producto presentado		Uno de los que presenta la práctica muestra dudas sobre alguna parte del desarrollo del producto presentado		Más de un integrante no muestra evidencia de que conoce el producto, o si el trabajo fue individual, el desarrollador duda sobre el desarrollo del producto que presenta.
	x 1(puntos se multiplican por 1)		x 0.5 (puntos se multiplican por 0.5)		x 0 (puntos se multiplican por 0)
Sobresaliente 20 %	Tiene 1 en todos los puntos anteriores. El producto entregado es sobresaliente, muestra tener la calidad para ser expuesto como un producto representativo de la carrera Hay evidencia de que los desarrolladores se documentaron y muestran aprendizajes más allá de lo esperado				No tiene 1 en todos los puntos anteriores, o el producto entregado no es sobresaliente y no muestra tener la calidad para ser expuesto como un producto representativo de la carrera o no hay evidencia de que los desarrolladores se documentaron y muestran aprendizajes más allá de lo esperado
	+20				0