UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL MAR DEL PLATA
ARQUITECTURA DE SISTEMAS OPERATIVOS
1er Año – 1er Cuatrimestre

Trabajo Práctico:

Comunicación y Sincronización entre Procesos

RESOLUCIÓN

1. ¿Qué se entiende por deadlock y livelock? De un ejemplo de cada uno fuera de los sistemas computacionales y analícelo.

   Deadlock (interbloqueo): situación donde dos o más procesos son incapaces de actuar porque cada uno está esperando que alguno haga algo.
   Ejemplo: Dos personas quieren jugar al arco y flecha, pero solo se dispone de un arco y una flecha, entonces una persona toma uno de los recursos (arco) y la otra persona toma el otro recurso (flecha), luego, ambas personas se quedan esperando a que la otra finalice su turno pero se ven impedidos mutuamente ya que ninguna de las dos personas disponen de ambos recursos para realizar el turno.
   Livelock (círculo vicioso): dos o más procesos cambian continuamente su estado en respuesta a cambios en los otros procesos, sin realizar trabajo útil.
   Ejemplo: Cuando dos personas se encuentran en un pasillo y se chocan y ambos se mueven hacia los costados para pasar y ambos se bloquean mutuamente en un ciclo que nunca termina

2. En un sistema conviven 3 procesos y 2 recursos. Uno de los recursos (R2) es de uso exclusivo y el otro (R1) puede ser compartido por hasta dos procesos.El proceso 1 necesita R1 , P2 necesita R1,R2 y P3 necesita R1.

   a. ¿Puede haber deadlock o livelock?

Si puede haber deadlock o livelock en el caso de que al menos uno de los procesos (P1,P2) no libere el R1 para que lo pueda tomar el P3.

¿Y si ahora R1 puede ser compartido por hasta 3 procesos?

El sistema estaría libre de deadlock.

3. Considere un sistema con 4 recursos del mismo tipo compartidos por 3 procesos. Cada recurso puede ser requerido a lo sumo por 2 procesos, y cada proceso puede requerir hasta 2 recursos. ¿En qué estado se encuentra el sistema? ¿Por qué?
   Si todos los procesos solicitan recursos simultáneamente, podrían cumplir sus solicitudes sin entrar en conflicto, ya que hay suficientes recursos disponibles.

4. En un sistema hay tres procesos (P1, P2 y P3) y tres recursos (R1, R2 y R3). Los tres recursos son de uso exclusivo. Se sabe que P1 requiere los tres recursos, P2 requiere de R1 y R2 y P3 solo requiere R3.

   a. ¿El sistema está libre de deadlock?
      Si todos los procesos solicitan sus recursos al mismo tiempo, se podría llegar a un estado de deadlock, ya que P1 tiene todos los recursos que P2 necesita, y P2 tiene un recurso que P1 necesita. Además, P3 podría no obtener su recurso (R3) si está siendo utilizado por P1. Por lo tanto, el sistema no está libre de deadlock.
   b. ¿P3 influye en que el sistema esté o no libre deadlock?
      Sí, P3 puede influir en la posibilidad de deadlock. Si P3 solicita R3 y P1 ya lo está utilizando, podría crear una situación donde P3 no puede avanzar debido a la retención del recurso por parte de P1, contribuyendo así al riesgo de deadlock.
   c. Si me aseguro que P2 no podrá pedir ningún recurso hasta que P1 haya liberado todos sus recursos. ¿El sistema está libre de deadlock? ¿Por qué?
      En ese caso si se liberan los recursos que adquirió el P1 (R1,R2,R3) y se informa a los demás procesos (P2,P3) mediante el uso de semáforos se puede garantizar que el sistema está libre de deadlock.

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL MAR DEL PLATA
ARQUITECTURA DE SISTEMAS OPERATIVOS
1er Año – 1er Cuatrimestre

5. Considere el siguiente programa. Note que el scheduler irá ejecutando estos procesos de manera concurrente, mezclando la ejecución de P1 y P2.

int x=10;	
Proceso 1	Proceso 2

while (true) {
x--;
x++;
if (x != 10)
printf("x is %d",x);
}
	while (true) {
	x--;
	x++;
	if (x != 10)
	printf("x is %d",x);
	}

a. Muestre una secuencia de ejecución (indicando el trace de programa/instrucción) en que imprima “x is 9”.

Proceso 1	Proceso 2
while (true) { ✅	
x - -; // x = 9	
x ++; // x = 10	
	while (true) { ✅
	x - -; // x = 9
	x ++; // x = 10
if(x != 10) 🚫	

while (true) { ✅	
x - -; // x = 9	
	if(x != 10) ✅
	printf("x is %d",x); // x is 9

b. Muestre una secuencia de ejecución (indicando el trace de programa/instrucción) en que imprima “x is 10”.

Proceso 1	Proceso 2
while (true) { ✅	
x - -; // x = 9	
	while (true) { ✅
	x - -; // x = 8
x ++; // x = 9	
if(x != 10) ✅	
	x ++; // x = 10
	if(x != 10) 🚫
printf("x is %d",x); // x is 10	

6. Considera que tiene un programa con dos procesos. Un proceso se encarga de imprimir y el otro se encarga de introducir ficheros para su impresión.

<table>
<tr><td colspan="2">int peticiones = 0;</td></tr>
<tr><td>Proceso Impresora</td><td>Proceso Imprime Fichero</td></tr>
<tr><td>

```
int R1;
While (true) {
    If ( peticiones > 0 ){
        extraerColaPeticion();
        R1 = peticiones;
        R1 = R1 - 1;
        peticiones = R1;
    }
}
```

</td><td>

```
int R2;
While (true) {
    insertarFicheroCola();
    R2 = peticiones;
    R2 = R2 + 1;
    peticiones = R2;
}
```

</td></tr>
</table>

Nota:

- La función extraerColaPeticion() extrae de la cola de peticiones un fichero y lo manda a imprimir. *(peticiones - 1).*
- La función insertarFicheroCola() inserta en la cola de peticiones un archivo para su impresión. *(peticiones + 1).*

a. ¿Presenta algún problema el código anterior si se ejecuta de manera concurrente? Analice el código.

Si ambos procesos intentan modificar peticiones al mismo tiempo, puede ocurrir una condición de carrera, lo que significa que los resultados de las operaciones pueden depender del orden en que se ejecuten los procesos. Además, nos encontramos con que dicha variable puede incrementarse/decrementarse en varias unidades sin sentido dentro del programa, posiblemente generando pérdida de información.

b. Si la respuesta es negativa, explique el porqué. Si la respuesta es positiva, muestre la secuencia donde se genera el problema (Trace del programa).

Proceso Impresora	Proceso Imprime Fichero
While (true) {	
	While (true) {
If (peticiones > 0){ 🚫	
	insertarFicheroCola(); // p = 1
While (true) {	
	R2 = peticiones; // R2 = 1
If (peticiones > 0){ ✅	
	R2 = R2 + 1; // R2 = 2
extraerColaPeticion(); // p = 0	
	peticiones = R2; // p = 2 🧐
R1 = peticiones; // R1 = 2	
	While (true) {
R1 = R1 - 1; // R1 = 1	
	insertarFicheroCola(); // p = 3
peticiones = R1; // p = 1 🧐	

7. Se tienen 3 procesos A, B y C. Implementar en máquina:
    a. El código con semáforos de manera tal que la secuencia sea ABC, ABC,ABC…

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

sem_t sem1, sem2, sem3;

void funcion1(){
  for(int i = 0; i<5; i++){
    sem_wait(&sem1); ///sem1 = 0
    printf("Soy el hilo A\n");
    sem_post(&sem2);
  }
    pthread_exit(0);
}

void funcion2(){
  for(int i=0; i<5; i++ ){
    sem_wait(&sem2);
    printf("Soy el hilo B\n");
    sem_post(&sem3);
  }
       pthread_exit(0);
}

void funcion3(){
  for(int i = 0; i<5; i++)  {
     sem_wait(&sem3);
     printf("Soy el hilo C\n");
     sem_post(&sem1);
```

```
 }
     pthread_exit(0);
}

int main()
{
   sem_init(&sem1, 0, 1); ///1 = abierto, 0 = cerrado representa a A
   sem_init(&sem2, 0, 0); ///representa a B
   sem_init(&sem3, 0, 0); ///representa a C

   pthread_t a, b ,c;

   pthread_create(&a, NULL, funcion1, NULL);
   pthread_create(&b, NULL, funcion2, NULL);
   pthread_create(&c, NULL, funcion3, NULL);



   pthread_join(a, 0);
   pthread_join(b, 0);
   pthread_join(c, 0);

   sem_destroy(&sem1);
   sem_destroy(&sem2);
   sem_destroy(&sem3);


   return 0;
}
```

b. ¿Y si quiero que la secuencia sea BCA, BCA, BCA…?

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

```
#include <pthread.h>

sem_t sem1, sem2, sem3;

void funcion1(){
  for(int i = 0; i<5; i++){
    sem_wait(&sem1); ///sem1 = 0
    printf("Soy el hilo A\n");
    sem_post(&sem2);
  }
    pthread_exit(0);
}

void funcion2(){
  for(int i=0; i<5; i++ ){
    sem_wait(&sem2);
    printf("Soy el hilo B\n");
    sem_post(&sem3);
  }
       pthread_exit(0);
}

void funcion3(){
  for(int i = 0; i<5; i++)  {
     sem_wait(&sem3);
     printf("Soy el hilo C\n");
     sem_post(&sem1);
  }
       pthread_exit(0);
}

int main()
{
    sem_init(&sem1, 0, 0); ///1 = abierto, 0 = cerrado representa a A
    sem_init(&sem2, 0, 1); ///representa a B
    sem_init(&sem3, 0, 0); ///representa a C
```

```
    pthread_t a, b ,c;

    pthread_create(&a, NULL, funcion1, NULL);
    pthread_create(&b, NULL, funcion2, NULL);
    pthread_create(&c, NULL, funcion3, NULL);



    pthread_join(a, 0);
    pthread_join(b, 0);
    pthread_join(c, 0);

    sem_destroy(&sem1);
    sem_destroy(&sem2);
    sem_destroy(&sem3);


    return 0;
}
```

8. ¿Tiene algún problema el siguiente programa? ¿Por qué?

P1	**P2**

sem_wait(Sem1);
sem_wait(Sem2);
// Sección Crítica //
sem_post(Sem2);
sem_post(Sem1);
	sem_wait(Sem2);
	sem_wait(Sem1);
	// Sección Crítica //
	sem_post(Sem1);
	sem_post(Sem2);

Sí, si ambos semáforos son inicializados en 1 con un q=1 en el sistema, el sem_wait() del inicio de cada proceso hará que ambos queden bloqueados, llevando al sistema a un deadlock.

9. Implementar en máquina la sincronización de los procesos A y B de tal manera que, siendo X una variable global:

A	B

x=199;
x=x+1;
Print(x);
	x=500;
	x=X/10;
	Print(x);

a. Siempre el resultado de la ejecución sea 200 y 50.

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

sem_t sem_A;
sem_t sem_B;
int x = 0;

void* proceso_A(void* arg) {
    sem_wait(&sem_A);
    x = 199;
    x = x + 1;      // x = 200
    printf("%d\n", x); // Imprime x (200)
    sem_post(&sem_B);
    return NULL;
}
```

```
void* proceso_B(void* arg) {
    sem_wait(&sem_B);
    x = 500;
    x = x / 10;     // x = 50
    printf("%d\n", x); // Imprime x (50)
    sem_post(&sem_A);
    return NULL;
}

int main() {
    sem_init(&sem_A, 0, 1); // Inicializa el semáforo con valor 1
    sem_init(&sem_B, 0, 0); // Inicializa el semáforo con valor 0

    pthread_t hilo_A, hilo_B;

    pthread_create(&hilo_A, NULL, proceso_A, NULL);
    pthread_create(&hilo_B, NULL, proceso_B, NULL);

    pthread_join(hilo_A, NULL);
    pthread_join(hilo_B, NULL);

    sem_destroy(&sem_A);
    sem_destroy(&sem_B);

    return 0;
}
```

b. Siempre el resultado de la ejecución sea 50 y 200.

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

sem_t sem_A;
```

```
sem_t sem_B;
int x = 0;

void* proceso_A(void* arg) {
    sem_wait(&sem_A);
    x = 199;
    x = x + 1;      // x = 200
    printf("%d\n", x); // Imprime x (200)
    sem_post(&sem_B);
    return NULL;
}

void* proceso_B(void* arg) {
    sem_wait(&sem_B);
    x = 500;
    x = x / 10;     // x = 50
    printf("%d\n", x); // Imprime x (50)
    sem_post(&sem_A);
    return NULL;
}

int main() {
    sem_init(&sem_A, 0, 0); // Inicializa el semáforo con valor 1
    sem_init(&sem_B, 0, 1); // Inicializa el semáforo con valor 0

    pthread_t hilo_A, hilo_B;

    pthread_create(&hilo_A, NULL, proceso_A, NULL);
    pthread_create(&hilo_B, NULL, proceso_B, NULL);

    pthread_join(hilo_A, NULL);
    pthread_join(hilo_B, NULL);

    sem_destroy(&sem_A);
    sem_destroy(&sem_B);
```

```
    return 0;
}
```

10. Dado el problema del Productor-Consumidor y los siguientes algoritmos:

sem_t espacioLibre, manipulaciónDelBuffer, elementoDisponible;	
Productor	**Consumidor**

While(true) {
producirUnElemento;
sem_wait(espacioLibre);
sem_wait(manipulaciónDelBuffer);
depositarElementoEnBuffer;
sem_post(manipulaciónDelBuffer);
sem_post(elementoDisponible);
}
	While(true) {
	sem_wait(manipulaciónDelBuffer);
	sem_wait(elementoDisponible);
	sacarElementoDelBuffer;
	sem_post(manipulaciónDelBuffer);
	sem_post(espacioLibre);
	consumirElemento;
	}

a. Inicialice los semáforos según corresponda. ¿Es válida la solución anterior? ¿Por qué?

```
sem_init(&espacio_libre, 0, TamañoDelBuffer);
sem_init(&manipulacion_buffer, 0, 1);
sem_init(&elemento_disponible, 0, 0);
```

En el código del Consumidor, se debe modificar el orden de los sem_wait() para asegurar que “elementoDisponible” se verifica antes de “manipulaciónDelBuffer”.