Ingeniería en Sistemas Computacionales

Fundamentos de Sistemas Operativos

Actividad 14

IS730547 – Santiago Cordova Berrelleza

IS727272 - Marco Ricardo Cordero Hernández

Jal., 15 de junio de 2023

1.- Cuatro personas están sentados en una mesa jugando dominó, se van turnando para tirar una ficha o pasar si no pueden jugar en el sentido contrario a las manecillas del reloj. Represente a los cuatro jugadores con procesos desde P(0) hasta P(3) y como sincronizaría los turnos usando un arreglo de semáforos.

R: Pseudocódigo

```
Semaphore sem[4] = {0, 0, 0, 0};


jugador (int i) {
      int next = (i + 1) % 4;
      while (!juego.terminado) {
            wait(sem[i]);
            tirar ficha o pasar
            signal(sem[next]);
      }
      exit(0);
}


main () {
      cobegin {
            jugador(0); jugador(1);
            jugador(2); jugador(3);
      }
      signal(sem[0]);
}
```

2.- Considere el ejemplo del productor consumidor que se muestra a continuación

```
#define TAM_BUFFER 10
Semaphore s = 1;
Semaphore n = 0;
Semaphore e = TAM_BUFFER;

productor()
```

```
{
    while(TRUE)
    {
        producir();
        wait(e);
        wait(s);
        añadir();
        signal(s);
        signal(n);
    }
}

consumidor()
{
    while(TRUE)
    {
        wait(n);
        wait(s);
        tomar();
        signal(s);
        signal(e);
        consumir();
    }
}
```

¿Cambiaría el significado del programa si se intercambian las siguientes sentencias?

1. wait(e); wait(s)
   Alterar el orden de estas líneas podría provocar una ejecución incorrecta, ya que, al poner el producto directamente en el buffer sin consultar su disponibilidad espacial, es posible que se exceda el tamaño máximo establecido para el mismo, provocando una inconsistencia de datos, y por ende, una ejecución errónea.

2. signal(s); signal(n)
   Este cambio también provocaría un consumo de datos erróneos o nulos para el consumidor, ya que al realizar esta modificación se está indicando que el consumidor puede comenzar a tomar datos del buffer aún cuando el productor no ha terminado de posicionar sus resultados en memoria.
   A pesar de una posible ejecución inesperada, la forma en que el código del consumidor está acomodada hace posible el caso de una probable ejecución exitosa, ya que al encontrarse en este nuevo orden (wait n, después wait s) las esperas del consumidor, algunas iteraciones del proceso probablemente resultarán en el comportamiento deseado.

3. wait(n); wait(s)
   Similar al inciso anterior, así como pueden existir ejecuciones correctas, es más probable que se encuentren aquellas consideradas como incorrectas. Si se desea esperar primero al término del posicionamiento de productos en el buffer antes que a la señal de permiso para comenzar a tomar elementos de este, es probable que los tiempos del programa sean favorables para su ejecución y todo resulte como se desea, no obstante, también existe el caso en que un desfase temporal afecte la ejecución y termine en datos corruptos.

Aparentemente, esta modificación no resulta perjudicial del todo, más bien, el orden de los algoritmos se definieron como tal para asegurar una ejecución correcta.

4. signal(s); signal(e)
   Ahora, como el primer punto, si se le da el paso al buffer, existe el caso en que este no se haya desplazado correctamente en nivel lógico de la memoria, cabiendo la posibilidad de corrupción de datos y posiciones ocupadas erróneamente al realizar la espera el productor.

3.- De ser posible implementar semáforos generales por medio de semáforos binarios. Se pueden usar las operaciones WaitB y SignalB y dos semáforos binarios espera y exmut. Considere lo siguiente:

```
struct strsemáforo {
     int contador;
     BinSemaphore espera=0;
};

typedef struct strsemáforo * semáforo;
BinSemaphore exmut=1;

void Wait(semáforo s)
{
   WaitB(exmut);
   s->contador--;
   if(s->contador!=0)
   {
        SignalB(exmut);
        WaitB(s->espera);
    }
    else
        Signal(exmut);
}

void Signal(semáforo s)
{
   Waitb(exmut);
   s->contador++;
   if(s->contador!=0)
      SignalB(s->espera);
   SignalB(exmut);
}
```

Inicialmente, s->contador tiene el valor deseado por el semáforo. Cada operación Wait disminuye s->contador y cada operación Signal lo incrementa. El semáforo binario exmut iniciado en 1, asegura que hay exclusión mutua en las actualizaciones de s->contador. El semáforo binario s->espera, iniciado a 0, se usa para suspender procesos.

De acuerdo a las especificaciones anteriores, muestren si la implementación de semáforos enteros a partir de semáforos binarios es correcta, y si no lo es indique en dónde es incorrecto, proponga la corrección.

R: Antes de demostrar siquiera alguna validez de lo propuesto, cabe recalcar que las condiciones que revisan el contador del semáforo son incorrectas, ya que al tratarse de un intento de implementación de semáforos generales (con contadores signados), estos valores pueden tomar números menores a ceros, haciendo desde antes de su análisis a esta implementación como incorrecta.

La manera de solucionar lo anterior simplemente sería modificando la comparación de la desigualdad por una revisión de valores menores a cero. Con esta modificación, se propone el siguiente código:

```
struct strsemáforo {
     int contador;
     BinSemaphore espera=0;
};

typedef struct strsemáforo * semáforo;
BinSemaphore exmut=1;

void Wait(semáforo s)
{
   WaitB(exmut);
   s->contador--;
   if(s->contador < 0)
   {
        SignalB(exmut);
        WaitB(s->espera);
    }
    else
        Signal(exmut);
}

void Signal(semáforo s)
{
   Waitb(exmut);
   s->contador++;
   if(s->contador <= 0)
      SignalB(s->espera);
   SignalB(exmut);
}
```

4.- La siguiente solución intenta resolver el problema de permitir que K de n procesos puedan ejecutar concurrentemente su sección crítica usando para esto sólo semáforos binarios.

Muestre si es posible que haya más de K procesos en la sección crítica para:

k=2 n=3

k=2 n=4

y se podría concluir que no funciona para valores de n mayores.

Muestre que problemas puede presentar esta solución y plantee una solución que resuelva correctamente con semáforos binarios dicho problema.

```
    // Constantes

    const int K=2;
    const int N=4;

    // Variables
    int contador;
    int bloqueados=0;
    binarysem mutex=1;
    binarysem retardo=0;

    void proceso()
    {
 1:     wait(mutex);
 2:     if(contador==0)
        {
 3:         bloqueados++;
 4:         wait(retardo);
        }
        else
 5:         contador--;
 6:     signal(mutex);

        // Sección crítica

 7:     wait(mutex);
 8:     if(bloqueados>0)
        {
 9:         signal(retardo);
10:         bloqueados--;
        }
11:     else
12:         contador++;
13:     signal(mutex);
    }

    void main()
    {
14:     contador=K;
        cobegin
        {
15:         proceso(); proceso(); proceso();proceso();
        }
    }
```

Suponga que el semáforo mutex no existe, la variable contador=1 y dos procesos ejecutan concurrentemente (paralelamente) las líneas de la 1 a la 6.

Proceso 1	Proceso 2	Contador	Bloqueados		Sem Retardo (Valor)
else contador --;		1	0		0 - cola vacia
	if(contador==0)	0	0		
SC					
	bloqueados ++;		1		
SC					
	wait(retardo)		1		0 - cola con P2

Suponga que el semáforo mutex si existe, la variable contador=1 y dos procesos ejecutan concurrentemente (paralelamente) las líneas de la 1 a la 6. ¿Qué pasa a diferencia del caso de la pregunta anterior?

Lo que posiblemente puede pasar en la primera es que no entren en caso de que no entren nunca al if de contador == 0 si es que se hubiera hecho los contador--, dejando el contador en -1, corriendo y posteriormente entrando a SC, con el semáforo de Mutex, nos aseguramos que se detenga para correr sin problema las primeras 6 líneas del código.

		inicial =1	inicial=0	incial =1	inicial =0
Proceso 1	Proceso 2	Contador	Bloqueados	Sem Mutex (Valor)	Sem Retardo (Valor)
wait(mutex);		1	0	0	
	wait(mutex);			0 - P2 en cola	
if(contador==0)					
else contador --;		0			
signal(mutex);				0 - cola vacia	
	if(contador==0)				
	bloqueados++;		1		
	wait(retardo);				0 - P2 en cola

¿Qué sucede si contador=0 y un proceso ejecuta las líneas de la 1 a la 4?, esto funcionaría correctamente?, proponga modificaciones

No, se bloquearía con entrar el primero en wait retardo, por lo tanto, el segundo proceso nunca recibiría el signal de mutex dejándolo bloqueado también.

```
if(contador==0)
    {
3:      bloqueados++;
4:      wait(retardo);

            contador++;

            signal(mutex);
    }
```

Con ese signal hacia mutex aseguramos que el segundo seguiría corriendo y el contador++ también para que no pase de nuevo.

5.- ¿Cuáles son tus aprendizajes de esta actividad?

Se ha visto y aprendido acerca de la técnica de sincronización de procesos conocida como semáforos a gran profundidad, exigiéndonos pensar de formas alternativas para ser capaces de resolver los problemas previos. También se ha tenido la oportunidad de ver múltiples implementaciones reales de la técnica mencionada a través de código ejecutable en Linux, lo cual agrega un nivel de aprendizaje adicional el cual sin duda alguna será útil para el resto del curso.