SEMAFOROS

1. DEFINICIONES

A continuación se presentan diversas definiciones de semáforos.

1.1. SEMÁFORO GENERAL

Un semáforo general **S** es una *variable entera*, inicializada a un valor no negativo, y a la que, aparte de su inicialización, sólo se puede acceder por medio de dos operaciones atómicas ***P*** y ***V***.

P(**S**) ***if*** **S** > 0 ***then*** **S** = **S** – 1
else suspender la ejecución del proceso

V(**S**) ***if*** hay procesos suspendidos en este semáforo
then despertar a uno de ellos
else **S** = **S** + 1

S es una variable compartida por los procesos.
La operación ***V*** despierta a uno de los procesos suspendidos pero no indica a cual.
Estas operaciones se realizan de forma atómica y esto tiene sus implicaciones:

- No hay modificación ni acceso simultáneo al semáforo.
- No hay posibilidad de entrelazado de instrucciones entre el chequeo del valor de **S** y su posible modificación en una operación ***P*** o ***V***.
- ***P*** y ***V*** son secciones críticas; pues son trozos de código donde se accede a variables compartida

1.2. SEMÁFORO BINARIO

Es un semáforo general pero sólo puede tomar valores 0 y 1.
En este caso las operaciones P y V se redefinen de la siguiente forma:

P(**S**) ***if*** S > 0 then **S** = **S** - 1
else suspender la ejecución del proceso

V(**S**) ***if*** hay procesos suspendidos en este semáforo
then despertar a uno de ellos
else **S** = 1

1.3. SEMÁFORO CON COLA DE PROCESOS BLOQUEADOS

En la definición de semáforo general no se especifica en que orden se despiertan los procesos suspendidos, en este caso los procesos suspendidos se mantienen en una cola FIFO y se despiertan en el orden en que fueron suspendidos

P(**S**) ***if*** S > 0 then **S**=**S**-1
else suspender la ejecución del proceso. // *El proceso se inserta al final de la cola FIFO*

V(**S**) ***if*** hay procesos suspendidos en este semáforo
then despertar al que está al principio de la cola FIFO
else **S**=**S**+1

Se puede modificar la gestión utilizando una pila, un árbol, una cola con prioridades ... etc.

1.4. SEMÁFORO CON ESPERA ACTIVA (BUSY-WAIT)

El proceso que no completa la operación ***P*** (por no ser positivo el valor del semáforo en el momento de realizarla) no lo hace porque se mantiene en un bucle, de ahí que se denomine de espera activa.

```
P(S)    loop                    //  lazo
            if S > 0 then
                S=S-1;
                exit;           // sale del lazo
            end if;
        end loop;

V(S)    S=S+1;
```

Comentarios:

- Dado que un bucle no puede ser atómico, consideraremos que no hay entrelazado de instrucciones entre el chequeo del valor de S y posible decremento: if atómico. Posible entrelazado entre ciclos del lazo.
- Incremento atómico de S en V(S)

1.1. CONSECUENCIAS DE LAS ANTERIORES DEFINICIONES

Consecuencias de interés que se derivan

1) **S** >= 0 — Se deriva de la definición. Dado que el semáforo está inicializado a un valor no negativo y que la operación P solo disminuye su valor cuando éste es mayor que 0 la consecuencia es evidente.

2) $\mathbf{S} = \mathbf{S_0} + \mathbf{NV} - \mathbf{NP}$ — Donde:

NV es el número de operaciones **V** ejecutadas sobre **S**
NP es el número de operaciones **P** completadas sobre **S**
$\mathbf{S_0}$ es el valor inicial del semáforo **S**

Si no hay procesos suspendidos la demostración es trivial

Si hay procesos suspendidos el valor del semáforo será 0 y el número de procesos que han completado una operación **P** será el valor inicial del semáforo más el número de operaciones **V** realizadas.

2. USOS DE LOS SEMAFOROS

Veremos dos utilidades de los semáforos:

1) Solución al problema de la sección crítica
2) Medio de implementar cualquier grafo de precedencia mediante la sentencia concurrente

2.1. SOLUCION AL PROBLEMA DE LA SECCIÓN CRÍTICA

Las soluciones vistas hasta el momento (soluciones software) consistían en codificar adecuadamente las secciones de entrada y de salida en el código de cada proceso Pi

```
begin
   repeat
      Sección de entrada
      Sección crítica
      Sección de salida
      Sección restante
   until false
end
```

Requerimiento a las soluciones

- Exclusión mutua
- Progreso
- Espera limitada

Soluciones ya vistas:

Soluciones software

Para 2 procesos (Peterson) y n procesos (Lamport)

Soluciones con instrucciones hardware especiales

Para 2 procesos y n procesos (Burns)

Características de las soluciones vistas:

-Dificultades de conseguir soluciones que cumplan los tres requerimientos (secciones de entrada y salida de difícil programación)

-Espera activa

2.1.1. SOLUCIÓN CON SEMÁFOROS PARA 2 PROCESOS

Consideremos la siguiente solución con semáforos para el acceso a una sección crítica por dos procesos:

Variables compartidas

S=1 // Inicialización

Código de P_1	Código de P_2
begin	**begin**
repeat	**repeat**
P(S)	**P(S)**
Sección crítica de P_1	Sección crítica de P_2
V(S)	**V(S)**
Sección restante de P_1	Sección restante de P_2
until false	**until false**
end	**end**

Comprobación de la satisfacción de las tres condiciones:

Exclusión mutua

No puede haber más de un proceso en la sección crítica
Sea **#SC** número de procesos en la sección crítica.
#SC=NP-NV ya que por construcción de la solución estarán en la sección crítica los procesos que hayan completado la operación P pero no hayan hecho todavía la operación V
Recordando que para todo semáforo tenemos que **S** = $\mathbf{S_0}$ + **NV** – **NP**
Llegamos a que **#SC=$\mathbf{S_0}$-S**
y como el semáforo esta inicializado a 1
#SC=1-S,
Como **S>=0**
Llegamos a que **#SC<=**1 lo que demuestra que hay exclusión mutua (el número de procesos en la sección crítica es siempre menor o igual que 1

Progreso

La opción de entrar en la sección crítica solo la tiene uno de los que solicita entrar y además es imposible el interbloqueo (ambos procesos suspendidos en **P(S**) y ninguno en la sección crítica) ya que tendriamos:

S = 0	Por haber procesos suspendidos y
#SC = 0	Ya que ninguno está en la sección crítica
S = 0 y #SC = 0	Imposible por (e3)

Espera limitada

Supongamos que P_1 quiere entrar en la sección crítica y P_2 está en la sección crítica. Veremos que es imposible que P_2 entre de nuevo hasta que lo haya hecho P_1:
Si P_1 está suspendido, S = 0 y P_2 en su sección crítica. Cuando P_2 sale, ejecuta **V(S)** y despierta el proceso suspendido (dejando S=0). Despierta P_1 y P_1 entra, P_2 no podrá entrar hasta que lo haya hecho P_1
Vemos que la condición de espera limitada no se satisfaría si el semáforo fuese con espera activa, ya que sería posible que P_2 saliese de su sección restante ejecutase V(S) (dejando S=1) y volviese a entrar

2.1.2. SOLUCIÓN CON SEMÁFOROS PARA *N* PROCESOS

La forma de la solución se generaliza para N procesos

S = 1 // Inicialización

Código de P_i

```
begin
   repeat
      P(S)
      Sección crítica de Pi
      V(S)
      Sección restante de Pi
   until false
end
```

- Satisface las exigencias de exclusión mutua y progreso
- La satisfacción de espera limitada depende del tipo de semáforo
- Para un semáforo con espera activa no se satisface el requerimiento de espera limitada

  Se demuestra viendo que es posible la postergación indefinida de un proceso. Considérese la siguiente secuencia de ejecución:
  1. P_1 ejecuta P(S) y entra en su sección crítica
  2. P_2 encuentra S=0 y se queda en el lazo de la operación P (el semáforo es de espera activa).
  3. P_1 completa un ciclo (protocolo de salida, sección restante, protocolo de entrada) y vuelve a entrar en la sección crítica
  4. P_2 encuentra S=0 y se queda en el lazo

  Esta secuencia puede repetirse indefinidamente causando la postergación de P_2.
- Para un semáforo con cola de procesos bloqueados se satisface el requerimiento de espera limitada

  Suponer P1 bloqueado en S. Habrá como máximo N-1 procesos delante de P1 en la cola d S. P1 entrará en su sección crítica después de un máximo de N-1 ejecuciones de P(S)
- Para un semáforo con conjunto de procesos bloqueados no se satisface el requerimiento de espera limitada con N>=3

  Se demuestra viendo que es posible la postergación indefinida de un proceso. Ya se ha demostrado que no es posible para el caso N=2.

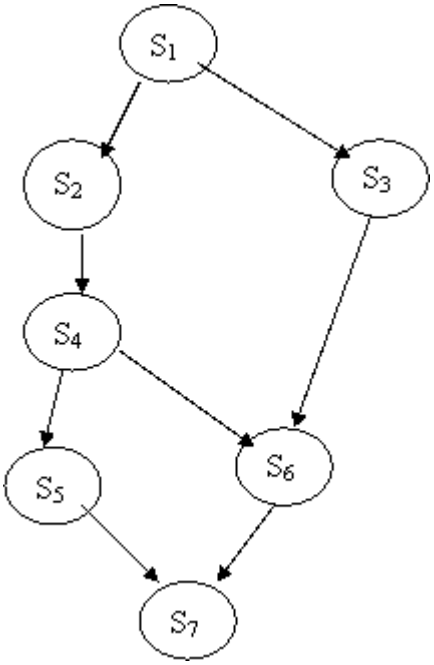
  Con N>=3 es posible que haya siempre 2 procesos bloqueados en S

  V(S) despierta a un proceso arbitrario

  Es posible que se ignore siempre a un proceso determinado, provocando para él una situación de postergación indefinida.

2.2. IMPLEMENTACIÓN DE UN GRAFO DE PRECEDENCIA CON SEMÁFOROS

Considérese el siguiente grafo de precedencia:



Una solución con semáforos, y no la única, podría ser la siguiente:

a, b, c, d, e, f, g: semáforos inicializados a 0

begin
    parbegin
        begin S_1; **V**(a); **V**(b) **end**;
        begin P(a); S_2; S_4; **V**(c); V(d) **end**;
        begin P(b); S_3; **V**(e) **end**;
        begin P(c); S_5; **V**(f) **end**;
        begin P(d); **P**(e); S_6; **V**(g) **end**;
        begin P(f); **P**(g); S_7 **end**;
    parend;
end;

Este método nos permite resolver el problema del enlace de S_3 en S_6.

3. IMPLEMENTACIÓN DE LOS SEMÁFOROS

Un semáforo es un concepto. En esta sección veremos las diferentes formas de construir un semáforo.

3.1. IMPLEMENTACIONES DE SEMÁFOROS CON ESPERA ACTIVA

Espera activa es aquella situación en la que el proceso espera su continuidad dentro del semáforo. Tiene el problema de consumir ciclos tontos del procesador.

3.1.1. IMPLEMENTACIÓN DEL SEMÁFORO BINARIO

Para la implementación nos apoyaremos en el uso de la instrucción Test&Set. Recordemos su funcionamiento:

```
function TS (var X: Boolean): Boolean;
  begin
     TS= X;
     X=true;
  end;
```

Con esta función una posible implementación del semáforo sería

TYPE Semaforo_binario:boolean; //el valor 1 del semáforo corresponde al valor FALSE

```
Procedure Pb(var Sb: semaforo_binario);
  begin
        while  TS(Sb) do;
  end;

Procedure Vb (Var Sb:semaforo_binario)
  begin
        Sb=false;
  end;
```

También admite la forma que utilizaremos a partir de ahora:

```
function TS (var X: Boolean): Boolean;
```

```
begin
    TS= X;
    X=false;
end;
```

```
TYPE  semaforo_binario:boolean; /*el valor 1 del semáforo corresponde al valor TRUE*/
```

```
Procedure Pb(var Sb: semaforo_binario);
  begin
        while not TS(Sb) do;
  end;

Procedure Vb (Var Sb:semaforo_binario)
  begin
        Sb=true;
  end;
```

3.1.2. APROXIMACIONES A LA IMPLEMENTACIÓN DEL SEMÁFORO GENERAL

Consideremos las definiciones de semáforo general

P(**S**) ***if*** S > 0 **then S** = **S** – 1
else suspender la ejecución del proceso

V(**S**) ***if*** hay procesos suspendidos en este semáforo
then despertar a uno de ellos
else **S** = **S** + 1

Vamos a modificarla de la siguiente manera

P(**S**) **S = S - 1**
if (S >= 0) **then**
else suspender la ejecución del proceso

V(**S**) **S = S + 1**
if hay procesos suspendidos en este semáforo /* S <=0*/
then despertar a uno de ellos

Que equivale a

P(**S**) **S = S - 1**
if (S < 0) **then** suspender la ejecución del proceso

V(**S**) **S = S + 1**
if (S <= 0) ***then*** despertar a uno de ellos

- Restamos primero del valor del semáforo, permitiendo que este tome valores negativos, el comportamiento es el mismo y no afecta a los procesos pues éstos **nunca** conocen el valor del semáforo, solo acceden a él a través de **P** y **V**. Además, el valor negativo del semáforo nos sirve así para determinar si hay procesos suspendidos en el semáforo (que no han completado la operación **P)**
- Para suspender la ejecución de un proceso utilizamos un semáforo binario auxiliar *delay* inicializado a 0 (la implementación de semáforos binarios ya se ha visto)

P(**S**) **S = S - 1**
if (S < 0) **then Pb**(delay)

V(**S**) **S = S + 1**
if (S <= 0) ***then*** **Vb**(delay)

Finalmente solo falta conseguir que las operaciones sean (o se comporten como) atómicas, lo que conseguiremos con otro semáforo binario auxiliar *mutex* (inicializado a 1) para conseguir la exclusión mutua.

P(**S**) **Pb** (mutex);
S = S - 1

```
                    if  (S < 0) then Pb(delay)
                    Vb (mutex);

V(S)                Pb (mutex);
                    S = S + 1
                    if  (S <= 0) then Vb(delay)
                    Vb (mutex);
```

Esta solución aparentemente correcta no es valida pues provocaría un interbloqueo: Un proceso comienza ejecutando una P(S), se encuentra con S<0 y se queda esperando en Pb(delay), como no libera mutex impide que otro proceso entre a hacer una V(S) y por lo tanto él tampoco puede salir de la Pb(delay).
Un paso más consiste en liberar mutex una vez que se ha accedido a las variables del semáforo.
La implementación quedaría por tanto

```
type semaforo=record
                integer: valor;                 /* el valor del semáforo   */
                semaforo_binario: mutex, delay;     /* mutex inicializado a 1   */
        end;                                        /* delay inicializado a 0    */


procedure P (var s: semaforo)
begin
        Pb (s.mutex);
        s.valor = s.valor –1;
        if (s.valor <0) then
                begin
                        Vb(s.mutex);
                        Pb(s.delay);
                end
        else
                Vb (s.mutex);
end;


procedure V (var s:semaforo)
begin
        Pb(s.mutex);
        s.valor = s.valor +1;
        if (s.valor <=0) then
                Vb (s.delay);
        Vb(s.mutex);
end;
```

§ s.mutex se utiliza para la exclusión mutua en el acceso al valor de semáforo
§ en s.delay esperan (de manera activa) los procesos que no pueden completar la operación P.

La implementación, aunque a simple vista también parece correcta sigue sin ser valida. Supongamos el siguiente caso:
El valor del semáforo es 0, dos procesos A y B hacen una operación P y dos procesos C y D hacen una operación V. Si la implementación fuese correcta, al hacer los cuatro procesos su operaciones, los procesos A y B habrían completado sus operaciones P y el valor del semáforo sería 0, independientemente de cómo se entrelazasen las instrucciones de los procesos. Supongamos ahora la siguiente secuencia de sucesos:

1. proceso A: inicia la operación P, le es apropiada la CPU justo después de haber ejecutado la instrucción (1) antes de hacer Pb(s.delay)
2. proceso B: inicia la operación P, le es apropiada la CPU justo después de haber ejecutado la instrucción (1) antes de hacer Pb(s.delay). En este momento el valor del semáforo es –2 y s.delay vale 0
3. proceso C: realiza la operación V. En este momento el valor del semáforo es –1 y s.delay vale 1
4. proceso D: realiza la operación V. En este momento el valor del semáforo es 0 y s.delay vale 1
5. proceso A: reanuda la ejecución. Completa su operación P. En este momentoel valor del semáforo es 0 y s.delay vale 0
6. procesoB: reanuda la ejecución: No puede completar la operación P pues s.delay vale 0

El fallo de la implementación se manifiesta porque s.delay es un semáforo binario y hacer varias operaciones Vb seguidas sobre él (es decir, varias operaciones V sobre el semáforo general cuando hay procesos que no han completado la operación P) es lo mismo que hacer una. Para solucionarlo:

- s.delay no es binario, es un semáforo general.
- Se impide que se hagan dos operaciones Vb seguidas sobre s.delay, es decir, impedimos que se hagan dos operaciones V sobre el semáforo general si hay algún proceso suspendido: cuando hay procesos pendientes de completar una operación P, cada operación V ha de ir seguida de la finalización de una operación P pendiente (implementación de Hemmendiger).
- Usamos un contador del número de operacions Vb que se han hecho sobre el semáforo binario, y cada proceso que completa una P realiza, si es necesario, una operación Vb adicional sobre s.delay (implementación de Kearns).

3.1.3. IMPLEMENTACIÓN DE HEMMENDINGER

Variables compartidas por todos los procesos

```
type
        semáforo = record
        integer: valor;         /* valor del semáforo*/
        semaforo_binario: mutex, delay; /* val_ini 1 y 0 */
   end;



procedure P(var s:semaforo)
begin
        Pb(s.mutex);
        s.valor = s.valor - 1;
        if (s.valor < 0) then
                begin
                        Vb(s.mutex);
                        Pb(s.delay);
                end;
        Vb(s.mutex);
end;


procedure V (var s:semáforo)
begin
        Pb (s.mutex);
        s.valor = s.valor + 1;
        if (s.valor <= 0) then
                Vb(s.delay);
        else
                Vb(s.mutex);
end;
```

Si el valor del semáforo es negativo (hay procesos suspendidos en s.delay), no se permite una secuencia de operaciones V. Cada operación V tiene que ir seguida de la finalización de una operación P suspendida hasta que el valor del semáforo sea no negativo (no se libera la exclusión mutua al realizar la operación V en este caso).

3.1.4. IMPLEMENTACIÓN DE KEARNS

Variables compartidas por todos los procesos

```
type
    semáforo = record
      mutex , delay:  semáforo_binario
      valor, contador_delay: integer
   end;
 /* val_ini delay=1, mutex=1, contador_delay=0*/
```

```
procedure P(var s: semáforo)
begin
   Pb(s.mutex)
   s.valor = s.valor -1;
   if s.valor < 0 then
      begin
         Vb(s.mutex);
         Pb(s.delay);
         Pb(s.mutex);
         s.contador_delay = s.contador_delay -1;
         if s.contador_delay > 0 then Vb(s.delay)
      end;
   Vb(s.mutex)
end;
```

```
procedure V(var s: semáforo
V(var s: semáforo)
begin
   Pb(s.mutex)
   s.valor = s.valor + 1;
   if s.valor <= 0 then
      begin
         s.contador_delay = s.contador_delay + 1;
         Vb(s.delay)
      end;
   Vb(s.mutex)
end;
```

3.1.5. IMPLEMENTACIÓN MEDIANTE INHABILITACIÓN DE INTERRUPCIONES

```
Variables compartidas
type
    semáforo = record
        valor: integer
   end;


procedure P(var s: semáforo)
begín
  Inhabilitar_interrupciones;
  while (s.valor = 0) do
       begin
          Habilitar_interrupciones;
          Inhabilitar_interrupciones ;
       end;
  s.valor = s.valor - 1;
end;



procedure V(var S: semáforo)
begin
  Inhabilitar_interrupciones ;
  s.valor = s.valor + 1;
  Habilitar_interrupciones;
end;
```

3.2. IMPLEMENTACIÓN CON ESPERA INACTIVA

La espera activa presenta un consumo de ciclos de CPU de los procesos bloqueados (que no completan una operación P).

Variables compartidas

```
type
    semáforo = record
        valor: integer;
        L: list of identificadores de proceso,
    end;


procedure P(var s:semáforo)
begin
    if s.valor > 0 then
        s.valor = s.valor -1;
    else
        begin
            añadir el proceso a s.L.;
            bloquear;
        end;
end;



procedure V(var s:semáforo)
begin
    if ListaVacia(s.L.)
        s.valor = s.valor + 1;
    else
        begin
            sacar identificador de procesos de s.L.;
            despertar (proceso);
        end;
end;
```

Partiendo de la modificación de la definición de semáforo, donde el semáforo toma valores negativos, que indican el número de procesos suspendidos, podemos tener la siguiente implementación:

Variables compartidas

```
type
    semáforo = record
        valor: integer;
        L: list of identificadores de proceso,
    end;


procedure P(var s:semáforo)
begin
    s.valor = s.valor -1;
    if s.valor < 0 then
        begin
            añadir el proceso a s.L.;
            bloquear;
        end;
end;



procedure V(var s:semáforo)
begin
    s.valor = s.valor + 1;
    if s.valor <= 0 then
        begin
            sacar un proceso de s.L.;
            despertar (Proceso);
        end;
end;
```

Comentarios a la implementación con espera inactiva

- La implementación del semáforo requiere
  - Una variable entera
  - Lista de identificadores de procesos.
  - La lista puede emplear cualquier estrategia de colas:
    FIFO, LIFO, por prioridades, aleatoria.
    añadir y *sacar* implementan esta estrategia.

- ***Bloquear*** suspende el proceso que la invoca. Pasa a espera.
  - El planificador de la CPU elija un proceso la Lista de Procesos Listos.
  - El proceso suspendido no consume ciclos de CPU.

- ***Despertar*** (P) reanuda la ejecución de un proceso P bloqueado (en espera). El proceso pasaría a la cola de procesos listos.

- En esta implementación no se ha garantizado que P y V sean atómicas. P y V son secciones críticas:
  - Necesidad de protección
  - Necesidad de código eficiente y libre de errores

- Protección de P(S) y V(S) con cierres hardware o soluciones software
  - Introduce espera activa en la entrada de las secciones críticas P(S) y V(S)
    - Secciones críticas breves (<= 10 instrucciones)
    - Secciones críticas generalmente vacías
    - Espera activa de muy breves periodos
  - Elimina la espera activa en la entrada de las secciones críticas de las aplicaciones
    - Secciones críticas largas
    - Secciones críticas generalmente ocupadas
    - La espera activa sería muy ineficiente

La implementación completa podria quedar como sigue (utilizando algoritmo de Lamport, aunque sería análogo utilizando otro algoritmo distinto)

```
type
   semáforo = record
         valor: integer;
         L: list of identificador proceso;
end;


Procedure P(s)
begin
   Seccion de entrada algoritmo de Lamport
   s.valor = s.valor -1;
   if s.valor < 0 then
      begin
         añadir el proceso a s.L.;
         bloquear;
      end;
  Sección de salida algoritmo de Lamport
end;



procedure V(s)
begin
   Seccion de entrada algoritmo de Lamport
   s.valor = s.valor + 1;
   if s.valor <= 0 then
      begin
         sacar identificador de  proceso de s.L.;
         despertar (Proceso);
      end;
     Seccion de salida algoritmo de Lamport
end;
```