

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.1 EXCLUSION MUTUA

DEFINICIÓN

La forma mas sencilla de comunicación entre los procesos de un programa concurrente es el uso común de variables de datos.

Esta forma tan sencilla de comunicación puede llevar a errores en el programa. El acceso concurrente puede hacer que la acción de un proceso interfiera en las acciones de otro, de forma no adecuada.

Ejemplo

Problema de los jardines.

Se desea controlar el numero de visitantes a unos jardines. La entrada y la salida se puede realizar por dos puntos. Se dispone de un computador con conexión en cada uno de los puntos, asociamos el proceso P1 a un punto de entrada y el proceso P2 al otro. Ambos procesos se ejecutan de forma concurrente y utilizan una única variable X para llevar la cuenta de visitantes. Cuando entra un visitante se ejecuta $X := X + 1$, y cuando sale, $X := X - 1$.

La actualización de la variable se lleva a cabo por la ejecución de instrucciones mas sencillas.

- 1) Copia el valor de X en un registro del procesador.
- 2) Incrementa / Decrementa el valor del registro.
- 3) Almacena el resultado en la dirección donde se guarda X.

En la ejecución concurrente de los procesos, el procesador entrelaza la ejecución.

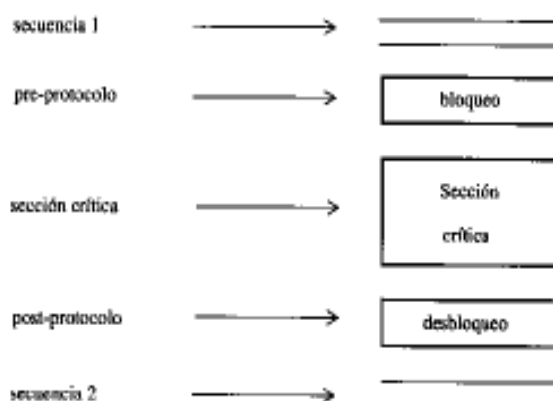
- 1) P1 carga el valor de X, en un registro.
- 2) P2 carga el valor de X, en un registro.
- 3) P2 incrementa el valor de su registro.
- 4) P1 incrementa el valor de su registro.
- 5) P1 guarda el valor de su registro en la dirección de memoria de X.
- 6) P2 guarda el valor de su registro en la dirección de memoria de X.

Se puede observar que se pierde un incremento de la variable X.

Solución: Identificar aquellas regiones de los procesos que acceden a variables compartidas y dotarlas de la posibilidad de ejecución como si fueran una única instrucción.

SECCION CRÍTICA. Partes de los procesos concurrentes, que no pueden ejecutarse de forma concurrente. Desde otros procesos se deben ver como si fueran una única instrucción.

Las secciones críticas se pueden agrupar en clases, siendo **mutuamente exclusivas** las secciones críticas de cada una. Para conseguir dicha exclusión se implementan protocolos software que bloqueen el acceso a una sección crítica mientras esta siendo usada por un proceso.



TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.1.2 BLOQUEO MEDIANTE EL USO DE VARIABLES COMPARTIDAS

Se asocia a cada recurso que se comparte un indicador (flag, variable compartida de tipo booleano). Antes de acceder al recurso, el proceso debe consultar el estado del indicador, si este es verdadero podrá acceder y si es falso no podrá.

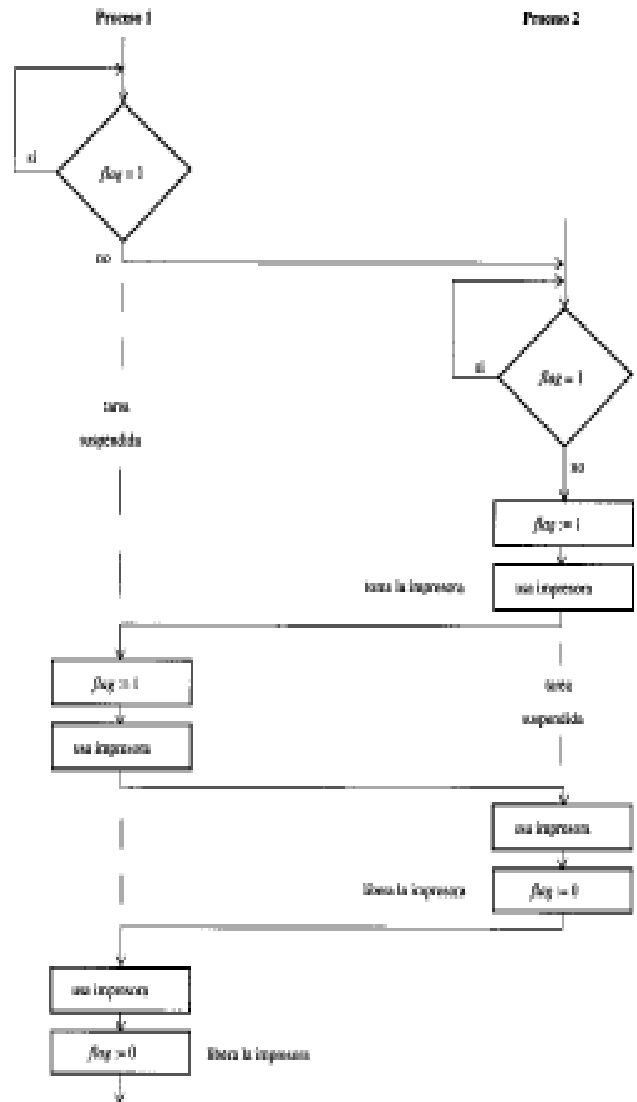
```
/*Exclusión Mutua :Uso de un indicador*/  
program/module Exclusión_Mutua_1;
```

```
var flag:boolean;
```

```
process P1;  
begin  
  loop  
    while flag = true do  
      /*Espera a que el recurso se libere*/  
    end;  
    flag := true;  
    /*Uso de la sección crítica*/  
    flag := false;  
    /*Resto del proceso*/  
  end;  
end;
```

```
process P2;  
begin  
  loop  
    while flag = true do  
      /*Espera a que el recurso se libere*/  
    end;  
    flag := true;  
    /*Uso de la sección crítica*/  
    flag := false;  
    /*Resto del proceso*/  
  end;  
end;
```

```
begin /*Exclusion_Mutua_1*/  
  flag := false;  
  cobegin  
    P1;  
    P2;  
  coend;  
end Exclusion_Mutua_1;
```



No resuelve el problema de la exclusión mutua. Al ser la comprobación y la puesta del indicador a falso operaciones separadas, puede que se entrelace el uso del recurso por ambos procesos.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

Intentamos solucionar el problema usando dos indicadores, asociamos un indicador a cada uno de los procesos. Ambos procesos pueden leer los dos indicadores, pero solo pueden modificar el que tienen asociado. Evita el problema de que los dos procesos actualicen de forma simultánea el mismo indicador.

```
/* Exclusión Mutua: Uso de dos indicadores */
program/module Exclusión_Mutua_2;

var flag1,flag2:boolean;

procedure bloqueo(var mi_flag,su_flag:boolean);
begin
    mi_flag := true;    /*Señala la intencion de usar el recurso */
    while su_flag do;end;    /* Espera a que se libere el recurso */
end bloqueo;

procedure desbloqueo(var mi_flag:boolean);
begin
    mi_flag := false;
end desbloqueo;

process P1;
begin
    loop
        bloqueo(flag1,flag2);
        /* Acceso a sección crítica */
        desbloqueo(flag1);
        /* Resto del proceso */
    end;
end P1;

process P2;
begin
    loop
        bloqueo(flag2,flag1);
        /* Acceso a sección crítica */
        desbloqueo(flag2);
        /* Resto del proceso */
    end;
end P1;

begin    /* Exclusion_Mutua_2 */
    flag1 := False;
    flag2 := False;
    cobegin
        P1;
        P2;
    coend;
end Exclusion_Mutua_2;
```

Inconvenientes :

Durante la espera de liberación del recurso el proceso permanece ocupado.

Espera Activa

Si ambos procesos llaman al bloqueo de forma simultánea, cada proceso puede establecer su propio indicador y comprobar el estado del otro. Ambos verán los indicadores contrarios como ocupados y permanecerán a la espera de que el recurso se libere, pero esto no sucederá, al no poder entrar ninguno en su sección crítica.

Interbloqueo (deadlock)

El interbloqueo se produce porque la desactivación del indicador asociado a un proceso se produce cuando se ha completado el acceso a la sección crítica.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

Una posible solución puede ser haciendo que el proceso desactive su propio indicador durante la fase de bloqueo, siempre que encuentre el indicador del otro proceso activado.

```
procedure bloqueo( var mi_flag, su_flag:boolean );
begin
  mi_flag := true;
  while su_flag do
    mi_flag := false;
    mi_flag := true;
  end;
end bloqueo;
```

Permite que en caso de interbloqueo se pueda proseguir siempre que no se produzca una completa sincronización entre los dos procesos.

Otro problema: puede permitir que un proceso deje su sección crítica y vuelva a entrar mientras que el otro proceso desactiva su indicador en la sección de bloqueo.

El que un proceso no pueda progresar porque se lo impida otro proceso o grupo de procesos se denomina **cierre (lockout o starvation)**.

Dos soluciones al problema de la exclusión mutua que evitan el interbloqueo y el cierre.

- Algoritmo de Peterson
- Algoritmo de Dekker

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.1.3 ALGORITMO DE PETERSON

Se introduce una variable adicional (turno), que solo resulta útil cuando se produzca un problema de petición simultánea de acceso a la sección crítica.

```
/* Exclusión Mutua: Solución Peterson */
program/module Exclusión_Mutua_P;

var flag1,flag2:boolean;
    turno:integer;

procedure bloqueo(var mi_flag,su_flag:boolean; su_turno:integer);
begin
    mi_flag := true;
    turno := su_turno;
    while su_flag and (turno = su_turno) do; end;
end bloqueo;

procedure desbloqueo(var mi_flag:boolean);
begin
    mi_flag := false;
end desbloqueo;

process P1
begin
    loop
        bloqueo(flag1,flag2,2);
        /* Uso del recurso sección crítica */
        desbloqueo(flag1);
        /* resto del proceso */
    end;
end P1;

process P2
begin
    loop
        bloqueo(flag2,flag1,1);
        /* Uso del recurso sección crítica */
        desbloqueo(flag2);
        /* resto del proceso */
    end;
end P2;

begin /* Exclusion_Mutua_P */
    flag1 := false;
    flag2 := false;
    cobegin
        P1;
        P2;
    coend;
end Exclusion_Mutua_P;
```

El algoritmo resuelve el problema de la exclusión mutua y garantiza que ambos procesos usarán de forma consecutiva el recurso en caso de que lo soliciten a la vez y se **impedirá el cierre del otro proceso**.

Ambos procesos gozan de la misma prioridad en la utilización del recurso.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.1.4 ALGORITMO DE DEKKER

La variable turno, ahora sirve para establecer la prioridad relativa de los procesos y su actualización se realiza en la sección crítica, lo que evita que pueda haber interferencias entre los procesos.

```
/* Exclusión Mutua: Solucion de Dekker */
program/module Exclusión_mutua_D;

var flag1, flag2:boolean;
    turno:integer;

procedure bloqueo(var mi_flag,su_flag:boolean; su_turno:integer);
begin
    mi_flag := true;
    while su_flag do          /*Otro proceso en la sección crítica */
        if turno = su_turno then
            mi_flag := false;
            while turno = su_turno do
                /* espera a que termine el otro proceso */
            end;
            mi_flag := true;
        end;
    end;
end bloqueo;

procedure desbloqueo(var mi_flag:boolean; su_turno:integer);
begin
    turno := su_turno;
    mi_flag := false;
end desbloqueo;

process P1
begin
    loop
        bloqueo(flag1,flag2,2);
        /* Uso del recurso. Sección crítica */
        desbloqueo(flag1,2);
        /* Resto del proceso */
    end P1;

process P2
begin
    loop
        bloqueo(flag2,flag1,1);
        /* Uso del recurso. Sección crítica */
        desbloqueo(flag2,1);
        /* Resto del proceso */
    end P1;

begin /*Exclusión Mutua_D */
    flag1 := false; flag2 := false;
    turno := 1;
    cobegin
        P1;
        P2;
    coend;
end Exclusion_Mutua_D;
```

El programa se inicia con turno = 1, lo que da prioridad al proceso P1.

Si ambos procesos piden a la vez acceso a la sección crítica, activan sus respectivos indicadores y comprueban el indicador del otro. Ambos encuentran que si.

Pasan a comprobar el turno.

P2 comprueba que no es su turno, desactiva su indicador y queda a la espera P1 comprueba que es su turno y evalúa el indicador de P2.

P1 entra en la sección crítica, y dará el turno a P2 antes de desactivar su indicador.

Esto permite que P2 entre en la sección crítica aunque P1 haga una nueva solicitud de acceso a la sección crítica.

Los algoritmos de Peterson y Dekker se pueden extender al caso más general en que haya n procesos en ejecución concurrente

No son las soluciones adecuadas, ya que la espera por el recurso siempre se realiza de forma ocupada (espera activa).

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.2 SEMÁFOROS

DEFINICIÓN

Permite resolver la mayoría de los problemas de sincronización entre procesos.

Semáforo binario, es un indicador de condición (S) que registra si un recurso esta disponible o no.

Solo puede tomar dos valores :

$S = 1 \rightarrow$ Recurso disponible.

$S = 0 \rightarrow$ Recurso NO disponible.

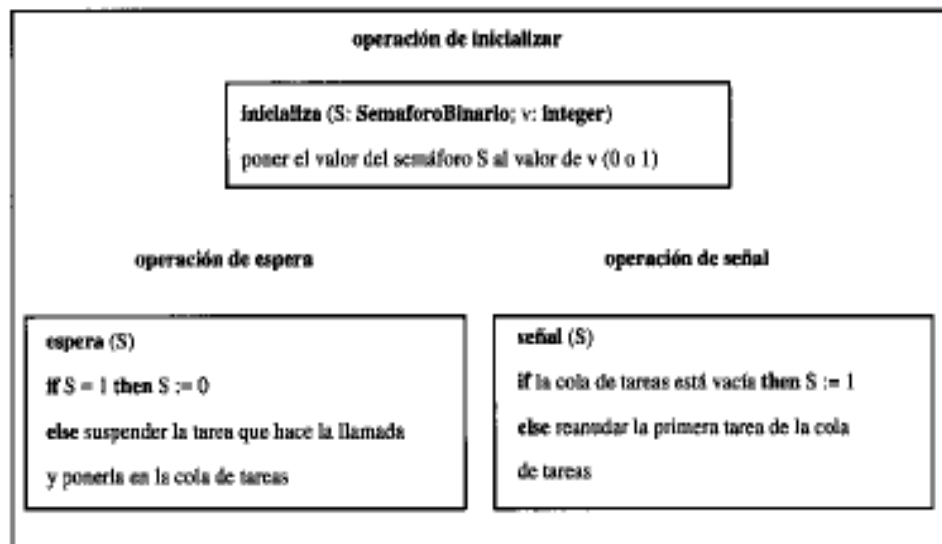
Los semáforos se implementan con una cola de tareas, a la cual se añaden los procesos que están a la espera del recurso.

Sólo se permiten tres operaciones sobre un semáforo :

- 1.- Inicializa.
- 2.- Espera (wait)
- 3.- Señal (wait)

Un semáforo binario se puede definir como un tipo de datos especial que sólo puede tomar los valores 0 y 1, con una cola de tareas asociada y con sólo tres operaciones para actuar sobre él.

Operaciones Primitivas



Las operaciones primitivas son operaciones que se implementan como acciones indivisibles, la comprobación y cambio de valor del indicador se efectúa realmente como una sola operación.

Inicializa. Se debe ejecutar antes de comenzar la ejecución concurrente. Su función exclusiva es dar un valor inicial al semáforo.

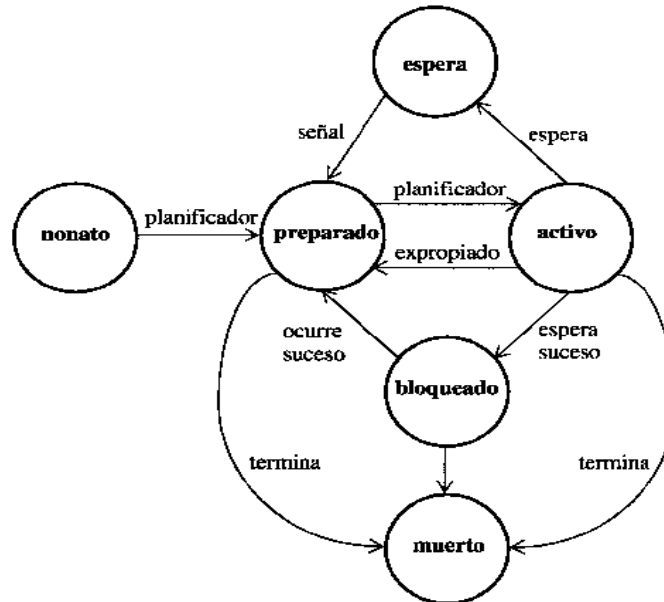
Espera. Si un proceso ejecuta espera y encuentra el semáforo a 1, lo pone a 0 y continua su ejecución. Si el semáforo está a 0 el proceso queda en estado de espera hasta que se libera el semáforo.

Señal. Al ejecutar señal puede haber varios procesos en la cola, el proceso que la dejará para pasar al estado preparado depende del esquema de gestión de la cola adoptado.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

El diagrama de transición de estados se puede ampliar con un nuevo estado que denominaremos de espera.



3.2.1 EXCLUSIÓN MUTUA CON SEMÁFOROS

Se realiza fácilmente utilizando semáforos.

La operación **Espera** se usará como procedimiento de bloqueo.

La operación **Señal** como procedimiento de desbloqueo.

Se emplearán tantos semáforos como clases de secciones críticas se establezcan.

```
process P1;
begin
  loop
    espera(S);
    /* Sección crítica */
    señal(S);
    /* Resto del proceso */
  end P1;
```

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.2.2 EJEMPLO

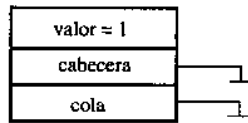
Esquema de comportamiento de una cola ligada a un semáforo utilizado para compartir un recurso entre tres procesos concurrentes (P1,P2,P3).

La cola del ejemplo usa estrategia FIFO.

El nombre del semáforo es AccesoImpresora.

1) Inicializa(AccesoImpresora,1)

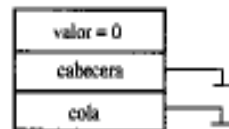
AccesoImpresora



El semáforo se inicializa a 1, dejando libre el acceso a la impresora.

2) Proceso P1 ejecuta: Espera(AccesoImpresora)

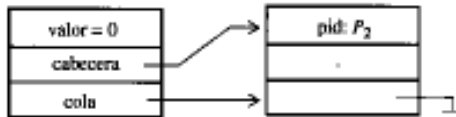
AccesoImpresora



P1, toma la impresora y pone el semáforo a 0, impidiendo el acceso de otro proceso

3) Proceso P1 bloqueado, P2 ejecuta: Espera(AccesoImpresora)

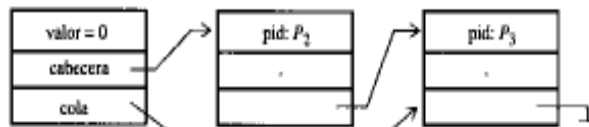
AccesoImpresora



P2 intenta acceder al recurso y se suspende en espera de que quede libre. Colocándose en la cola del semáforo.

4) Proceso P3 ejecuta: Espera(AccesoImpresora)

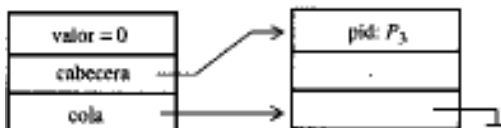
AccesoImpresora



P3 se suspende y se añade a la cola.

5) Proceso P1 ejecuta: Señal(AccesoImpresora)

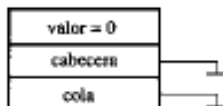
AccesoImpresora



P1 deja el recurso.
P2 pasa al estado preparado y tomará el recurso cuando entre en ejecución.

6) Proceso P2 ejecuta: Señal(AccesoImpresora)

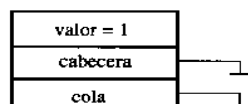
AccesoImpresora



P2 deja el recurso.
P3 pasa al estado preparado.

7) Proceso P3 ejecuta: Señal(AccesoImpresora)

AccesoImpresora



P3 deja el recurso.
La impresora esta disponible cualquier recurso.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.2.3 SINCRONIZACIÓN

El uso de semáforos hace que se pueda programar fácilmente la sincronización entre dos procesos.

Las operaciones de **espera** y **señal** no se usan dentro del mismo proceso.

El proceso que ejecuta **espera** queda bloqueado hasta que el otro proceso ejecuta **señal**.

Es común emplear la palabra **señal** para denominar un semáforo que se usa para sincronizar dos procesos. **señal** tendrá dos operaciones, **espera** y **señal**.

3.2.4. EJEMPLO

Sincronización entre dos procesos utilizando un semáforo.

```
/* Sincronización con semáforo */
program/module Sincronizacion;
var sincro:semaforo;

process P1      /*Proceso que espera */
begin
    .....
    espera(sincro);
    .....
end P1;

process P2      /*Proceso que señala */
begin
    .....
    señal(sincro);
    .....
end P2;

begin /*Sincronización */
    inicializa(sincro,0);
    cobegin
        P1;
        P2;
    coend
end Sincronizacion;
```

El semáforo se inicializa a cero.

Cuando P1 ejecuta espera, se suspende hasta que P2 ejecuta señal.

La sincronización se produce perfectamente incluso si P2 ejecuta señal antes que P1 ejecute espera, ya que P2 incrementaría el semáforo y permitiría que P1 lo decremente y siga su ejecución cuando alcanza la operación espera.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.2.5 EJEMPLO

Problema del productor-consumidor, caracteriza aquellas situaciones en las que existe un conjunto de procesos que producen información que otros procesos consumen, siendo diferentes las velocidades de producción y consumo. Hace falta que se establezca una sincronización entre los procesos de manera que la información no se pierda ni se duplique.

Dos procesos P1 y P2:

- 1) P1 produce datos que consume P2.
- 2) P1 almacena datos en algún sitio hasta que P2 esta listo para usarlos.
- 3) Para almacenar datos se dispone de un búfer común al productor y al consumidor.

Intento de solución SIN semáforos:

```
/* Productor-Consumidor: Solución sin semáforos */  
program/modulo Productor_Consumidor;
```

```
var BufferComun:búffer;
```

```
process Productor;  
var x:dato;  
begin  
  loop  
    produce(x);  
    while Lleno do  
      /*Espera */  
    end;  
    Poner(x);  
  end;  
end Productor;
```

```
process Consumidor;  
var x:dato;  
begin  
  loop  
    while Vacio do  
      /*Espera*/  
    end;  
    Tomar(x);  
    Consume(x);  
  end;  
end Cosnumidor;
```

```
begin  
  cobegin  
    Productor;  
    Consumidor;  
  coend;  
end Productor_Consumidor;
```

La solución no es satisfactoria por:

a.- Las funciones Poner(x) y Tomar(x) utilizan el mismo buffer, lo cual plantea el problema de la exclusión mutua.

b.- Ambos procesos utilizan una espera activa.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

Solución utilizando semáforos:

Autor: agvuned@gmail.com

El acceso al búfer esta protegido por las operaciones espera y señal del semáforo AccesoBuffer.

Antes de ejecutar las operaciones espera(NoLleno)

/*Productor-Consumidor: Solución CON semáforos */

program/module Productor_Consumidor;

var

BufferComun:buffer;

AccesoBuffer,NoLleno,NoVacio:semaforo;

process Productor;

var x_dato;

begin

loop

produce(x);

espera(AccesoBuffer);

if Lleno then

señal(AccesoBuffer);

espera(NoLleno);

espera(AccesoBuffer);

end;

Poner(x);

señal(AccesoBuffer);

señal(NoVacio);

end;

end Productor;

process Consumidor;

var x:dato;

begin

loop

espera(AccesoBuffer);

If Vacio then

señal(AccesoBuffer);

espera(NoVacio);

espera(AccesoBuffer);

end;

Tomar(x);

señal(AccesoBuffer);

señal(NoLleno);

Consume(x);

end;

end;

end Consumidor;

begin

inicializa(AccesoBuffer,1);

inicializa(NoLleno,1);

inicializa(NoVacio,0);

cobegin

Productor;

Consumidor;

coend;

end Productor_Consumidor;

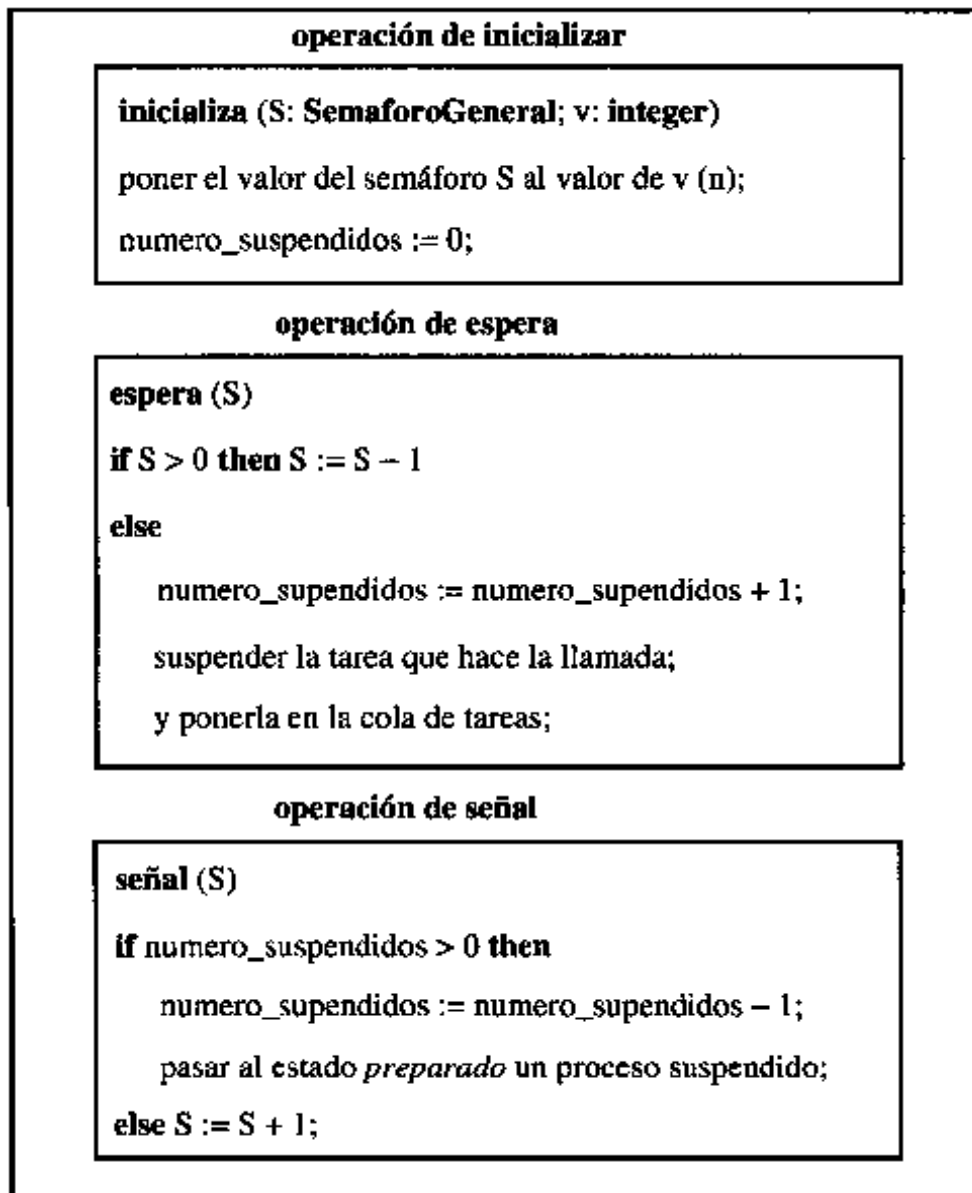
TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.2.6 VERSIÓN MÁS GENERAL DE LOS SEMÁFOROS

El semáforo binario resulta adecuado cuando hay que compartir un recurso que pueden compartir varios procesos.

Cuando hay que proteger un conjunto de recursos similares, se puede usar una versión más general del concepto de semáforo que lleve la cuenta del número de recursos disponibles.
El semáforo se inicializa con el número total de recursos disponibles (n).
Las operaciones de **espera** y **señal** se diseñan de modo que se impida el acceso al recurso protegido por el semáforo cuando el valor éste es menor o igual a cero.
Cada vez que se solicita y obtiene un recurso, el semáforo se decrementa.
Cada vez que se libera un recurso el semáforo se incrementa.
Si la operación de **espera** se ejecuta cuando el semáforo tiene un valor menor que 1, el proceso debe quedar en espera de que la ejecución de una operación **señal** libere alguno de los recursos.
La ejecución de las operaciones del semáforo son indivisibles.
Si el semáforo se asocia a un código concurrente, su valor inicial restringe el número máximo de ejecuciones concurrentes que se pueden realizar. Si el valor inicial es 0, ningún proceso conseguirá entrar; si es 1, sólo un proceso conseguirá entrar.



TEMA 3

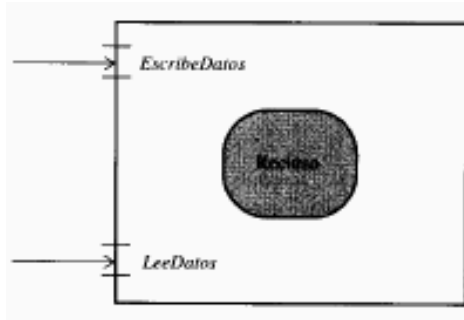
SINCRONIZACION Y COMUNICACION DE PROCESOS

3.3 MONITORES

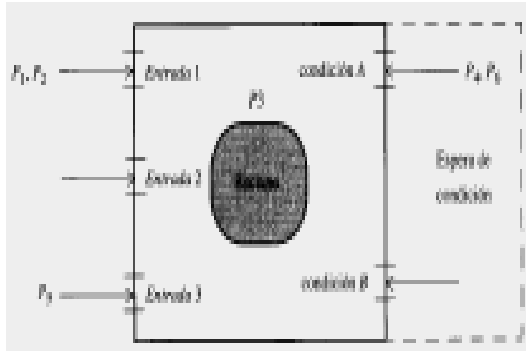
DEFINICIÓN

Monitor. Conjunto de procedimientos que proporciona el acceso con exclusión mutua a un recurso o conjunto de recursos compartidos. Los procedimientos van encapsulados dentro de un modulo que tiene

la propiedad especial de sólo un proceso puede estar activo cada vez para ejecutar un procedimiento del monitor.



Dos procedimientos para acceder a los datos, EscribirDatos y LeeDatos, representan las puertas por las que se puede acceder. Si un proceso desea usar el recurso para escribir datos, llama a EscribirDatos y si no existe un proceso que esté ya accediendo al monitor se le permite la entrada para escribir nuevos datos. Si existe otro proceso que está usando LeeDatos o EscribirDatos entonces el proceso que intenta acceder se para y se suspende en espera de que salga el que está dentro.



Monitor mas complejo, ahora se tienen tres puntos de entrada o procedimientos para uso del recurso (Entrada1, Entrada2, Entrada3), así como dos condiciones que proporcionan sincronización entre los procesos que están esperando a usar el recurso (condicionA y condicionB). El proceso P3 está usando el monitor y dos esperando entrar en la Entrada1 (P1 y P2) y uno en la Entrada3 (P5). Dos procesos han entrado con anterioridad y se han suspendido esperando la condicionA (P4 y P6). La espera de los procesos para acceder a uno de los procedimientos no se hace en colas separadas para cada procedimiento, sino que se utiliza una única cola, porque solo un procedimiento se puede ejecutar a la vez

Un **monitor presenta una ventaja** frente a un semáforo u otro mecanismo para la exclusión mutua y es que ahora ésta está implícita.

Los monitores **no** proporcionan por si mismos un mecanismo para la sincronización de tareas.

Una variable que se utilice como mecanismo de sincronización en un monitor se conoce como **variable de condición**. A cada causa diferente por la que un proceso deba esperar se asocia una variable de condición.

Sobre las variables de condición solo se puede actuar con dos procedimientos: **espera** y **señal**.

Cuando un proceso ejecuta una operación de espera se suspende y se coloca en una cola asociada a dicha variable de condición.

La diferencia con el semáforo es que ahora la ejecución de esta operación siempre suspende el proceso que la emite. La suspensión del proceso hace que se libere la posesión del monitor, lo que permite que entre otro proceso.

Cuando un proceso ejecuta una operación de **señal** se libera un proceso suspendido en la cola de la variable de condición utilizada.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.3.1 SINTAXIS DEL MONITOR

monitor:nombre_del_monitor;

declaración de los tipos y procedimientos que se importan y exportan;
declaración de las variables locales del monitor;

declaración de las variables de condición;

```
procedure Prc1(...);
begin
    .....
end Prc1;

procedure Prc2(...);
begin
    .....
end Prc2;

procedure Prcn(...);
begin
    .....
end Prcn;

begin
    Inicialización del monitor;
end;
```

El monitor no tiene por que exportar todos sus procedimientos. Sólo aquellos que sean públicos (puertas de entrada), manteniendo como privados aquellos a los que sólo se puede acceder desde otros procedimientos definidos dentro del monitor.

La inicialización del monitor sólo se ejecuta una vez y antes de que se realice ninguna llamada a los procedimientos del monitor.

3.3.2 EJEMPLO

Uso de un monitor mostrando como se pueden realizar las operaciones de espera y señal de un semáforo binario.

Las palabras *sespera* y *sseñal* designan las operaciones de espera y señal del semáforo binario. Las palabras *espera* y *señal* las respectivas del monitor.

```
monitor:semáforo;
from condiciones import condicion,espera,señal;
export sespera, sseñal;

var
    ocupado:boolean;
    libre:condicion;

procedure sespera(var libre:condicion);
begin
    if ocupado then
        espera(libre);
        ocupado := true;
    end;
end sespera;

procedure sseñal(var libre:condicion);
begin
    ocupado := false;
    señal(libre);
end sseñal;

begin
    ocupado := false;
end.
```

Se denomina condiciones a la biblioteca donde se encuentra la definición de las variables de condición y de los procedimientos *espera* y *señal* de los monitores. La variable **ocupado** indica el estado del semáforo. El primer proceso que realiza una invocación a la operación **sespera** puede acceder al recurso controlado por el monitor (**ocupado = false**). Cualquier otro proceso que ejecute la operación **sespera** debe esperar a que se ejecute una operación **sseñal** para poder acceder al recurso.

Cuando se ejecuta una operación **sseñal**, la variable *ocupado* pasa a valer false y la **condicion** libre recibe una señal que hace que uno de los procesos que esperan en su cola pase a completarla ejecución de *sespera* y pueda acceder al recurso.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.3.3 EJEMPLO

Monitor para el empleo del buffer en el problema del productor-consumidor.

La manipulación de la variable compartida se realizará de forma exclusiva por llevarla a cabo dentro de procedimientos del monitor (poner y tomar).

La sincronización de las acciones de poner y tomar datos del buffer se llevará a cabo mediante el uso de variables de condición (*espaciodisponible* y *itemdisponible*).

```

monitor productor_consumidor;
from condiciones import condicion,espera,señal;
export poner,tomar;

```

```

const tamaño = 32;
var
  buff:array[0..tamaño-1] of dato;
  cima,base:0..tamaño-1;
  espaciadisponible,itemdisponible:condicion;
  numeroenbuffer:integer;

```

```

procedure poner(var item:dato);
begin
  if numeroenbuffer = tamaño then
    espera(espaciadisponible);
    buff[cima] := item;
    numeroenbuffer := numeroenbuffer + 1;
    cima := (cima + 1) mod tamaño;
    señal(itemdisponible);
  endif;
end poner;

```

```

procedure tomar(var item:dato):
begin
  if numeroenbuffer = 0 then
    espera(itemdisponible);
    item := buff[base];
    numeroenbuffer := numeroenbuffer - 1;
    base := (base + 1) mod tamaño;
    señal(espaciadisponible);
  end;
end tomar;

```

```

begin /*Iniciación */
  numeroenbuffer := 0;
  cima := 0;
  base := 0;
end .

```

Un proceso que desee poner un dato en el buffer deberá esperar al monitor en la cola de entrada si éste está ocupado.

Una vez dentro si el buffer esta lleno se suspenderá cuando ejecute **espera**(espaciadisponible), colocándose en la cola de esta variable de condición.

La espera durará hasta que la ejecución por otro proceso de la operación **señal** del procedimiento tomar deje espacio en el buffer.

Si un proceso llama a tomar cuando no hay nada en el buffer se suspenderá cuando ejecute

espera(itemdisponible), colocándose en la cola de esta variable de condición.

Un proceso que ponga un dato liberará a un proceso que esta suspendido en espera del mismo.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.3.4 IMPLEMENTACION DE UN MONITOR

Como en cada instante sólo se puede ejecutar un proceso, en cada monitor se utiliza un semáforo (exmut) inicializado a 1, al que se invoca antes de entrar al monitor **espera**(exmut) y después de dejar el monitor **señal**(exmut).

Si un proceso P1 está utilizando el monitor y ejecuta la operación **señal**(condi) y hay un proceso P2 en espera en la cola asociada a condi, al proceso P2 se le permite reanudar su ejecución, con lo que se pueden dar tres posibilidades:

- 1) Una operación **señal** sólo se permite como la última acción de un proceso antes de dejar el monitor.
- 2) Continúa el proceso que emite la señal hasta que termina con el procedimiento monitor o espera en otra condición.
- 3) Continúa el proceso desbloqueado por la señal y el proceso que la emite espera a que aquel acabe con el uso del monitor o a que se suspenda en una condición.

En los dos últimos casos, para implementar la espera de los procesos señalados y que puedan reanudar su ejecución, es posible utilizar un semáforo (siguiente) inicializado a 0.

Para la ejecución en exclusiva de cada procedimiento de entrada al monitor, estos se sustituyen por el siguiente segmento de código:

```
espera(exmut);  
.....  
cuerpo del procedimiento;  
.....  
if cuenta_siguientes > 0 then  
    señal(siguiente);  
else  
    señal(exmut);  
end;
```

La variable entera cuenta_siguientes, contador del número de procesos suspendidos en la cola del semáforo siguiente.

Cuando un proceso termina de ejecutar un procedimiento del monitor pasa la posesión de éste a un proceso en espera en la cola del semáforo siguiente. Cuando no hay ninguno en la cola de espera de este semáforo, el monitor se pasa a un proceso que espere en la cola del semáforo exmut del monitor.

Las operaciones de entrada y salida deben de hacerse con exclusión mutua a nivel de hardware para asegurar la integridad de las colas y de las señales.

Las variables de condición se pueden implementar utilizando, para cada una, un semáforo binario y una variable entera ambas inicializadas a 0.

Denominamos scondi al semáforo y scuenta a la variable entera.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

Las operaciones de **espera** y **señal** del monitor se pueden expresar de la forma siguiente:

```
/* operación espera de una variable de condicion: espera(condi) */  
scuenta := scuenta + 1;  
if cuenta_siguientes > 0 then  
    señal(siguiente);  
else  
    señal(exmut);
```

```
end;  
s cuenta := s cuenta - 1;
```

3.4 MENSAJES

DEFINICIÓN

Los mensajes proporcionan una solución al problema de la concurrencia de procesos que integra la sincronización y la comunicación entre ellos y resulta adecuado tanto para sistemas centralizados como para distribuidos.

La comunicación entre mensajes necesita siempre de:

- Un proceso emisor.
- Un proceso receptor.

Las operaciones básicas para una comunicación mediante mensajes son:

- Enviar(mensaje)
- Recibir(mensaje)

Las acciones de transmisión de información y sincronización se ven como actividades inseparables.

La comunicación por mensajes requiere que se establezca un **enlace** entre el receptor y el emisor.

Aspectos importantes en los enlaces:

- a) Cómo y cuantos enlaces se pueden establecer entre los procesos.
- b) La capacidad de mensajes del enlace.
- c) El tipo de los mensajes.

Su implementación varía dependiendo de:

- 1) El modo de referenciar a los procesos.
- 2) El modelo de sincronización.
- 3) Almacenamiento y estructura del mensaje.

3.4.1 MODOS DE REFERENCIAR A LOS PROCESOS

COMUNICACIÓN DIRECTA

Ambos procesos el que envía y el que recibe, nombran de forma explícita al proceso con el que se comunican.

enviar(Q,mensaje) -> Envía un mensaje al proceso Q.

recibir(P,mensaje) -> Recibe un mensaje del proceso P.

Cada proceso debe conocer la identidad de su pareja en la comunicación. No resulta muy adecuado para diseñar rutinas de servicio de un sistema operativo.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

COMUNICACIÓN INDIRECTA

Los mensajes se envían y reciben a través de una entidad intermedia denominada **buzón** o puerto. Un buzón es un objeto en el que los procesos dejan mensajes y del cual pueden tomarlos.

Ofrece mayor versatilidad, permiten comunicación :

- De uno a uno.
- De uno a muchos.
- De muchos a muchos.

Cada buzón tiene un identificador que lo distingue.

Las operaciones básicas tienen la forma:

- **enviar**(buzónA,mensaje) -> envía un mensaje al buzónA.
- **recibir**(buzónA,mensaje) -> recibe un mensaje del buzónA.

El buzón establece un enlace que puede ser usado por más de dos procesos, y permite que la comunicación de un proceso con otro se pueda hacer por distintos buzones.

Cuando existen varios procesos que recogen información del mismo buzón se plantea el problema de quien debe recoger un mensaje. Distintas soluciones:

- a) Permitir que un buzón sólo pueda ser compartido por dos procesos.
- b) Permitir que cada vez sólo un proceso pueda ejecutar una operación de recibir.
- c) Que el sistema identifique al receptor del mensaje

3.4.2 MODELOS DE SINCRONIZACIÓN

- a) **Asíncrona**. El proceso que envía el mensaje sigue su ejecución sin preocuparse de si el mensaje se recibe o no.
El sistema operativo se encarga de recoger el mensaje del emisor y almacenarlo a la espera de una operación de recibir.
Tiene limitada la cantidad de memoria que puede utilizar una pareja mediante comunicación directa, o un buzón en comunicación indirecta.
- b) **Síncrona**. El proceso que envía sólo prosigue su tarea cuando se recibe el mensaje. Necesita que ambos procesos se “junten” para realizar una comunicación, lo que se denomina **encuentro**(randezvous).
Si no se ha emitido una señal de recibir cuando se ejecuta la operación de enviar, el proceso emisor se suspende hasta que la ejecución de una operación recibir le saca de ese estado.
Cuando el proceso que envía el mensaje continua sabe que su mensaje ha sido recibido.
Una pareja emisor-receptor no puede tener más de un mensaje pendiente.
- c) **Invocación Remota**. El proceso que envía sólo prosigue su ejecución cuando ha recibido una respuesta del receptor.
También conocida como **encuentro extendido**, ya que el receptor puede realizar un número arbitrario de cálculos antes de enviar la respuesta.

La operación recibir, tanto en comunicación directa como indirecta, se suele realizar de forma que el proceso que ejecuta la operación toma un mensaje si lo encuentra y se suspende si no lo encuentra. Este modo de funcionamiento plantea el problema de una espera indefinida en caso de que un fallo impida que llegue el mensaje.

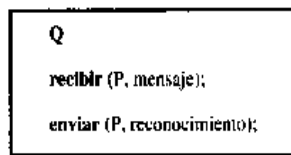
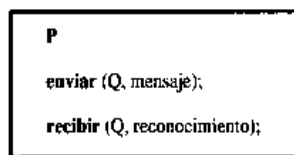
TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

Una solución consiste en proporcionar una operación de recibir sin bloqueo, que en algunos sistemas se denomina **aceptar**, de modo que si el mensaje esta presente se devuelve y si no lo está se devuelve un código de error.

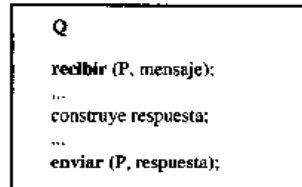
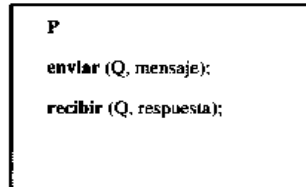
Otra solución mas aceptable consiste en especificar en la operación de recibir un tiempo máximo de espera del mensaje.

Se pueden relacionar las distintas formas de enviar un mensaje. Dos operaciones asíncronas pueden constituir una relación síncrona si se envía una señal de reconocimiento.



Dos procesos P y Q se comunican de forma directa asíncrona. El proceso P envía el mensaje a Q y después se suspende en espera de un mensaje de reconocimiento por parte de Q. Cuando el proceso Q recibe el mensaje envía un mensaje de reconocimiento a P, que hace que este pueda proseguir.

La operación de envío con buzón también puede ser síncrona si se diseña de modo que el remitente se suspende hasta que el mensaje ha sido recibido.



Como una señal asíncrona se puede utilizar para construir los otros dos métodos se podría pensar que este método es el mas flexible y es el que debería utilizarse. Sin embargo tiene un inconveniente, al no saber cuando se recibe el mensaje la mayoría se programan para recibir uno de reconocimiento, con lo que se vuelven síncronas. Son difíciles de depurar.

3.4.3 ALMACENAMIENTO Y ESTRUCTURA DEL MENSAJE

El intercambio de información se puede realizar de dos formas:

- Por valor:
Se realiza una copia del mensaje desde el espacio de direcciones del emisor al espacio de direcciones del receptor.
Tiene la ventaja de que mantiene desacoplada la información que maneja el emisor y el receptor, asegurando la integridad de la información.
Inconvenientes: gasto de memoria, tiempo que implica la copia y uso de memoria intermedia.
- Por referencia:
Se pasa sólo un puntero al mensaje.
Inconvenientes: necesita mecanismos de seguridad para compartir la información entre los procesos

El método de sincronización de la invocación remota utiliza necesariamente la transferencia por valor. Los métodos asíncronos y síncronos pueden utilizar ambos modos.

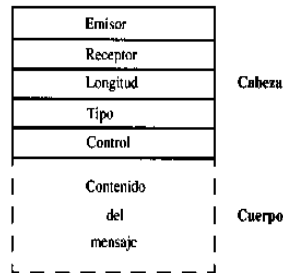
TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

Los S.O. asocian a cada enlace una cola de mensajes pendientes de ser recibidos. En la comunicación síncrona se puede considerar que la cola tiene una longitud nula, ya que los dos procesos deben encontrarse para proceder al intercambio del mensaje. En la comunicación asíncrona y en la invocación remota la implementación de la cola se realiza normalmente con una capacidad de mensajes finita. Cuando el proceso emisor envía un mensaje si la cola está llena se suspende hasta que queda espacio en la cola. Si el mensaje se pasa por referencia se guarda en la cola los punteros a los mensajes. Si el mensaje se pasa por valor se guarda el valor del mensaje en la cola. En algunos casos se considera que la capacidad de la cola es infinita.

Dependiendo de la estructura de los mensajes se pueden dividir en tres tipos:

- a. Longitud fija.
Resulta una implementación física.
Permite una asignación sencilla y eficaz, principalmente en las transferencias por valor.
Inconvenientes: dificulta la tarea de programación.
- b. Longitud variable.
Más adecuado en las transferencias por referencia.
La longitud del mensaje puede formar parte de la información transmitida.
La programación resulta más sencilla.
Tiene una realización más sencilla.



- c. De tipo definido.
Se adaptan mejor a la comunicación con buzones.
A cada buzón que utilice un proceso se le puede asignar el tipo de dato para dicho mensaje y solo los mensajes con esa estructura pueden ser enviados por ese enlace.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.4.4 EJEMPLO

Utilizar la comunicación por mensajes para la realización de un semáforo binario.

Se utiliza un buzón asíncrono con una cola ilimitada. El buzón es conocido tanto por el procedimiento de espera como por el de señal.

```
program/module semáforo;  
type  
  mensaje = ... ; /* Definición del tipo de mensaje */  
const  
  nulo = ... ; /* Mensaje vacío */  
  
procedure espera(var S:integer);  
var
```

Autor: agvuned@gmail.com

El buzón creado se identifica de forma exclusiva con el semáforo.

El semáforo está libre cuando hay un mensaje en la cola.

Si un proceso ejecuta una señal de espera (lo que equivale a una operación de recibir), y existe un mensaje puede proseguir su ejecución.

Si no existen mensajes en la cola y un proceso ejecuta una operación de espera se suspenderá hasta que se señale por otro proceso (que envía un

```

    temp.:mensaje;
begin
    recibir(Sbuzon,temp.);
    S := 0;
end espera;

procedure señal(var S:integer);
begin
    enviar(Sbuzon,nulo);
end señal;

procedure inicializa(var S:integer, valor:boolean);
begin
    if valor = 1 then
        enviar(Sbuzon,nulo);
    end;
    S := valor;
end inicializar;

begin
    crear_buzon(Sbuzon);
end semaforo;

```

3.4.5 EJEMPLO

Solución al problema del productor-consumidor utilizando mensajes. Los mensajes transmiten los datos producidos y consumidos. Las soluciones dependen del tipo de sistema de mensajes que se utilice. En el ejemplo se supone que es asíncrono.

```

process ProductorX;
var x:mensaje;
begin
    loop
        produce(x);
        enviar(buzon,x);
        /*Otras operaciones X*/
    end;
end ProductorX;

```

Esta solución no es válida si la cola del buzón tiene una capacidad finita.

Un proceso que ejecuta una operación de enviar no se suspende si la cola del buzón esta llena, luego esta solución puede llevar a que se pierdan mensajes

```

process ConsumidorY;
var y:mensaje;
begin
    loop
        recibir(buzon,y);
        consume(y);
        /*Otras operaciones Y*/
    end;
end ConsumidorY;

```

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

```

/* Problema del Productor-Consumidor: Solución con mensajes */
program/module Productor_Consumidor;
type
    mensaje = ...; /*definición del tipo de datos del mensaje*/
const
    tamañoQ = ...; /*tamaño de la cola de los buzones */
    nulo = ...; /*mensaje vacio */
var
    n:integer;

```

```

process ProductorX;
var x:mensaje;
begin
    loop
        recibir(buzonP,x);
        produce(x);

```

Autor: agvuned@gmail.com

Se utilizan dos buzones, buzónP y buzónC.

El buzónC se emplea para que los productores dejen los datos que producen y sean recibidos por algún consumidor.

En el buzónP dejan mensajes vacíos los consumidores cada vez que han consumido un dato recibido.

Cada productor debe de recoger un mensaje del buzónP antes de llenarlo y enviarlo al buzónC.

Que buzónP este vacío significa que la cola del buzónC está llena.

Inicialmente se envían tamañoQ mensajes nulos al buzónP, lo

```

    enviar(buzonC,x);
    /* Otras operaciones X */
end;
end ProductorX;

process ConsumidorY;
var y:mensaje;
begin
    loop
        recibir(buzonC,y);
        consume(y);
        enviar(buzonP,y);
        /*Otras operaciones Y */
    end;
end ConsumidorY;

begin
    crear_buzon(buzonP);
    crear_buzon(buzonC);
    for n:= 1 to tamañoQ do
        enviar(buzonP,nulo);
    end;
cobegin
    Productores;
    Consumidores;
coend;
end Productor_Consumidor;

```

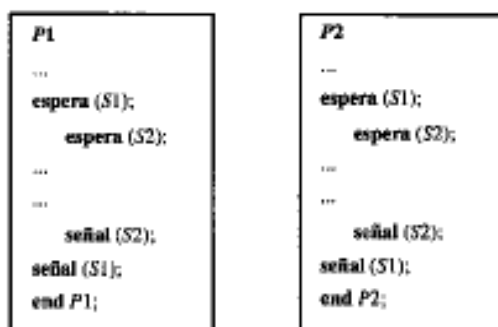
TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.5 INTERBLOQUEO

DEFINICIÓN

Interbloqueo. Consiste en que dos o más procesos entran en un estado que imposibilita a cualquiera de ellos salir del estado en que se encuentra.



Los recursos están protegidos por los semáforo S1 y S2.

Si los procesos acceden a los recursos en el mismo orden no hay problema.

P1 toma S1 y también S2, y posteriormente los libera en orden inverso.

```

P1
...
espera (S1);
    espera (S2);
...
...
señal (S2);
señal (S1);
end P1;

```

```

P2
...
espera (S2);
    espera (S1);
...
...
señal (S1);
señal (S2);
end P2;

```

Si uno de los procesos desea utilizar los recursos en orden inverso, podría ocurrir que P1 tomara el primer recurso (S1) y P2 el segundo recurso (S2) y se quedarán ambos en espera de que el otro liberara el recurso que posee.

3.5.1 CARACTERIZACIÓN DEL INTERBLOQUEO

Para que se de una situación de interbloqueo se deben cumplir de forma simultanea:

- 1) **Exclusión mutua.** Los procesos usan de forma exclusiva los recursos que han adquirido. Si otro proceso solicita el recurso debe esperar a que sea liberado.
- 2) **Retención y espera.** Los procesos retienen los recursos que han obtenido mientras esperan para adquirir otros recursos que están siendo retenidos por otros procesos.
- 3) **No existencia de expropiación.** Los recursos no se pueden quitar a los procesos que los tienen.
- 4) **Espera circular.** Existe una cadena circular de procesos en la que cada uno retiene al menos un recurso que se solicita por el siguiente proceso de la cadena.

Formas principales de tratar el problema del interbloqueo:

- Evitar que se entre en esa situación. Con dos métodos.
 - Prevención de los interbloqueos.
 - Evitación de los interbloqueos.
- Permitir que se pueda entrar y recuperarse de ella.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.5.2 PREVENCIÓN DE INTERBLOQUEOS

Se trata de evitar cualquier posibilidad que pueda llevar a la situación de interbloqueo.

Es el método más utilizado, pero puede llevar a una pobre utilización de los recursos.

Como las cuatro condiciones son necesarias para que se produzca interbloqueo basta con evitar una de ellas para que no se produzca.

EXCLUSIÓN MUTUA

Se debe mantener ya que es necesario mantener recursos que no se puedan compartir.

RETENCIÓN Y ESPERA

Para que no se cumpla esta condición se debe garantizar que un proceso que posee un recurso no pueda pedir otro.

Haciendo que la petición de todos los recursos que necesita un proceso se realice bajo la premisa de todos o ninguno.

Cuando el conjunto completo de recursos necesarios por el proceso están disponibles se le concede y en caso contrario, se suspende en espera de que todos lo estén.

La espera debe hacerse sin poseer ningún recurso.

No resulta adecuado cuando hay recursos que sólo se van a utilizar al final del proceso y que sin embargo se retienen mientras se hace uso de otros. Se puede evitar pidiendo conjuntos de recursos con la condición de o todos o ninguno, pero solo cuando no se disponga de otros.

El método plantea dos problemas:

- 1) Pobre uso de los recursos. Puede haber recursos retenidos que no estén en uso y otros que aunque estén retenidos, no se pueden asignar por ser requeridos junto con otros que si lo están.
- 2) Puede resultar difícil que un conjunto de recursos se encuentren disponibles a la vez, lo que puede producir una espera indefinida del proceso que los necesita.

NO EXISTENCIA DE EXPROPIACIÓN

Se puede evitar permitiendo la expropiación.

Dos estrategias :

- a) Si un proceso que tiene uno o más recursos solicita otro éste le es concedido si está disponible. En caso de que este en uso, el proceso que lo reclama debe esperar y permitir que los recursos de que dispone se puedan expropiar.

El proceso libera los recursos de que ya dispone que se añaden a la lista de recursos disponibles y a la vez a la lista de recursos solicitados por el proceso que los acaba de liberar.

El proceso sólo puede volver a activarse cuando pueda ganar el acceso a todos los recursos que necesita.

- b) Si un proceso solicita algunos recursos primero comprueba que están libres, si es así se le asignan. Si no están libres se mira si están asignados a otros procesos que están esperando, en cuyo caso se expropian y pasan al proceso que puede entrar en ejecución disponiendo de ellos. Si hay algún recurso que no está libre o que no puede ser expropiado, el proceso se suspende y no puede volver a ejecución hasta que no disponga de todos.

Inconveniente: Puede llevar a que haya procesos que se vean relegados durante un tiempo excesivamente grande.

Se aplica con aquellos recursos que pueden ser fácilmente salvados y restaurados.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

ESPERA CIRCULAR

Para que no se produzca esta condición se ordenan los recursos asignándoles a cada tipo de ellos un número entero y se impone que se pidan en ese orden.

Las peticiones de todos los recursos pertenecientes a un mismo tipo deben realizarse con una única petición y no incrementalmente.

Por ejemplo, si el proceso P1 pide el recurso con número de orden 3, a continuación sólo puede pedir recursos con número de orden superior a 3. Si desea pedir a la vez dos recursos siempre deberá pedir primero el recurso con menor número de orden.

Desventaja: los recursos no se piden el orden que se necesitan si no en el orden establecido.

Aquellos procesos que necesiten los recursos en un orden distinto del establecido pueden verse obligados a pedir recursos antes de que los necesiten, acaparándolos innecesariamente.

3.5.3 EVITACIÓN DE INTERBLOQUEOS

El objetivo es imponer condiciones menos restrictivas que el método de la prevención, para conseguir un mejor aprovechamiento de los recursos.

Cuando se vislumbra la posibilidad de que ocurra un interbloqueo este se soslaya. Esto se consigue haciendo que se considere el caso de que produzca un bloqueo cada vez que se asigna un recurso, si se prevé esta posibilidad el recurso no se concede.

La técnica más utilizada es el **n algoritmo del banquero**, que asegura que el número de recursos asignados a todos los procesos nunca puede exceder el número de recursos totales del sistema.

Nunca se puede hacer una asignación peligrosa, es decir, asignar recursos de forma que no queden suficientes para las necesidades todos los procesos.

3.5.4 EJEMPLO

Tres procesos P1, P2 y P3 y un conjunto de 10 recursos, por ejemplo bloques de memoria.

Cuando se crean los procesos declaran que en cualquier instante no podrán necesitar mas de:

- P1 -> 5 bloques.
- P2 -> 6 bloques.
- P3 -> 4 bloques.

Se construye una tabla para llevar la cuenta de los recursos disponibles y necesarios.

Proceso	Usados	Posibles Necesarios	Máximos Necesarios
P ₁	5	0	5
P ₂	6	0	6
P ₃	4	0	4
Total Disponibles		10	

Cuando se van pidiendo recursos el S.O. actualiza la tabla asegurándose de que no se alcanza un posible estado de interbloqueo.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

Un estado es **seguro**, si todos los procesos que ya tienen concedidos recursos tienen la posibilidad de ser completados en algún orden determinado. Incluso con la posibilidad de que cada uno de ellos utilizase el máximo de los recursos declarados

Proceso	Usados	Posibles Necesarios	Máximos Necesarios
P ₁	2	3	5
P ₂	1	5	6
P ₃	3	1	4
Total Disponibles		4	

Ejemplo de estado seguro.

Se cumplen las necesidades de P1 y P3.

No se pueden cumplir las de P2, pero eventualmente la finalización de P1 y/o P3, permitiría que P2 se pudiera completar.

Proceso	Usados	Posibles Necesarios	Máximos Necesarios
P ₁	3	2	5
P ₂	4	2	6
P ₃	2	2	4
Total Disponibles		1	

Ejemplo de estado **NO** seguro.

En este caso no se cumplen las necesidades máximas de ninguno de los procesos.

No se puede asegurar que no se produzca interbloqueo.

Características del algoritmo del banquero:

- a) Se permiten las condiciones:
 - Exclusión mutua.
 - Retención y espera.
 - No existencia de expropiación.
- b) Los procesos solicitan los recursos que necesitan.
Mientras esperan a alguno se les permite mantener recursos de que disponen sin que se les puedan expropiar.
- c) Los procesos piden los recursos al S.O. de uno en uno.
- d) El sistema puede aceptar o rechazar cada petición .
- e) Una petición que conduce a un estado seguro se concede, la petición que no conduce a un estado seguro se rechaza.

Inconvenientes del algoritmo del banquero.

- a) La gestión de los recursos suele ser conservadora.
- b) El algoritmo requiere que los procesos conozcan por adelantado sus necesidades máximas.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

3.5.4 DETECCIÓN DE LOS INTERBLOQUEOS

Este tipo de métodos se usa en aquellos S.O. en los que se permite que se produzca interbloqueo. Se procede comprobando periódicamente si se ha producido el interbloqueo. Si es así, se identifican los procesos y recursos puestos en juego para poder dar una solución..

Es necesario conservar actualizada la información sobre las peticiones y asignaciones de los recursos a los procesos.

En los métodos de detección y recuperación hay que tener presente la sobrecarga que conlleva para el S.O. el mantener la información necesaria y el algoritmo de recuperación, así como las posibles pérdidas que puedan ocurrir en el intento de recuperar al sistema.

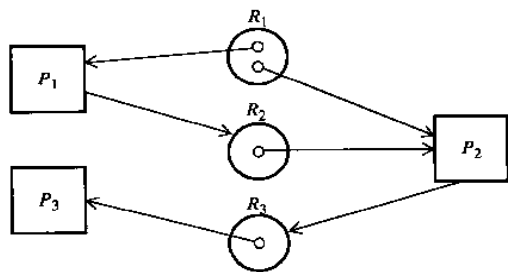
GRAFOS DE ASIGNACIÓN DE RECURSOS

Se usan para facilitar la detección de los interbloqueos.

Se usa un grafo dirigido, los nodos del grafo son procesos y recursos y cada arco conecta el nodo de un proceso con el nodo de un recurso.

Los procesos se representan con cuadrados y los recursos de un mismo tipo con un círculos grandes.

Dentro de un círculo grande se representa mediante círculos pequeños el número de recursos que hay de ese tipo



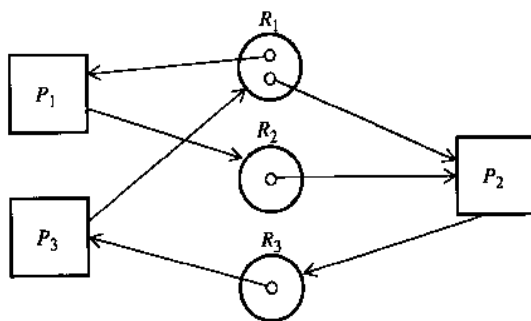
Tres procesos (P1,P2,P3) y tres tipos de recursos (R1,R2,R3), por ejemplo un disco, una cinta y una impresora. Hay dos unidades de disco, una cinta y una impresora.
 P1 tiene la propiedad de un disco y esta solicitando una cinta.
 P2 tiene la propiedad de un disco y una cinta y solicita la impresora.
 P3 tiene la propiedad de la impresora

El recurso solicitado se indica mediante un arco que va desde el nodo proceso solicitante hasta el nodo recurso solicitado.
 La propiedad de un recurso se indica con un arco que va desde el recurso al proceso.

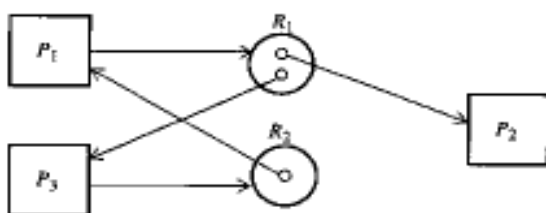
Si el grafo no contiene ningún ciclo no hay ningún proceso que este bloqueado, pero si lo contiene puede existir un interbloqueo.
 Si sólo hay elemento por cada tipo de recurso la existencia de un ciclo es una condición necesaria y suficiente para que exista interbloqueo.
 Si cada tipo de recurso tiene varios elementos, entonces la condición de existencia de un ciclo es necesaria pero no suficiente para asegurar que existe interbloqueo. En este caso una condición suficiente es la existencia de un ciclo en el que no hay ningún camino que salga de algunos de los nodos que lo forman que a su vez no sea ciclo.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS



El grafo contiene dos ciclos:
 (R1,P1,R2,P2,R3,P3,R1) y (R1,P2,R3,P3,R1)
 El proceso P1 esta esperando al recurso R2 que lo tiene P2 que a su vez espera por R2 que lo tiene P3, que a su vez espera que P1 o P2 dejen libre el recurso R1.



En este caso se tiene un ciclo:
 (R1,P3,R2,P1,R1), pero el camino (R1,P2) no es un ciclo.
 P2 liberara a R1 y dejara que P1 lo pueda tomar. Con este recurso P1 puede liberar a R2 y dejara que P3 siga con su ejecución.

Se puede usar un método de reducción del grafo con el fin de detectar el interbloqueo y de obtener los procesos que pueden completarse y los que permanecerán bloqueados. Para ello se determinan los procesos a los que se pueden conceder los recursos que tienen solicitados. Los procesos que cumplen

esta condición se dice que pueden reducir el grafo. La reducción de estos procesos se realiza quitando los arcos que unen estos procesos con los recursos y a los que unen los recursos con los procesos.

Si el grafo puede reducirse para todos los procesos no existe interbloqueo. En caso contrario, los procesos irreducibles constituyen el conjunto de procesos interbloqueados.

3.5.5 RECUPERACIÓN DE INTERBLOQUEOS

Una vez que se ha detectado el interbloqueo se debe romper para que los procesos puedan finalizar su ejecución y liberar los recursos.

Para la ruptura de la espera circular se pueden realizar varias opciones. La idónea sería suspender algunos de los procesos bloqueados para tomar sus recursos y reanudar su ejecución una vez que se haya deshecho el interbloqueo. Esta solución sólo puede ser factible en casos muy particulares.

Las dos opciones que se suelen utilizar son:

- Reiniciar uno o más procesos bloqueados. Factores a tener en cuenta para que la reiniciación sea menos traumática.
 - La prioridad del proceso.
 - El tiempo de procesamiento utilizado y el que le resta..
 - El tipo y el número de recursos que posee.
 - El número de recursos que necesita para finalizar.
 - El número de otros procesos que se verían involucrados en su reiniciación.
- Expropiar los recursos de algunos de los procesos bloqueados.
 - El procedimiento consiste en ir expropiando recursos de algunos procesos de forma sucesiva hasta que se consiga salir del interbloqueo.
 - La elección de los recursos que se expropián se basa en criterios similares a la reiniciación.

TEMA 3

SINCRONIZACION Y COMUNICACION DE PROCESOS

- Otro aspecto a tener en cuenta, es el estado al que pasan los procesos expropiados. La solución sería pasarlos a un estado anterior en el que se rompa el bloqueo. Pero para que esto sea posible se necesita que el sistema disponga de una utilidad que registre los estados de los distintos procesos en tiempo de ejecución, con la consiguiente carga adicional sobre el S.O.
- En algunos sistemas de tiempo real el interbloqueo puede tener resultados inaceptables. En otros sistemas el interbloqueo se rechaza, por el costo de tiempo y medios adicionales que conlleva la situación.

Se puede obtener una mayor eficiencia combinando los distintos métodos. Una solución puede ser:

- a) Agrupar los recursos en clases disjuntas.
- b) Para evitar el interbloqueo entre las clases se ordenan en la forma que se ha señalado para evitar la espera circular.
- c) Usar en cada clase el método más apropiado de evitar en ella el interbloqueo.

Ejemplo

Considérese las cuatro clases de recursos siguientes, ordenadas en la forma en que aparecen:

- 1) Espacio de intercambio. Área de bloques de memoria secundaria para intercambio de procesos con la memoria principal.

- 2) Recursos de los procesos. Impresoras, ficheros, discos, cintas, etc.
- 3) Memoria principal. Asignable en páginas o segmentos.
- 4) Recursos internos. Canales de E/S, etc.

Las estrategias más apropiadas para cada clase son:

- a) Espacio de intercambio. Lo usual es que se conozca de antemano la capacidad máxima de almacenamiento. Lo más adecuado es la prevención del interbloqueo, por el método usado para evitar la condición de retención y espera. Haciendo que se solicite a la vez toda la memoria necesaria. Se puede usar el método de evitación del interbloqueo.
- b) Recursos de los procesos. Normalmente los procesos declaran antes de la ejecución los recursos de una determinada clase que van a necesitar. El método de evitación de interbloqueos, parece el más adecuado. También se puede utilizar el método de la ordenación de procesos.
- c) Memoria principal. El método es el de la prevención mediante expropiación , ya que cuando un proceso resulta expropiado se pasa a memoria secundaria.
- d) Recursos internos. Prevención por ordenación de recursos. Con estos recursos no suele ser necesario realizar ninguna elección en tiempo de ejecución entre las peticiones pendientes.