COMUNICACIÓN Y SINCRONIZACIÓN ENTRE PROCESOS. INTERBLOQUEOS

Capítulo 4

Índice

Introducción

- Multiprogramación, multiprocesamiento y procesamiento distribuido
- Necesidad de sincronizar y comunicar procesos
- Métodos básicos de comunicar procesos
  - Compartición de datos
  - Intercambio de información

Exclusión mutua

- Se denomina Sección Crítica (SC) de un proceso a aquellas partes de su código que no pueden ejecutarse de forma concurrente
- Protocolo: Código dedicado a asegurar que la sección crítica se ejecuta de forma exclusiva

Exclusión mutua

Requisitos resolver para la exclusión mutua:

- Sólo un proceso debe tener permiso para entrar en la SC en un momento dado.
- Cuando se interrumpe un proceso en una región no crítica no debe interferir el resto de procesos
- No puede demorarse un proceso indefinidamente en una sección crítica.
- Cuando ningún proceso está en su SC, cualquier proceso que solicite entrar debe hacerlo sin dilación
- No se deben hacer suposiciones sobre la velocidad relativa de los procesos ni el número de procesadores
- Un proceso permanece en su SC sólo por un tiempo finito.

Exclusión mutua: solución sw

- La responsabilidad de mantener la Exclusión Mutua recae sobre los procesos.
- Es necesaria una memoria principal compartida accesible a todos los procesos.
- Existe una exclusión mutua elemental en el acceso a la memoria.
- Algoritmo de Peterson
- Algoritmo de Dekker
- Inconvenientes:
  - La espera de acceso a un recurso se realiza de forma ocupada.
  - Presentan dificultades antes una cantidad elevada de procesos concurrentes.

Solución sw: Algoritmo de Peterson


```
booleano señal[2];
int turno;

void main()
{
    señal[0] = false;
    señal[1] = false;
    cobegin
      P0();P1();
    coend;
}
```

```
void P0()
{
  while (true)
  {
    . . .
    señal[0] = true;
    turno = 1;
    while (señal[1] &&
     turno==1);
    /*Sección Crítica*/
    señal[0] = false;
    . . .
  }
}
```

```
void P1()
{
  while (true)
  {
    . . .
    señal[1] = true;
    turno = 0;
    while (señal[0] &&
     turno==0);
    /*Sección Crítica*/
    señal[1] = false;
    . . .
  }
}
```

Exclusión mutua: solución hw

- Inhabilitación de interrupciones
- Sistemas monoprocesador. Sólo aplicable a nivel de núcleo

```
while (true)
{
    . . .
    Inhabilitar interrupciones;
    /*Sección crítica*/
    Habilitar interrupciones;
    . . .
}
```

- Se degrada la eficiencia del procesador
- Instrucciones especiales de máquina
- Se realizan varias acciones atómicamente: leer y escribir, leer y examinar, …
- No están sujetas a interferencias de otras instrucciones

Semáforos

- Tipo Abstracto de Datos
- Datos:
  - Contador entero
  - Cola de procesos en espera
- Operaciones:
  - **Inicializar**: Inicia el contador a un valor no negativo
  - **P()**: Disminuye en una unidad el valor del contador. Si el contador se hace negativo, el proceso que ejecuta P se bloquea.
  - **V()**: Aumenta en una unidad el valor del contador. Si el valor del contador no es positivo, se desbloquea un proceso bloqueado por una operación P.
- Las operaciones son atómicas a nivel hardware
- Se denomina **semáforo binario** aquel en el que el contador sólo toma valor 0 ó 1.
- El proceso que espera entrar en la SC no usa el procesador, está bloqueado.

Semáforo general: primitivas


```
struct TSemáforo
{
    int contador;
    TColaProcesos Cola;
}
void inicializar(TSemáforo
    s, int n)
{
   s.contador=n;
}
```

```
void P(TSemáforo s)
{
   s.contador--;
   if (s.contador<0)
   {
      poner este proceso en
     s.cola;
      bloquear este
     proceso;
   }
}
```

```
void V(TSemáforo s)
{
   s.contador++;
   if (s.contador<=0)
   {
      quitar un proceso p de
    s.cola;
      poner el proceso p en
    la cola de listos;
   }
}
```

Semáforo binario: primitivas


```
struct TSemáforo_bin
{
    int contador;
    TColaProcesos cola;
}

Void inicializar_B
    (TSemáforo_bin s, int n)
{
   s.contador=n;
}
```

```
void P_B(TSemáforo_bin s)
{
   if (s.contador == 1)
s.contador = 0;
   else
   {
      poner este proceso en
s.cola;
      bloquear este
proceso;
   }
}
```

```
void V_B(TSemáforo_bin s)
{
   if (s.cola.esvacia())
s.contador = 1;
   else
   {
      quitar un proceso p de
s.cola;
      poner el proceso p en
la cola de listos;
   }
}
```

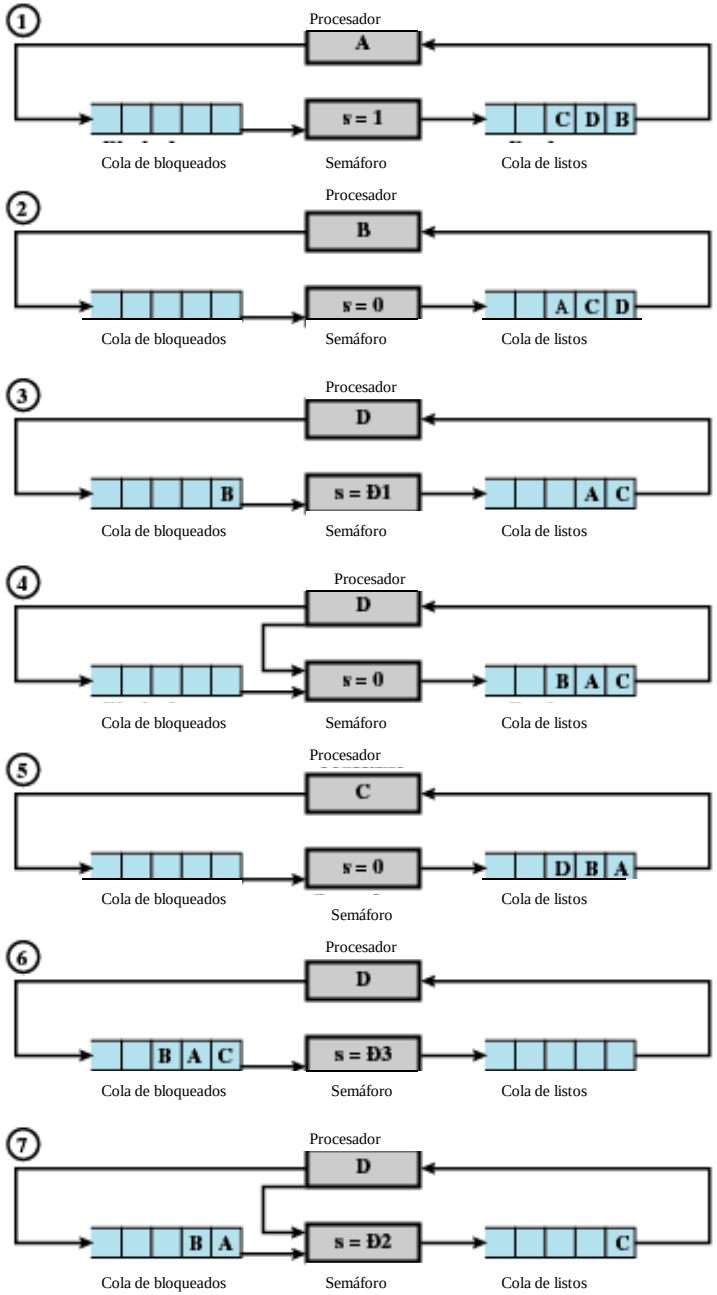
Semáforos: exclusión mutua

- El valor asignado al contador indicará la cantidad de procesos que pueden ejecutar concurrentemente la sección crítica
- Los semáforos se deben inicializar antes de comenzar la ejecución concurrente de los procesos.

```
TSemáforo s;
void Pi();
{
   while (true)
   {
      . . .
      P(s);
      sección crítica;
      V(s)
      . . .
   }
}
```

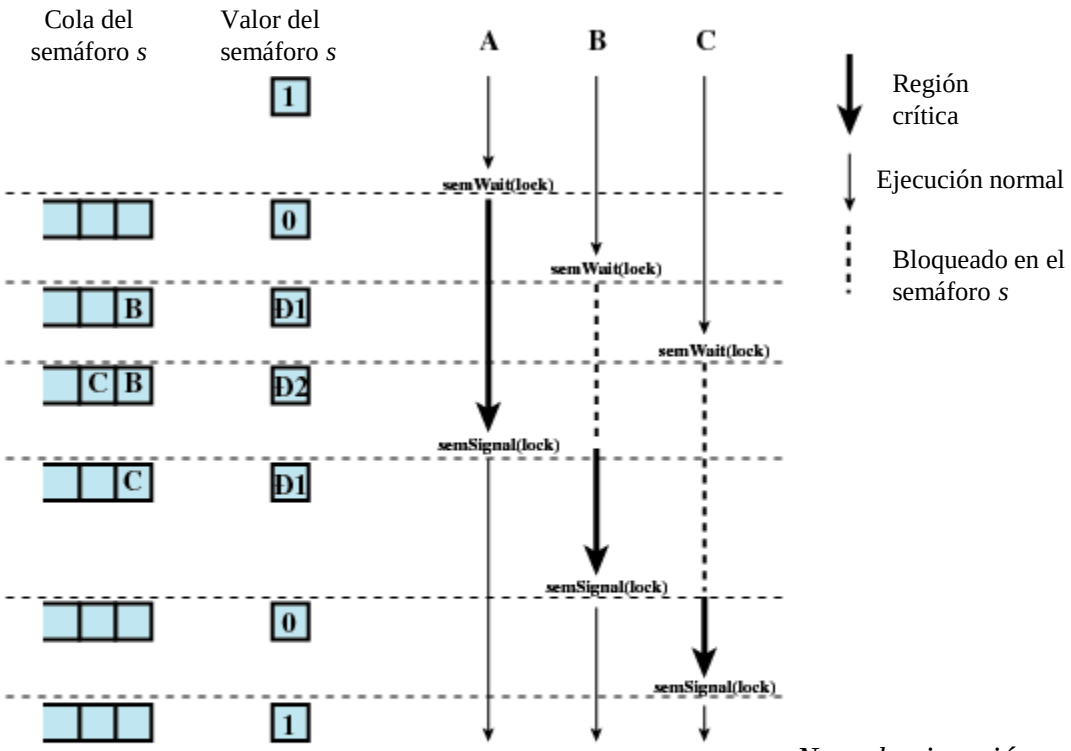
```
void main
{
   inicializar(s, 1)
   cobegin
      P1(); P2(); … ; Pn();
   coend
}
```

Semáforos



Ejemplo de mecanismo semáforo

Semáforos



Nota: la ejecución normal sucede en paralelo pero las regiones críticas se serializan

Procesos accediendo a datos compartidos protegidos por un semáforo

Semáforos: sincronización procesos

- El uso de semáforos permite la sincronización entre procesos
- Problema del productor – consumidor
- Uno o más productores generan cierto tipo de datos y los sitúan en una zona de memoria o buffer. Un único consumidor saca elementos del buffer de uno en uno. El sistema debe impedir la superposición de operaciones sobre el buffer. **Solución:**

Tamaño de buffer ilimitado

```
void main()
{
   inicializar(s, 1);
inicializar(n, 0);
   cobegin
      productor();
      consumidor();
   coend;
}
```

```
TSemáforo s,n;
void productor()
{
   while (true)
   {
      producir();
      P(s);
      añadir_buffer();
      V(s);
      V(n);
   }
}
```

```
Tvoid consumidor()
{
   while (true)
   {
      P(n);
      P(s);
      coger_ buffer();
      V(s);
      consumir();
   }
}
```

Semáforos: sincronización procesos

2 Semáforos : s y n

- El uso de semáforos permite la sincronización entre procesos
- Pr
- Uno o más p eneran cierto tipo de datos y los sitúan en una zona de

s para garantizar el acceso a la sección crítica

del buffer de uno en uno. El sobre el buffer. **Solución:**

n para gestión de P y C →

sincronización del consumidor

Tamaño de buffer ilimitado

```
void main()
{
   inicializar(s, 1);
inicializar(n, 0);
   cobegin
      productor();
      consumidor();
   coend;
}
```

```
TSemáforo s,n;
void productor()
{
   while (true)
   {
      producir();
      P(s);
      añadir_buffer();
      V(s);
      V(n);
   }
}
```

```
Tvoid consumidor()
{
   while (true)
   {
      P(n);
      P(s);
      coger_ buffer();
      V(s);
      consumir();
```

Al comienzo hay 0 elementos

s=1, sólo 1 proceso

Semáforos: sincronización procesos

Problema del productor – consumidor: **Tamaño de buffer limitado**

```
#define tamaño_buffer N
TSemáforo e, s, n;

void main()
{
   inicializar(s, 1);
   inicializar(n, 0);
   inicializar(e,tamaño_buffer);
   cobegin
      productor();
      consumidor();
   coend;
}
```

```
void productor()
{
   while (true)
   {
      producir();
      P(e);
      P(s);
      añadir_buffer();
      V(s);
      V(n);
   }
}
```

```
void consumidor()
{
   while (true)
   {
      P(n);
      P(s);
      coger_ buffer();
      V(s);
      V(e);
      consumir();
   }
}
```

3 Semáforos : s, n y e para gestión de buffer

procesos

Problema del productor – consumidor: **Tamaño de buffer limitado**

```
#define tamaño_buffer N
TSemáforo e, s, n;

void main()
{
   inicializar(s, 1);
   inicializar(n, 0);
   inicializar(e,tamaño_buffer);
   cobegin
      productor();
      consumidor();
   coend;
}
```

```
void productor()
{
   while (true)
   {
      producir();
      P(e);
      P(s);
      añadir_buffer();
      V(s);
      V(n);
   }
}
```

Al producir un elemento,

Se introduce al buffer = P

```
void consumidor()
{
   while (true)
   {
      P(n);
      P(s);
      coger_ buffer();
      V(s);
      V(e);
      consumir();
   }
}
```

Al consumir un elemento,

Se libera del buffer = P

Semáforos: sincronización procesos-lectores/escritores (prioridad lectores)

- Se dispone de una zona de memoria o fichero a la que acceden unos procesos (lectores) en modo lectura y otros procesos en modo escritura (escritores).
  - Los lectores pueden acceder al fichero de forma concurrente.
  - Los escritores deben acceder al fichero de manera exclusiva entre ellos y con los lectores.
- El sistema debe coordinar el acceso a la memoria o al fichero para que se cumplan las restricciones.

```
TSemáforo mutex, w;
int lectores;
void main()
{
   inicializar(mutex, 1);
   inicializar(w, 1);
   lectores=0;
   cobegin
      escritor1();…; escritorn();
            lector1(); … ; lectorm();
   coend;
}
```

```
void escritori()
{
    . . .
    P(w);
    escribir();
    V(w);
    . . .
}
```

```
void lectorj()
{
    . . .
    P(mutex);
    lectores++;
    if (lectores==1) P(w);
    V(mutex)
    leer();
    P(mutex);
    lectores--;
    if (lectores==0) V(w);
    V(mutex)
    . . .
}
```

Semáforos: sincronización procesos-lectores/escritores (prioridad lectores)

¿Cómo se sabe cual es el último?

Añadiendo una variable →

Nuevo problema de SC (semáforo MUTEX)

que acceden unos procesos (lectores) en modo s).

- Los lectores pueden acceder al fichero de for
- Los escritores deben acceder al fichero de m
- El sistema debe coordinar el acceso a la memoria

Sólo realiza la V el último

```
TSemáforo mutex, w;
int lectores;
void main()
{
   inicializar(mutex, 1);
   inicializar(w, 1);
   lectores=0;
   cobegin
      escritor1(); ... escritorn();
      lector1(); ... lectorm();
   coend
}
```

```
void escritori()
{
   ...
   P(w);
   escribir();
   V(w);
   ...
}
```

```
void lectorj()
{
   ...
   P(mutex);
   lectores++;
   if (lectores==1) P(w);
   V(mutex);
   leer();
   P(mutex);
   lectores--;
   if (lectores==0) V(w);
   V(mutex)
   ...
}
```

El primer lector realiza la P,

el resto NO, pueden leer a la vez

Semáforos: sincronización procesos-lectores/escritores (prioridad escritores)


```
TSemáforo mutex1, mutex2, w, r;
int lectores, escritores;
void main()
{
   inicializar(mutex1, 1);
inicializar(mutex2, 1);
   inicializar(w, 1); inicializar(r, 1);
   lectores = 0; escritores = 0;
   cobegin
      escritor1();…; escritorn();
lector1(); … ; lectorm();
   coend;
}
```

```
void lectori()
{
   . . .
   P(r);
   P(mutex1);
   lectores++;
   if (lectores==1)
P(w);
   V(mutex1);
   V(r);
   leer();
   P(mutex1);
   lectores--;
   if (lectores==0)
V(w);
   V(mutex1);
   . . .
}
```

```
void escritorj()
{
   . . .
   P(mutex2);
   escritores++;
   if (escritores==1) P(r);
   V(mutex2);
   P(w);
   escribir();
   V(w);
   P(mutex2);
   escritores--;
   if (escritores==0) V(r);
   V(mutex2)
   . . .
}
```

Lectores/escritores (prioridad escritores)

Supongamos que llegan por éste orden: un lector, un escritor, lector, escritor

Sig. Lector para por el nuevo semáforo r

```
TSemáforo mutex1, mutex2, w, r;
int lectores, escritores;
void main()
{
   inicializar(mutex1, 1);
inicializar(mutex2, 1);
   inicializar(w, 1); inicializar(r, 1);
   lectores = 0; escritores = 0;
   cobegin
      escritor1();…; escritorn();
lector1(); … ; lectorm();
   coend;
}
```

```
void lectori()
{
   . . .
   P(r);
   P(mutex1);
   lectores++;
   if (lectores==1)
P(w);
   V(mutex1);
   V(r);
   leer();
   P(mutex1);
   lectores--;
   if (lectores==0)
V(w);
   V(mutex1);
   . . .
}
```

```
void escritorj()
{
   . . .
   P(mutex2);
   escritores++;
   if (escritores==1) P(r);
   V(mutex2);
   P(w);
   escribir();
   . . .
   escritores--;
   if (escritores==0) V(r);
   V(mutex2)
   . . .
}
```

Adelanta el nuevo lector

Semáforos: sincronización procesos-lectores/escritores (acceso según llegada)


```
TSemáforo mutex, fifo,  w;
int lectores;
void main()
{
   inicializar(mutex, 1);
inicializar(fifo, 1); inicializar(w,
1);
   lectores = 0;
   cobegin
      escritor1();…; escritorn();
lector1(); … ; lectorm();
   coend;
}
```

```
void lectori()
{
    . . .
    P(fifo);
    P(mutex);
    lectores++;
    if (lectores==1)
P(w);
    V(mutex);
    V(fifo);
    leer();
    P(mutex);
    lectores--;
    if (lectores==0)
V(w);
    V(mutex);
    . . .
}
```

```
void escritorj()
{
    . . .
    P(fifo);
    P(w);
    V(fifo);
    escribir();
    V(w);
     . . .
}
```

Semáforos: sincronización de procesos: problema barbería

Una barbería tiene una sala de espera con **n** sillas, y una habitación con un sillón donde se atiende a los clientes. Si no hay clientes el barbero se duerme. Si un cliente entra en la barbería y todas las sillas están ocupadas, entonces se va, sino, se sienta en una de las sillas disponibles .Si el barbero está dormido, el cliente lo despertará.

El sistema debe coordinar el barbero y los clientes

```
#define sillas n
TSemáforo mutex, clientes, barbero;
int espera;
void main()
{  inicializar(mutex, 1);
inicializar(clientes, 0);
   inicializar(barbero, 0);  espera=0;
    cobegin
      barbero();
      cliente1(); cliente2(); …
clientem();
   coend;
}
```

```
void barbero()
{
    while (true)
    {
       P(clientes);
       P(mutex);
       espera=espera-1;
       V(barbero);
       V(mutex);
       cortar_pelo();
    }
}
```

```
void clientei()
{
       P(mutex);
       if (espera<sillas)
       {
          espera=espera+1;
          V(clientes);
          V(mutex);
          P(barbero);
          se_corta_pelo();
       }
       else V(mutex);
}
```

Semáforos: sinc procesos: problem

Clientes : sincroniza al barbero

barbero : clientes de uno en uno

Mutex: variable espera

Una barbería tiene una sala de espera con **n** sillas, y una habitación con un sillón donde se atiende a los clientes. Si no hay clientes el barbero se duerme. Si un cliente entra en la barbería y todas las sillas están ocupadas, entonces

Lo primero ejecuta P y bloquea excepto si han llegado antes clientes

y han ejecutado la operación V correspondiente.

```
#define sillas n
TSemáforo mutex, clientes, barbero;
int espera;
void main()
{  inicializar(mutex, 1);
inicializar(clientes, 0);
```

```
void barbero()
{
   while (true)
   {
      P(clientes);
      P(mutex);
      espera=espera-1;
      V(barbero);
      V(mutex);
      cortar_pelo();
```

```
void clientei()
{
      P(mutex);
      if (espera<sillas)
      {
         espera=espera+1;
         V(clientes);
         V(mutex);
         P(barbero);
         se_corta_pelo();
      }
      else V(mutex);
}
```

que el barbero vaya despertando uno a uno con V

En este semáforo se bloquean los clientes al ejecutar P esperando

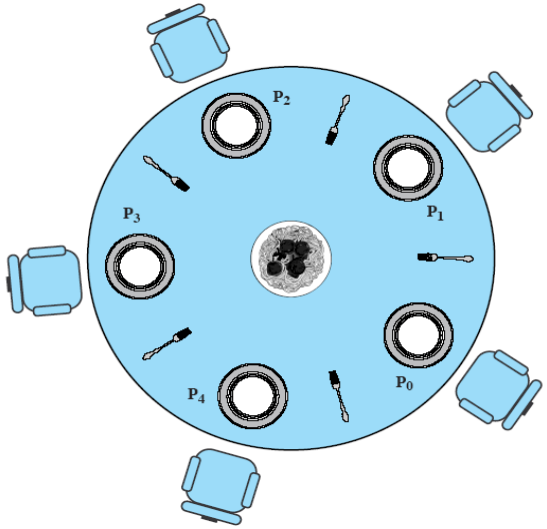
Semáforos: sincronización de procesos: problema filósofos

Cinco filósofos se dedican a pensar y a comer en una mesa circular. En el centro de la mesa hay un cuenco con arroz, y la mesa está puesta con cinco platos y cinco palillos, uno por cada filósofo. Cuando un filósofo tiene hambre se sienta en la mesa a comer en su sitio. El filósofo sólo puede coger un palillo cada vez y no le puede quitar un palillo a un compañero que lo tenga en la mano. Cuando un filósofo tiene los dos palillos come sin soltarlos hasta que termine y vuelve a pensar. El sistema debe coordinar los filósofos para evitar la espera indefinida y no se mueran de hambre.

```
TSemáforo palillo[5];
void main()
{  int i;
   for (i=0; i<5; i++)
inicializar(palillo[i], 1);
   cobegin
      filósofo(0); filósofo(1); …
filósofo(4);
   coend;
}
```

```
void filósofo(int i)
{
    while (true)
    {
       pensar();
       P(palillo[i]);
       P(palillo[(i+1)%5]);
       comer();
       V(palillo[i]);
       V(palillo[(i+1)%5]);
    }
}
```

Solución que mantiene exclusión mutua pero se produce interbloqueo cuando acuden a comer todos a la vez



Semáforos: sincronización de procesos – Filósofos

Solución: Sólo se pueden sentar en la mesa 4 filósofos a la vez

Cinco filósofos se dedican a pensar y a comer en una mesa circular. En el centro de la mesa hay un cuenco con arroz, y la mesa está puesta con cinco platos y cinco palillos, uno por cada filósofo. Cuando un filósofo tiene hambre se sienta en la mesa a comer en su sitio. El filósofo sólo puede coger un palillo cada vez y no le puede quitar un palillo a un compañero que lo tenga en la mano. Cuando un filósofo tiene los dos palillos come sin soltarlos hasta que termine y vuelve a pensar.

```
TSemáforo palillo[5],silla;
void main()
{
   int i;
   for (i=0; i<5; i++)
inicializar(palillo[i], 1);
   inicializar(silla, 4);
   cobegin
      filósofo(0); filósofo(1); …
filósofo(4);
   coend;
}
```

```
void filósofo(int i)
{
   while (true)
   {
      pensar();
      P(silla);
      P(palillo[i]);
      P(palillo[(i+1)%5]);
      V(silla);
      comer();
      V(palillo[i]);
      V(palillo[(i+1)%5]);
   }
}
```

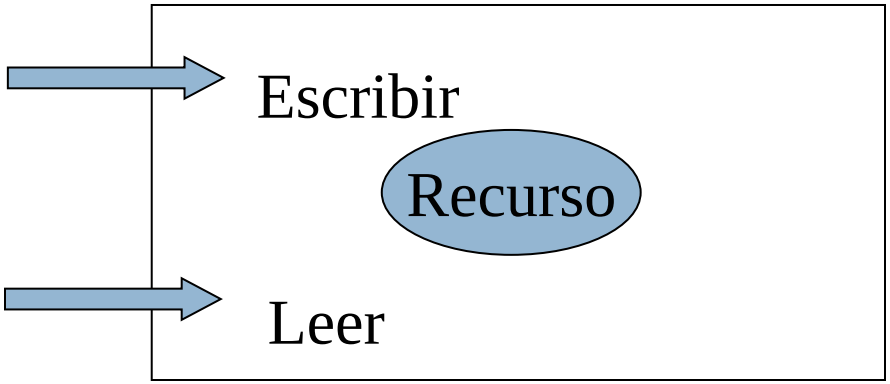
Solución que mantiene exclusión mutua y evita interbloqueos

Semáforos: limitaciones

- Resulta difícil construir un programa correcto mediante semáforos. No es sencillo recordar qué semáforo está asociado a cada recurso o variable.
- Las operaciones P y V se distribuyen por todo el programa y no es fácil advertir el efecto global que provocan.
- El usuario es responsable tanto de la gestión de la exclusión mutua como de la sincronización entre los procesos.
- Cuando se examina un recurso y este está ocupado el proceso siempre se bloquea.

Monitores

- Tipo Abstracto de Datos: Datos locales, procedimientos y una secuencia de inicio.
- Los datos locales sólo están accesibles desde los procedimientos del monitor.
- A un monitor sólo puede entrar un proceso en un instante dado, de modo que si un proceso quiere usar un monitor y existe otro proceso que ya lo está usando, entonces el proceso que quiere entrar se suspende hasta que salga el que está dentro.
- Si los datos del monitor representan a algún recurso, el monitor ofrecerá un servicio de exclusión mutua en el acceso a ese recurso.



Monitores: sincronización

- El monitor proporciona sincronización por medio de variables de condición.
- Procedimientos para operar con las variables de condición:
  - **Espera**(condición): Suspende la ejecución del proceso que llama bajo la condición. Se dispone de una cola de procesos a cada variable de condición.
  - **Señal**(condición): Reanuda la ejecución de algún proceso suspendido en el procedimiento anterior. Si no hay procesos suspendidos no hace nada.
- Cuando un proceso se bloquea en una cola de una variable condición, sale del monitor, permitiendo que otro proceso pueda entrar en él.
- La propia naturaleza del monitor garantiza la exclusión mutua, sin embargo, la sincronización entre los procesos es responsabilidad del programador.

Monitores: sincronización

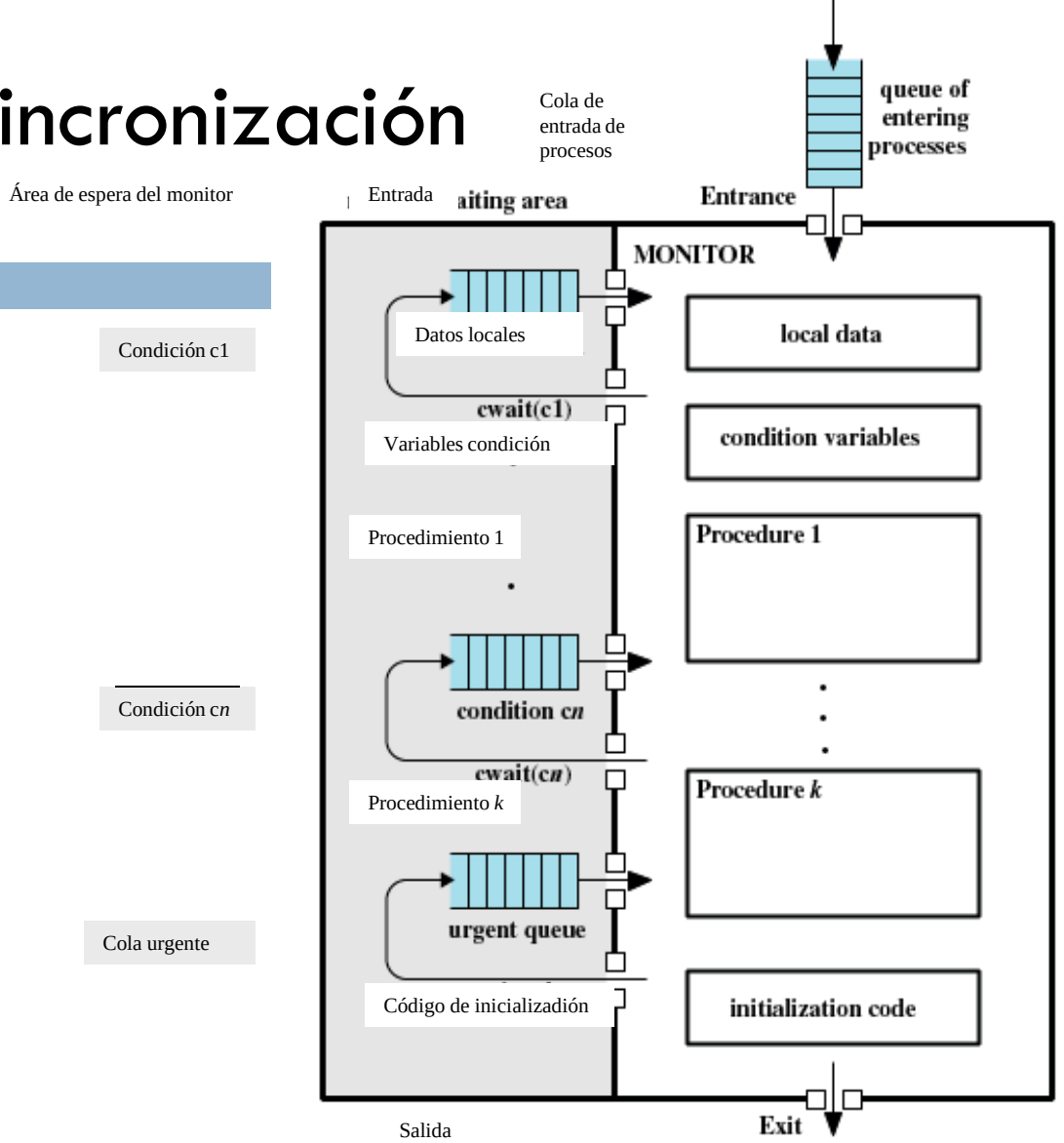


Figura 5.15. Estructura de un monitor ure of a Monitor

Mensajes

- El paso de mensajes resuelve la comunicación y la sincronización de procesos. Adecuado para sistemas centralizados y distribuidos.
- Primitivas:
  - Enviar(destino, mensaje)
  - Recibir(origen, mensaje)
- Las primitivas son atómicas a nivel hardware.

Mensajes

- Direccionamiento
  - **Directo:** Se nombra de forma explícita en la primitiva el proceso al que se refieren.

    Enviar (Procesoi, mensaje)
  - **Indirecto:** Los mensajes se envían y se reciben a través de una entidad intermedia llamada buzón.

    Enviar (buzón, mensaje)
    - Se desacopla el emisor y el receptor
    - Los conjuntos de emisores y receptores no tienen porqué tener la misma cardinalidad.
    - La asociación de procesos a buzones puede ser estática o dinámica.

Mensajes

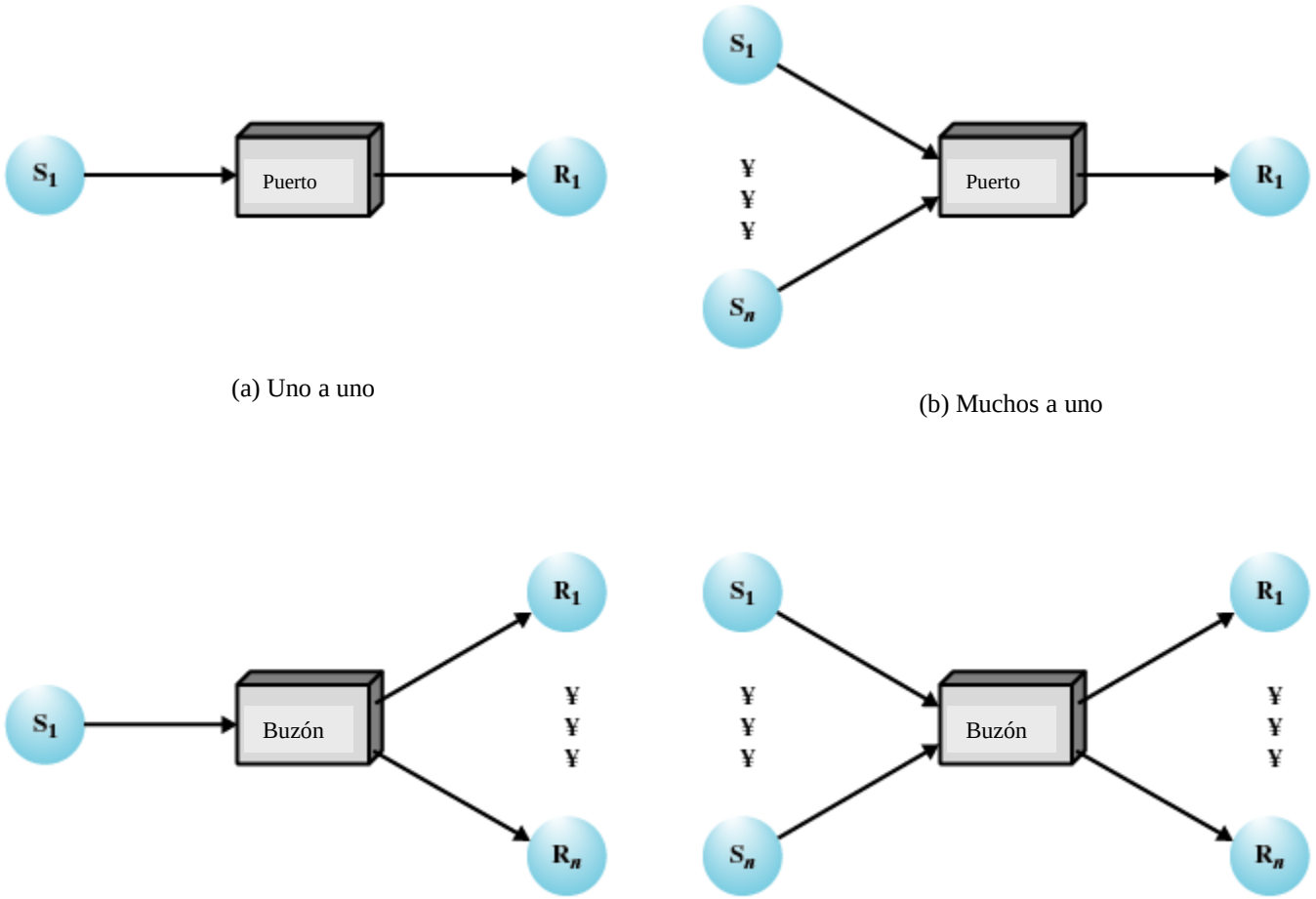


Figura 5.18. Comunicación indirecta de procesos

Mensajes: sincronización

- Modelos de sincronización: Enviar
  - **Bloqueante:** El proceso que envía sólo prosigue su tarea cuando el mensaje ha sido recibido
  - **No Bloqueante:** El proceso que envía un mensaje sigue su ejecución sin preocuparse de si el mensaje se recibe o no.
  - **Invocación remota:** El proceso que envía el mensaje sólo prosigue su ejecución cuando ha recibido una respuesta explícita del receptor.

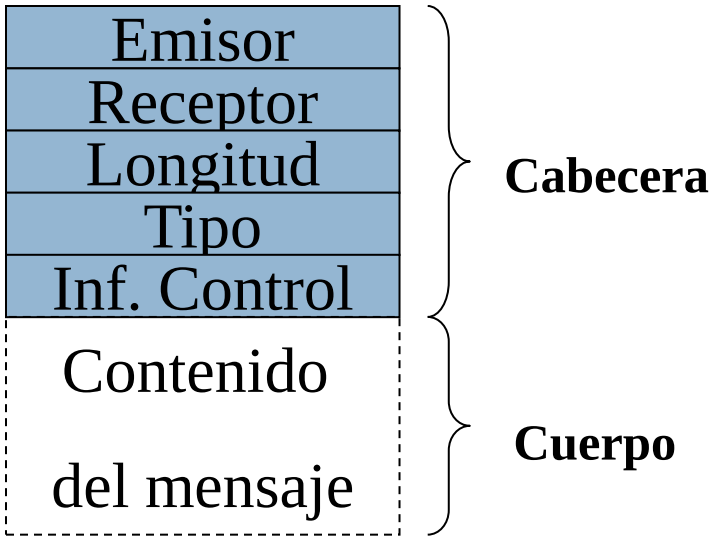
Mensajes: sincronización

- Modelos de sincronización: Recibir
  - **Bloqueante**: El proceso que realiza recibir un mensaje lo recoge si éste existe o bien se bloquea si el mensaje no está.
  - **No Bloqueante**: El proceso que realiza recibir un mensaje especifica un tiempo máximo de espera del mensaje.

    Recibir(buzón, mensaje, tiempo_espera)

Mensajes: estructura

- Intercambio de información:
  - Por valor: Se realiza una copia del mensaje desde el espacio de direcciones del receptor.
  - Por referencia: Se transmite sólo un puntero al mensaje.
- Clasificación
  - Longitud fija
  - Longitud variable
  - De tipo definido



Interbloqueos: Definición y caracterización

- **Definición 1**: Un conjunto de procesos está en un interbloqueo si cada proceso está esperando un recurso que sólo puede liberar otro proceso del conjunto.
  - Los procesos adquieren algún recurso y esperan a que otros recursos retenidos por otros procesos se liberen.

```
void Proceso1()
{
   . . .
   P(S1)
   P(S2)
   . . .
   V(S2)
   V(S1)
}
```

```
void Proceso2()
{
   . . .
   P(S2)
   P(S1)
   . . .
   V(S1)
   V(S2)
}
```

Interbloqueos: Definición y caracterización

- **Definición 2**: Se dice que el estado de un sistema se puede reducir por un proceso P si se pueden satisfacer las necesidades del proceso con los recursos disponibles.
- **Definición 3**: Se dice que un sistema está en un estado seguro si el sistema puede asignar todos los recursos que necesitan los procesos en algún orden.

Interbloqueos: Definición y caracterización

- **Caracterización del interbloqueo:** Condiciones necesarias para que se dé un interbloqueo:
  - Exclusión mutua
  - Retención y espera
  - No existencia de expropiación
  - Espera circular
- Un sistema está libre de interbloqueos si existe una secuencia de reducciones del estado actual del sistema que incluye a todos los procesos, o si se encuentra en un estado seguro.

Interbloqueos: Definición y caracterización

- Descripción del estado de un sistema:
  - Representación matricial
  - Representación gráfica
- Representación matricial: Para representar el estado del sistema se usan dos matrices y un vector: matriz de solicitud S, matriz de asignación A y vector E con la cantidad de elementos de cada tipo de recurso.

A [i, j] ≡ Cantidad de elementos del recurso j que tiene asignado el proceso i

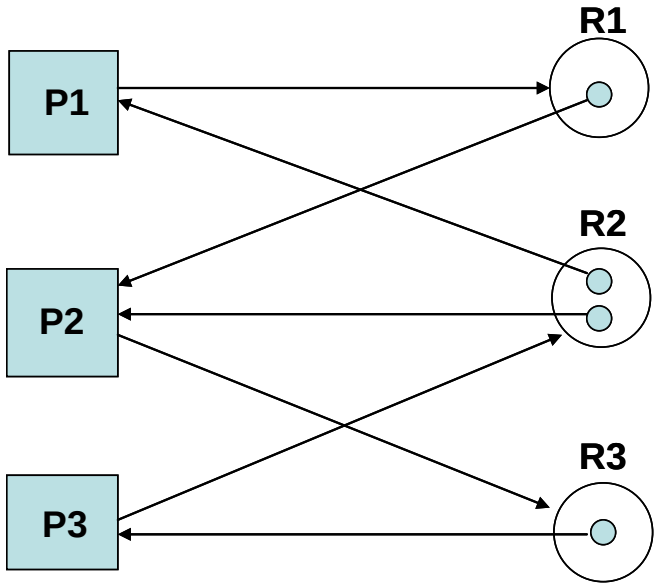
S [i, j] ≡ Cantidad de elementos del recurso j que solicita el proceso i

E[i] ≡ Cantidad de elementos del recurso i

Interbloqueos: Definición y caracterización: Ejemplo

- Sea un sistema formado por tres procesos: P1, P2 y P3; y los recursos siguientes: una impresora R1, dos unidades de disco R2 y una cinta R3.
- Dada la siguiente situación:
  - El proceso P1 posee uno de los recursos R2 y solicita R1
  - El proceso P2 posee uno de los recursos R2 y un recurso R1 y solicita el recurso R3.
  - El proceso P3 posee el recurso R3
  - y solicita el recurso R2.
- Representaciones matricial y gráfica:

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad E = (1, 2, 1)$$



Interbloqueos: Estrategias de actuación

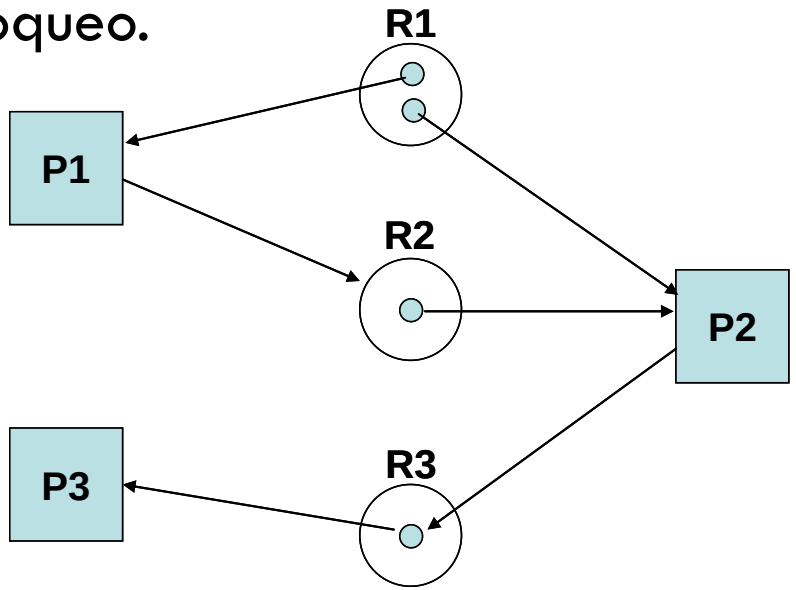
- **Prevención:** Evitar cualquier posibilidad que pueda llevar a una situación de interbloqueo fijando una serie de restricciones
- **Predicción:** Evitar el interbloqueo analizando la información disponible y los recursos que necesitará cada proceso
- **Detección:** No se establecen restricciones y el sistema se limita a detectar situaciones de interbloqueo
- **No actuación:** Se ignora la presencia de interbloqueos

Interbloqueos: Prevención

- Evitar una de las condiciones del interbloqueo
  - **Retención y espera:** Un proceso con un recurso no puede pedir otro.
  - **No existencia de expropiación:** Permitir la expropiación de recursos no utilizados.
  - **Espera circular:** Se solicitan los recursos según un cierto orden establecido.
- Afecta al rendimiento del sistema:
  - Puede provocar infrautilización de recursos
  - Puede provocar esperas muy dilatadas de los procesos

Interbloqueos: Detección

- Se comprueba si se ha producido un interbloqueo
  - Definir intervalos de activación del algoritmo de detección
  - Detectar los procesos a los que afecta el interbloqueo
- Definir estrategia de recuperación del sistema
- Si no existen ciclos: No hay interbloqueo
- Existen ciclos: Puede existir interbloqueo.
  - Si sólo hay un elemento por cada tipo de recurso, la existencia de un ciclo es condición necesaria y suficiente para el interbloqueo.
  - Si hay algún camino que no sea ciclo, que sale de alguno de los nodos que forman el ciclo, entonces no hay interbloqueo.



Interbloqueos: Recuperación

- Romper el interbloqueo para que los procesos puedan finalizar su ejecución y liberar los recursos.
  - Reiniciar uno o más procesos bloqueados

  Considerar:
    - Prioridad del proceso
    - Tiempo de procesamiento utilizado y el que resta
    - Tipo y número de recursos que posee
    - Número de recursos que necesita para finalizar
    - Número de procesos involucrados en su reiniciación.
  - Expropiar los recursos de algunos de los procesos bloqueados

Interbloqueos: Soluciones combinadas

- Agrupar los recursos en clases disjuntas
- Se evita el interbloqueo entre las clases
- Usar en cada clase el método más apropiado para evitar o prevenir en ella el interbloqueo