Semáforos y Monitores

Soluciones al problema de exclusión mutua

Roberto Gómez C.

El problema del productor/consumidor

- Dos procesos comparten un almacen (buffer) de tamaño fijo.
  - productor: coloca información en el almacen (buffer)
  - consumidor: obtiene información del almacen
- Problema:
  - productor desea colocar algo en el almacen lleno
  - consumidor desea obtener en el almacen vació
- Solución
  - irse a dormir cuando el almacen este lleno (productor) o cuando se encuentre vacio (consumidor)

Roberto Gómez C.

Planteamiento de la solución

- Dos variables compartidas por los procesos:
  - cont: número elementos en el almacen (buffer)
  - N: número máximo elementos (tamaño buffer)
- Actividades productor:
  - verificar si count = N
    - si es verdad => a dormir
    - si no es verdad => añadir un elemento
- Actividades consumidor:
  - verificar si count = 0
    - si es verdad => a dormir
    - si no es verdad => quitar un elemento

Primitivas de la solución

- Dos primitivas de comunicación son usadas entre los procesos que bloquean el CPU.
- Primitiva dormir (sleep)
  - provoca el bloqueo de quien hizo la llamada
  - será suspendido hasta que alguien lo despierte
- Primitiva despertar (wakeup)
  - despierta al proceso que invocó una primitiva sleep
  - tiene un parámetro: el proceso a despertar

Roberto Gómez C.

Solución problema productor/consumidor

```
#define N 100                   /* Número de espacios en el almacen (buffer) */
int   cont = 0;                 /* Número de elementos en el almacen (buffer) */

void productor( )
 {
    while (TRUE)  {             /* ciclo infinito                  */
       produce_elem(&elemento); /*  genera el siguiente elemento   */
       if (cont == N)           /*  si el almacen (buffer) esta lleno */
           sleep();             /*  entonces  se duerme            */
       intro_elem(elemento);    /*  colocar elemento en almacen    */
       cont = cont+1            /*  incrementa conta. elementos alma. */
       if (cont == 1)           /*  estaba vacio el almacen (buffer) */
            wakeup(consumidor);
    }
}
```

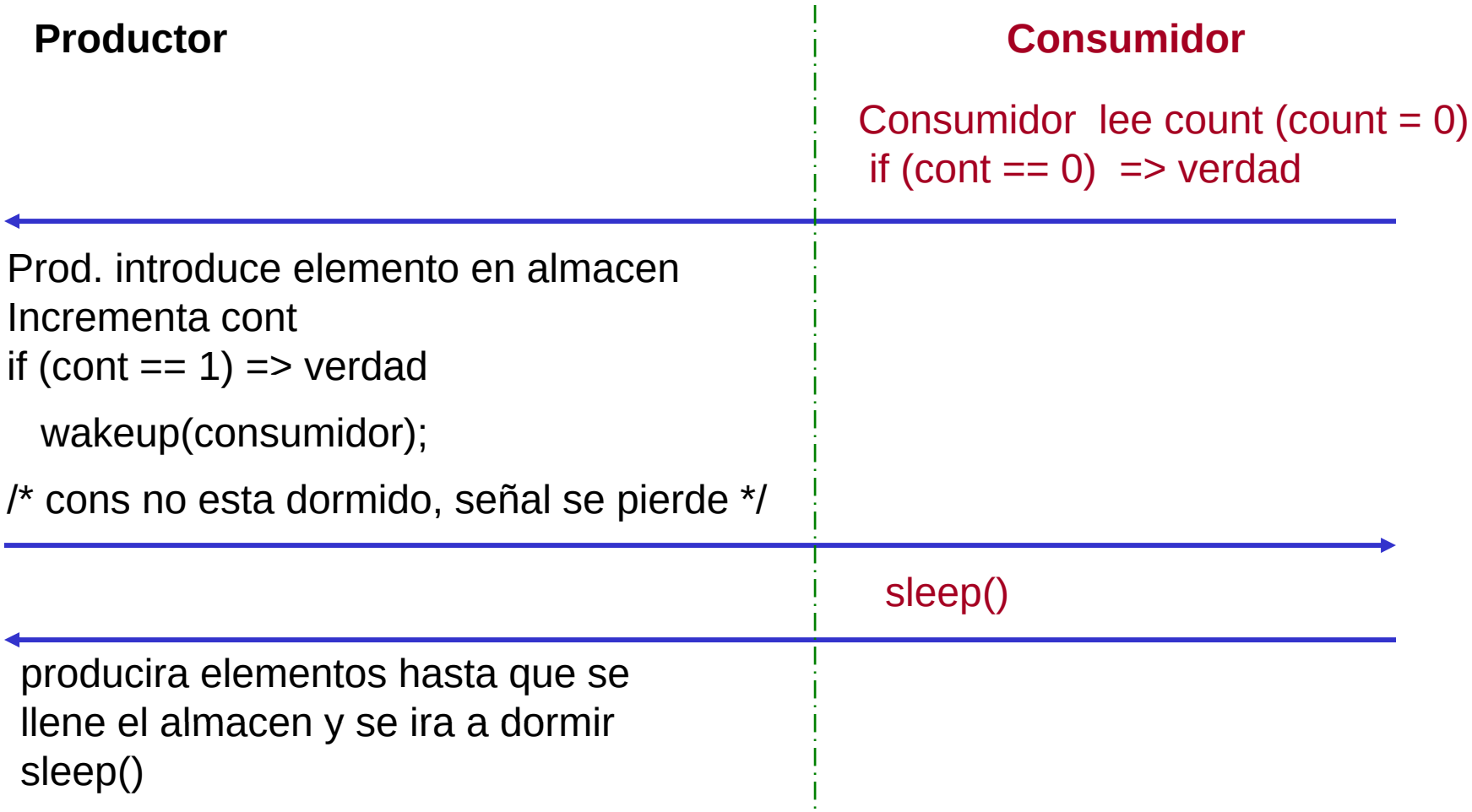
Solución problema productor/consumidor

```
void consumidor( )
{

    while (TRUE) {                  /*  ciclo infinito                    */
        if (cont == 0)              /*  si el almacen (buffer) esta vacío */
            sleep();                /*  entonces  se duerme               */
        retira_elem(&elemento);     /*  retira elemento del almacen       */
        cont = cont - 1;            /*  decrementa conta. elementos alma  */
        if (cont == N-1)            /*  estaba lleno el almacen (buffer)  */
            wakeup(productor);
        consume_elem(elemento);     /*  consume el elemento               */
    }
}
```

Problemas de la solución

- *Almacen vacío*



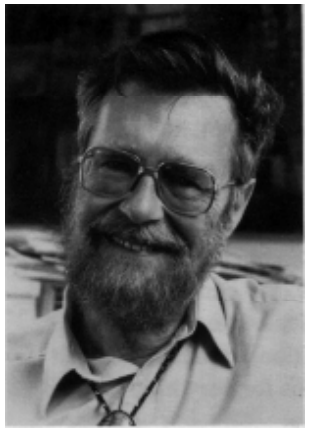
AMBOS DORMIRAN POR SIEMPRE

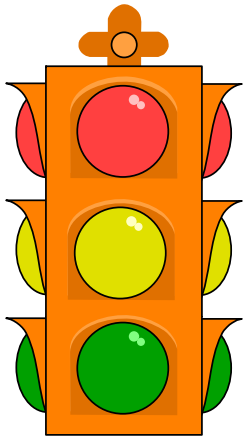
Posible solución

- Añadir un bit de espera de despertar
- Sin embargo si se tienen dos o más procesos un solo bit no es suficiente.

Los semáforos

- Definidos por Dijkstra en 1965
  - Dijkstra, E. W., *Cooperating sequential processes*, `Programming Languages', Genuys, F. (ed.), Academic Press, 1965





- Variable protegida cuyo valor solo puede ser accesado y alterado por dos operaciones: P(S) y V(S)
- Nombres provienen del holandes proberen (probar) y verhogen (incrementar).

Algo sobre Dijkstra

"In his later career, he seems to personify almost perfectly the class of computer scientist who is inclined to never touch a computer, to do his best to keep his students from touching computers, and to present computer science as a branch of pure mathematics. (Yes, this is a caricature.) "

"Dijkstra is among the brightest lights in theoretical computer science; I think it is generally conceded that he has often found the answer before otherfolk were aware of the problem.

-Muf Mastery

http://max.cs.kzoo.edu/~jatkins/dijkstra.htm

Las operaciones de semáforos

P(S)

if (S > 0) **then**

S:=S-1

else

esperar por S

V(S)

if (alguien espera por S) **then**

deja pasar al proceso

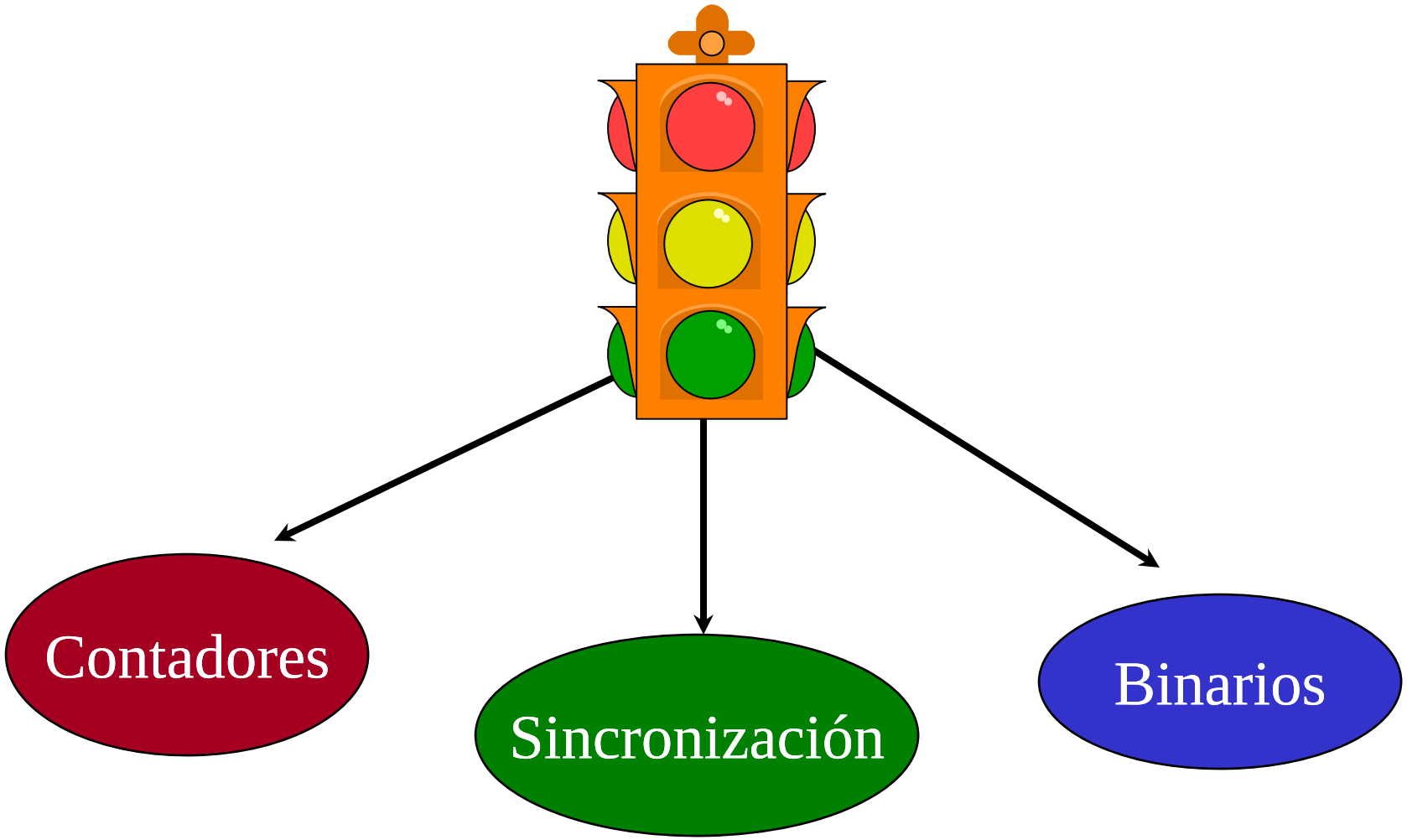
else

S:=S+1

Características principales

- Desbloqueando procesos
  - El semáforo informa a los procesos que se encuentran bloqueados.
  - El administrador de procesos que elige quien pasa a ejecución
- Atomicidad
  - Se garantiza que al iniciar una operación con un semáforo, ningún otro proceso puede tener acceso al semáforo hasta que la operación termine o se bloquee.
  - En ese tiempo ningún otro proceso puede simultáneamente modificar el mismo valor de semáforo

Tipos de semáforos



Semáforos generales o contadores

- Útiles cuando un recurso será asignado, tomándolo de un conjunto de recursos idénticos
- Semáforo es inicializado con el número de recursos existentes:
  - P(S) decrementa S en 1; indicando que un recurso ha sido suprimido del conjunto.
    - Si S = 0 entonces no hay más recursos y el proceso se bloquea
  - V(S) incrementa S en 1; indicando que un recurso ha sido regresado al conjunto.
    - Si un proceso esperaba por un recurso, éste se despierta.

Semáforos binarios

- Sólo pueden tomar dos valores: 0 o 1
- Generalmente se inicializan con un valor de 1.
- Son usados por dos o más procesos para garantizar que sólo uno puede entrar en sección crítica.
- Antes de entrar a sección crítica un proceso ejecuta un P(S) y un V(S) antes de salir de ella
- Cada proceso tiene la estructura siguiente:

```
while(1)
        P(entrar)
                <Sección Crítica>
        V(entrar)
do
```

Semáforos sincronía

- Solución varios problemas sincronización.
- Sean dos procesos concurrentes P_1 y P_2 que se encuentran corriendo:
  - P_1 con enunciado S_1
  - P_2 con enunciado S_2
  - Se desea que S_2 sea ejecutado después de que S_1 haya terminado.
  - *Solución:* Semáforo síncrono (inicializado en 0).

P_1:		P_2:	
	----------		----------
	----------		----------
	S_1		P(sincro)
	V(sincro)		S_2
	----------		----------
	----------		----------

Solución prod/cons semáforos (1)

```
#define N 100                   /* Número de espacios en el almacen (buffer) */
int   cont = 0;                 /* Número de elementos en el almacen (buffer) */

typedef int semaphore

semaphore ocupado = 1;
semaphore vacio = N;
semaphore lleno = 0;
```

Solución prod/cons semáforos (2)

```
void productor( )
{
   while (TRUE)  {                    /* ciclo infinito                          */
     produce_elem(&elemento);      /*  genera el siguiente elemento          */
     P(&vacio);                        /*  decrementa contador espacios vacíos */
     P(&ocupado);                    / *  entra en la sección crítica          */
     intro_elem(elemento);          /*  colocar elemento en el almacen      */
     V(&ocupado);                    /*  sale de la sección crítica            */
     V(&lleno);                       /*  incrementa cont entradas ocupadas   */
   }
}
```

Solución prod/cons semáforos (3)

```
void consumidor( )
{
   while (TRUE)   {              /*  ciclo infinito                          */
      P(&lleno);                 /*  decrementa cont. entradas ocupadas */
      P(&ocupado);               /*  entra en la sección crítica             */
      retira_elem(elemento);     /*  toma un elemento del almacen            */
      V(&ocupado);               /*  sale de la sección crítica              */
      V(&vacio);                 /*  incrementa contador entradas vacías */
      haz_algo(elemento);        /*  hace algo con el elemento               */
     }
}
```

Ejemplo ejecución (1)

PRODUCTOR

produce_elem(&elemento)

P(&vacio)

vacio=99

P(&ocupado)

ocupado= 0

intro_elem(elemento)

V(&ocupado)

ocupado=1

Temina quantum, cambio CPU

CONSUMIDOR

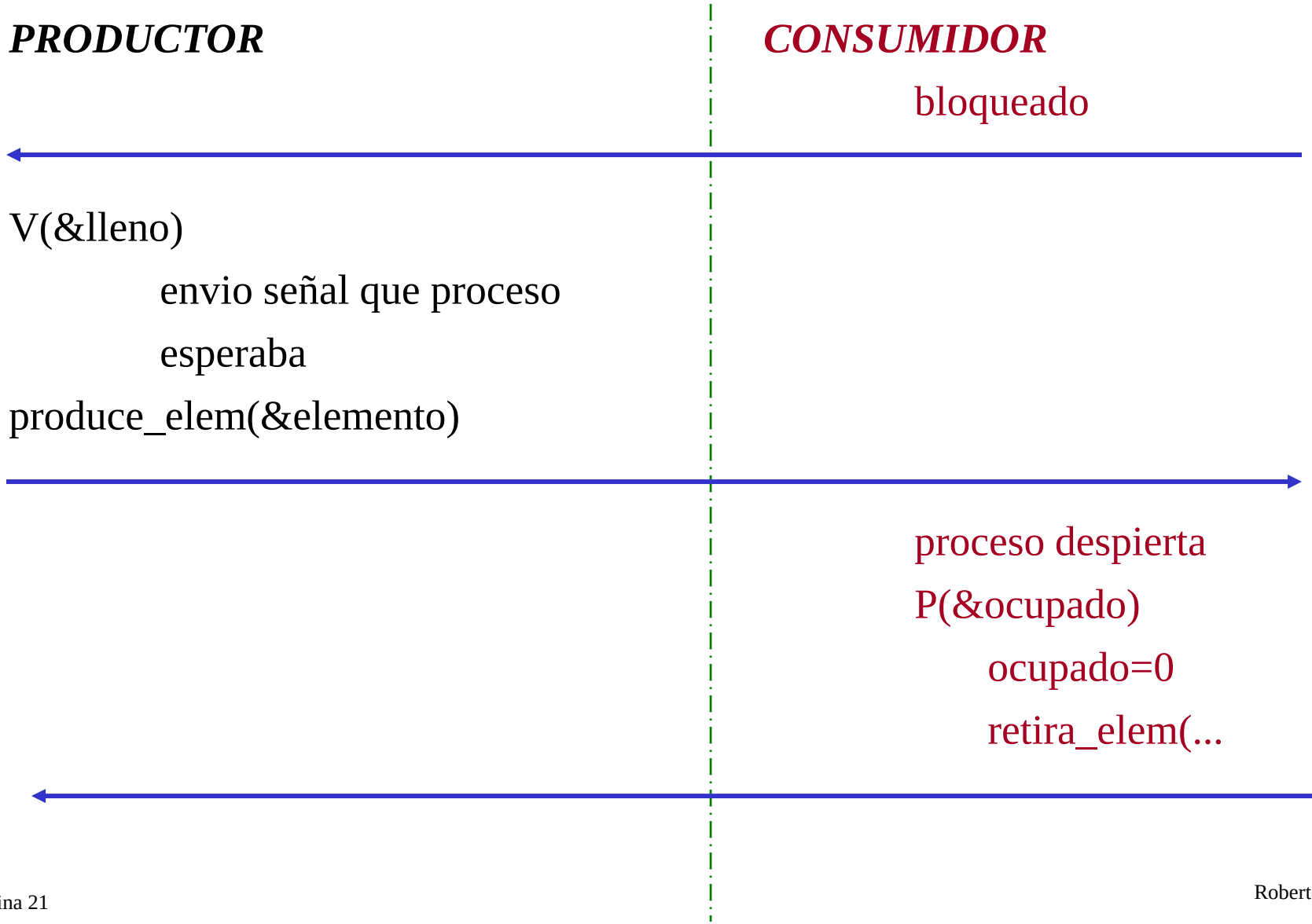
P(&lleno)

proceso

bloqueado

Temina quantum, cambio CPU

Ejemplo ejecución (2)



Ejemplo ejecución (3)

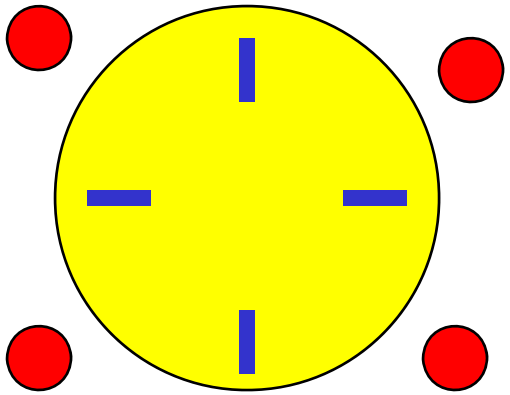
PRODUCTOR	***CONSUMIDOR***
P(&vacio)	
vacio=98	
P(&ocupado)	
proceso espera liberación	
de la variable ocupado	
	V(&ocupado)
	envio señal a quien
	esperaba ocupado
	V(&vacio)
	vacio=99
	haz_algo(elemento)
proceso despierta	
intro_elem(elemento)	
V(&ocupado)	
ocupado=1	

Problemas clásicos de exclusión mutua

- El problema de la cena de filósofos
- El problema del Barbero dormilón
- El problema de los fumadores
- El problema de la panadería
- El problema de los lectores/escritores

La cena de filosofos

Cinco filósofos se sientan en una mesa redonda. En el centro de la mesa hay un plato de arroz. El arroz es tan escurridizo que un filósofo necesita dos palillos para comerlo, pero cada filosofo solo posee un palillo, luego existe el mismo número de filósofos que de palillos. La vida de un filósofo consta de periodos alternados de comer y pensar. Cuando un filósofo siente hambre, intenta coger el palillo de la izquierda y si lo consigue, lo intenta con el de la derecha. Si logra asir dos palillos toma unos bocados y después deja los cubiertos y sigue pensando.



Solución con semáforos

```
#define N5                          /* número de filosofos*/
#define IZQUIERDA  (i+N-1)%N        /* numero del vecino izquierdo */
#define DERECHA  (i+1)%N            /* numero del vecino derecho */
#define PENSANDO 0                  /* filosofo pensando*/
#define HAMBRIENTO 1                /* filosofo intenta tomar palillos*/
#define COMIENDO 2                  /* filosofo comiendo*/

int state[N];                       /* arreglo almacena estado de cada uno */
semaphore mutex = 1;                /* exclusión mutua para seccion critica */
semaphore s[N];                     /* un semaforo por filosofo*/
```

Solución

```
void filosofo (int i)
/* i: numero de filosofo, de 0 a N-1 */
{
    while (TRUE) {
        pensando();
        toma_palillos(i);
        comiendo();
        dejar_palillos(i);
    }
}
```

```
toma_paliilos (int i)
/* i: numero de filosofo, de 0 a N-1 */
{
        /* intenta entrar a SC */
        P(mutex);
        /* indica que esta hambriendo */
        state[i] = HAMBRIENTO;
        /* intenta tomar palillos */
        test(i);
        /* sale de seccion critica */
        V(mutex);
        /* bloqueado si no adquiere palillos*/
        P(s[i]);
}
```

Solución

```
void dejar_palillos (int i)
/* i: numero de filosofo, de 0 a N-1 */
{
        P(mutex);
        state[i] = PENSANDO;
        /* verificar si los vecinos
           pueden comer */
        test(IZQUIERDA);
        test(DERECHA);
        V(mutex);
}
```

```
test (int i)
/* i: numero de filosofo, de 0 a N-1 */
{
    if  ((state[i] == HAMBRIENTO) &&
        (state[IZQUIERDA] != COMIENDO) &&
        (state[DERECHA] != COMIENDO) )  {
            state[i] = COMIENDO;
            V(s[i]);
    }
}
```

Problema Barbero Dormilón

Una barbería tiene una sala de espera con n sillas, y otra sala donde se encuentra el sillón de afeitado. Si no hay clientes a los que servir, el barbero se echa a dormir en el sillón. Si entra un cliente en la barbería y todas las sillas están ocupadas, el cliente se va. Si el barbero está ocupado afeitando, el cliente se sienta en una de las sillas disponibles. Si el barbero está dormido, el cliente despierta al barbero. Escribir un programa que coordine al barbero y los clientes mediante semáforos.

Problema de los fumadores

Considere un sistema con tres procesos fumadores y un proceso agente. Cada fumador esta continuamente enrrollando y fumando cigarrillos. Sin embargo, para enrrollar y fumar un cigarrillo, el fumador necesita tres ingredientes: tabaco, papel, y fósforos. Uno de los procesos fumadores tiene papel, otro tiene el tabaco y el tercero los fósforos. El agente tiene una cantidad infinita de los tres materiales. El agente coloca dos de los ingredientes sobre la mesa. El fumador que tiene el ingrediente restante enrrolla un cigarrillo y se lo fuma, avisando al agente cuando termina. Entonces, el agente coloca dos de los tres ingredientes y se repite el ciclo. Escriba un programa para sincronizar al agente y a los fumadores.

Problema Panadería de Lamport

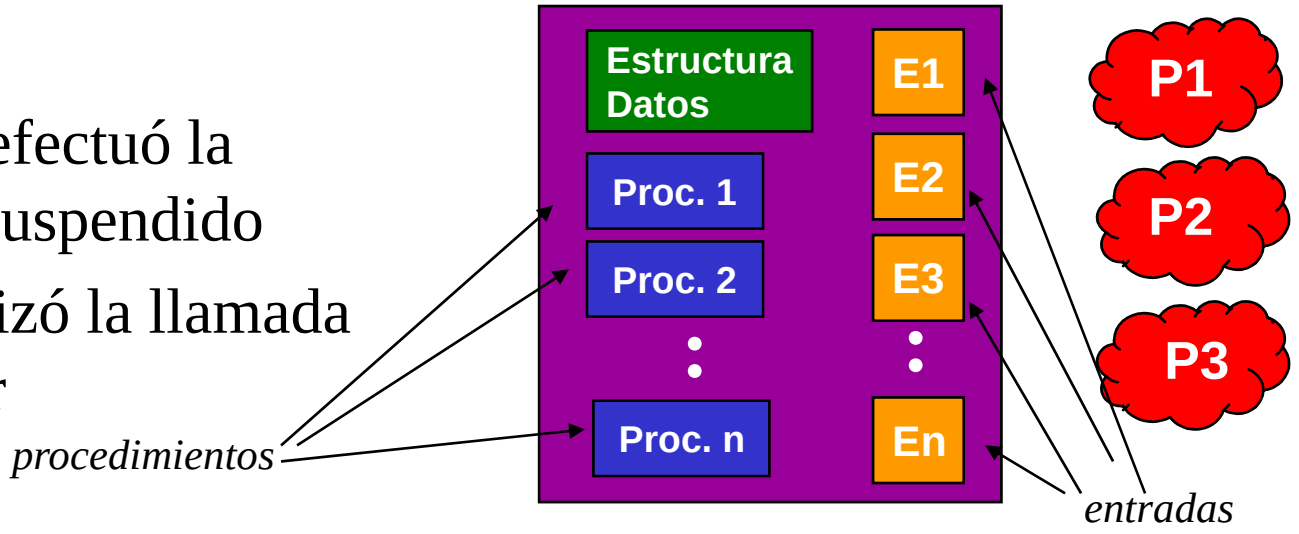
Una panadería tiene una variedad de panes y pasteles vendidos por n vendedores. Cada uno de los cuales toma un número al entrar. El cliente espera hasta oír su número. Cuando el vendedor se desocupa, llama al siguiente número. Escriba un procedimiento para los vendedores y otro para los clientes.

Los monitores

- Otra opción para la sincronización de procesos.
- Introducido por Hoare
  - C. A. R. Hoare's seminal research paper on monitors, "*Monitors: An Operating System Structuring Concept*," Communications of the ACM, Vol. 17, No. 10, October 1974, pp. 549-557
- Se trata de una construcción concurrente que contiene los datos y procedimientos necesarios para la asignación de un recurso compartido.
- Para lograr la asignación de un recurso una función del proceso debe llamar una entrada del monitor.

Procesos y monitores

- Procesos no tienen acceso directo a las estructuras de datos.
- Solo un proceso puede estar activo en un monitor en cada momento.
- Cuando se llama a un entrada la implementación monitor verifica si no hay otro proceso dentro del monitor:
  - *si:* proceso efectuó la llamada es suspendido
  - *no:* el que hizó la llamada puede entrar



Operaciones sincronía en procesos

- Para sincronizar procesos se usan variables de tipo condición y dos operaciones:

1. wait()

- proceso descubre que no puede continuar (almacen lleno) ejecuta *wait* en alguna variable de condición proceso se bloquea y permite que otro proceso entre al monitor

2. signal

- proceso que entra puede despertar a otro con una instrucción *signal* sobre la variable de condición que el otro proceso espera

Comentarios variables condición

- Las variables de condición NO son contadores.
- No se acumulan señales para su uso posterior (como los semáforos)
- Si una variable de condición queda señalada sin que nadie la espere, la señal se pierde.
- La instrucción WAIT debe aparecer antes de SIGNAL
  - regla que hace más sencilla la implementación

Monitores y problema prod/cons

- Exclusión mutua es automática,
  - el monitor garantiza que solo el productor o el consumidor estaran ejecutandose
- Si el almacen esta totalmente ocupado, el productor realizará una operación WAIT
  - se encontrará bloqueado hasta que consumidor ejecute un SIGNAL
- Si el almacen esta totalmente vacio el consumidor realizará una operación WAIT
  - se encontrará bloqueado hasta que productor ejecute un SIGNAL

Solución prod/cons con monitores

```
monitor ProdCons
    condition lleno, vacio;
    integer  cont, N;

    procedure producir;
    begin
       if (cont = N) then
            wait(lleno);
       introducir_elemento;
       cont := cont +1;
       if (cont = 1) then
            signal(vacio);
    end;

    procedure retirar;
    begin
       if (cont = 0) then
            wait(vacio)
       retirar_elemento;
       cont := cont-1
       if (cont = N-1) then
            signal(lleno);
     end;

cont :=0; N=100;
end monitor;
```

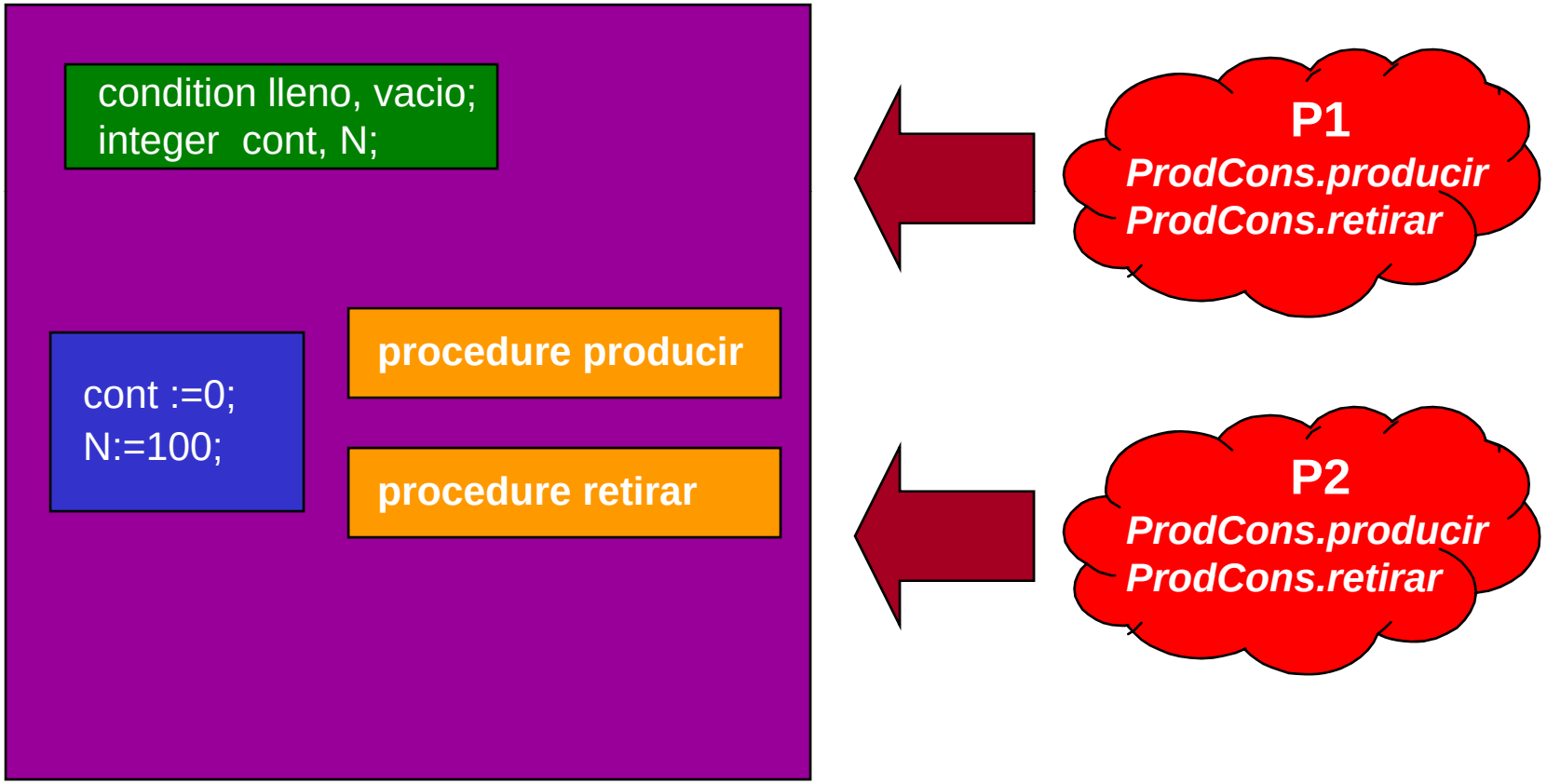
Llamando entradas monitor

```
procedure productor;
begin
   while true do
   begin
      producir_elemento;
     ProdCons.producir;
    end
end;
```

```
procedure consumidor;
begin
    while true do
    begin
       ProdCons.retirar;
       consumir_elemento;
     end
end;
```

Esquema solución prod/cons con monitores

monitor ProdCons



Problema lectores/escritores

- En algunos sistemas computacionales se cuenta con dos tipos de procesos
  - procesos lectores: leen datos
  - procesos escritores: escriben datos
- Un ejemplo de este tipo de sistema es uno de reservación aerea.
- Muchos lectores pueden acceder la base de datos a la vez
  - varios lectores no cambian el contenido de la base de datos
- Un escritor debe tener acceso exclusivo
  - un escrito puede modificar los datos

Planteamiento solución lec/esc

- Si el escritor esta activo
  - ningún lector o escritor puede estar activo
- Esta exclusión necesita realizarse solo a nivel registro (i.e. no toda la base de datos)
  - no es necesario garantizar a un escritor el acceso exclusivo a la base de datos completa
- Problema propuesto por Courtois, Heymans y Parnas:
  - Courtois, P.J., Heymans, F., and Parnas, D.L. *Concurrent Control with Readers and Writers*, CACM, Vol. 14, No. 10, October 1971, pp.667-668

Solución lectores/escritores

```
monitor LectoresEscritores
var
    lectores:                integer;
    alguien_escribe:                 boolean;
    puede_leer, puede_escribir:   condition;

    procedure comenzar_lectura;
    begin
        if   ( (alguien_escribe) o en_cola(puede_escribir) ) then
            wait(puede_leer)
        lectores:= lectores + 1;
        signal(puede_leer)
    end;

    procedure terminar_lectura;
    begin
            lectores:= lectores - 1;
            if   (lectores = 0 ) then
                signal(puede_escribir);
    end;
```

Solución lectores/escritores

```
procedure comenzar_escritura;
begin
        if   ( (lectores > 0) or (alguien_escribe) ) then
            wait(puede_escribir)
        alguien_escribe:=true;
end;

procedure terminar_escritura;
begin
        alguien_escribe:=false;
        if  ( en_cola(puede_leer))  then
            signal(puede_leer)
        else
            signal(puede_escribir)
end
```

Solución lectores/escritores

```
begin
    lectores:=0;
    alguien_escribe:= false;
end;
```

Ultimo comentario sobre solución

- El monitor lectores/escritores puede usarse para controlar el acceso a una base de datos completa, un subconjutno de la base consistente de varios registros o aún sobre un simple registro