

SISTEMAS OPERATIVOS

Procesos concurrentes y problemas en la
comunicación y la sincronización

Jesús Carretero Pérez
Alejandro Calderón Mateos
José Daniel García Sánchez
Francisco Javier García Blas
José Manuel Pérez Lobato
María Gregoria Casares Andrés

Contenido

- **Concurrencia.**
- Condiciones de carrera.
- Exclusión mutua y sección crítica.
- Semáforos.
- El problema del productor consumidor.
- El problema de los lectores escritores.

Proceso concurrente

Dos procesos son concurrentes cuando se ejecutan de manera que sus intervalos de ejecución se solapan.



Si hay concurrencia

No hay concurrencia

Tipos de concurrencia

● Concurrencia aparente: Hay más procesos que procesadores.

- Los procesos se multiplexan en el tiempo
- Pseudoparalelismo

1 CPU



2 CPUs: Si hay concurrencia



Tipos de concurrencia

- **Concurrencia real:** Cada proceso se ejecuta en un procesador.
 - Se produce una ejecución en paralelo.
 - Paralelismo real. 4 CPUs



Modelos de programación concurrente

- Multiprogramación con un único procesador
 - El sistema operativo se encarga de repartir el tiempo entre los procesos (Multiplexación: planificación apropiativa).
- Multiprocesador
 - Se combinan paralelismo real y multiplexación.
 - Normalmente más procesos que CPU's.
- Sistema distribuido en varios computadores
- Varios computadores conectados por red.

Ventajas de la ejecución concurrente

- Facilita la programación.
 - Diversas tareas se pueden estructurar en procesos separados.
 - Servidor Web: Un proceso encargado de atender a cada petición.
- Acelera la ejecución de cálculos.
 - División de cálculos en procesos ejecutados en paralelo.
Ejemplos: Simulaciones, Evaluación de carteras financieras, Mercado eléctrico.
- Mejora la interactividad de las aplicaciones.
 - Se pueden separar las tareas de procesamiento de atención a usuarios, ejemplo: Impresión y edición.
- Mejora el aprovechamiento de la CPU.
 - Se aprovechan las fases de E/S de una aplicación para procesamiento de otras.

Tipos de procesos concurrentes

- Independientes

Procesos que se ejecutan concurrentemente pero sin ninguna relación.

- No necesitan comunicarse.
- No necesitan sincronizarse.

Ejemplo: Dos intérpretes de comandos de dos usuarios ejecutados en distintos terminales

- Cooperantes

Procesos que se ejecutan concurrentemente con alguna interacción entre ellos.

- Pueden comunicarse entre si.
- Pueden sincronizarse.

Ejemplo: Servidor de transacciones organizado en proceso receptor y procesos de tratamiento de peticiones.

Interacciones entre procesos

- Acceso a recursos compartidos
 - Procesos que comparten un recurso
 - Procesos que compiten por un recurso

Ejemplo: Servidor de peticiones en la que distintos procesos escriben en un registro de actividad (log).
- Comunicación.
 - Procesos que intercambian información.

Ejemplo: Receptor de peticiones debe pasar información a proceso de tratamiento de petición.
- Sincronización.
 - Un proceso debe esperar a un evento en otro proceso.

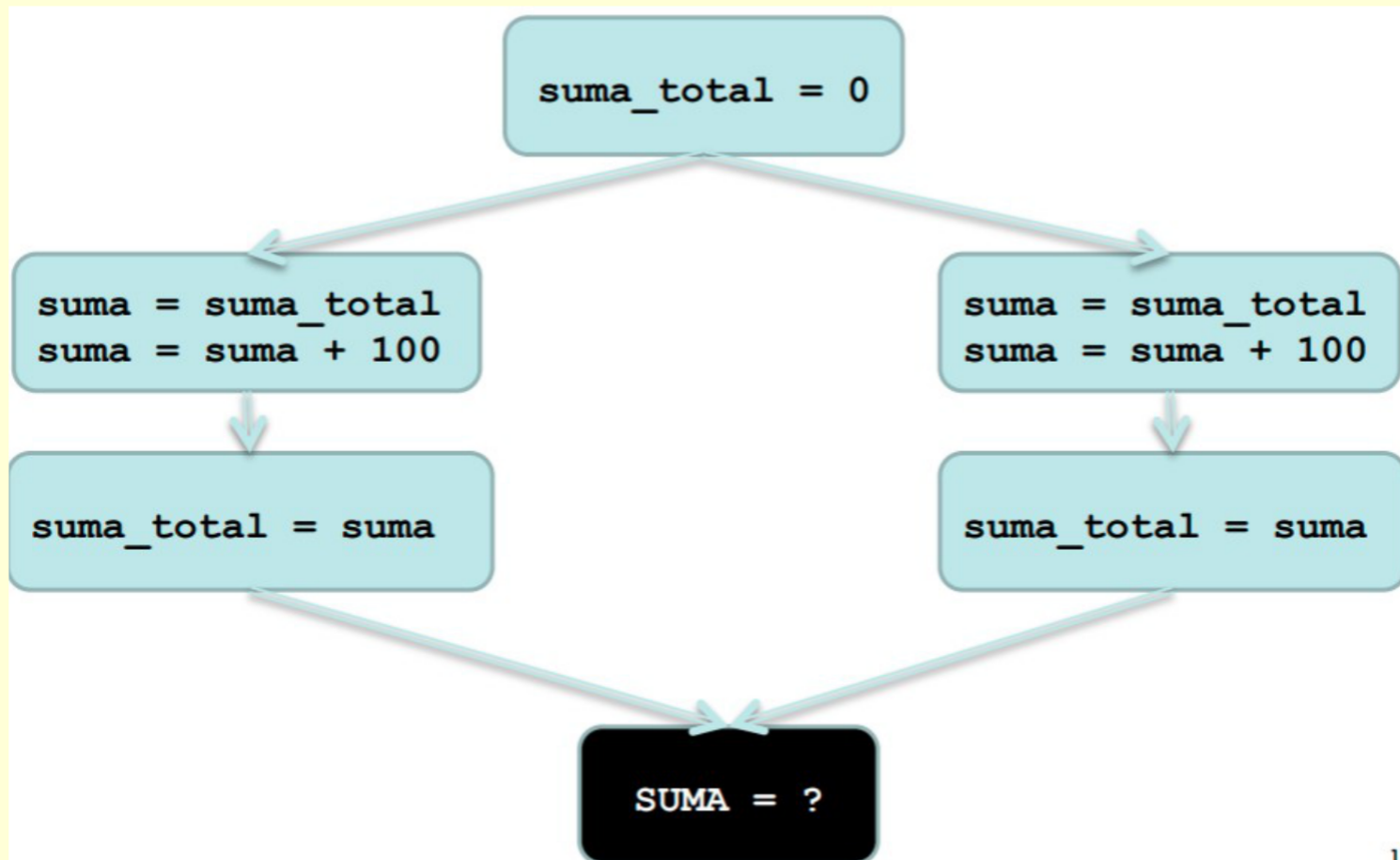
Ejemplo: Un proceso de presentación debe esperar a que todos los procesos de cálculo terminen.

Contenido

- Concurrencia.
- Condiciones de carrera.
- Exclusión mutua y sección crítica.
- Semáforos.
- El problema del productor consumidor.
- El problema de los lectores escritores.

Lea el programa 02Thread.c para aprender a usar
thrad

Condiciones de carrera



Condición de carrera: Es un posible error de cálculos que se puede producir en una concurrencia de procesos a causa de fallas en la sincronización del uso de memoria (`suma_tota`).

Secuencias posibles

```
suma_total = 0
```

```
suma1 = suma_total  
suma1 = suma1 + 100
```

```
suma_total = suma1
```

```
suma2 = suma_total  
suma2 = suma2 + 100
```

```
suma_total = suma2
```

200

```
suma_total = 0
```

```
suma1 = suma_total  
suma1 = suma1 + 100
```

```
suma2 = suma_total  
suma2 = suma2 + 100
```

```
suma_total = suma2
```

```
suma_total = suma1
```

100

Condiciones de carrera (ver 03aConCarr.c)

// compilar: gcc suma.c -lpthread

Ejecutar: ./a.out

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <pthread.h>
```

```
#define NUMTH 10
```

```
int suma_total = 0;
```

```
// todos los threads comparten suma_total
```

```
void suma() {
```

```
// suma 100 a suma_total
```

```
    int i;
```

```
    int suma=suma_total;
```

```
    suma = suma + 100;
```

```
    for (i=0;i<10000;i++);    // loop consumidor de tiempo, no hace nada
```

```
    suma_total=suma;
```

```
}
```

```
int main() {
```

```
    pthread_t th[NUMTH];    // declara 10 thread
```

```
    int i;
```

```
    // crea 10 threads y ejecuta suma
```

```
    for (i=0;i<NUMTH;i++) pthread_create(&th[i], NULL, (void *)suma, NULL);
```

```
    for (i=0;i<NUMTH;i++) pthread_join(th[i], NULL); // El proceso principal espera a los 10
```

```
    printf("Suma Total=%d\n", suma_total);
```

```
}
```

Resultados

Suma Total=700

Suma Total=1000

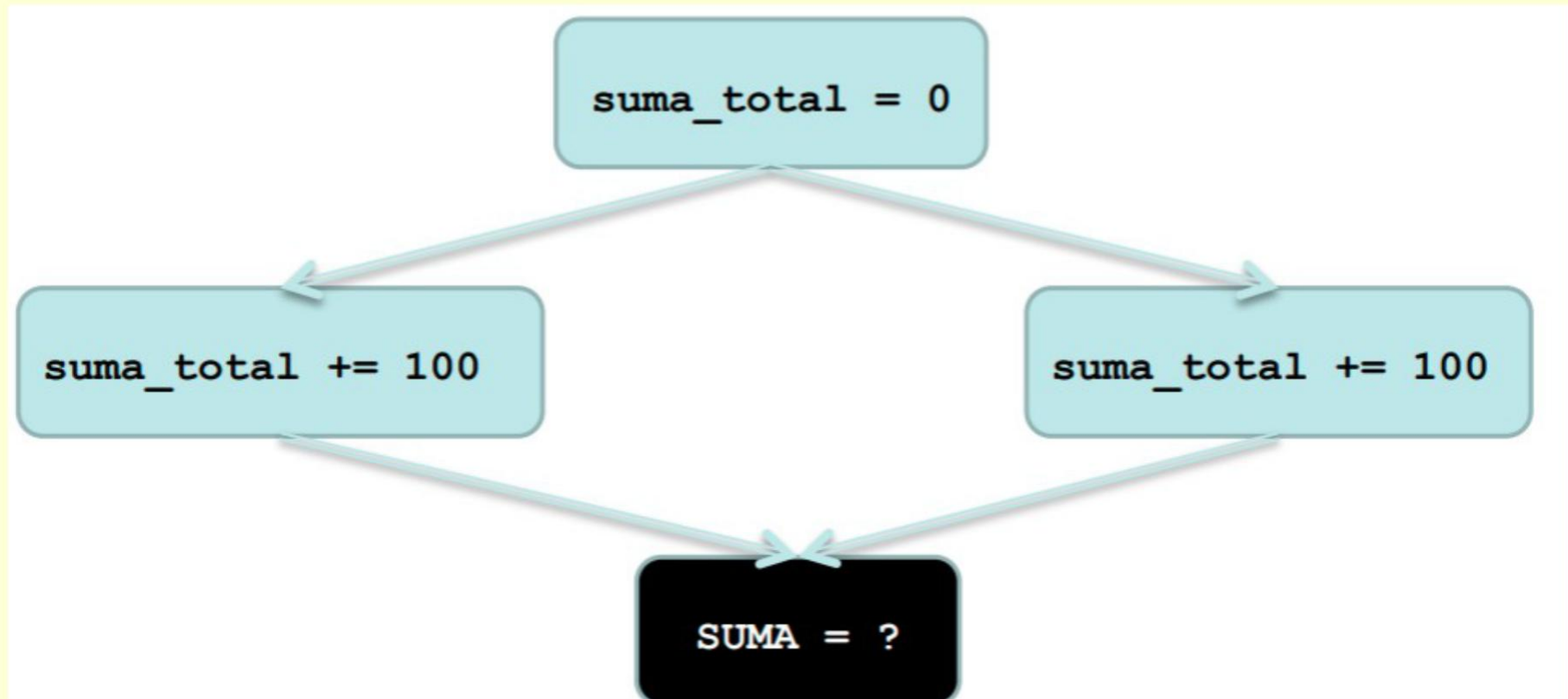
Suma Total=1000

Suma Total=900

Suma Total=800

- Cada vez se obtiene un resultado distinto.
- No se sabe cual es el resultado correcto
- ¿Qué está pasando?

Otra alternativa



/*03bConCarr.c: Se crean y ejecutan 10 threads (=procesos) que suman 100 cada uno, se espera Suma= 1000, y es así. Compilar: c++ 03bConCarr.c -lpthread */

```
#include <thread>           // incluye librería
#include <iostream>
#define NUMTH 10
using namespace std;
int suma_total = 0;          // todos los threads comparten suma_total
void suma() {
    int i;
    for (i=0;i<100000;i++); // consumidor de tiempo
    suma_total +=100;
}
int main() {
    thread th[NUMTH];
    int i;
    for (i=0;i<NUMTH;i++) th[i] = thread(suma); // Crea 10 threads
    for (i=0;i<NUMTH;i++) th[i].join();         // espera a los 10 threads
    printf("Suma=%d\n", suma_total);
}
```

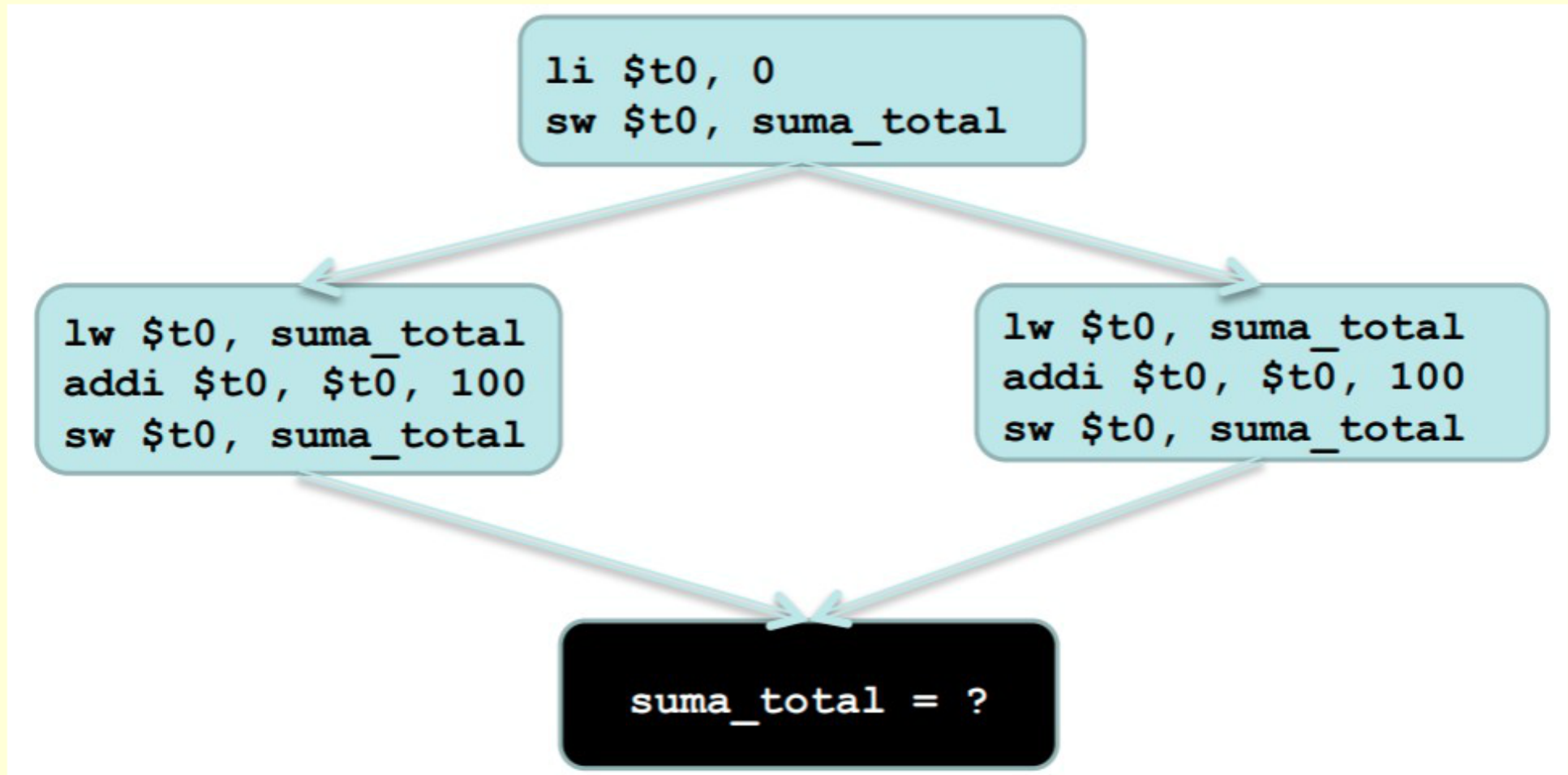
/* SALIDAS: Suma; esperamos que Suma= 1000. No hay condición de carrera

Suma=1000

Suma=1000

¿Puede darse condición de carrera?; Si, pero es menos frecuente

Instrucciones máquina



Solución a Condiciones de carrera

- Es necesario garantizar que el orden de ejecución de los procesos no afecte al resultado.
- El funcionamiento de un proceso y su resultado debe ser independiente de su velocidad relativa de ejecución con respecto a otros procesos.
- **Solución: Conseguir que un conjunto de instrucciones se ejecute de forma atómica: se ejecuta todo o nada:**

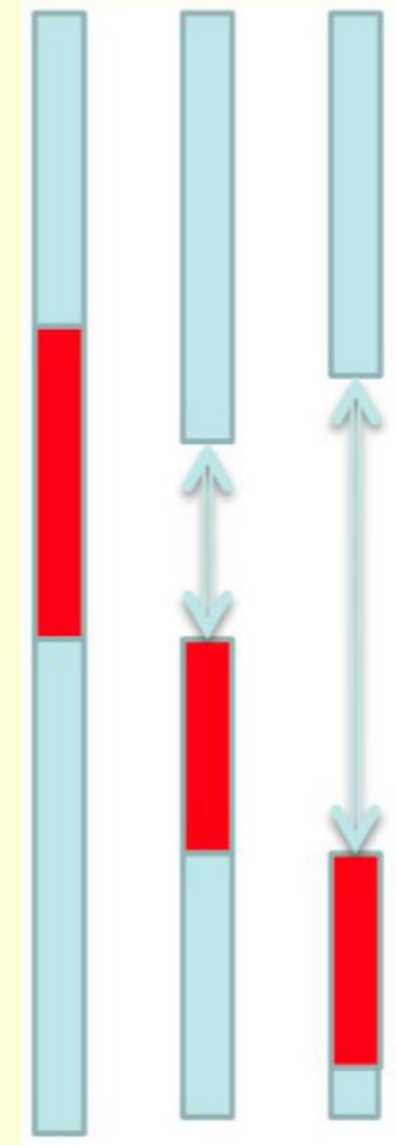
Exclusión mutua

Contenido

- Concurrencia.
- Condiciones de carrera.
- Exclusión mutua y sección crítica.
- Semáforos.
- El problema del productor consumidor.
- El problema de los lectores escritores.

Solución con exclusión mutua

- **Sección crítica**: Segmento de código que manipula un **recurso** y debe ser ejecutado de forma atómica (todo o nada).
- Se asocia a un **recurso**, un **mecanismo** de gestión de exclusión mutua:
Solamente un proceso puede ejecutar la sección crítica.

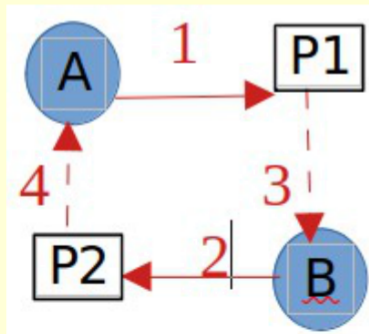


Problema 1 de la sección crítica

- Puede producir **Interbloqueos**.

➤ Se produce cuando los **procesos requieren más de un recurso**, y los recursos se **asignan a un solo proceso**:

- 1: El proceso P1 entra en la sección crítica del recurso A.
- 2: El proceso P2 entra en la sección crítica del recurso B.
- 3: El proceso P1 solicita el recurso B (queda a la espera de que P2 la abandone).
- 4: El proceso P2 solicita el recurso A (queda a la espera de que P1 la abandone)



Ninguno puede avanzar

Problema 2 de la sección crítica

- Puede haber **Inanición**: Un proceso queda indefinidamente bloqueado en espera de entrar en una sección crítica.
 - El proceso P1 entra en la sección crítica del recurso A.
 - El proceso P2 solicita entrar en la sección crítica del recurso A.
 - El proceso P3 solicita entrar en la sección crítica del recurso A.
 - El proceso P1 abandona la sección crítica del recurso A.
 - El proceso P2 entra en la sección crítica del recurso A.
 - El proceso P1 solicita entrar en la sección crítica del recurso A.
 - El proceso P2 abandona la sección crítica del recurso A.
 - El proceso P1 entra en la sección crítica del recurso A.
 - ...

El proceso P3 nunca consigue entrar en la sección crítica del recurso A

Inanición



El proceso P3 nunca llega a conseguir entrar en la sección crítica

Condiciones para la **solución** de la exclusión mutua

- Se permite que solo un proceso ejecute la sección crítica de un recurso.
- **No debe ser posible que un proceso que solicite acceso a una sección crítica sea postergado indefinidamente.**
- Cuando ningún proceso esté en una sección crítica, cualquier proceso que solicite su entrada lo hará sin demora.
- No se puede hacer suposiciones sobre la velocidad relativa de los procesos ni el número de procesadores.
- Un proceso permanece en su sección crítica durante un tiempo finito.

Sección crítica: Mecanismo de sincronización

- Cualquier mecanismo que solucione el problema de la sección crítica debe proporcionar sincronización entre procesos:
 - Cada proceso debe **solicitar permiso** para entrar en la sección crítica
 - Cada proceso debe **indicar cuando abandona** la sección crítica.

Código no crítico ...

...

<Entrada en sección crítica>

Código de sección crítica

<Salida de sección crítica>

...

Código no crítico

Alternativas de implementación

- Desactivar **interrupciones**.
 - El proceso no sería interrumpido.
 - Solamente sería válido en sistemas monoprocesador.
- Instrucciones máquina (en la lógica del programa fuente)
 - Test prueba and set (hace) y swap (deshace).
 - Implica espera activa.
 - Son posibles inanición e interbloqueo.
- Soporte del sistema operativo: que responde a instrucción máquina.

Solución de Peterson (solo para 2 procesos)

Código en los procesos:

- Asume que instrucciones LOAD y STORE son atómicas, no interrumpibles.
- Los 2 procesos comparten 2 variables:

int turno; Boolean flag[2]

Turno: indica **quien entrará en la sección crítica**.

Flag: indica si un proceso está listo para entrar en la sección crítica.

– flag[i] = true implica que P_i está listo.

Programa limitado que resuelve el interbloqueo, **con espera**

ocupada/activa: No vale la pena estudiarlo, porque solo resuelve para dos procesos.

Contenido

- Concurrencia.
- Condiciones de carrera.
- Exclusión mutua y sección crítica.
- **Semáforos.**
- El problema del productor consumidor.
- El problema de los lectores escritores.

Perspectiva Operativa de: SO + Programación/ejecución de programación:

SO: La planificación de procesos optimiza la cantidad de procesos en ejecución; **pero genera el problema de condición de carrera (errores en los cálculos) → solución:**

Programación: Exclusión mutua de procesos en secciones críticas (uso de semáforos); **pero genera problema de interbloqueo de procesos, y de inanición → Solución:**

Gestión de recursos: interbloqueo e inanición (lección 5)

Solución: Semáforos de Dijkstra

Sincronización de procesos mediante un mecanismo de señalización → semáforo.

Un semáforo es un **mecanismo** de sincronización de acceso a una sección crítica que opera **recursos compartido por varios procesos**. Se puede implementar con una variable entera $\underline{s} = n$ (n : capacidad máxima de la sección crítica) y tres operaciones asociadas:

- 1) **init** : $s = n$ // $n > 0$. Si $s < 0$: $-s$ = largo de la cola de espera
- 2) **wait** : $s = s - 1$; if($s < 0$) este proceso se bloquea;
else: este proceso entra a la sección crítica;
- 3) **signal**: $s = s + 1$; if($s \leq 0$) este proceso sale de sección crítica
y desbloquee a otro proceso;
else: no hace nada.

Código de proceso que implementa el **Mecanismo**

- Semáforo asociado a la sección crítica de un recurso.

- (1) Semáforo iniciado a 1.

- (2) Entrada en la sección crítica:

- (3) Salida de la sección crítica:

```
sem_t s;
```

Código no crítico

```
s_init(&s, ,1);
```

Código no crítico

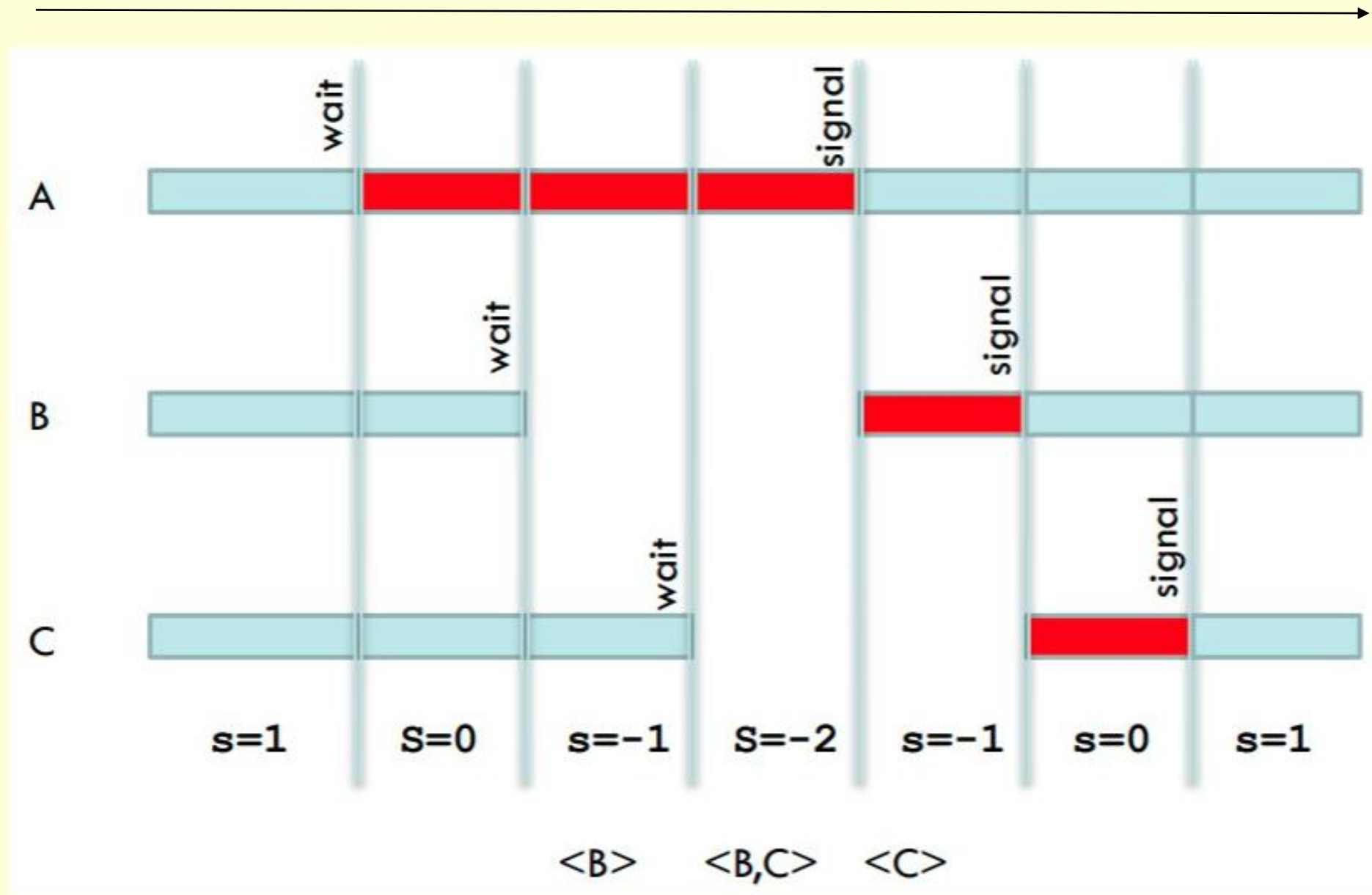
```
s_wait(&s);
```

Código atómico de
sección crítica

```
s_signal(&s);
```

Código no crítico

Sección crítica y aplicación de semáforo: procesos A, B y C 31



Contenido

- Concurrencia.
- Condiciones de carrera.
- Exclusión mutua y sección crítica.
- Semáforos.
- El problema del productor consumidor.
- El problema de los lectores escritores.

El problema del productor-consumidor

- El productor produce elementos de información.
- El consumidor consume elementos de información.
- Se tiene un espacio de almacenamiento intermedio.



Búfer v, infinito y compartido, sin semáforo:

- Productor

```
for (;;) {
    x= producir();
    v[fin] = x;
    fin++;
}
```

Hay que introducir sincronización

- Consumidor

```
for (;;) {
    while (inicio==fin)
        {}
    y=v[inicio];
    inicio++;
    procesar(y);
}
```

Espera activa



Programa 05aProdCons.c: Productor-Consumidor con condición de carrera y espera activa

Búfer infinito con semáforo *s* (recursos): *v*, inicio y fin

- Productor

```
semaforo  s=1
for (;;) {
    x= producir();
    semWait(s);
    v[fin] = x;
    fin++;
    semSignal(s);
}
```

- Consumidor

```
semaforo  s=1
for (;;) {
    while (inicio==fin)
        {}
    semWait(s);
    y=v[inicio];
    inicio++;
    semSignal(s);
    procesar(y);
}
```



**Espera
activa**

Programa 05bProdCons.c: Productor-Consumidor sin condición de carrera y espera activa

Búfer infinito n (recursos): inicio y fin

- Productor

```
semaforo s=1; semaforo n=0;
```

```
for (;;) {  
    x= producir();  
    semWait(s);  
    v[fin] = x;  
    fin++;  
    semSignal(s);  
    semSignal(n)  
}
```

- Consumidor

```
int m;  
for (;;) {  
    semWait(n);  
    semWait(s);  
    y=v[inicio];  
    inicio++;  
    semSignal(s);  
}
```



Programa 05cProdCons.c: Productor-Consumidor sin condición de carrera ni espera activa; pero persiste el buffer infinito

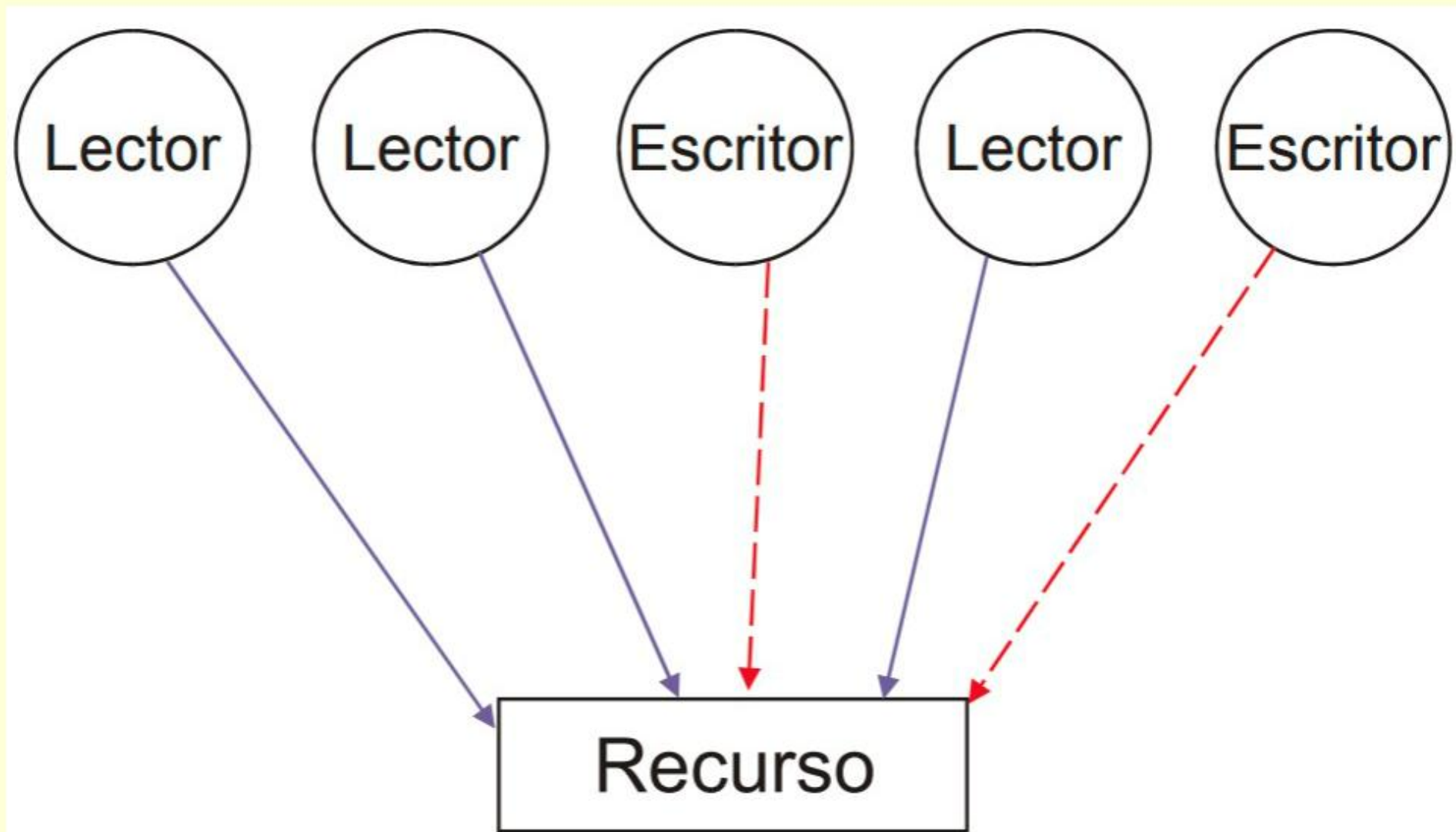
Contenido

- Concurrencia.
- Condiciones de carrera.
- Exclusión mutua y sección crítica.
- Semáforos.
- El problema del productor consumidor.
- El problema de los lectores escritores.

Problema de los lectores-escriptores

- Problema que se plantea cuando se tiene un área de almacenamiento compartida.
 - Múltiples procesos leen información.
 - Múltiples procesos escriben información.
- Condiciones:
 - Cualquier número de lectores pueden leer de la zona de datos concurrentemente.
 - Solamente un escritor puede modificar la información a la vez.
 - Durante una escritura ningún lector puede realizar una consulta.

El problema de los lectores-escriptores



Diferencias con otros problemas

- Exclusión mutua:
 - En el caso de la exclusión mutua solamente se permitiría a un proceso acceder a la información.
 - No se permitiría concurrencia entre lectores.
- Productor consumidor:
 - En el productor/consumidor los dos procesos modifican la zona de datos compartida.
- Objetivos de restricciones adicionales:
 - Proporcionar una solución más eficiente > se debe crear un algoritmo específico.

Condiciones de gestión:

- Los lectores tienen prioridad
 - Si hay algún lector en la sección crítica otros lectores pueden entrar.
 - Un escritor solamente puede entrar en la sección crítica si no hay ningún proceso.
 - Problema: Inanición para escritores.
- Los escritores tienen prioridad
 - Cuando un escritor desea acceder a la sección crítica no se admite la entrada de nuevos lectores.

Los lectores tienen prioridad. Ver 06LecEsc

- **Lector** `int nlect; semaforo lec=1; semaforo = escr=1;`

```
for(;;) {
```

```
    semWait(lec);
```

```
    nlect++;
```

```
    if (nlect==1)
```

```
        semWait(escr);
```

```
    semSignal(lec);
```

```
    realizar_lect();
```

```
    semWait(lec);
```

```
    nlect--;
```

```
    if (nlect==0)
```

```
        semSignal(escr);
```

```
    semSignal(lec);
```

```
}
```

```
for(;;) {
```

```
    semWait(escr);
```

```
    realizar_escr();
```

```
    semSignal(escr);
```

```
}
```

Tarea: Diseñar una solución para escritores con prioridad