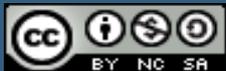


SISTEMAS OPERATIVOS: COMUNICACIÓN Y SINCRONIZACIÓN ENTRE PROCESOS



Procesos concurrentes y sincronización

A recordar...

Antes de clase

Clase

Después de clase

Preparar los pre-requisitos.

Estudiar el material asociado a la **bibliografía**:
las transparencias solo no son suficiente.
Preguntar dudas (especialmente tras estudio).

Ejercitar las competencias:

- ▶ Realizar todos los **ejercicios**.
- ▶ Realizar los **cuadernos de prácticas** y las **prácticas** de forma progresiva.

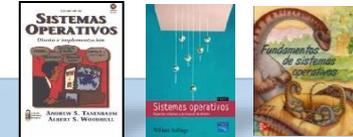
Lecturas recomendadas

Base



1. Carretero 2020:
 1. Cap. 6
2. Carretero 2007:
 1. Cap. 6.1 y 6.2

Recomendada



1. Tanenbaum 2006:
 1. (es) Cap. 5
 2. (en) Cap. 5
2. Stallings 2005:
 1. 5.1, 5.2 y 5.3
3. Silberschatz 2006:
 1. 6.1, 6.2, 6.5 y 6.6

Contenidos

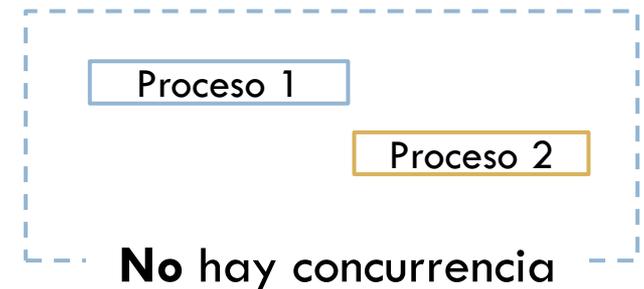
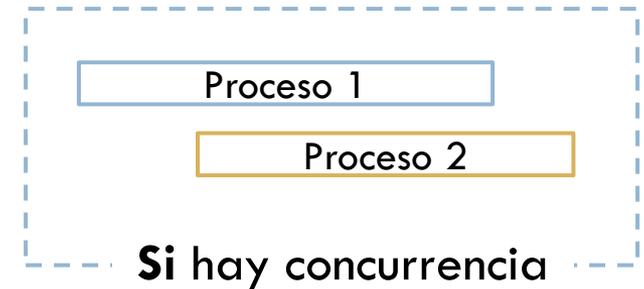
- Introducción (definiciones):
 - Procesos concurrentes.
 - Concurrencia, comunicación y sincronización
 - Sección crítica y condiciones de carrera
 - Exclusión mutua y sección crítica.
- Mecanismos de sincronización (I):
 - Primitivas básicas iniciales
 - Semáforos.
- Problemas clásicos de concurrencia (I):
 - Productor-consumidor
 - Lectores-escriores
- Mecanismos de sincronización de *threads* (II)
 - Semáforos
 - Llamadas al sistema para semáforos.
 - Problemas clásicos de concurrencia.
 - Mutex y variables condición
 - Llamadas al sistema para mutex.
 - Problemas clásicos de concurrencia.
- Caso estudio: desarrollo de servidores concurrentes

Contenidos

- Introducción (definiciones):
 - **Procesos concurrentes.**
 - **Concurrencia, comunicación y sincronización**
 - Sección crítica y condiciones de carrera
 - Exclusión mutua y sección crítica.
- Mecanismos de sincronización (I):
 - Primitivas básicas iniciales
 - Semáforos.
- Problemas clásicos de concurrencia (I):
 - Productor-consumidor
 - Lectores-escriores
- Mecanismos de sincronización de *threads* (II)
 - Semáforos
 - Llamadas al sistema para semáforos.
 - Problemas clásicos de concurrencia.
 - Mutex y variables condición
 - Llamadas al sistema para mutex.
 - Problemas clásicos de concurrencia.
- Caso estudio: desarrollo de servidores concurrentes

Proceso concurrente

- Dos procesos son concurrentes cuando se ejecutan de manera que sus intervalos de ejecución se solapan.
- Por defecto se espera el mismo resultado en ambos casos.



Cómo conseguir concurrencia

Tipos de concurrencia

- ❑ **Concurrencia aparente:**
Hay más procesos que procesadores.
 - ❑ Los procesos se multiplexan en el tiempo.
 - ❑ Pseudoparalelismo.
- ❑ **Concurrencia real:**
Cada proceso se ejecuta en un procesador.
 - ❑ Los procesos se simultanean en el tiempo.
 - ❑ Se produce una ejecución en paralelo.
 - ❑ Paralelismo real.

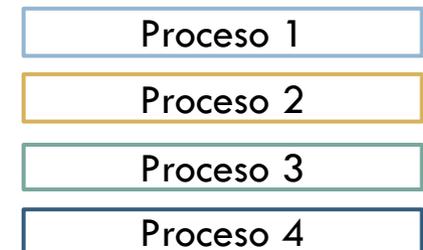
1 CPU



2 CPU



4 CPU

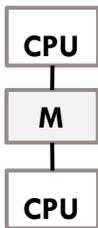


Cómo conseguir concurrencia

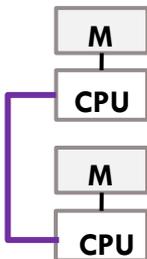
Modelos de programación concurrente

CPU

- Multiprogramación con un único procesador
 - El sistema operativo se encarga de repartir el tiempo entre los procesos
 - planificación expulsiva/no expulsiva.



- Multiprocesador
 - Se combinan paralelismo real y pseudoparalelismo.
 - Normalmente más procesos que procesadores (CPU).



- Sistema distribuido
 - Varios computadores conectados por red.

Ventajas de la ejecución concurrente

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

Desventajas de la ejecución concurrente

- Compartición de recursos.
 - ▣ La compartición de recursos precisa de sincronización.
 - ▣ Ejemplo: variable compartida con actualizaciones/lecturas (w-w, w-r).

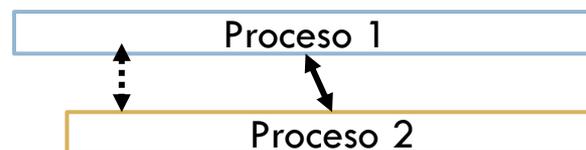
- Dificultad para depurar y localizar errores.
 - ▣ Las ejecuciones no son siempre deterministas ni reproducibles.
 - ▣ Ejemplos: entrelazados de ejecución particulares con problemas.

- Dificultades del S.O. para gestión óptima de recursos.
 - ▣ Dificultades del sistema operativos para la gestión de recursos de forma óptima.

Interacciones entre procesos

Tipos de servicios de interacción

- Comunicación:
 - ▣ Permiten la transferencia de información entre procesos.
 - ▣ Ejemplo: un proceso envía datos medidos para su procesamiento.
 - ▣ Mecanismos: archivos, tuberías, memoria compartida, paso de mensajes.
- Sincronización:
 - ▣ Permiten la espera hasta que ocurra un evento en otro proceso (deteniendo su ejecución hasta que ocurra)
 - ▣ Ejemplo: un proceso de presentación debe esperar a que todos los procesos de cálculo terminen.
 - ▣ Mecanismos: señales, pipes, semáforos, mutex, conditions, paso de mens.



Interacciones entre procesos

Tipos de procesos concurrentes

Relación	Influencia de un proceso en otro	Problemas potenciales
Independientes	<ul style="list-style-type: none">• No comunicación<ul style="list-style-type: none">• Resultado de un proceso no afecta a otros• No sincronización<ul style="list-style-type: none">• Temporización no puede afectar	
Compiten	<ul style="list-style-type: none">• No comunicación• Si posible sincronización	<ul style="list-style-type: none">• Excl. Mutua• Interbloqueo• Inanición
Cooperan	<ul style="list-style-type: none">• Si comunicación<ul style="list-style-type: none">• Por compartición, con recurso renovable (conocidos indirectamente)• Por comunicación, con recurso consumible (conocidos directamente)• Si posible sincronización	<ul style="list-style-type: none">• Interbloqueo• Inanición Compartición añade: <ul style="list-style-type: none">• Excl. Mutua• Coherencia datos

Contenidos

- Introducción (definiciones):
 - Procesos concurrentes.
 - Concurrencia, comunicación y sincronización
 - **Sección crítica y condiciones de carrera**
 - Exclusión mutua y sección crítica.
- Mecanismos de sincronización (I):
 - Primitivas básicas iniciales
 - Semáforos.
- Problemas clásicos de concurrencia (I):
 - Productor-consumidor
 - Lectores-escritores
- Mecanismos de sincronización de *threads* (II)
 - Semáforos
 - Llamadas al sistema para semáforos.
 - Problemas clásicos de concurrencia.
 - Mutex y variables condición
 - Llamadas al sistema para mutex.
 - Problemas clásicos de concurrencia.
- Caso estudio: desarrollo de servidores concurrentes

Interacciones entre procesos

Tipos de procesos concurrentes

Relación	Influencia de un proceso en otro	Problemas potenciales
Independientes	<ul style="list-style-type: none">• No comunicación<ul style="list-style-type: none">• Resultado de un proceso no afecta a otros• No sincronización<ul style="list-style-type: none">• Temporización no puede afectar	
Compiten	<ul style="list-style-type: none">• No comunicación• Si posible sincronización	<ul style="list-style-type: none">• Excl. Mutua• Interbloqueo• Inanición
Cooperan	<ul style="list-style-type: none">• Si comunicación<ul style="list-style-type: none">• Por compartición, con recurso renovable (conocidos indirectamente)• Por comunicación, con recurso consumible (conocidos directamente)• Si posible sincronización	<ul style="list-style-type: none">• Interbloqueo• Inanición Compartición añade: <ul style="list-style-type: none">• Excl. Mutua• Coherencia datos

Dos procesos con recurso compartido

escenario base

```
int suma = 0 ;
```

```
suma += 10 ;
```

```
suma += 20 ;
```

Dos procesos con recurso compartido

escenario base

```
int suma = 0 ;
```

```
suma += 10 ;
```

```
suma += 20 ;
```

```
suma == 30
```

Por defecto se espera que el resultado de ejecución paralela igual a la ejecución secuencial

Dos procesos con recurso compartido sección crítica

```
int suma = 0 ;
```

```
suma += 10 ;
```

Sección crítica:

Segmento de código que manipula un recurso (y debe ser ejecutado de forma atómica: w-w, w-r).

```
suma += 20 ;
```

```
suma == 30
```

Dos procesos con recurso compartido escenario base

```
int suma = 0 ;
```

```
lw $t0 suma ;
```

```
addi $t0 $t0 10;
```

```
sw $t0 suma;
```

Sección crítica no atómica:
La distinta velocidad en
la ejecución de procesos
permite distintos
entrelazados.

```
lw $t0 suma ;
```

```
addi $t0 $t0 20;
```

```
sw $t0 suma;
```

¿ suma ?

Dos procesos con recurso compartido condiciones de carrera

```
int suma = 0 ;
```

```
lw $t0 suma ;
```

```
addi $t0 $t0 10;
```

```
sw $t0 suma;
```

Sección crítica no atómica:
La distinta velocidad en la ejecución de procesos permite distintos entrelazados.

```
lw $t0 suma ;
```

```
addi $t0 $t0 20;
```

```
sw $t0 suma;
```

```
¿ suma ?
```

Condición de carrera:
La velocidad en la ejecución afecta al resultado.

Dos procesos con recurso compartido condiciones de carrera

int suma = 0 ;

lw \$t0 suma ;

addi \$t0 \$t0 10;

sw \$t0 suma;

lw \$t0 suma ;

addi \$t0 \$t0 20;

sw \$t0 suma;

¿ suma ?

Dos procesos con recurso compartido condiciones de carrera

```
int suma = 0 ;
```

```
lw $t0 suma ;
```

```
addi $t0 $t0 10;
```

```
sw $t0 suma;
```

```
lw $t0 suma ;
```

```
addi $t0 $t0 20;
```

```
sw $t0 suma;
```

```
? suma ?
```

Dos procesos con recurso compartido condiciones de carrera

```
int suma = 0 ;
```

```
lw $t0 suma ;
```

```
addi $t0 $t0 10;
```

```
sw $t0 suma;
```

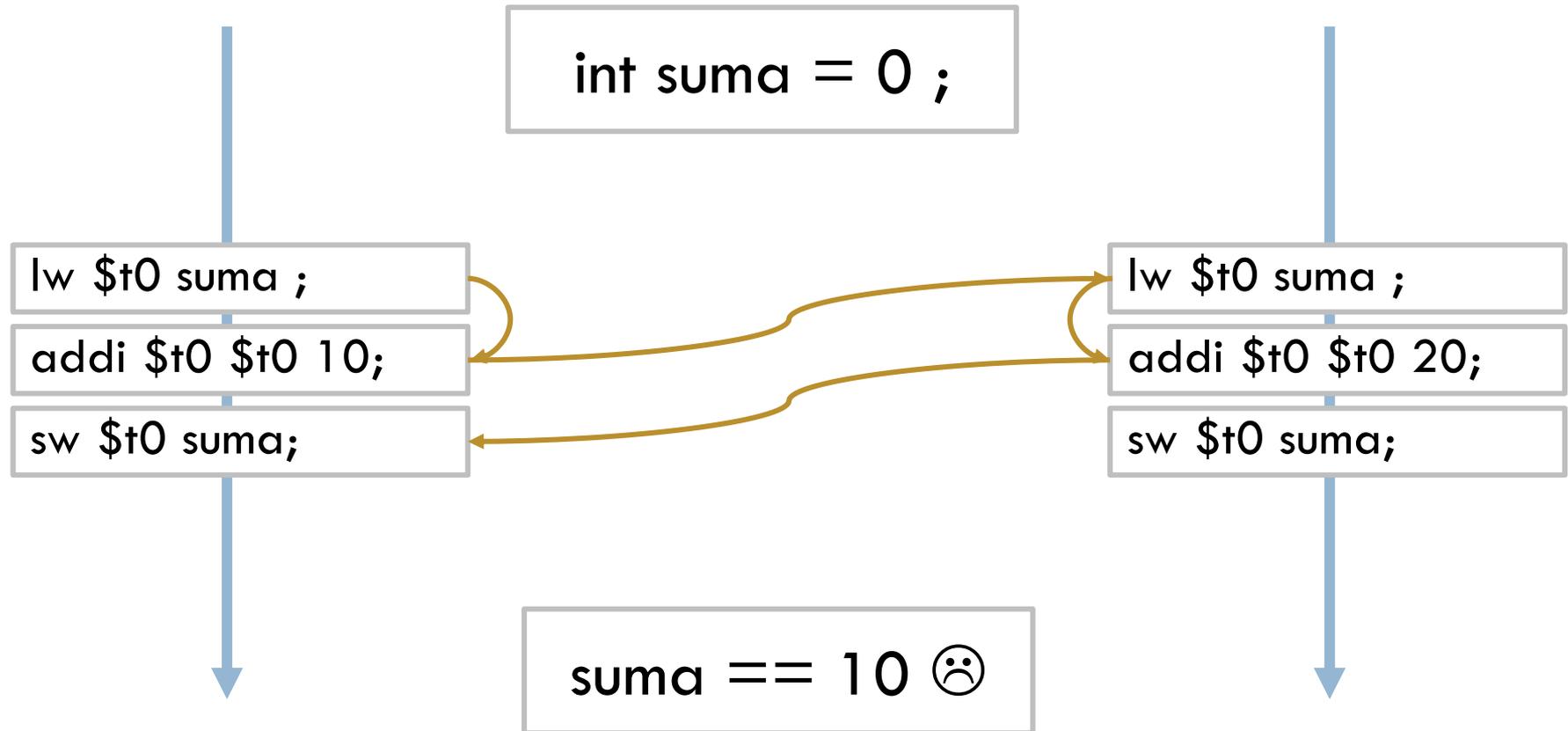
```
lw $t0 suma ;
```

```
addi $t0 $t0 20;
```

```
sw $t0 suma;
```

```
? suma ?
```

Dos procesos con recurso compartido condiciones de carrera



Dos procesos con recurso compartido condiciones de carrera

```
int suma = 0 ;
```

```
lw $t0 suma ;
```

```
addi $t0 $t0 10;
```

```
sw $t0 suma;
```

```
lw $t0 suma ;
```

```
addi $t0 $t0 20;
```

```
sw $t0 suma;
```

```
suma == 20 ☹️
```

Dos procesos con recurso compartido condiciones de carrera

```
int suma = 0 ;
```

```
lw $t0 suma ;
```

```
addi $t0 $t0 10;
```

```
sw $t0 suma;
```

```
lw $t0 suma ;
```

```
addi $t0 $t0 20;
```

```
sw $t0 suma;
```

```
suma == 20 ☹️
```

Se puede dar en un multiprocesador (cachés) o en monoprocesador con multitarea.

Dos procesos con recurso compartido condiciones de carrera

- Es necesario garantizar que el orden de ejecución 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.

```
lw $t0 suma ;
```

```
addi $t0 $t0 10;
```

```
sw $t0 suma;
```

```
lw $t0 suma ;
```

```
addi $t0 $t0 20;
```

```
sw $t0 suma;
```

```
suma == 30 😊
```

Dos procesos con recurso compartido condiciones de carrera

- Las instrucciones dentro de la sección crítica (acceden a variable) se han de ejecutar **de forma atómica**:
 - **La sección crítica de un proceso** se ejecuta en exclusión mutua con respecto a las secciones críticas de **otros procesos**.

```
lw $t0 suma ;
```

```
addi $t0 $t0 10;
```

```
sw $t0 suma;
```

Objetivo:
conseguir que las
instrucciones de la
sección crítica se ejecuten
de forma atómica

```
lw $t0 suma ;
```

```
addi $t0 $t0 20;
```

```
sw $t0 suma;
```

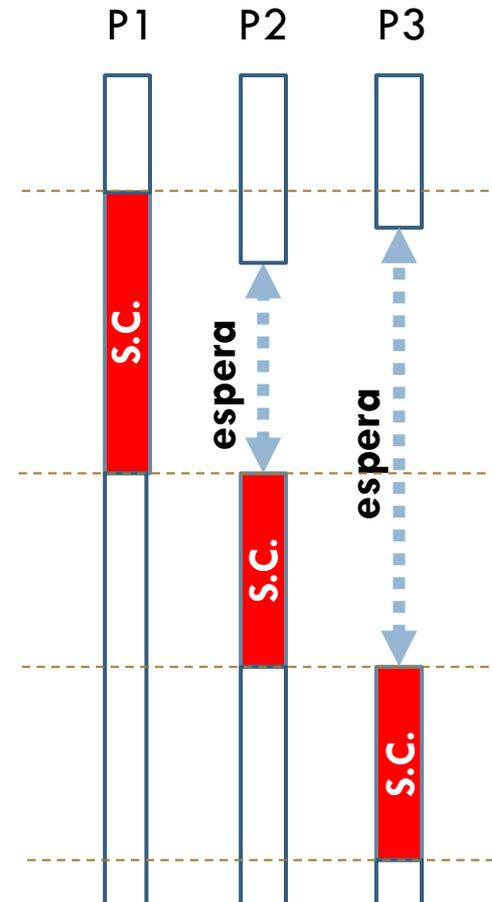
```
suma == 30 😊
```

Contenidos

- Introducción (definiciones):
 - Procesos concurrentes.
 - Concurrencia, comunicación y sincronización
 - Sección crítica y condiciones de carrera
 - **Exclusión mutua y sección crítica.**
- Mecanismos de sincronización (I):
 - Primitivas básicas iniciales
 - Semáforos.
- Problemas clásicos de concurrencia (I):
 - Productor-consumidor
 - Lectores-escriores
- Mecanismos de sincronización de *threads* (II)
 - Semáforos
 - Llamadas al sistema para semáforos.
 - Problemas clásicos de concurrencia.
 - Mutex y variables condición
 - Llamadas al sistema para mutex.
 - Problemas clásicos de concurrencia.
- Caso estudio: desarrollo de servidores concurrentes

Exclusión mutua (objetivo)

- **Exclusión mutua:** solamente un proceso puede estar a la vez en la sección crítica de un recurso.
- **Sección crítica:** segmento de código que manipula (w-w, w-r) un recurso y debe ser ejecutado de forma atómica.
- **Mecanismo de exclusión:** mecanismo asociado a un recurso para la gestión de su exclusión mutua.



Mecanismo de exclusión mutua

```
int suma = 0 ;
```

Entrar en sección crítica

```
lw $t0 suma ;
```

```
addi $t0 $t0 10;
```

```
sw $t0 suma;
```

Salir de sección crítica

Mecanismo de exclusión:

- Ha de sincronizar los procesos.
- Cada proceso al entrar en S.C. pide permiso.
- Cada proceso informa para abandonar S.C.

Entrar en sección crítica

```
lw $t0 suma ;
```

```
addi $t0 $t0 20;
```

```
sw $t0 suma;
```

Salir de sección crítica

```
suma == 30 😊
```

Mecanismo de exclusión mutua condiciones que ha de cumplir

1. Exclusión mutua

Se obliga a que solo un proceso puede estar simultáneamente en la sección crítica de un recurso.

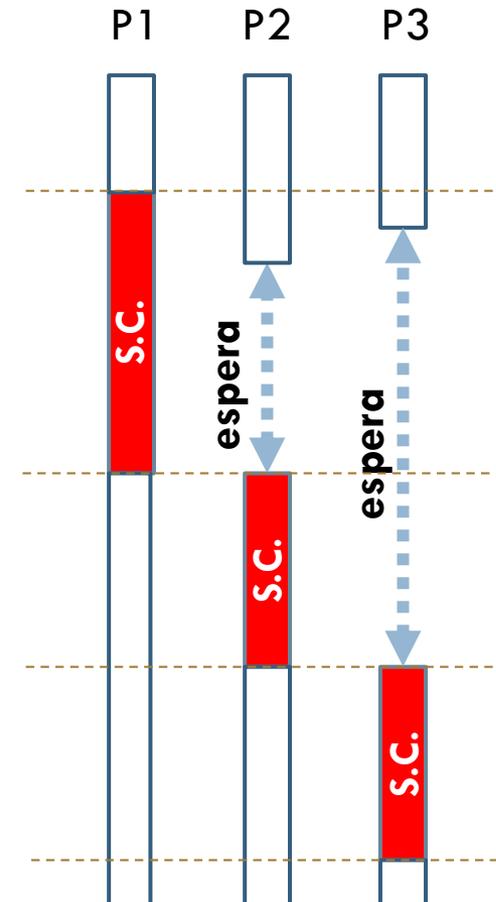
2. Progreso (no interbloqueo/*deadlock*)

Cuando ningún proceso este en una sección crítica, cualquier proceso que solicite su entrada lo hará sin demora.

3. Espera acotada (no inanición/*starvation*)

Debe existir una cota superior en el número de veces otros procesos entran en la s.c. después de que un proceso pida entrar y antes de que se otorga.

- ▣ Un proceso permanece en su sección crítica durante un tiempo finito.
- ▣ No se puede hacer suposiciones sobre la velocidad de los procesos ni el número de procesadores.
- ▣ Un proceso que termina en su sección no crítica no debe interferir en otros procesos.



Problemas en secciones críticas

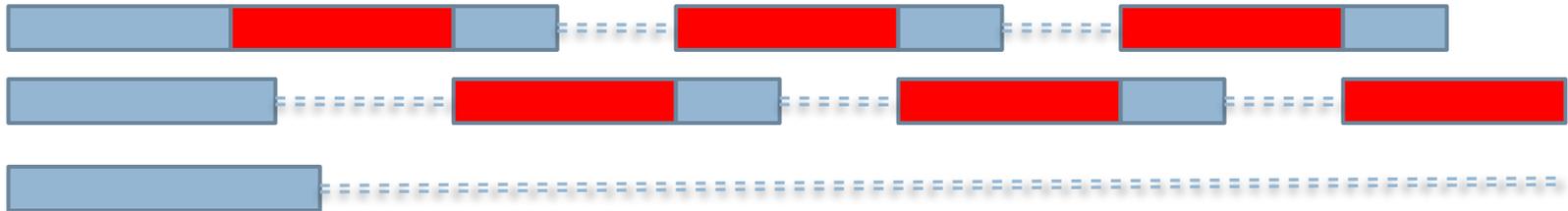
Inanición

- 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

Problemas en secciones críticas

Inanición

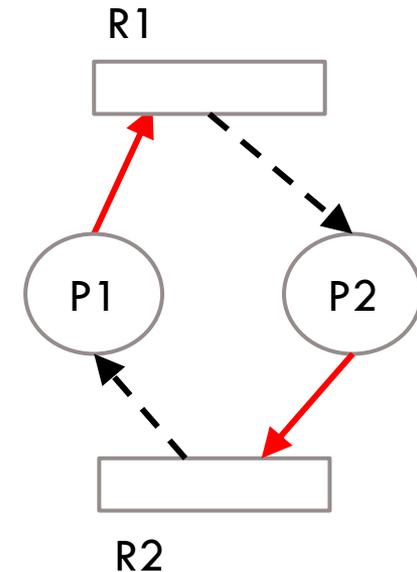


El proceso P3 nunca llega a conseguir entrar en la sección crítica

Problemas en secciones críticas

Interbloqueos

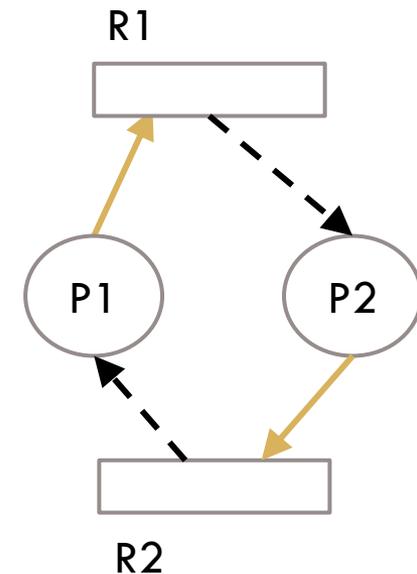
- Se produce con exclusión mutua para más de un recurso, condiciones necesarias son:
 1. **Exclusión mutua:** solo un proceso puede usar un recurso cada vez. Si otro proceso solicita ese recurso debe esperar a que esté libre.



Problemas en secciones críticas

Interbloqueos

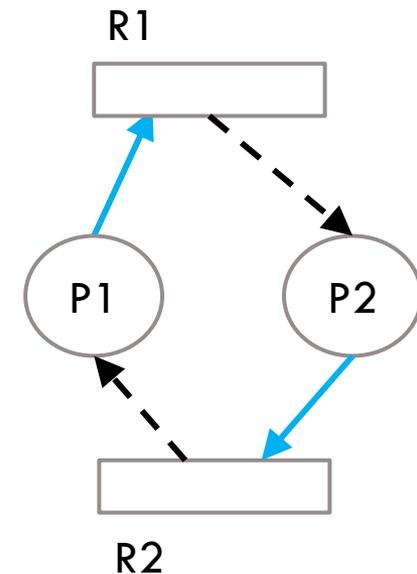
- Se produce con exclusión mutua para más de un recurso, condiciones necesarias son:
 - 1. Exclusión mutua:** solo un proceso puede usar un recurso cada vez. Si otro proceso solicita ese recurso debe esperar a que esté libre.
 - 2. Retención y espera:** un proceso retiene unos recursos mientras espera que se le asignen otros.



Problemas en secciones críticas

Interbloqueos

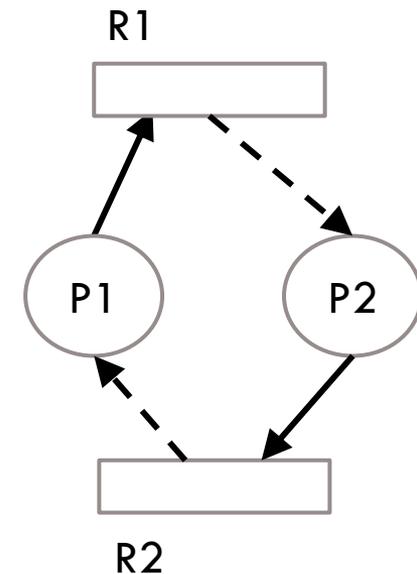
- Se produce con exclusión mutua para más de un recurso, condiciones necesarias son:
 - 1. Exclusión mutua:** solo un proceso puede usar un recurso cada vez. Si otro proceso solicita ese recurso debe esperar a que esté libre.
 - 2. Retención y espera:** un proceso retiene unos recursos mientras espera que se le asignen otros.
 - 3. No expropiación:** un proceso no puede ser forzado a abandonar un recurso que retiene.



Problemas en secciones críticas

Interbloqueos

- Se produce con exclusión mutua para más de un recurso, condiciones necesarias son:
 1. **Exclusión mutua:** solo un proceso puede usar un recurso cada vez. Si otro proceso solicita ese recurso debe esperar a que esté libre.
 2. **Retención y espera:** un proceso retiene unos recursos mientras espera que se le asignen otros.
 3. **No expropiación:** un proceso no puede ser forzado a abandonar un recurso que retiene.
 4. **Espera circular:** existe una cadena cerrada de procesos $\{P_0, \dots, P_n\}$ en la que cada proceso tiene un recurso y espera un recurso del siguiente proceso en la cadena.



Ninguno puede avanzar

Contenidos

- Introducción (definiciones):
 - Procesos concurrentes.
 - Concurrencia, comunicación y sincronización
 - Sección crítica y condiciones de carrera
 - Exclusión mutua y sección crítica.
- **Mecanismos de sincronización (I):**
 - Primitivas básicas iniciales
 - Semáforos.
- Problemas clásicos de concurrencia (I):
 - Productor-consumidor
 - Lectores-escritores
- Mecanismos de sincronización de *threads* (II)
 - Semáforos
 - Llamadas al sistema para semáforos.
 - Problemas clásicos de concurrencia.
 - Mutex y variables condición
 - Llamadas al sistema para mutex.
 - Problemas clásicos de concurrencia.
- Caso estudio: desarrollo de servidores concurrentes

Alternativas de implementación

- Aproximación software:
 - **Dekker** (Dijkstra, con 4 intentos)
 - **Peterson**
 - ...

- Aproximación por hardware:
 - **Desactivar interrupciones.**
 - Solamente válido en sistemas monoprocesador (y proceso no interrumpible).
 - **Instrucciones máquina especiales:** test_and_set o swap.
 - Implica espera activa (son posibles inanición e interbloqueo mal usadas).

- Soporte del S.O. (y lenguaje de programación):
 - **Semáforos**
 - **Monitores**
 - **Paso de mensaje**
 - ...

- (1) Conocer mecanismos y cómo usarlos para exclusión mutua.
- (2) Conocer cómo implementar unos mecanismos en función de otros.

Tipo aprox.	Mecanismo	semáforos	cerrojos	condiciones	...
software	Dekker
	Petterson

hardware	Desactivar interrupts.
	test_and_set
	swap

S.O. + lenguaje	semáforos
	cerrojos
	condiciones
	monitores
	paso de mensajes

Contenidos

- Introducción (definiciones):
 - Procesos concurrentes.
 - Concurrencia, comunicación y sincronización
 - Sección crítica y condiciones de carrera
 - Exclusión mutua y sección crítica.
- **Mecanismos de sincronización (I):**
 - **Primitivas básicas iniciales**
 - Semáforos.
- Problemas clásicos de concurrencia (I):
 - Productor-consumidor
 - Lectores-escritores
- Mecanismos de sincronización de *threads* (II)
 - Semáforos
 - Llamadas al sistema para semáforos.
 - Problemas clásicos de concurrencia.
 - Mutex y variables condición
 - Llamadas al sistema para mutex.
 - Problemas clásicos de concurrencia.
- Caso estudio: desarrollo de servidores concurrentes

Test-and-set

- Instrucción test-and-set
 - Espera activa
 - No caché en 'lock'

```
while (test_and_set(&lock) == 1) ;  
critical section  
lock = 0;  
remainder section
```

```
volatile int lock = 0 ;  
while (test_and_set(&lock) == 1) ;  
critical section  
lock = 0;  
remainder section
```

Solución de Peterson

- Limitaciones:
 - ▣ SOLO para 2 procesos.
 - ▣ Asume que instrucciones LOAD y STORE son atómicas, no interrumpibles.
- Los 2 procesos comparten 2 variables:
 - ▣ **int turno;**
 - indica quien entrará en la sección crítica.
 - $\text{turno} = 1$ implica que P_1 entrará.
 - ▣ **bool flag[2];**
 - indica si un proceso tiene intención en entrar en la sección crítica.
 - $\text{flag}[i] = \text{true}$ implica que P_i está listo para entrar.

Peterson: algoritmo para proceso P_i

2 procesos: P_i y P_j (con $j=1-i$)

- $i=0 \Rightarrow j=1$ ($1-i$)
- $i=1 \Rightarrow j=0$ ($1-i$)

```
do
{
  flag[j] = TRUE;
  turn = i;
  while (flag[i] &&
        turn == i);

  critical section

  flag[j] = FALSE;
  remainder section
} while (TRUE);
```

```
do
{
  flag[i] = TRUE;
  turn = j;
  while (flag[j] && turn == j);

  critical section

  flag[i] = FALSE;
  remainder section
} while (TRUE);
```

Contenidos

- Introducción (definiciones):
 - Procesos concurrentes.
 - Concurrencia, comunicación y sincronización
 - Sección crítica y condiciones de carrera
 - Exclusión mutua y sección crítica.
- **Mecanismos de sincronización (I):**
 - Primitivas básicas iniciales
 - **Semáforos.**
- Problemas clásicos de concurrencia (I):
 - Productor-consumidor
 - Lectores-escritores
- Mecanismos de sincronización de *threads* (II)
 - Semáforos
 - Llamadas al sistema para semáforos.
 - Problemas clásicos de concurrencia.
 - Mutex y variables condición
 - Llamadas al sistema para mutex.
 - Problemas clásicos de concurrencia.
- Caso estudio: desarrollo de servidores concurrentes

Semáforos (Dijkstra)

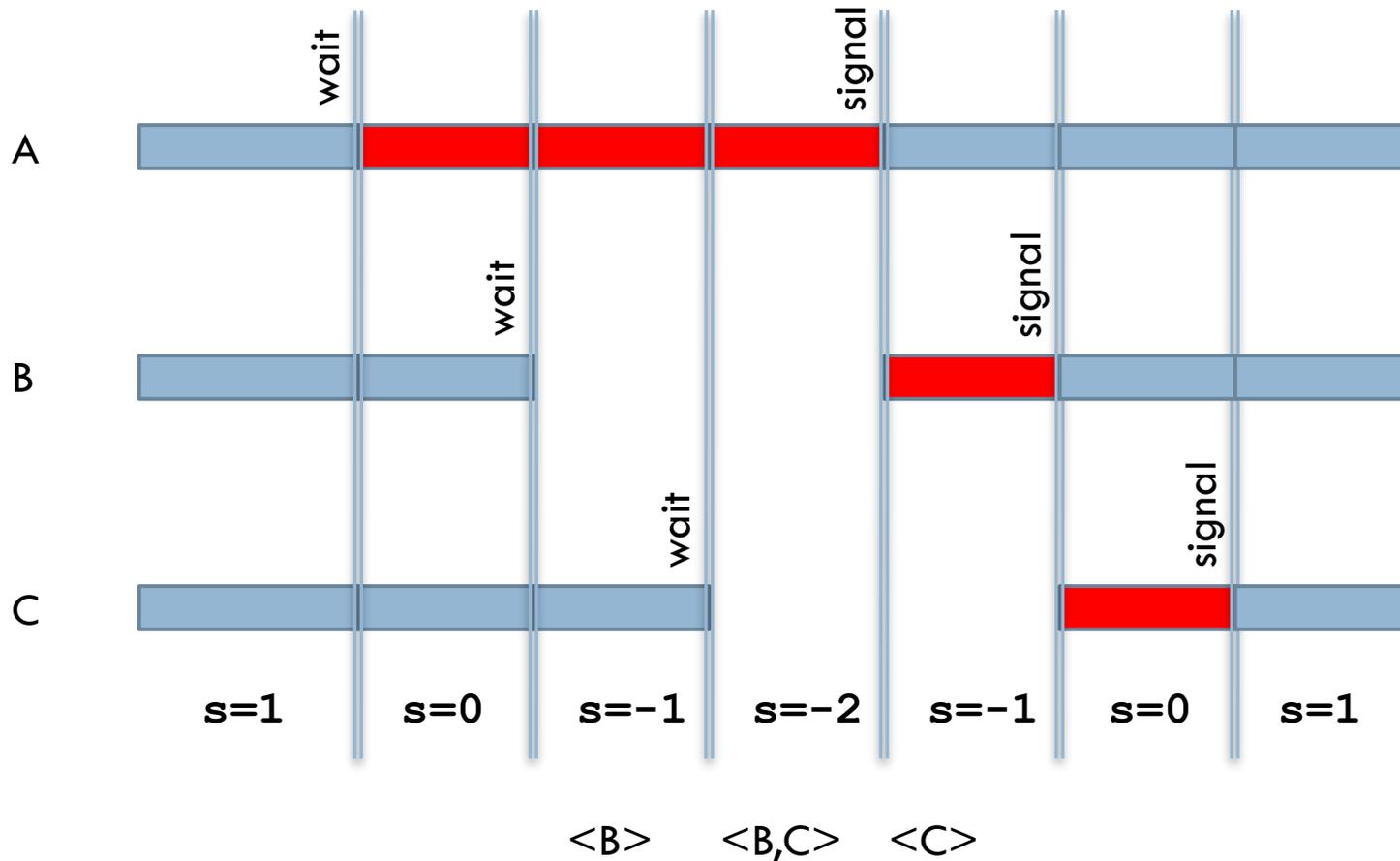
- Se puede ver un semáforo como una variable entera con tres operaciones atómicas asociadas.
- Operaciones atómicas asociadas:
 - ▣ **Iniciación** a un valor no negativo.
 - ▣ **semWait:**
 - Decrementa el contador del semáforo y si $(s < 0)$ → El proceso llamante se bloquea.
 - ▣ **semSignal:**
 - Incrementa el valor del semáforo y si $(s \leq 0)$ → Desbloquea un proceso.

Secciones críticas y semáforos

- A la sección crítica de un recurso se le asocia un semáforo:
 - ▣ Semáforo **iniciado a 1**.
- **semWait**: entrada en la sección crítica.
- **semSignal**: salida de la sección crítica.

```
s = 1 ;  
  
// sección no crítica  
...  
semWait(s);  
// sección crítica  
semSignal(s);  
...  
// sección no crítica
```

Secciones críticas y semáforos



Ejemplos de uso de semáforos

```
M.C.:
semaforo s=0;
```

```
A:
...
P(s);
...
B:
...
V(s);
...
```

“La señal”

```
M.C.:
semaforo s=1;
```

```
A:
P(s);
<SC>
V(s);
B:
P(s);
<SC>
V(s);
```

“El mutex”

```
M.C.:
semaforo s=10;
```

```
A:
P(s);
<max. 10>
V(s);
B:
P(s);
<max. 10>
V(s);
```

“El equipo”

Contenidos

- Introducción (definiciones):
 - Procesos concurrentes.
 - Concurrencia, comunicación y sincronización
 - Sección crítica y condiciones de carrera
 - Exclusión mutua y sección crítica.
- Mecanismos de sincronización (I):
 - Primitivas básicas iniciales
 - Semáforos.
- **Problemas clásicos de concurrencia (I):**
 - Productor-consumidor
 - Lectores-escriores
- Mecanismos de sincronización de *threads* (II)
 - Semáforos
 - Llamadas al sistema para semáforos.
 - Problemas clásicos de concurrencia.
 - Mutex y variables condición
 - Llamadas al sistema para mutex.
 - Problemas clásicos de concurrencia.
- Caso estudio: desarrollo de servidores concurrentes

Problemas clásicos de concurrencia

Tipo aprox.	Mecanismo	P-C	LL-EE	...
software	Dekker	P-C con Dekker
	Petterson	P-C con Petterson	<no aplica +2>	...

hardware	Desactivar interrupts.
	test_and_set
	swap

S.O. + lenguaje	semáforos	P-C con sem.	LL-EE con sem.	...
	cerrojos
	condiciones
	monitores
	paso de mensajes

Problemas clásicos de concurrencia

(1) Conocer los problemas clásicos de concurrencia para detectar cuándo aparecen [*].

- P-C: productor-consumidor
- LL-EE: lectores y escritores
- ...

[*] Pueden aparecer 1 o combinación de varios.

(2) Conocer la solución a los problemas clásicos de concurrencia para usarse como plantillas cuando aparecen

Tipo aprox.	Mecanismo	P-C	LL-EE	...
		P-C con Dekker
		P-C con Petterson	<no aplica +2>	...
	
	pts.
	
	
	
	semáforos	P-C con sem.	LL-EE con sem.	...

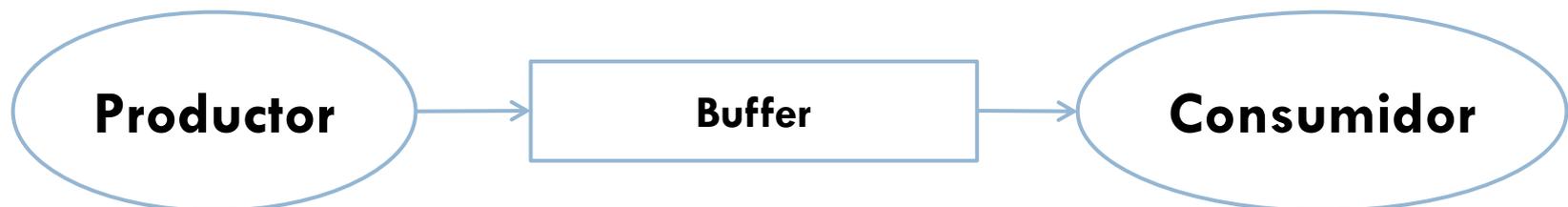
	
	
	s
	

Contenidos

- Introducción (definiciones):
 - Procesos concurrentes.
 - Concurrencia, comunicación y sincronización
 - Sección crítica y condiciones de carrera
 - Exclusión mutua y sección crítica.
- Mecanismos de sincronización (I):
 - Primitivas básicas iniciales
 - Semáforos.
- **Problemas clásicos de concurrencia (I):**
 - **Productor-consumidor**
 - Lectores-escriores
- Mecanismos de sincronización de *threads* (II)
 - Semáforos
 - Llamadas al sistema para semáforos.
 - Problemas clásicos de concurrencia.
 - Mutex y variables condición
 - Llamadas al sistema para mutex.
 - Problemas clásicos de concurrencia.
- Caso estudio: desarrollo de servidores concurrentes

El problema del productor-consumidor

- Un proceso produce elementos de información.
- Un proceso consume elementos de información.
- Se tiene un espacio de almacenamiento intermedio.
 - ▣ Infinito
 - ▣ Acotado (finito de tamaño N)



Búfer infinito

Hay que introducir
sincronización

55

<https://computationstructures.org/lectures/synchronization/synchronization.html>

Alejandro Calderón Mateos 

MEMORIA COMPARTIDA:

```
int inicio, fin;  
char v[N];
```

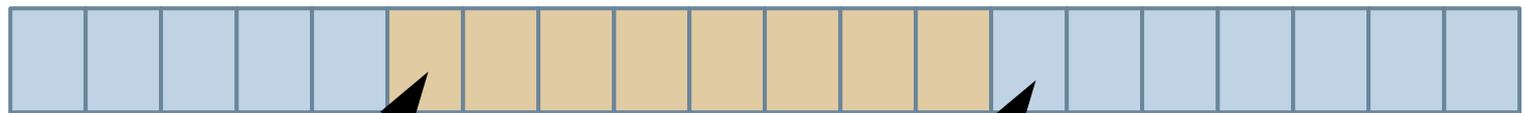
PRODUCER:

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

CONSUMIDOR:

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

Espera activa



inicio

ARCOS @ UC3M

fin

Búfer infinito

MEMORIA COMPARTIDA:

```
int inicio, fin;
char v[N];
semaforo s=1;
```

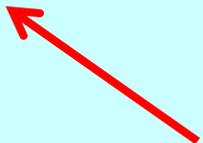
PRODUCER:

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

CONSUMIDOR:

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

**Espera
activa**



Búfer infinito

MEMORIA COMPARTIDA:

```
int inicio, fin;  
char v[N];  
semaforo s=1; semaforo n=0;
```

PRODUCER:

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

CONSUMIDOR:

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

Contenidos

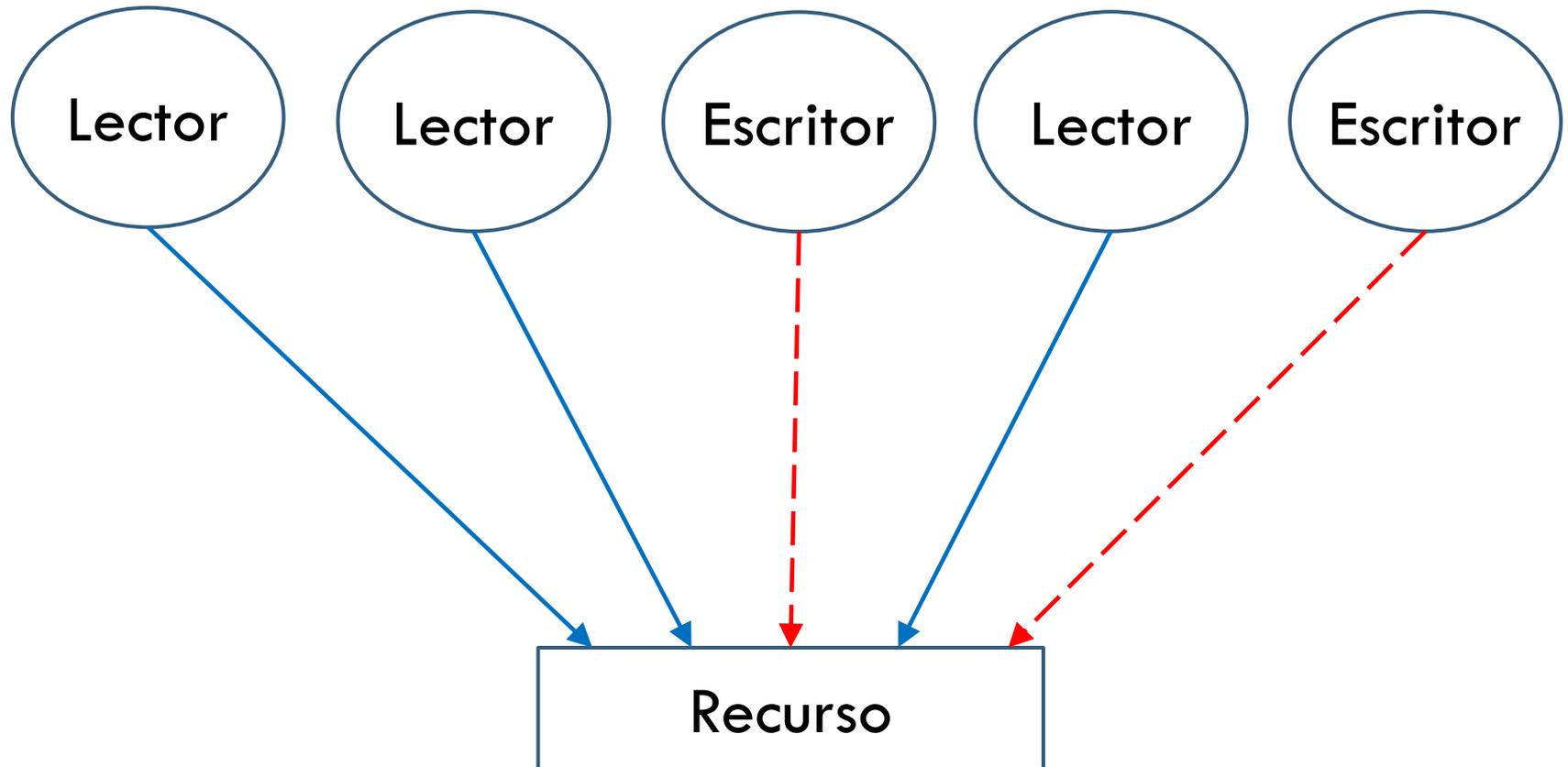
- Introducción (definiciones):
 - Procesos concurrentes.
 - Concurrencia, comunicación y sincronización
 - Sección crítica y condiciones de carrera
 - Exclusión mutua y sección crítica.
- Mecanismos de sincronización (I):
 - Primitivas básicas iniciales
 - Semáforos.
- **Problemas clásicos de concurrencia (I):**
 - Productor-consumidor
 - **Lectores-escriptores**
- Mecanismos de sincronización de *threads* (II)
 - Semáforos
 - Llamadas al sistema para semáforos.
 - Problemas clásicos de concurrencia.
 - Mutex y variables condición
 - Llamadas al sistema para mutex.
 - Problemas clásicos de concurrencia.
- Caso estudio: desarrollo de servidores concurrentes

Problema de los lectores-escritores

- 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: **posible varios lectores a la vez**.
 - **Solamente un escritor** puede modificar la información **a la vez**.
 - **Durante una escritura ningún lector** puede leer.

Problema de los lectores-escritores



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 dos lectores no precisan exclusión mutua en la sección crítica.
 - ▣ **Objetivo:** proporcionar una solución más eficiente.

Alternativas de gestión

A. Los lectores tienen prioridad.

- ▣ Si hay algún lector en la sección crítica entonces otros lectores pueden entrar.
- ▣ Un escritor solamente puede entrar en la sección crítica si no hay ningún proceso.
- ▣ **Problema:** inanición para escritores.

B. 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 (1/4) interacción de escritores (sección crítica)

MEMORIA COMPARTIDA:

```
int nlect; semaforo lec=1; semaforo escr=1;
```

ESCRITOR:

```
for(;;) {  
    semWait(escr);  
    realizar_escr();  
    semSignal(escr);  
}
```

Los lectores tienen prioridad (2/4)

interacción de lectores entre sí

MEMORIA COMPARTIDA:

```
int nlect; semaforo lec=1; semaforo escr=1;
```

LECTOR:

```
for(;;) {
    semWait(lec);
    nlect++;
    if (nlect==1)
        semWait(escr);
    semSignal(lec);

    realizar_lect();

    semWait(lec);
    nlect--;
    if (nlect==0)
        semSignal(escr);
    semSignal(lec);
}
```

ESCRITOR:

```
for(;;) {
    semWait(escr);
    realizar_escr();
    semSignal(escr);
}
```

Los lectores tienen prioridad (3/4)

varios lectores con un escritor

MEMORIA COMPARTIDA:

```
int nlect; semaforo lec=1; semaforo escr=1;
```

LECTOR:

```
for(;;) {
    semWait(lec);
    nlect++;
    if (nlect==1)
        semWait(escr);
    semSignal(lec);

    realizar_lect();

    semWait(lec);
    nlect--;
    if (nlect==0)
        semSignal(escr);
    semSignal(lec);
}
```

ESCRITOR:

```
for(;;) {
    semWait(escr);
    realizar_escr();
    semSignal(escr);
}
```

NOTA: nlect se incrementa y consulta de forma NO atómica entre lectores...

Los lectores tienen prioridad (4/4)

varios lectores con un escritor

MEMORIA COMPARTIDA:

```
int nlect; semaforo lec=1; semaforo escr=1;
```

LECTOR:

```
for(;;) {  
    semWait(lec);  
    nlect++;  
    if (nlect==1)  
        semWait(escr);  
    semSignal(lec);  
  
    realizar_lect();
```

```
    semWait(lec);  
    nlect--;  
    if (nlect==0)  
        semSignal(escr);  
    semSignal(lec);  
}
```

ESCRITOR:

```
for(;;) {  
    semWait(escr);  
    realizar_escr();  
    semSignal(escr);  
}
```

Los lectores tienen prioridad

MEMORIA COMPARTIDA:

```
int nlect; semaforo lec=1; semaforo escr=1;
```

LECTOR:

```
for(;;) {  
    semWait(lec);  
    nlect++;  
    if (nlect==1)  
        semWait(escr);  
    semSignal(lec);  
  
    realizar_lect();
```

```
    semWait(lec);  
    nlect--;  
    if (nlect==0)  
        semSignal(escr);  
    semSignal(lec);  
}
```

ESCRITOR:

```
for(;;) {  
    semWait(escr);  
    realizar_escr();  
    semSignal(escr);  
}
```

Tarea: Diseñar una solución para escritores con prioridad

Los escritores tienen prioridad

MEMORIA COMPARTIDA:

```
int nlect, nescr = 0; semaphore lect, escr = 1;
semaphore x, y, z = 1;
```

LECTOR:

```
for(;;) {
    semWait(z);
    semWait(lect);
    semWait(x);
    nlect++;
    if (nlect==1)
        semWait(escr);
    semSignal(x);
    semSignal(lect);
    semSignal(z);
    // doReading();
    semWait(x);
    nlect--;
    if (nlect==0)
        semSignal(escr);
    semSignal(x);
}
```

ESCRITOR:

```
for(;;) {
    semWait(y);
    nescr++;
    if (nescr==1)
        semWait(lect);
    semSignal(y);
    semWait(escr);
    // doWriting();
    semSignal(escr);
    semWait(y);
    nescr--;
    if (nescr==0)
        semSignal(lect);
    semSignal(y);
}
```

SISTEMAS OPERATIVOS: COMUNICACIÓN Y SINCRONIZACIÓN ENTRE PROCESOS



Procesos concurrentes y sincronización