



Sistemas Operativos

SpinLocks

Rodrigo Urrea

Agenda Auxiliar

P0.

Repaso

P1.

Lectores
Escritores

P2.

Función
team

Po.

Repaso

SpinLock

- ▶ Herramienta de sincronización primitiva.
- ▶ No dependen ni del Sistema Operativo ni del Scheduler.
- ▶ Utiliza la instrucción de máquina SWAP.

Modo de uso:

- ▶ Creación:
`int lk = OPEN;`
- ▶ Inicio sección crítica:
`spinLock(&lk);`
- ▶ Fin sección crítica:
`spinUnlock(&lk);`



Consistent Locking Behavior

Cuando 2 cores quieren acceder a una dirección **d**, uno de ellos modificandola, se debe dar que:

1. Core X obtiene mutex
2. Core X accede a **d**
3. Core X libera mutex
4. Core Y obtiene mutex
5. Core Y accede a **d**
6. Core Y libera mutex

Sanitize verifica que se cumpla con esta metodología

P1.

Lectores Escritores

Lectores Escritores usando SpinLocks

La siguiente implementación es incorrecta. Haga un diagrama de threads que muestre que un lector puede entrar junto con un escritor.

```
void enterRead() {
    if (readers == 0)
        spinLock(&write_lck);
    spinLock(&mutex_lck);
    readers++;
    spinUnlock(&mutex_lck);
}
void enterWrite() {
    spinLock(&write_lck);
}
```

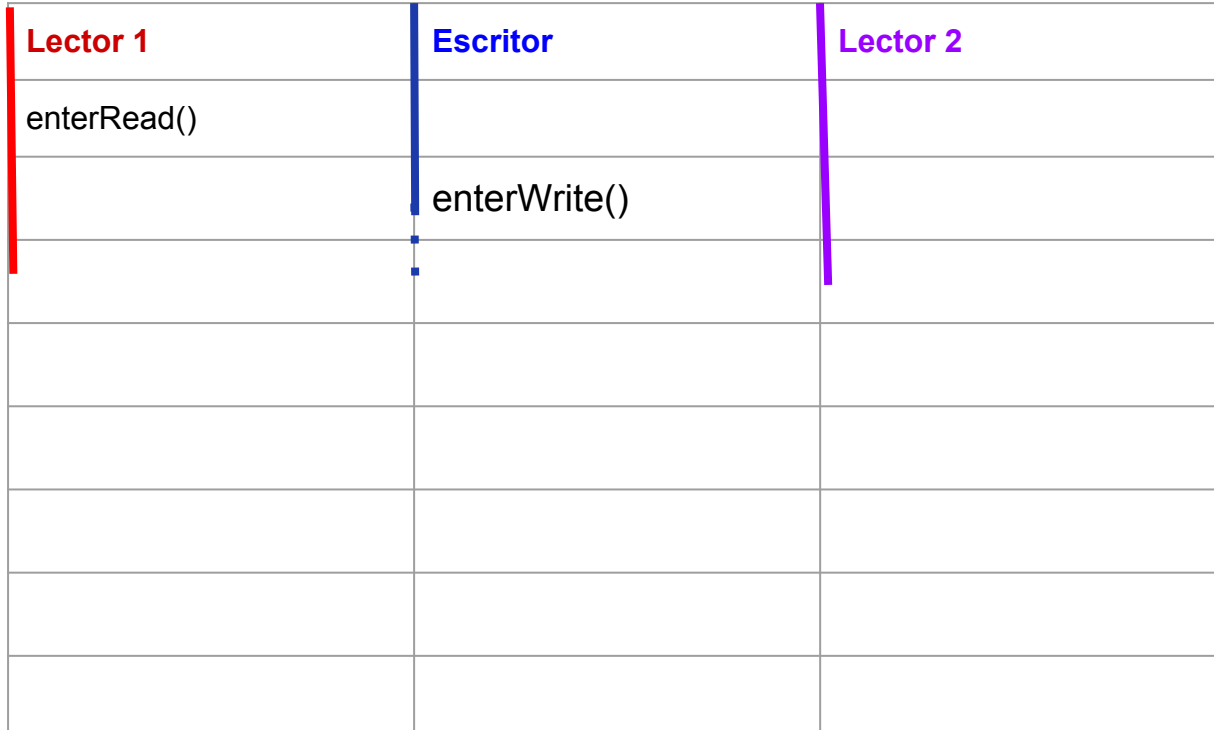
```
void exitRead() {
    spinLock(&mutex_lck);
    readers--;
    spinUnlock(&mutex_lck);
    if (readers == 0)
        spinUnlock(&write_lck);
}
void exitWrite() {
    spinUnlock(&write_lck);
}
```

Lectores Escritores usando SpinLocks

Lector 1	Escritor	Lector 2
enterRead() 		

readers=1

Lectores Escritores usando SpinLocks



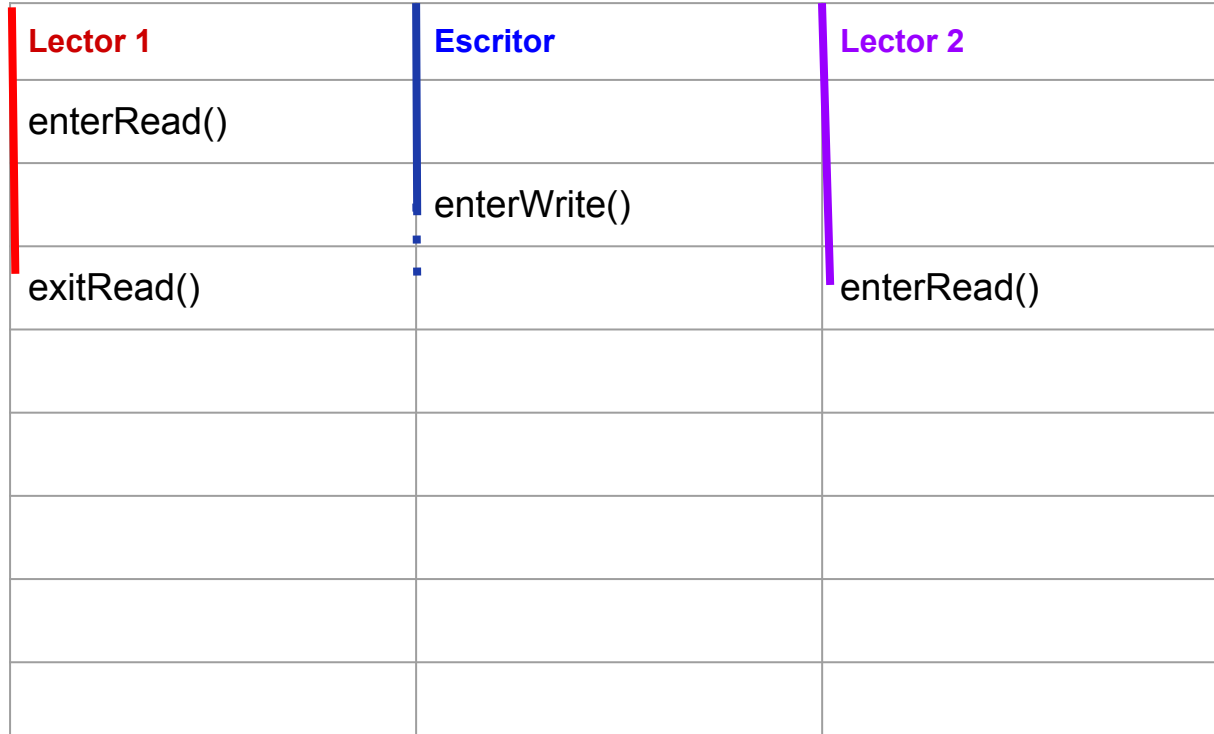
readers=1

Lectores Escritores usando SpinLocks



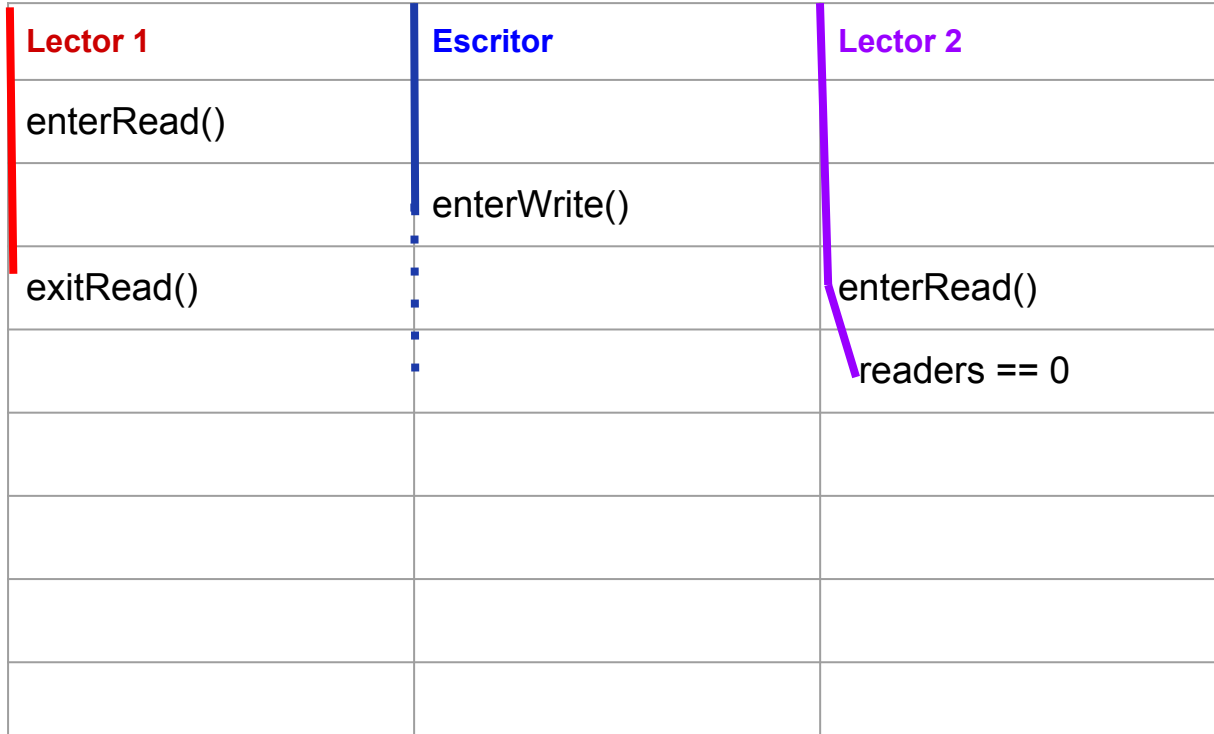
readers=1

Lectores Escritores usando SpinLocks



readers=1

Lectores Escritores usando SpinLocks



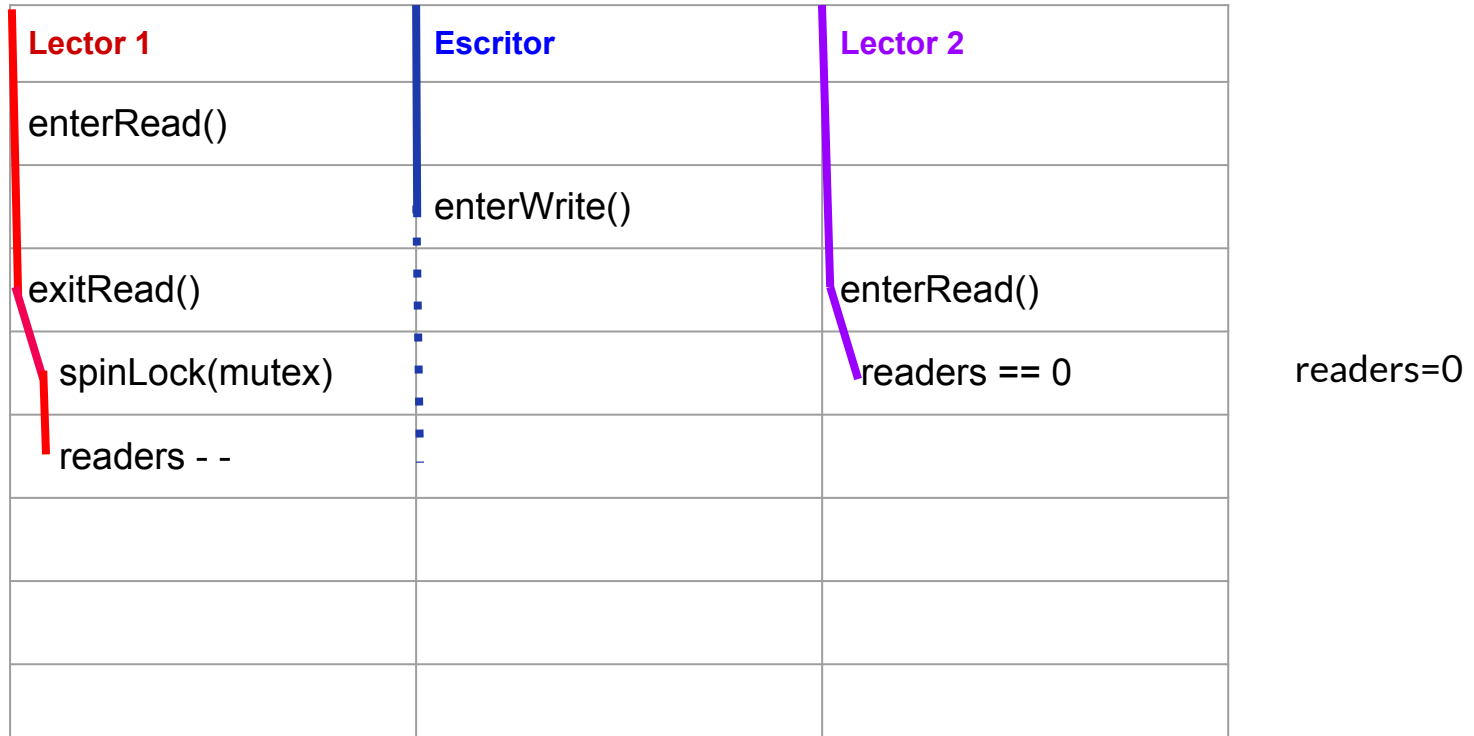
readers=1

Lectores Escritores usando SpinLocks

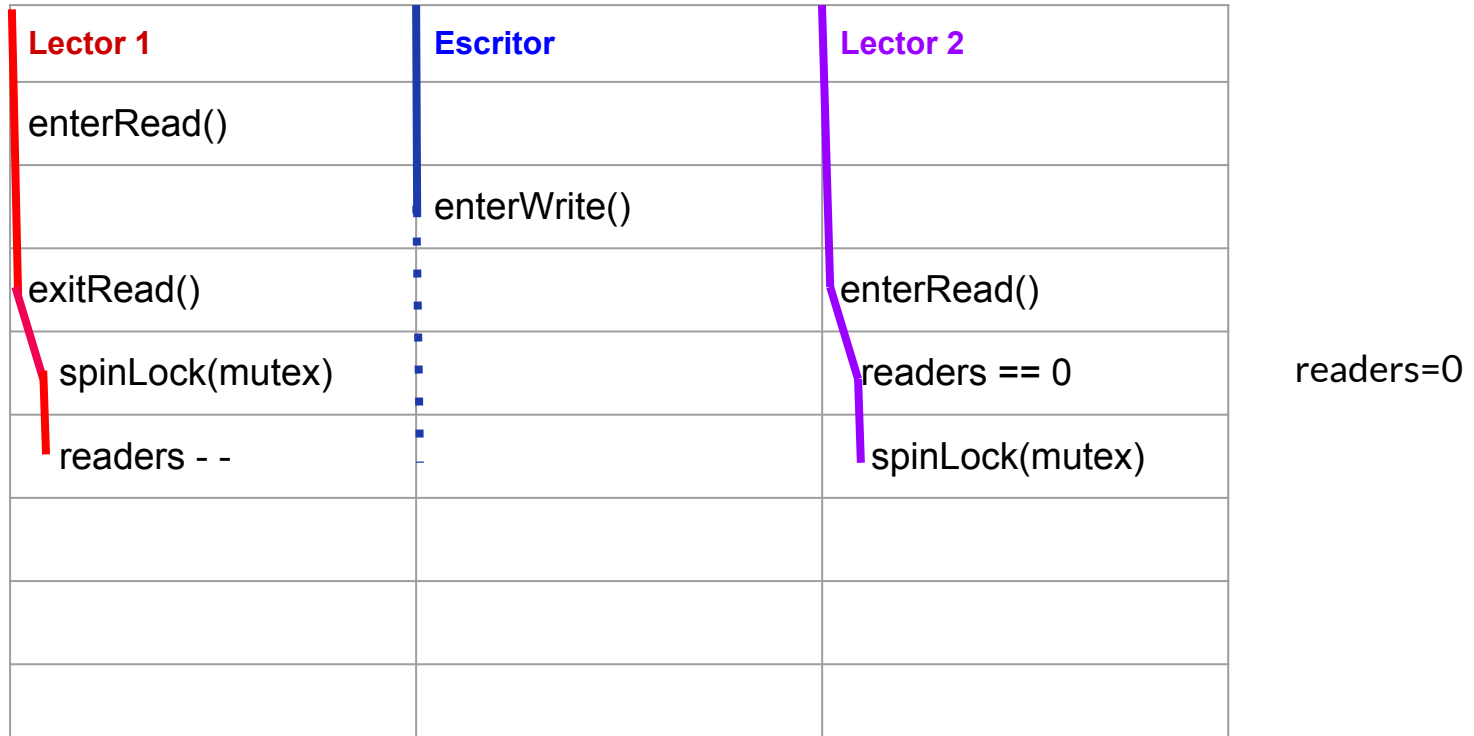


readers=1

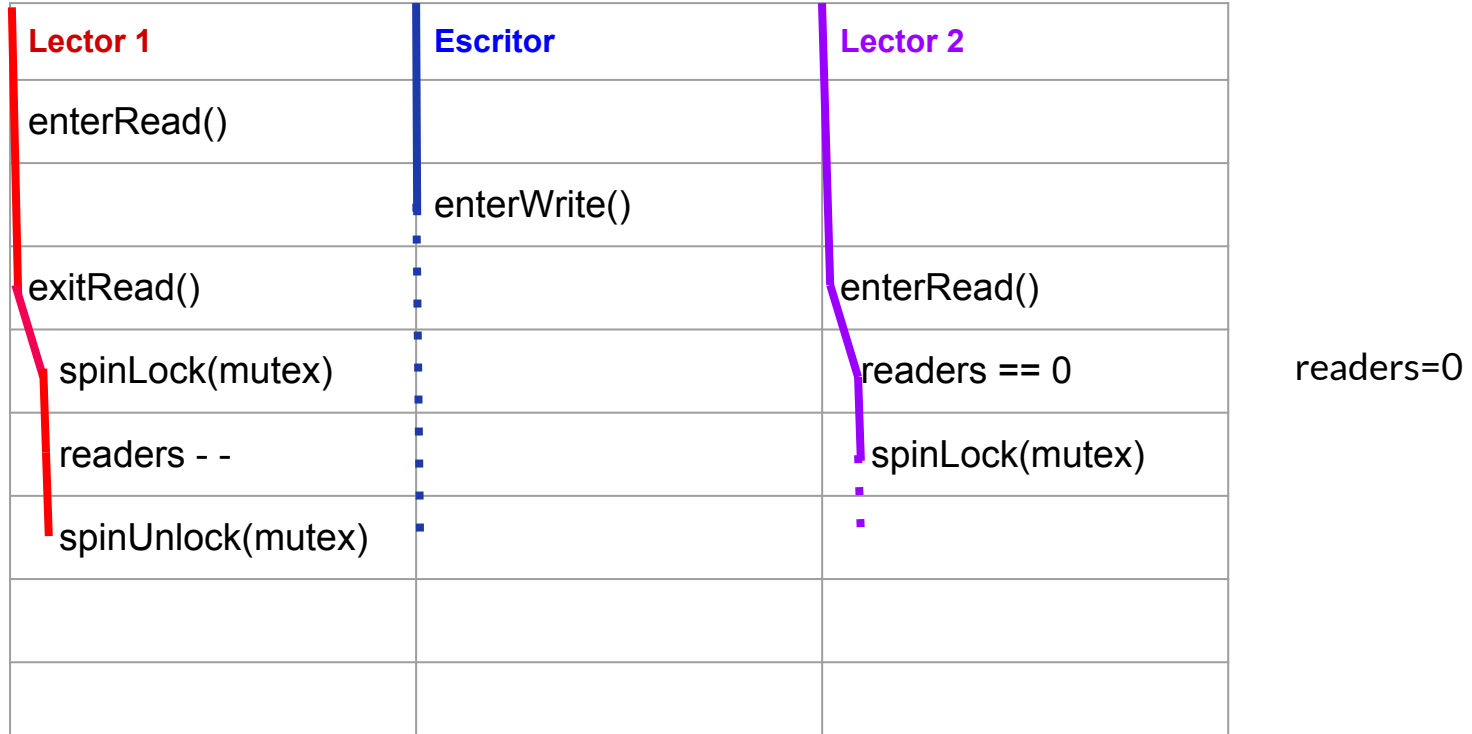
Lectores Escritores usando SpinLocks



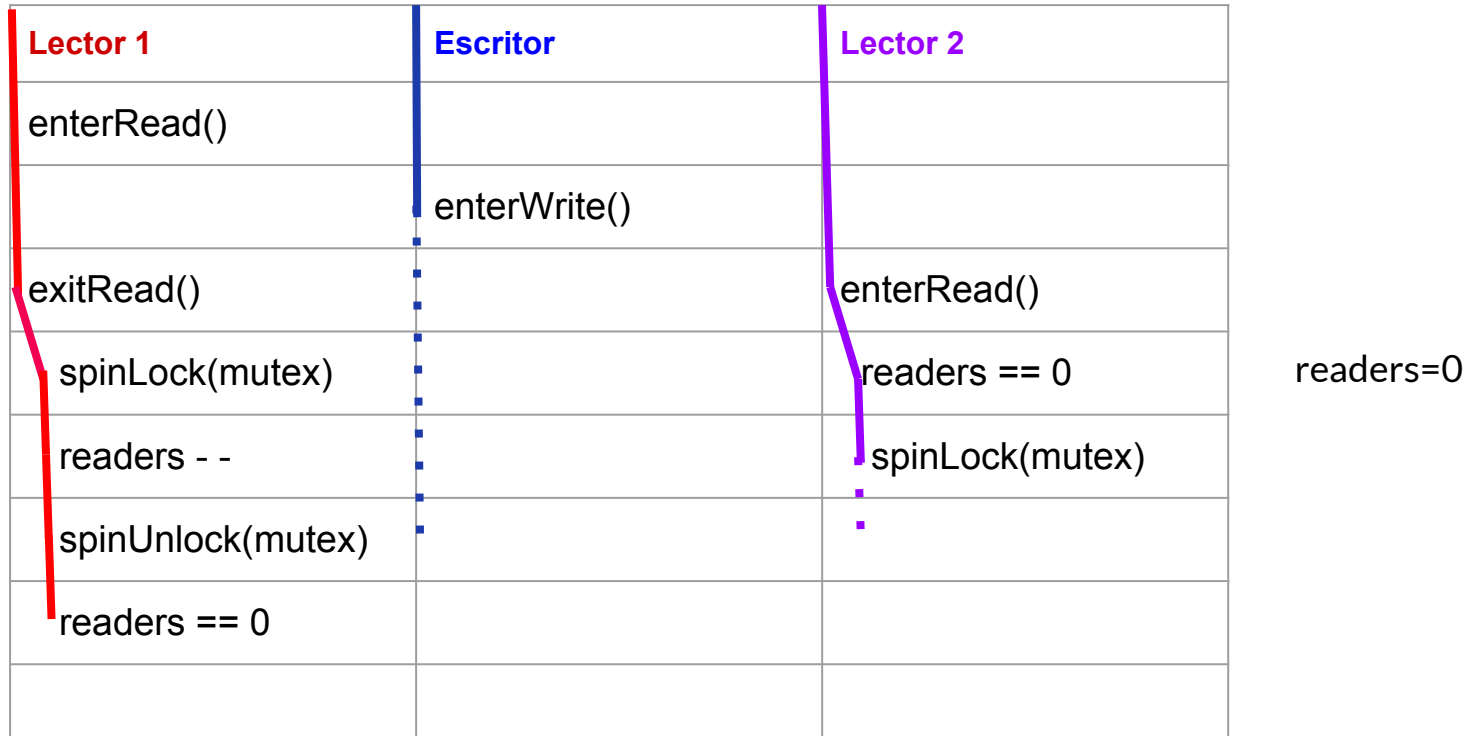
Lectores Escritores usando SpinLocks



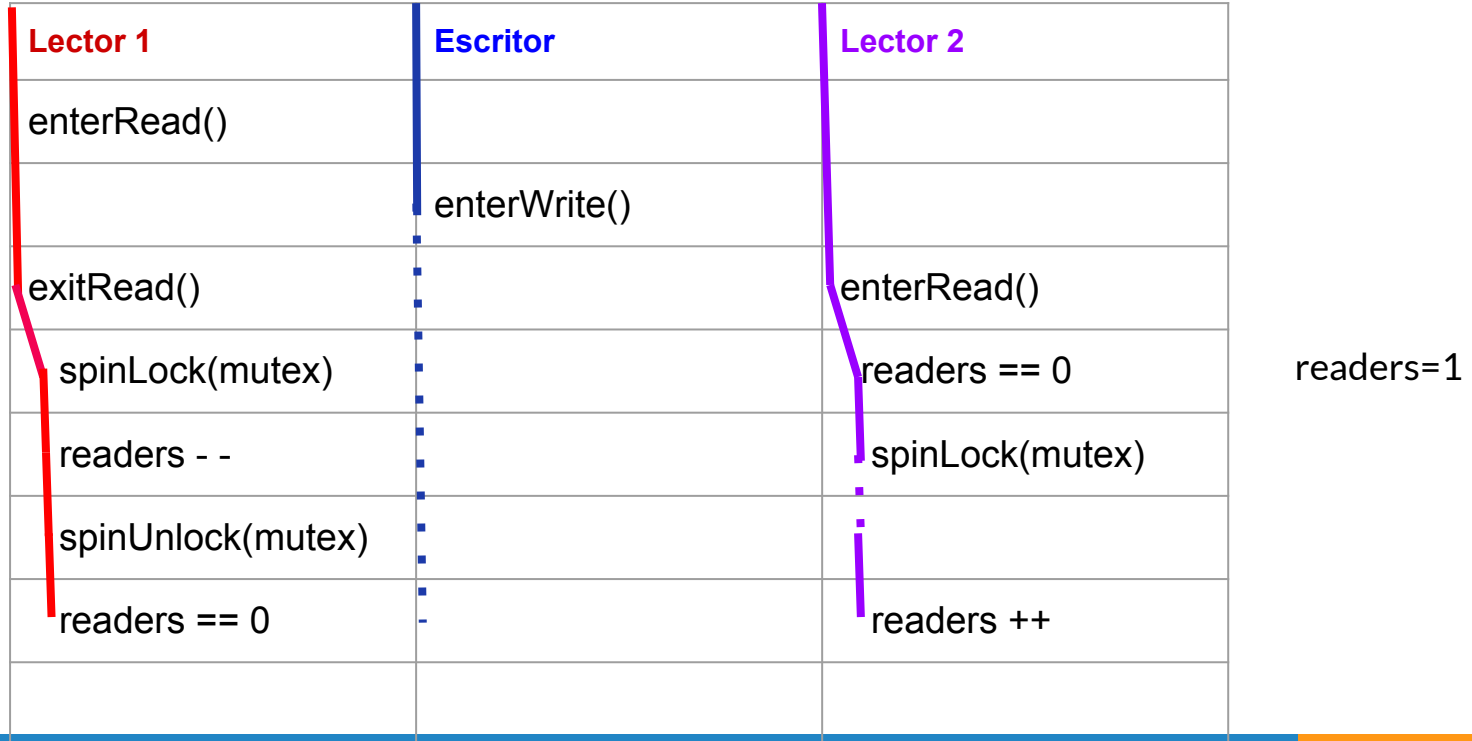
Lectores Escritores usando SpinLocks



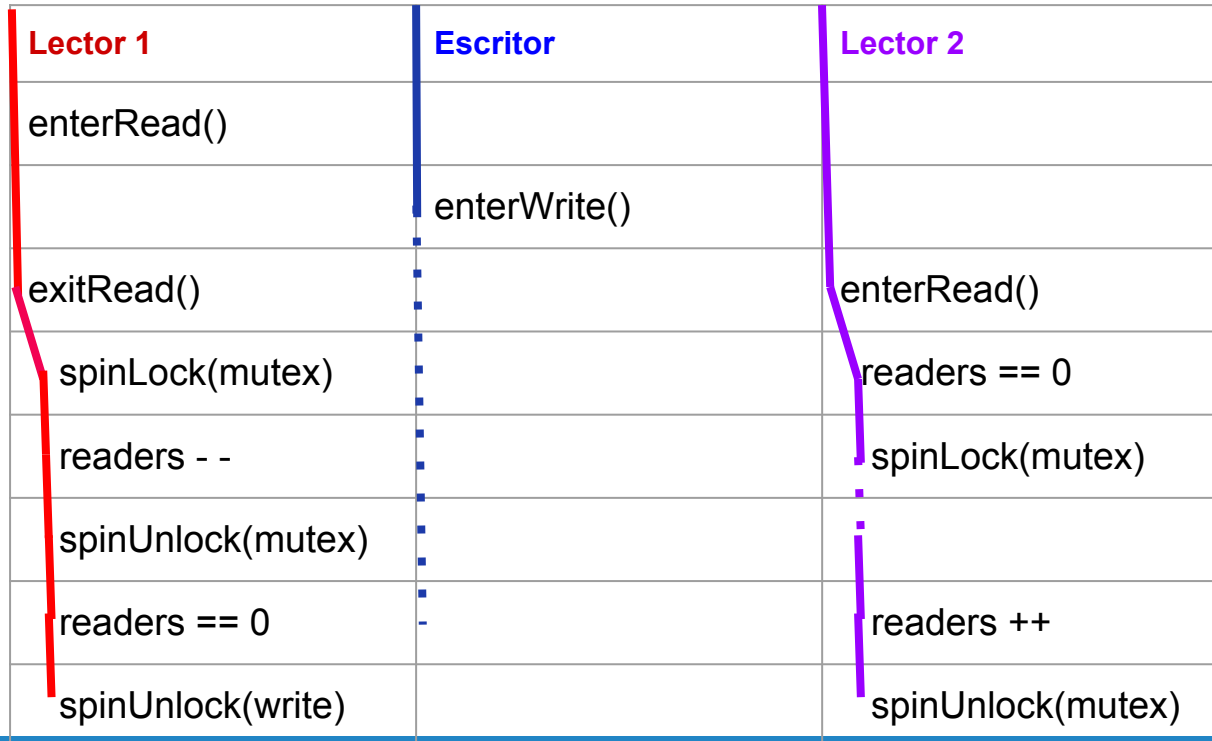
Lectores Escritores usando SpinLocks



Lectores Escritores usando SpinLocks

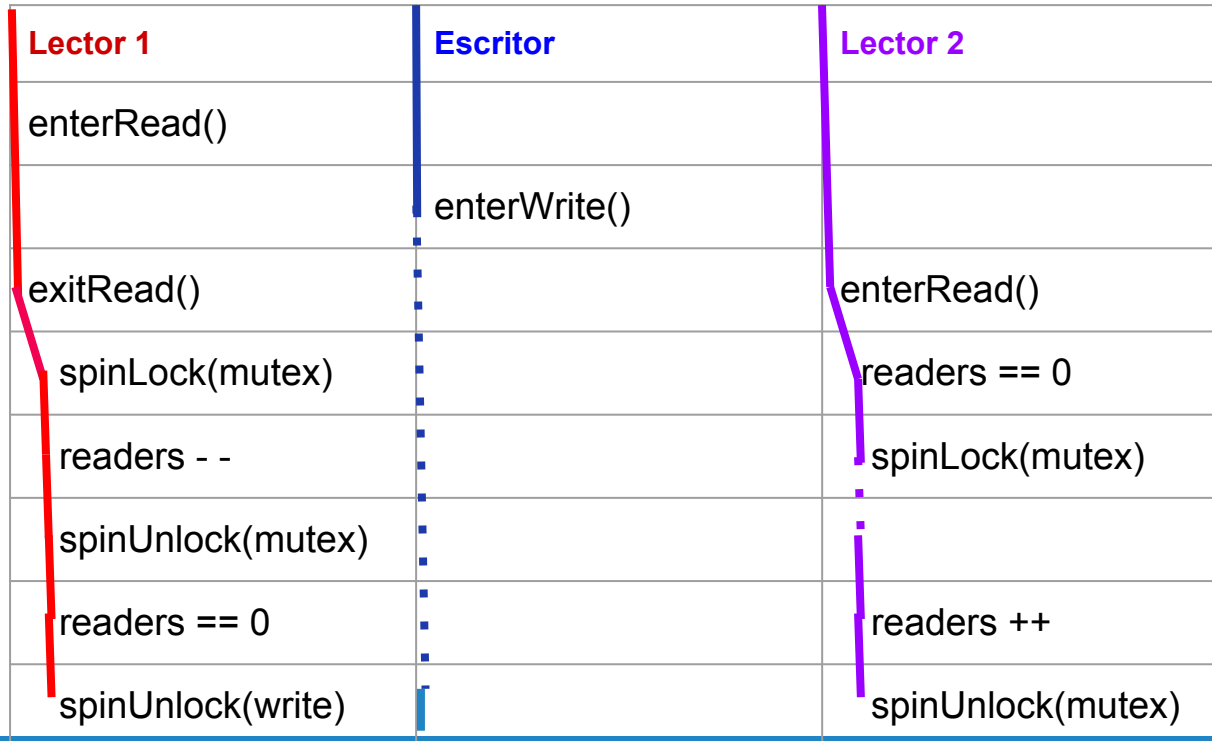


Lectores Escritores usando SpinLocks



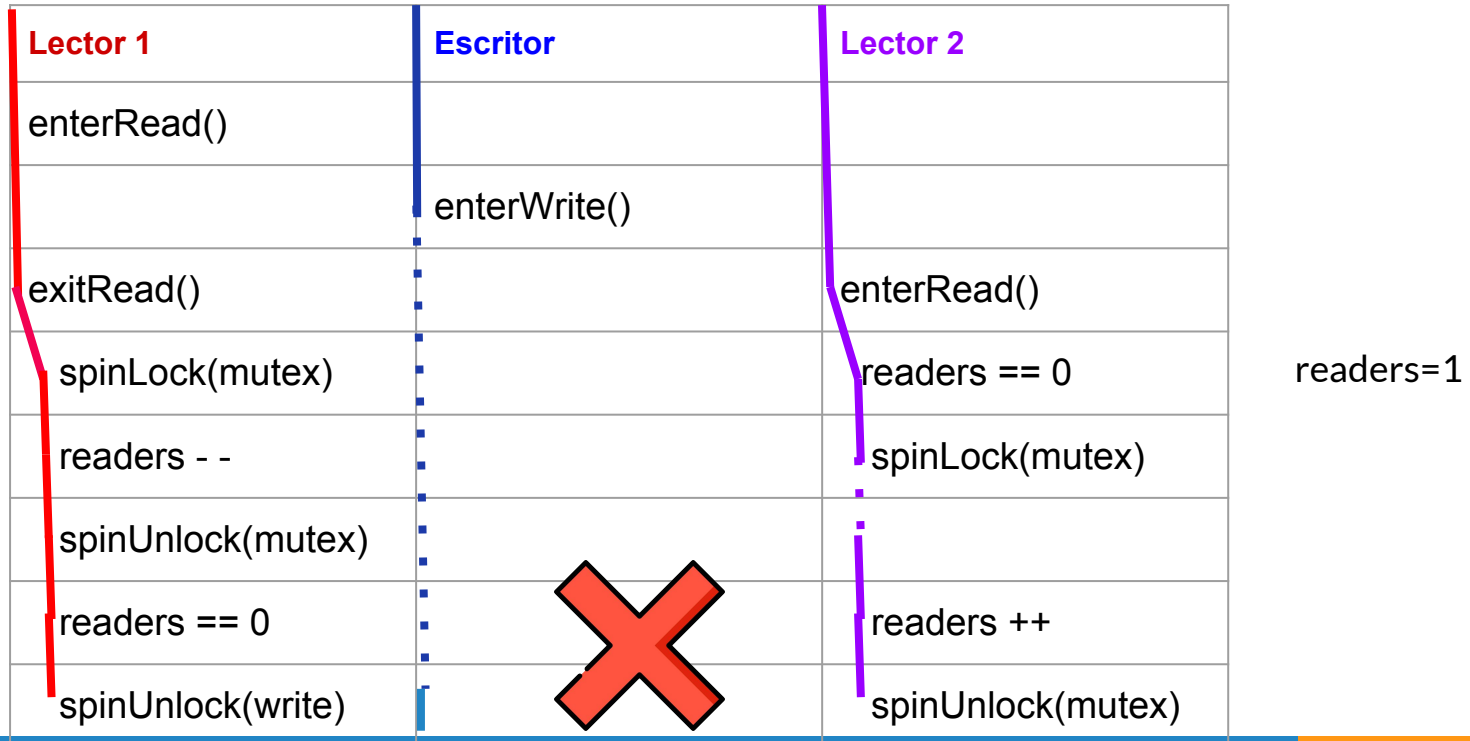
readers=1

Lectores Escritores usando SpinLocks



readers=1

Lectores Escritores usando SpinLocks



Lectores Escritores usando SpinLocks

Modifique la solución anterior para que funcione correctamente

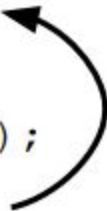
```
void enterRead() {
    if (readers == 0)
        spinLock(&write_lck);
    spinLock(&mutex_lck);
    readers++;
    spinUnlock(&mutex_lck);
}
void enterWrite() {
    spinLock(&write_lck);
}
```

```
void exitRead() {
    spinLock(&mutex_lck);
    readers--;
    spinUnlock(&mutex_lck);
    if (readers == 0)
        spinUnlock(&write_lck);
}
void exitWrite() {
    spinUnlock(&write_lck);
}
```

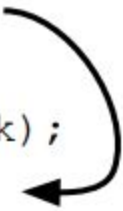
Lectores Escritores usando SpinLocks

Modifique la solución anterior para que funcione correctamente

```
void enterRead() {  
    spinLock(&mutex_lck);  
    if (readers == 0)  
        spinLock(&write_lck);  
    spinLock(&mutex_lck);  
    readers++;  
    spinUnlock(&mutex_lck);  
}
```



```
void exitRead() {  
    spinLock(&mutex_lck);  
    readers--;  
    spinUnlock(&mutex_lck);  
    if (readers == 0)  
        spinUnlock(&write_lck);  
    spinUnlock(&mutex_lck);  
}
```



P2.

Función Team

Función Team

Considere una máquina multicore en la que no existe un núcleo de sistema operativo y por lo tanto no hay un scheduler de procesos.

Se necesita formar múltiples equipos de 5 cores cada uno. Para ello, los cores invocan la función team indicando su nombre como argumento.

Esta función espera hasta que 5 cores hayan invocado team, retornando un arreglo de 5 strings con los nombres del equipo completo.

Este es un ejemplo del código de un core:

```
int player(char* name) {  
    for(;;) {  
        char** mTeam = team(name);  
        play(mTeam);  
        sleep();  
    }  
}
```

Función Team

Se debe programar la función team cuyo encabezado es:

```
char** team(char *name)
```

Se dispone de **spin-locks**. Necesitará usar variables globales y **malloc**.

Restricción: Dado que no hay un núcleo de sistema operativo, la única forma válida de esperar a que se forme el equipo es utilizando un **spin-lock**. Otras formas de busy-waiting no están permitidas. No hay fifoqueues (¡pero sí **malloc**!).

Recapitulación Auxiliar

P0.

Repaso

P1.

Lectores
Escritores

P2.

Función
team

P-Extra.

Función Robar (C2-2017)

Función Robar (C2-2017)

Considere una máquina con 8 cores físicos que comparten la memoria, sin un núcleo de sistema operativo y por lo tanto no hay scheduler de procesos.

A cada core se le asignan inicialmente 100 euros. Un core puede robar **cantidad** euros del core número **desde** invocando la función:

```
void robar(int desde, int cantidad);
```

En tal caso se le resta **cantidad** euros a la tarea **desde** y se le suman al core que invocó robar.

El parámetro **cantidad** es siempre mayor que cero.

Función Robar (C2-2017)

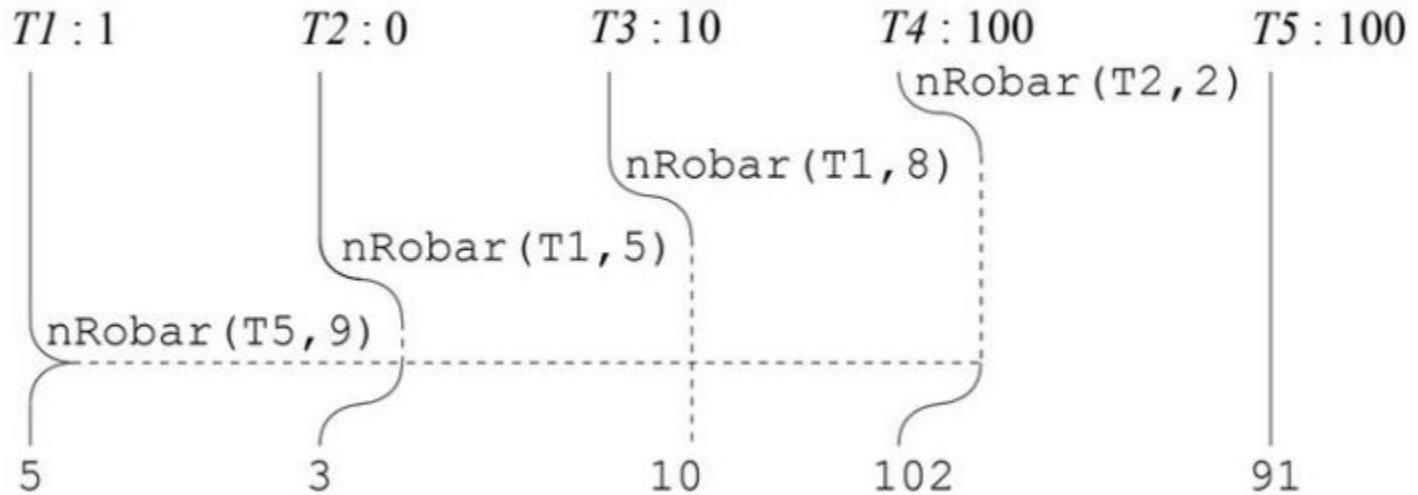
Un primer invariante es que un core no puede almacenar una cantidad negativa de euros. Si el core desde no posee suficiente dinero para robarle cantidad entonces robar debe esperar hasta que el core desde sí posea al menos la cantidad requerida.

El segundo invariante es que en un instante dado el core T no puede estar esperando robarle c euros al core U si U tiene al menos c euros.

Observe que cuando el core T lograr robar dinero, varios otros cores pueden estar esperando poder robarle dinero a T . No está especificado en qué orden deben robarle el dinero a T .

Ayuda: Use una matriz m de 8 por 8 punteros a **spin-locks**. Si $m[i][j]$ no es nulo quiere decir que $m[i][j]$ es la dirección de un **spin-lock** en el que el core i espera para robarle al core j .

Función Robar (C2-2017)





Sistemas Operativos

SpinLocks

Rodrigo Urrea