¿Qué es un semáforo?

Una primitiva sincronización.

Indicar si el siguiente multiprograma termina.
Suponga atomicidad línea a línea.
Inicialmente x=50.

```
while(0<x<100){        while (0<x<100) {
    x=x+1                  x=x-1
}                      }
```

A veces.

Suponga atomicidad línea a línea.
Inicialmente x=0

```
while(true) {          while(true) {
    x=x+1                  x=x+1
    x=x-1                  x=x-1
}                      }
```

Indique qué valores puede tomar x

x = {0,1,2}

Inicialmente x=0.
No suponga atomicidad, es decir cada incremento o decremento es: leer la memoria, operar, escribir en la memoria.

```
while(true) {          while(true) {
    x=x+1                  x=x+1
    x=x-1                  x=x-1
}                      }
```

Indique que valores puede tomar x.

Primer bucle	Segundo bucle
a : load a0, x b : add a0, a0, 1	1 : load a0, x 2 : add a0, a0, 1

c : store a0, x d : load a0, x e : sup a0, a0, 1 f : store a0, x	3 : store a0, x 4 : load a0, x 5 : sup a0, a0, 1 6 : store a0, x

x = {0,1,2}

Podemos perder un decremento tq:

	x = 0
abc	x = 1
123	x = 2
de	a0 = 2
456	x = 1
f	x = 1

De esta forma podemos repetir y llegar a cualquier valor de x positivo

Podemos perder un aumento tq:

	x = 0
ab	a0 = 1
123	x = 1
c	x = 1
def	x = 0
456	x = -1

Entonces x puede tomar todos los valores posibles en los enteros.

Suponga atomicidad línea a línea. La variable i es privada de la componente de la izquierda, la variable j es privada de la componente derecha, el arreglo a es compartido.
Inicialmente i=j=0 y a=[2,2,...,2].

```
while(i<N) {    while(j<N) {
    a[i]=0          a[j]=1
    i++             j++
}               }
```

Indique que valores del arreglo a son posibles a la salida.

a = {0,0,...,0}
a = {1,1,...,1}
a = {1,1,1…0,0,0}
a = {0,0,0…,1,1,1}

Suponga atomicidad línea a línea. La variable i es privada de la componente de la izquierda, la variable j es privada de la componente derecha, el arreglo a es compartido.
Inicialmente i=j=0, a=[2, 2, ..., 2], el semáforo s=0.

```
while(i<N) {        while(j<N) {
    sem_wait(s)         a[j]=1
    a[i]=0              j++
    i++                 sem_post(s)
}                   }
```

Indique que valores del arreglo a son posibles a la salida.

a = {0,0,...0,0} (Opción sincronizada)

Suponga atomicidad línea a línea. La variable i es privada de la componente de la izquierda, la variable j es privada de la componente derecha, el arreglo a es compartido.
Inicialmente i=j=0, a=[2, 2, ..., 2], el semáforo s=1 y t=0.

```
while(i<N) {        while(j<N) {
    sem_wait(s)         sem_wait(t)
    a[i]=0              a[j]=1
    i++                 j++
    sem_post(t)         sem_post(s)
}                   }
```

Indique que valores del arreglo a son posibles a la salida.

a = {1,1,...,1,1}

Se tiene la siguiente implementación de locks.

```
typedef struct __lock_t {
  int flag;
} lock_t;
```

```
void init(lock_t *mutex) {
  // 0 -> disponible, 1 -> tomado
  mutex->flag = 0;
}
```

```
void lock(lock_t *mutex) {
  if (mutex->flag == 0) // TEST la bandera
    if (mutex->flag == 0) // reTEST de la bandera!
      mutex->flag = 1; // gané el CTF!!!!
}
```

```
void unlock(lock_t *mutex) {
  mutex->flag = 0; // devuelvo la bandera
}
```

Decir si funciona:

A veces.

Para el siguiente multiprograma, decir si el ***Inv*ariante** se cumple.

Pre: barco=raton=0 $\wedge$ s1=1 $\wedge$ 0<N	
B: while(true) {	R: while(true) {
1 wait(s1);	a raton = raton + 1;
2 barco = barco + 1;	b post(s1);
}	}
Inv: |barco−raton|$\leq N$	

En el caso sincronizado, se cumple ya que la resta da 0 siempre.

N = 5
yo busco que la resta sea mayor a 5

si r es muy grande, luego la resta me da seguramente mayor a N por ejemplo
r = 40 → s1 = 41
luego sumamos en b
b = 1 → s1 = 40
tenemos que
1 - 40 = -39 en módulo es 39 lo que es mayor a N = 5

Por lo tanto, el invariante se mantiene a veces.