

1. Repaso clase anterior

Primitivas

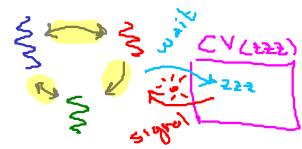
① Hilos - Concurrency

$$\text{Hilos} \rightarrow f()$$

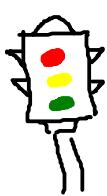
② Locks - Exclusión mutua



③ Variables de condición (CV) - Sincronización



2. Semáforo



① Exclusión Mutua



② Sincronización



→ coding



Facilita programación



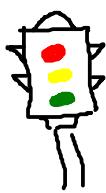
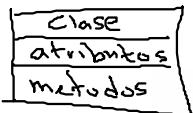
Usando semáforos puedo implementar

1. Lock

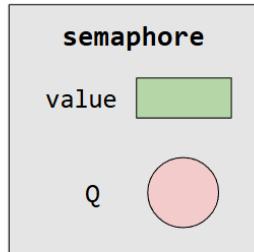
$$\text{Lock} = f(\text{semaf.}) \quad \checkmark$$

2. CV

$$CV = f(\text{semaf.}) \quad \checkmark$$



1. Init.
2. wait
3. post



```
typedef struct {
    int value;
    struct process *Q;
} sem_t;
```

1. init

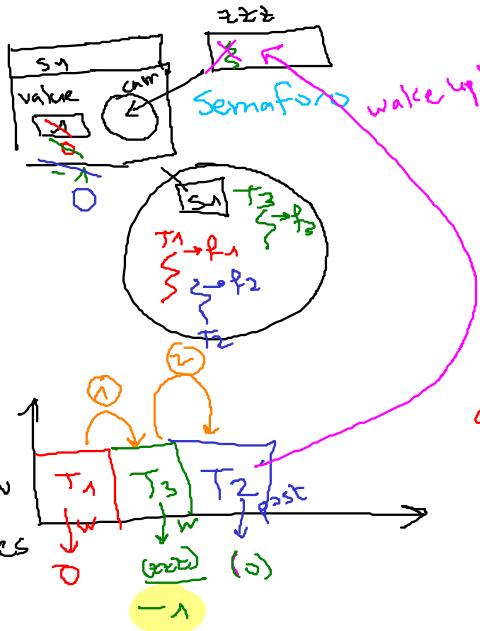
```
sem_init(sem_t *s, int value) {
    s->value = initval
}
```

2. wait

```
void sem_wait(sem_t *s) { // Must be executed atomically
    s->value--;
    if (s->value < 0) {
        add this process to s->Q;
        block();
    }
}
```

3. post

```
void sem_post(sem_t *s) { // Must be executed atomically
    s->value++;
    if (s->value <= 0) {
        remove a process P from s->Q;
        wakeup(P);
    }
}
```



value 1

Sem s1;

Sem-init(&s1, 1);

⋮

→ f1();

→ sem-wait(&s1);

⋮

→ f2();

→ sem-post(&s1);

⋮

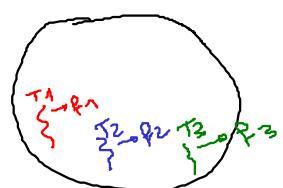
→ f3();

→ sem-wait(&s1);

⋮

⋮

Otra situación (BLOQUEO)



```
main()
Sem s1; ✓
Sem-init(&s1, 1);
```

```
⋮
```

T1 ✓ → f1

T2 ✓ → f2

T3 ✓ → f3

T4 ✓ → f4

sem-wait(&s1)

⋮

sem-wait(&s1)

⋮

sem-wait(&s1)

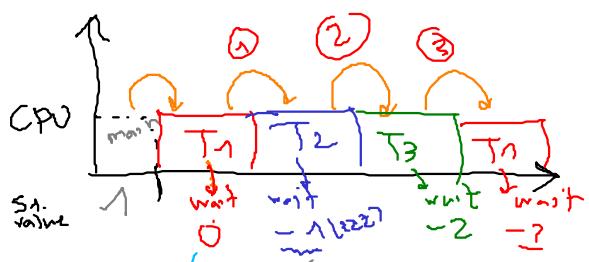
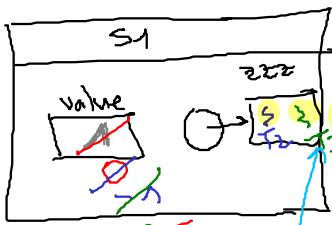
⋮

sem-wait(&s1)

⋮

sem-wait(&s1)

⋮



counts hits my bloqueos

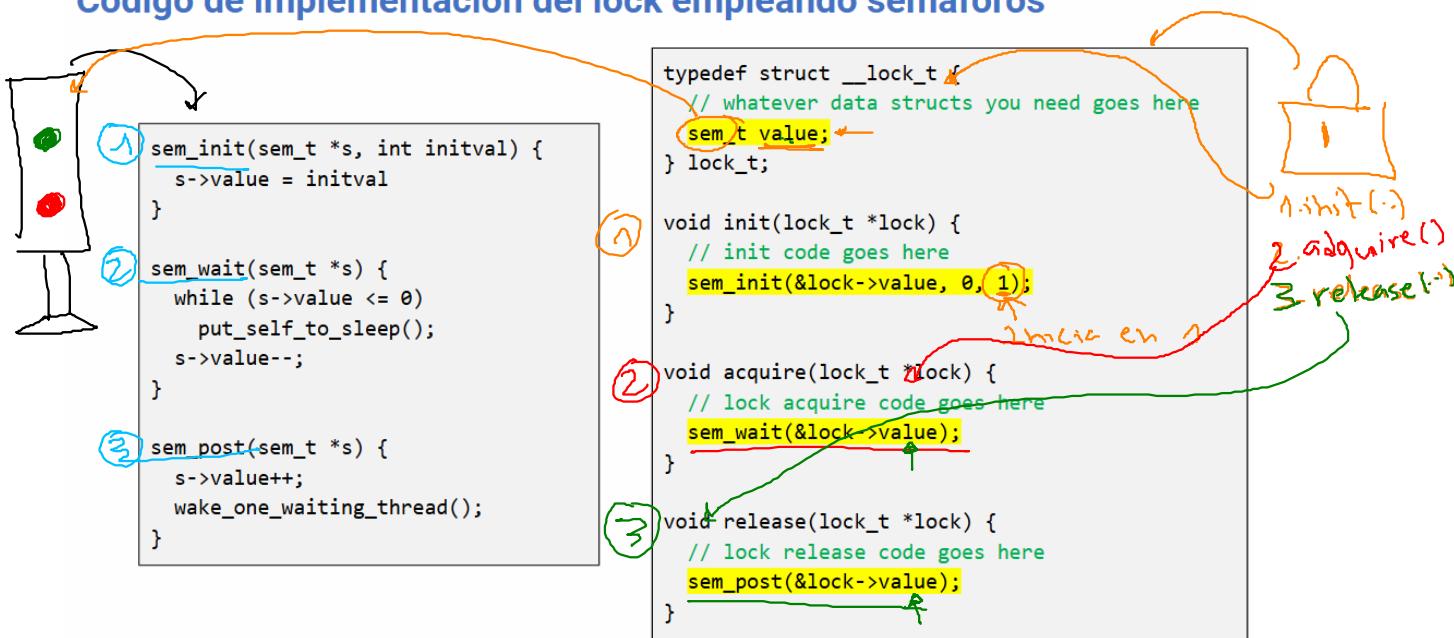
4. Uso de semáforos (Bihano)

a. Semáforo como candado (lock)

$$\boxed{\text{padilla}} = f \left(\boxed{\text{verde} \text{ rojo}} \right)$$

→ Puedes usar un semáforo inicializado en 1 para asegurar exclusión mutua

Código de implementación del lock empleando semáforos



2. Mediante el uso de semáforos puedes implementar variables de condición

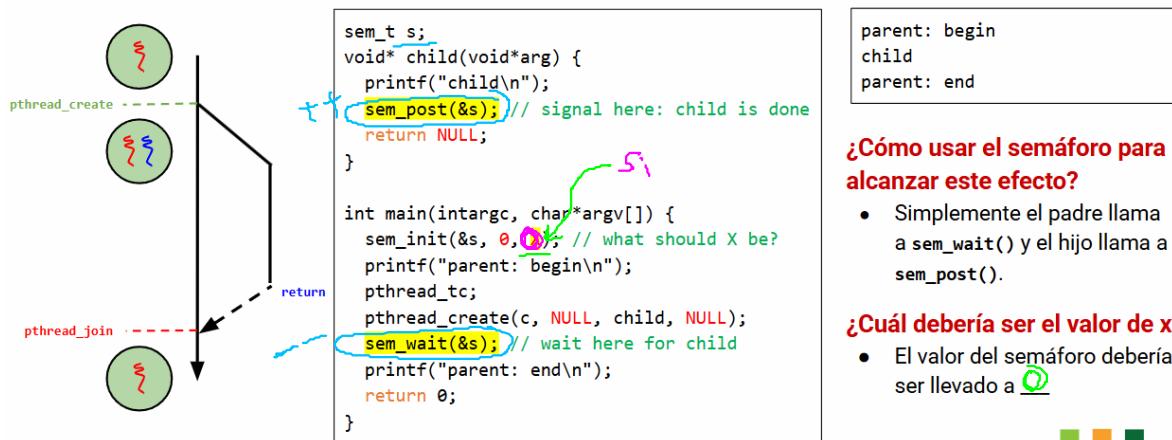


Puedes lograr sincronización entre hilos?

Rta: Sí

↓
¿Cómo?

- Los **semáforos** también son útiles para **ordenar eventos** en un programa concurrente lo que los hace aptos para ser usados como **primitiva para controlar el orden de ejecución**.



¿Cómo usar el semáforo para alcanzar este efecto?

- Simplemente el padre llama a `sem_wait()` y el hijo llama a `sem_post()`.

¿Cuál debería ser el valor de x?

- El valor del semáforo debería ser llevado a 0



- Puedes usar un semáforo inicializado en 0 para permitir sincronización

Para explicar lo que es Prueba de escritorio