


1. Sobre el final de la materia

- Parcial 2 (Módulo 3 - Concurrencia).
- Proyecto Final steps: 
- Pasteles (Bonus)
- Dudas Parcial 1 (Jose David - Revisar correo).

La presentación final del proyecto incluye los siguientes ítems a saber:

- El repositorio del código implementado con su documentación. Deben organizar un repositorio de código que incluya todos los recursos relevantes para ejecutar la aplicación desarrollada para resolver el desafío. También es obligatorio incluir un archivo README donde expliquen los requisitos del sistema, los pasos necesarios para instalar dependencias y ejecutar la aplicación. **Evaluado como práctica 4 del laboratorio.**
- Un reporte escrito siguiendo el formato de la siguiente [plantilla](#). **Evaluado como parte de la nota del Trabajo Final.**

Realizar una presentación al curso de máximo 20 minutos del proyecto ejecutado, mostrando un demo funcional del mismo. **Evaluado como parte de la nota del Trabajo Final.**

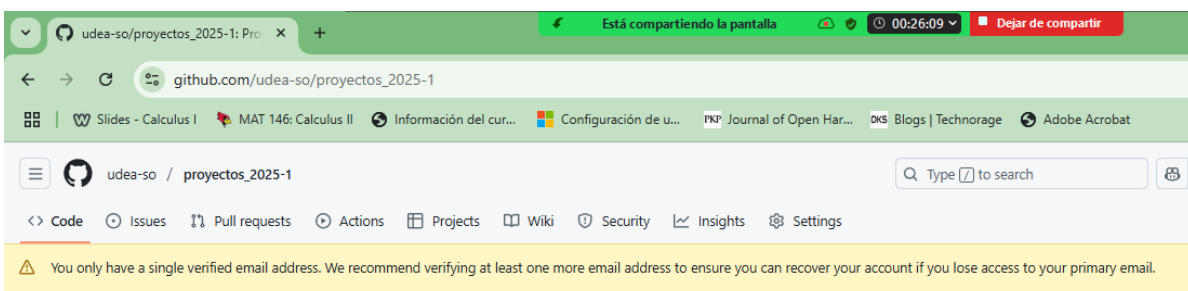
Fecha límite para enviar repositorio e informe técnico: **10 de julio de 2025 por el Moodle de Ude@.**

Fecha para las presentaciones: **15 de julio de 2025 en horario de clase.**

Usen Vibe coding

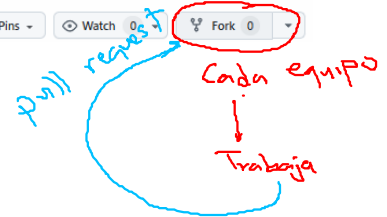
→ Reps en github:

https://github.com/udea-so/proyectos_2025-1/tree/main/virtual



proyectos_2025-1 Public

Edit Pins Watch 0 Fork 0



2. Repaso clase anterior

Primitivas

① Hilos - Concurrencia

→ f()

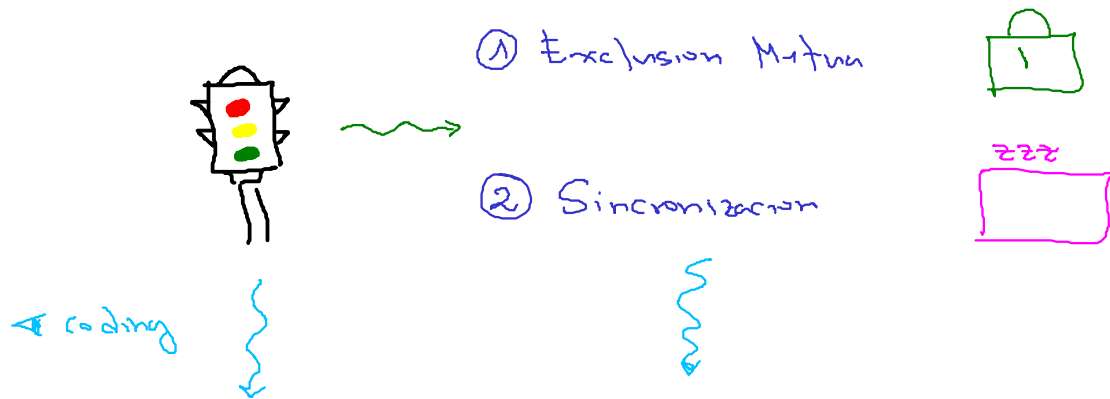
② Locks - Exclusion mutua



③ Variables de condición (CV) - Sincronización



3. Semaforo



Usando semaforos puedo implementar

1. Lock

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

2. CV

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

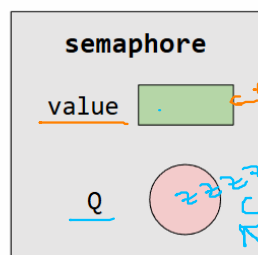
Clase
atributos
metodos



1. Init.

2. wait

3. post



1. init

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

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);
    }
}
```

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


Semaforo



4. Uso de semaforos

a. Semaforo como candado (lock)



- Puedo usar un semaforo inicializado en  para asegurar exclusion mutua

Código de implementación del lock empleando semáforos

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

sem_wait(sem_t *s) {
    while (s->value <= 0)
        put_self_to_sleep();
    s->value--;
}

sem_post(sem_t *s) {
    s->value++;
    wake_one_waiting_thread();
}
```

```
typedef struct __lock_t {
    // whatever data structs you need goes here
    sem_t value;
} lock_t;

void init(lock_t *lock) {
    // init code goes here
    sem_init(&lock->value, 0, 1);
}

void acquire(lock_t *lock) {
    // lock acquire code goes here
    sem_wait(&lock->value);
}

void release(lock_t *lock) {
    // lock release code goes here
    sem_post(&lock->value);
}
```

2. Mediante el uso de semaforos puedo implementar variables de condición

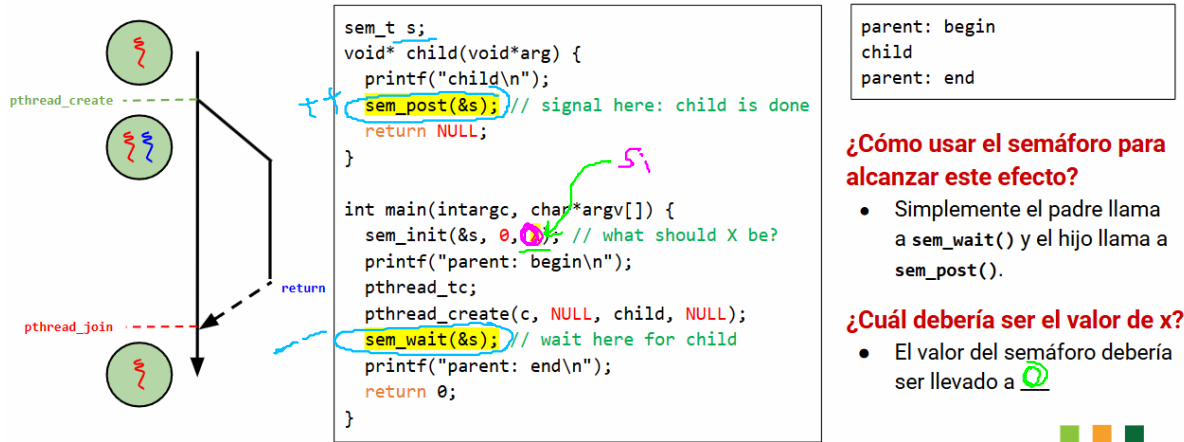


Puedo lograr sincronización entre hilos?

Rta: Si

↓
Como?

- 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**.



- Puedo usar un semáforo inicializado en 0 para permitir sincronización

↓
Para explicar hagamos Prueba de escritor