## 1° Cuatrimestre 2024

1. Revisando el diseño aplicado en algunos proyectos, se encontró el uso de las siguientes herramientas para resolver problemas de concurrencia. Para cada uno de los problemas enuncie ventajas o desventajas de utilizar la solución propuesta y menciona cual utilizaría usted.

- Renderizado de videos 3D en alta resolución, utilizando programación asincrónica.
- Aplicación que arma una nube de palabras a partir de la API de Twitter, utilizando barriers y mutex.
- Una aplicación para realizar una votación en vivo para un concurso de televisión, optimizada con Vectorización.

2. Programación asincrónica. Elija verdadero o falso y explique brevemente por qué:

- El encargado de hacer poll es el thread principal del programa.
- El método poll es llamado únicamente cuando la función puede progresar.
- El modelo piñata es colaborativo.
- La operación asincrónica inicia cuando se llama a un método declarado con async.

3. Para cada uno de los siguientes fragmentos de código indique si es o no es un busy wait. Justifique en cada caso (Nota: mineral y batteries_produced son locks).

```
for _ in 0..MINERS {
    let lithium = Arc::clone(&mineral);
    thread::spawn(move || loop {
        let mined = rand::thread_rng().gen();
        let random_result: f64 = rand::thread_rng().gen();

        *lithium.write().expect("failed to mine") += mined;
        thread::sleep(Duration::from_millis((5000 as f64 *
random_result) as u64));
    })
}
```

```
for _ in 0..MINERS {
    let lithium = Arc::clone(&mineral);
    let batteries_produced = Arc::clone(&resources);
    thread::spawn(move || loop {
        let mut lithium = lithium.write().expect("failed");
        if lithium >= 100 {
            lithium -= 100;
            batteries_produced.write().expect("failed to produce") += 1
        }
        thread::sleep(Duration::from_millis(500));
    })}
```

4. Dada la siguiente estructura, nombre si conoce una estructura de sincronización con el mismo comportamiento. Indique posibles errores en la implementación.

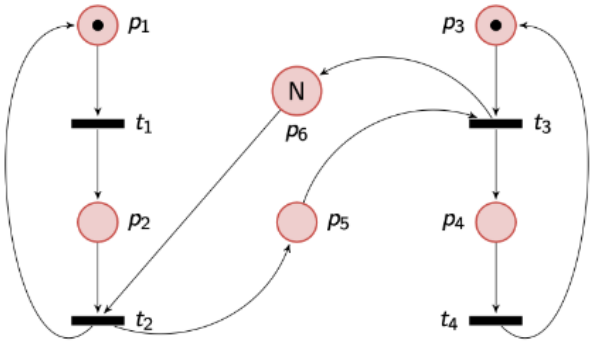
```
pub struct SynchronizationStruct {
    mutex: Mutex<i32>,
    cond_var: Condvar,
}

impl SynchronizationStruct {
    pub fn new(size: u16) -> SynchronizationStruct {
        SynchronizationStruct {
            mutex: Mutex::new(size),
            cond_var: Condvar::new(),
        }
    }

    pub fn function_1(&self) {
        let mut amount = self.mutex.lock().unwrap();
        if *amount <= 0 {
            amount = self.cond_var.wait(amount).unwrap();
        }
        *amount -= 1;
    }

    pub fn function_2(&self) {
        let mut amount = self.mutex.lock().unwrap();
        *amount += 1;
        self.cond_var.notify_all();
    }
}
```

5. Dados la siguiente red de Petri y fragmento de código, indique el nombre del problema que modelan. Indique si la implementación es correcta o describa cómo mejorarla.




```
fn main() {
    let sem = Arc::new(Semaphore::new(0));
    let buffer = Arc::new(Mutex::new(Vec::with_capacity(N)));

    let sem_cloned = Arc::clone(&sem);
    let buf_cloned = Arc::clone(&buffer);
    let t1 = thread::spawn(move || {
        loop {
            // heavy computation
            let random_result: f64 = rand::thread_rng().gen();
            thread::sleep(Duration::from_millis((500 as f64 *
random_result) as u64));

            buf_cloned.lock().expect("").push(random_result);
            sem_cloned.release()
        }
    });

    let sem_cloned = Arc::clone(&sem);
    let buf_cloned = Arc::clone(&buffer);
    let t2 = thread::spawn(move || {
        loop {
            sem_cloned.acquire();
            println!("{}", buf_cloned.lock().expect("").pop());
        }
    });

    t1.join().unwrap();
    t2.join().unwrap();
}
```