Programación Concurrente

Examen parcial 7 de mayo de 2025

Nombre: ..... Padrón: .....

Ejercicio 1

Para cada uno de los siguientes fragmentos de código indique si es o no es un busy wait. Justifique en cada caso.

1.
```
for _ in 0..MINERS {
    let copper = Arc::clone(&resource);
    thread::spawn(move || loop {
        let mined_amount = rand::thread_rng().gen_range(1..10);
        *copper.write().expect("failed to mine") += mined_amount;
        let delay = rand::thread_rng().gen_range(3000..7000);
        thread::sleep(Duration::from_millis(delay));
    });
}
```

2.
```
fn philosopher(id: usize, first_chopstick: Arc<Semaphore>, second_chopstick
Arc<Semaphore>) {
    loop {
        println!("Philosopher {}: thinking", id);
        thread::sleep(Duration::from_millis(rng.gen_range(500..1500)));

        loop {
            println!("Philosopher {}: taking first chopstick", id);
            let first_access = first_chopstick.acquire();

            println!("Philosopher {}: attempting second chopstick", id);
            match second_chopstick.try_acquire() {
                Ok(second_access) => {
                    println!("Philosopher {} eating", id);
                    let delay = rand::thread_rng().gen_range(3000..7000);
                    thread::sleep(Duration::from_millis(delay));
                    break;
                }
                Err(_) => {
                    println!("Philosopher {} was unsuccessful", id);
                    drop(first_access);
                }
            }
        }
    }
}
```

3.
```
struct Data {
    value: Option<i32>,
}

fn main() {
    let data = Arc::new(Mutex::new(Data { value: None }));

    let c1 = Arc::clone(&data);
    let t1 = thread::spawn(move || {
        _ // does some work
        let mut lock = c1.lock().unwrap();
        lock.value = Some(42);
        …
    });

    let c2 = Arc::clone(&data);
```

```
        let t2 = thread::spawn(move || {
            _
            loop {
                let lock = c2.lock().unwrap();
                if lock.value.is_some() {
                    println!("Valor obtenido: {}", lock.valor.unwrap());
                    break;
                }
            }
            _
        });

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

Ejercicio 2

Dado el siguiente fragmento de código, indique el nombre del problema que modela. Indique si la implementación es correcta o describa cómo mejorarla.

```
fn main() {
    const N: usize = 5;

    let producers_waiting = Arc::new(Semaphore::new(0));
    let consumer_done = Arc::new(Semaphore::new(0));
    let data = Arc::new(Mutex::new(None))

    let producers_waiting_clone = producers_waiting.clone();
    let consumer_done_clone = consumer_done.clone();

    let consumer = thread::spawn(move || loop {
        println!("[Consumer] Sleeping...");
        producers_waiting_clone.acquire();

        println!("Doing my job with data {}...", data.lock().expect("can read"));
        thread::sleep(Duration::from_secs(2));

        consumer_done_clone.release();
        println!("Job finished");
    });

    let producers: Vec<_> = (0..N).map(|id| {
        let producers_waiting_clone = producers_waiting.clone();
        let consumer_done_clone = consumer_done.clone();

        thread::spawn(move || loop {
            let delay = rand::thread_rng().gen_range(3000..7000);
            thread::sleep(Duration::from_millis(delay));
            println!("[Producer {}] Arrived", id);
            producers_waiting_clone.release();

            *data.lock().expect("can't set data") = Some(id)

            println!("[Producer {}] waiting to be consumed", id);
            consumer_done_clone.acquire();

            println!("[Producer {}] Leaving", id);
        })
    }).collect();

    consumer.join().unwrap();
    for p in producer { p.join().unwrap(); }
```

Modele una red de Petri para el problema del punto anterior. Si hubiera propuesto mejoras, inclúyalas.

Ejercicio 3

Se quiere abrir una estación de servicio YPF con 10 surtidores. Por el momento solamente se logró la habilitación de 3 tanques internos de nafta, el objetivo es a futuro poder habilitar más. Los surtidores atienden a los autos que vienen a cargar combustible, pero deben acceder a los tanques internos de un surtidor a la vez. Además, cuando llega el camión cisterna de YPF a recargar el combustible disponible en la estación de servicio, este debe asegurarse de que ningún tanque esté siendo usado por un surtidor ya que eso representa un riesgo de seguridad.

Diseñe el sistema utilizando el modelo de actores, y para cada entidad defina cuáles son los estados internos y los mensajes que intercambian.

Ejercicio 4

Describa y justifique con que modelo de concurrencia modelaría la implementación para cada uno de los siguientes casos de uso

a. Una aplicación que encripta múltiples archivos .PDF con una clave privada.
b. El backend para una aplicación de un restaurante donde múltiples mozos pueden sumar pedidos y cobrar en todas las mesas.
c. Una aplicación que consulta periódicamente las cotizaciones de Bitcoin en cientos de exchanges y brokers.
d. Una aplicación que subtitula para videos, convirtiendo voz a texto con una red neuronal.

Ejercicio 5

Verdadero o Falso. Justifique

a. Procesos, hilos y tareas asincrónicas todos poseen espacios de memoria independientes entre sí.
b. El scheduler del sistema operativo puede detener una tarea asincrónica puntual y habilitar la ejecución de otra para el mismo proceso.
c. En vectorización, tanto las operaciones verticales como las operaciones horizontales devuelven resultados vectoriales
d. Un hilo esperando sobre una condvar sólo puede despertarse cuando otro hilo hace signal de la misma.