# Universidad Nacional del Altiplano

**Educando mentes, Cambiando el mundo**


## Facultad de Ingeniería Mecánica Eléctrica, Electrónica y Sistemas

**Escuela Profesional de Ingeniería de Sistemas**

# Problemas clasicos de Paralelismo y Concurrencia

**PARALELISMO, CONCURRENCIA Y SISTEMAS DISTRIBUIDOS**

**Ing. ROMERO FLORES ROBERT ANTONIO**

**estudiante**

↬ **Larota Pilco David Brahyan** ↫

30 de abril de 2024

# 1 | Introducción

## 1.1 | Definición de paralelismo y concurrencia

**Paralelismo**

El paralelismo es un concepto utilizado en informática para referirse a la ejecución simultánea de múltiples tareas o instrucciones en un sistema. Se refiere a la capacidad de dividir una tarea en partes más pequeñas y realizarlas al mismo tiempo, aprovechando la capacidad de procesamiento paralelo de los sistemas informáticos. El paralelismo puede mejorar significativamente el rendimiento y la eficiencia, ya que permite realizar varias tareas al mismo tiempo en lugar de secuencialmente.

**Concurrencia**

la capacidad de múltiples partes de un sistema (como procesos, subprocesos o tareas) para ejecutarse de manera simultánea e independiente. Esto permite que distintas operaciones se realicen de forma concurrente, compartiendo recursos y compitiendo por ellos, lo que puede incrementar la eficiencia y el rendimiento del sistema. Sin embargo, la concurrencia también presenta desafíos, como la gestión adecuada de la sincronización y la exclusión mutua para evitar condiciones de carrera y garantizar la consistencia de los datos.

## 1.2 | Importancia de abordar estos problemas

Abordar los problemas de paralelismo y concurrencia es esencial en informática debido a su impacto en el rendimiento, la eficiencia de los recursos, la escalabilidad y la experiencia del usuario. El uso eficiente de la concurrencia y el paralelismo mejora el rendimiento al permitir la ejecución simultánea de tareas, optimiza la utilización de recursos compartiéndolos entre procesos, y facilita la escalabilidad al distribuir la carga de trabajo. Además, es fundamental para aprovechar al máximo el hardware multinúcleo y garantizar la fiabilidad y tolerancia a fallos de los sistemas, al evitar problemas como las condiciones de carrera y garantizar la consistencia de los datos.

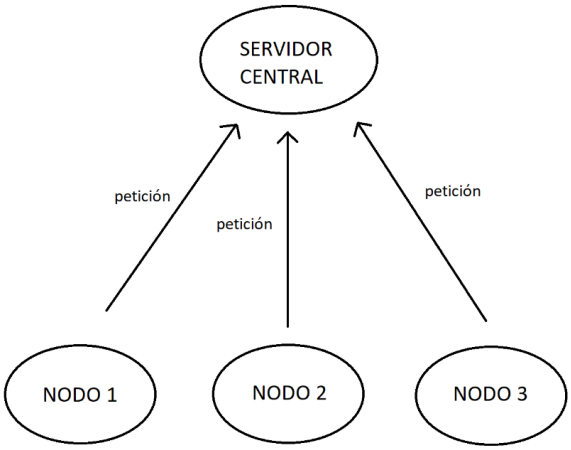
# 2 | Problemas de Paralelismo

## 2.1 | Exclusión mutua

La exclusión mutua es un concepto fundamental en el diseño de sistemas concurrentes, donde múltiples procesos o hilos comparten recursos comunes. Se refiere a la técnica que garantiza que, en un momento dado, solo un proceso tenga acceso a un recurso compartido, evitando así que varios procesos intenten modificar el recurso simultáneamente, lo que podría conducir a resultados no deseados o inconsistentes. La exclusión mutua se implementa mediante mecanismos como los semáforos, los mutex (mutual exclusion), los cerrojos (locks) u otros mecanismos de sincronización, que permiten que un proceso

bloquee temporalmente el acceso al recurso mientras lo utiliza, impidiendo que otros procesos lo accedan hasta que se libere.

Figura 1: Imagen que representa la Exclusión Mutua




## 2.2 | Condición de carrera

Una condición de carrera es un fenómeno que ocurre en sistemas concurrentes cuando el resultado de la ejecución depende del orden o la sincronización de eventos entre múltiples procesos o hilos. Sucede cuando el resultado de una operación depende del tiempo y la secuencia de ejecución de los procesos, y puede variar de una ejecución a otra. Las condiciones de carrera pueden surgir cuando varios procesos intentan acceder y modificar simultáneamente recursos compartidos sin la adecuada sincronización.

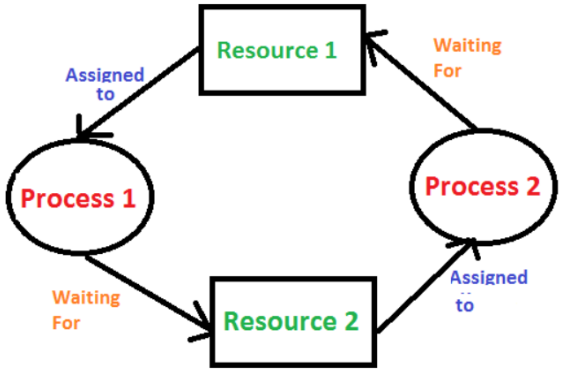
Figura 2: Imagen que representa la Condición de carrera


## 2.3 | Deadlock

Un deadlock, también conocido como **bloqueo mutuo**, es una situación en la que dos o más procesos o hilos quedan atrapados en un estado en el que ninguno puede continuar ejecutándose porque cada uno está esperando que el otro libere un recurso necesario. En otras palabras, los procesos se bloquean entre sí, creando un punto muerto en la ejecución del programa.

Un deadlock generalmente ocurre cuando los procesos tienen acceso exclusivo a recursos compartidos y adquieren múltiples recursos pero los retienen mientras esperan la liberación de otros recursos que están en posesión de otros procesos.

Figura 3: Imagen que representa la Deadlock




### 2.4 | Starvation

El **starvation** es un problema en sistemas concurrentes donde un proceso o hilo queda permanentemente excluido de acceder a un recurso o de completar su tarea, a pesar de que la planificación del sistema le permite ejecutarse. Este fenómeno ocurre cuando otros procesos o hilos en el sistema tienen prioridad sobre el proceso hambriento y monopolizan los recursos necesarios, impidiéndole avanzar.

## 3 | Problemas de Concurrencia

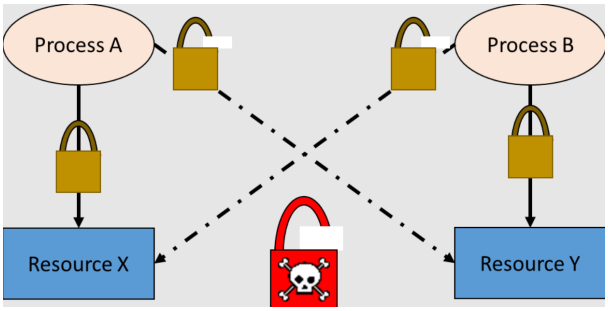
### 3.1 | Inanición

La **inanición** es un problema que ocurre en sistemas concurrentes cuando un proceso o hilo queda permanentemente excluido de acceder a recursos o de completar su tarea, a pesar de estar listo y esperando para ejecutarse. Este fenómeno se produce debido a que otros procesos en el sistema tienen prioridad sobre él y monopolizan los recursos necesarios, impidiéndole avanzar.

### 3.2 | Interbloqueo

El interbloqueo, también conocido como **deadlock**, es una situación en la que dos o más procesos o hilos quedan atrapados en un estado en el que ninguno puede continuar ejecutándose porque cada uno está esperando que el otro libere un recurso necesario. En otras palabras, los procesos se bloquean entre sí, creando un punto muerto en la ejecución del programa.

Figura 4: Imagen que representa la Interbloqueo




## 3.3 | Contención de recursos

La contención de recursos se refiere a la competencia o disputa que surge cuando múltiples procesos o hilos intentan acceder a un recurso compartido al mismo tiempo. Esta situación puede conducir a conflictos y problemas de rendimiento si los recursos no están correctamente gestionados.

# 4 | Técnicas de sincronización

## 4.1 | Monitores

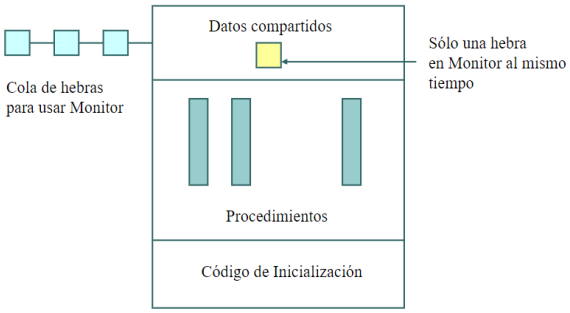
Los monitores son estructuras de control de acceso a recursos compartidos en programación concurrente. Fueron introducidos por C. A. R. Hoare en 1974 como una forma de sincronizar el acceso a datos compartidos entre múltiples procesos o hilos.

Un monitor combina un tipo de datos abstracto (TDA) con un conjunto de procedimientos o métodos asociados, y proporciona un mecanismo de sincronización para garantizar que solo un hilo pueda ejecutar un procedimiento del monitor en un momento dado. Esto evita problemas de concurrencia, como las condiciones de carrera y los interbloqueos.

**Los monitores constan de dos partes principales**

- Variables y procedimientos internos: Estos son los datos y las operaciones que componen el TDA del monitor. Están protegidos por el monitor y solo pueden ser accedidos por los hilos a través de los procedimientos definidos en el monitor.
- Condición de espera y señalización: Los monitores incluyen un mecanismo para permitir que los hilos esperen a que se cumpla una determinada condición y luego sean notificados cuando esto sucede. Esto se logra mediante las operaciones de espera (wait) y señalización (signal) proporcionadas por el monitor.

Figura 5: Imagen que representa las Monitores




## 4.2 | Semáforos

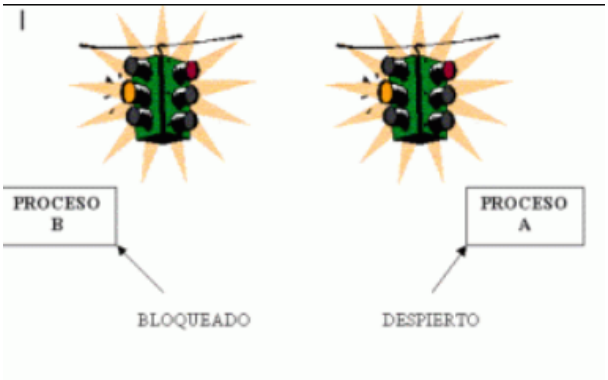
Los semáforos son una de las primitivas de sincronización más utilizadas en programación concurrente. Fueron propuestos por primera vez por Edsger Dijkstra en 1965 como un mecanismo para controlar el acceso a recursos compartidos entre procesos o hilos.

Un semáforo es una variable entera que se utiliza para controlar el acceso a recursos compartidos mediante dos operaciones fundamentales:

- P(s) (espera): Si el valor del semáforo es mayor que cero, se decrementa en uno y el proceso continúa su ejecución. Si el valor es cero, el proceso se bloquea hasta que el valor del semáforo sea mayor que cero.
- V(s) (señal): Incrementa el valor del semáforo en uno y despierta a uno de los procesos que estén esperando a que el valor del semáforo sea mayor que cero.

Los semáforos pueden tener diferentes variantes, como semáforos binarios (con valores 0 y 1), semáforos contadores (con valores no negativos) y semáforos mutex (para exclusión mutua).

Figura 6: Imagen que representa los Semaforos




## 4.3 | Barreras

Las barreras son estructuras de sincronización utilizadas en programación concurrente para coordinar la ejecución de múltiples hilos o procesos, asegurando que estos alcancen

un punto de sincronización antes de continuar con la ejecución.

Una barrera consta de un contador interno que se decrementa cada vez que un hilo o proceso alcanza la barrera. Cuando el contador alcanza cero, todos los hilos o procesos que esperan en la barrera pueden continuar con su ejecución.

Las barreras son útiles cuando se necesita que todos los hilos o procesos lleguen a un punto de sincronización antes de proceder, como en el caso de dividir una tarea en subprocesos independientes que deben completarse antes de combinar los resultados.

Las barreras son especialmente útiles en aplicaciones paralelas donde se realiza un procesamiento en paralelo y se necesita sincronizar los resultados parciales antes de continuar con la siguiente fase del procesamiento.

# 5 | Modelos de ejecución concurrente

## 5.1 | Modelo de lectores-escritores

El modelo de lectores-escritores es un problema clásico de sincronización en programación concurrente que involucra el acceso a un recurso compartido por múltiples procesos. En este modelo, hay dos tipos de procesos:

1. **Lectores:** Procesos que solo leen el recurso compartido pero no lo modifican.
2. **Escritores:** Procesos que leen y escriben en el recurso compartido.

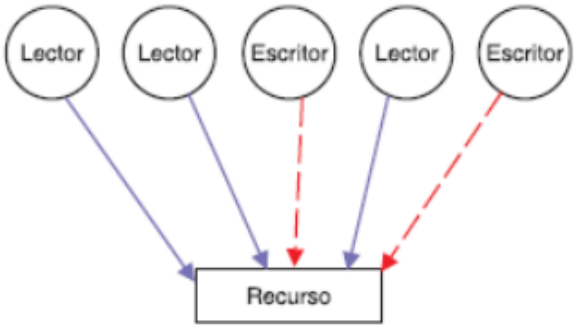
El objetivo del modelo es permitir el acceso concurrente de múltiples lectores al recurso compartido mientras se garantiza la exclusión mutua entre los escritores y entre un escritor y cualquier lector. Esto significa que los lectores pueden acceder al recurso compartido simultáneamente, pero los escritores deben tener un acceso exclusivo al recurso para evitar problemas de consistencia de datos.

El modelo de lectores-escritores se puede implementar de varias maneras, utilizando diferentes técnicas de sincronización como semáforos, mutex, monitores u otras estructuras.

**Algunas variantes del modelo incluyen:**

- Lectores preferidos: Se da prioridad a los lectores sobre los escritores, lo que puede llevar a la inanición de los escritores si hay una alta concurrencia de lectores.
- Escritores preferidos: Se da prioridad a los escritores sobre los lectores, lo que puede llevar a la inanición de los lectores si hay una alta concurrencia de escritores.
- Alternancia justa: Se alterna entre lectores y escritores de manera justa para evitar la inanición de ambos.

Figura 7: Imagen que representa el modelo Lector y Escritor




## 5.2 | Modelo de productor-consumidor

El modelo de productor-consumidor es otro problema clásico de sincronización en programación concurrente, que involucra la coordinación entre dos tipos de procesos:

1. **Productores:** Procesos que generan datos o recursos y los colocan en un búfer o cola compartida.
2. **Consumidores:** Procesos que toman los datos o recursos del búfer o cola y los utilizan o procesan.

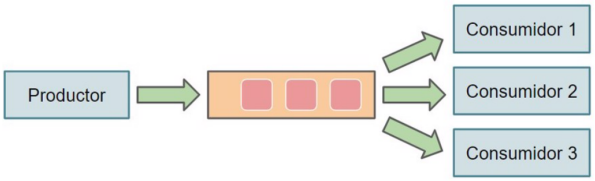
El objetivo del modelo es permitir que los productores y consumidores trabajen de manera concurrente, evitando problemas como la sobreproducción o la falta de datos. (NetMentor, 2024)

El modelo de productor-consumidor se puede implementar utilizando una estructura de datos compartida, como un búfer o una cola, y mecanismos de sincronización, como semáforos o mutex, para garantizar la exclusión mutua y la sincronización entre los productores y consumidores.

**Algunas variantes del modelo incluyen**

- **Productor único, consumidor único:** Un único productor genera datos y un único consumidor los consume.
- **Productores múltiples, consumidor único:** Varios productores generan datos y un único consumidor los consume.
- **Productor único, consumidores múltiples:** Un único productor genera datos y varios consumidores los consumen.
- **Productores múltiples, consumidores múltiples:** Varios productores generan datos y varios consumidores los consumen.

Figura 8: Imagen que representa el modelo Productor Consumidor




## 5.3 | Modelo de filósofos comensales

El problema de los filósofos comensales, también conocido como el problema de los filósofos hambrientos, es un problema clásico de sincronización en programación concurrente propuesto por Edsger Dijkstra en 1965. Se plantea de la siguiente manera: (*Programación concurrente*, 2023)

Imagina que hay cinco filósofos sentados alrededor de una mesa redonda, y entre cada par de filósofos hay un tenedor. Cada filósofo pasa su tiempo pensando y comiendo. Para comer, un filósofo necesita usar los dos tenedores que están a su lado.

El desafío consiste en diseñar un algoritmo de sincronización que permita a los filósofos alternar entre los estados de pensar y comer de manera segura, evitando posibles interbloqueos o inaniciones.

**Algunas soluciones al problema de los filósofos comensales incluyen**

- **Solución del camarero:** Introduce un camarero que controla el acceso a los tenedores y evita la competencia entre los filósofos por los recursos compartidos.
- **Solución de asignación de recursos:** Los filósofos adquieren los tenedores de uno en uno y los liberan en el mismo orden, evitando la posibilidad de interbloqueos.
- **Solución de jerarquía de recursos:** Se asigna un número a cada tenedor y se establece una jerarquía para su adquisición, evitando así ciclos de espera y garantizando que un filósofo pueda comer siempre y cuando pueda adquirir los tenedores de manera secuencial.

Figura 9: Imagen que representa el modelo de los filosofos

# 6 | Referencias

## Referencias

NetMentor. (2024, abril). *Patrón productor consumidor.* Descargado de https://www.netmentor.es/entrada/patron-productor-consumidor ([Online; accessed 26. Apr. 2024])

*Programación concurrente.* (2023, noviembre). Descargado de https://ferestrepoca.github.io/paradigmas-de-programacion/progconcurrente/concurrente_teoria/index.html ([Online; accessed 26. Apr. 2024])