Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3086 Programación de microprocesadores

Ciclo 2 de 2,024

Laboratorio 08: sincronización con semáforos

Realizado por grupo #1:
Pablo Daniel Barillas Moreno, Carné No. 22193
José Antonio Mérida Castejón, Carné No. 201105

Instrucciones:

Esta actividad se realizará individualmente. Al finalizar los períodos de laboratorio o clase, deberá dejar constancia de sus avances en Canvas, según indique su catedrático. Al finalizar la actividad, adjuntar los archivos .pdf y .cpp para solucionar los ejercicios:

- Desarrolle el programa solución en C++, empleando Pthreads, semáforos y/o barreras.
- Incluir video corto con explicación de funcionamiento del programa.

Ejercicio 01

24 puntos: 4 pts cada inciso. Crea un cuadro comparativo para ayudar a entender las diferencias y similitudes entre varios mecanismos de sincronización y control de hilos en la programación concurrente. Los conceptos a comparar son:

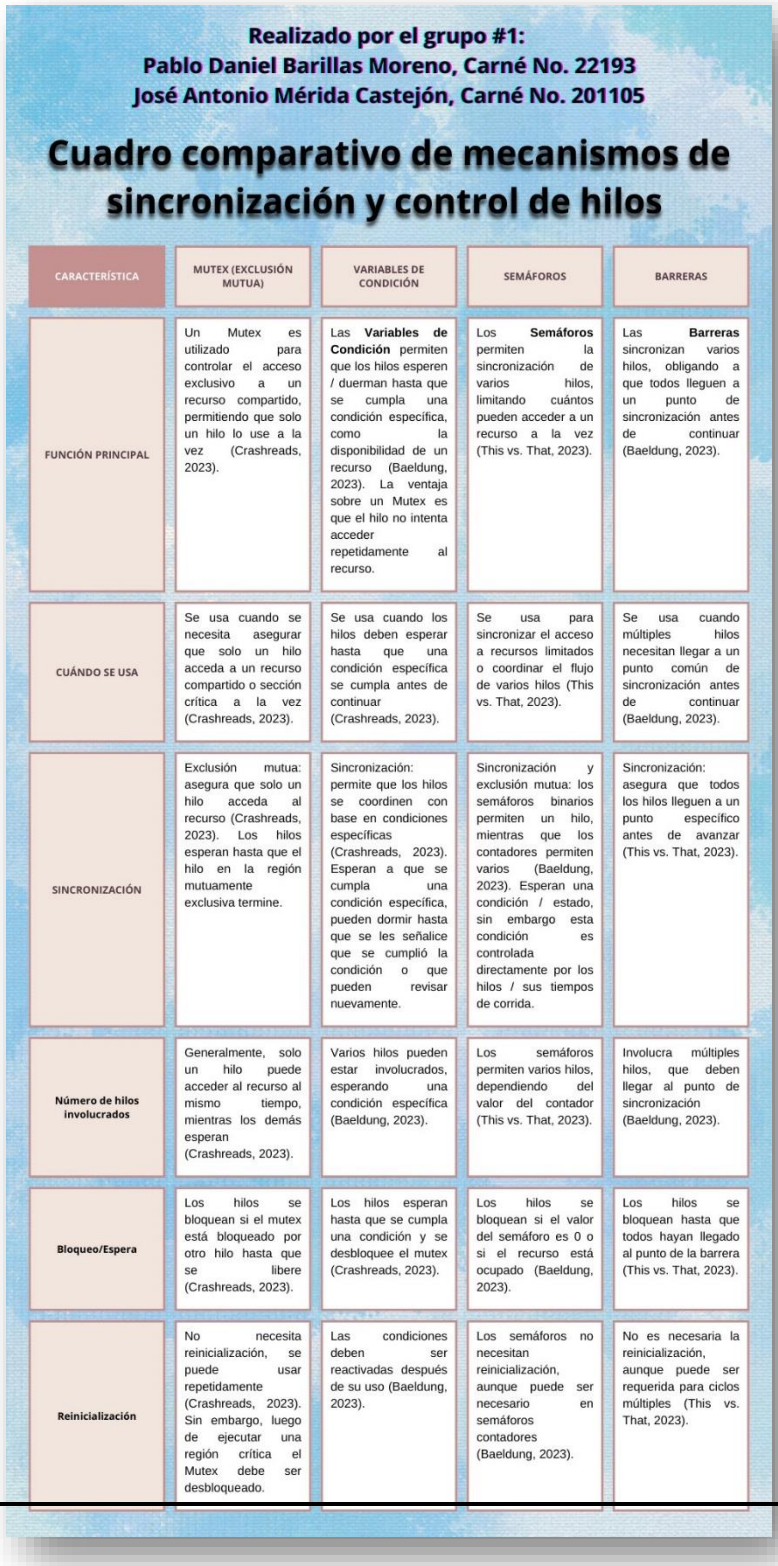
- Mutex (Exclusión Mutua)
- Variables de Condición
- Semáforos
- Barreras
-
-

Toma en cuenta los siguientes aspectos antes de generar el cuadro comparativo:
Identificar las características clave a comparar:

1. **Función principal:** cuál es el propósito o la función principal de cada mecanismo.
2. **Cuándo se usa:** explica en qué situaciones es más adecuado utilizar cada mecanismo.
3. **Sincronización:** ¿se utiliza para sincronización de hilos o para exclusión mutua?
4. **Número de hilos involucrados:** ¿cuántos hilos pueden participar en la operación?
5. **Bloqueo/Espera:** ¿el mecanismo implica que uno o más hilos esperen/bloqueen la ejecución hasta que se cumpla una condición o hasta que otros hilos terminen?
6. **Reinicialización:** ¿Es posible o necesario reinicializar el mecanismo después de su uso?

Enlace a cuadro comparativo para ayudar a entender las diferencias y similitudes entre varios mecanismos de sincronización y control de hilos en la programación concurrente:

https://www.canva.com/design/DAGTFfDOikQ/kIa9ayLNWIy_LyCHeuk5Q/view?utm_content=DAGTFfDOikQ&utm_campaign=designshare&utm_medium=link&utm_source=editor



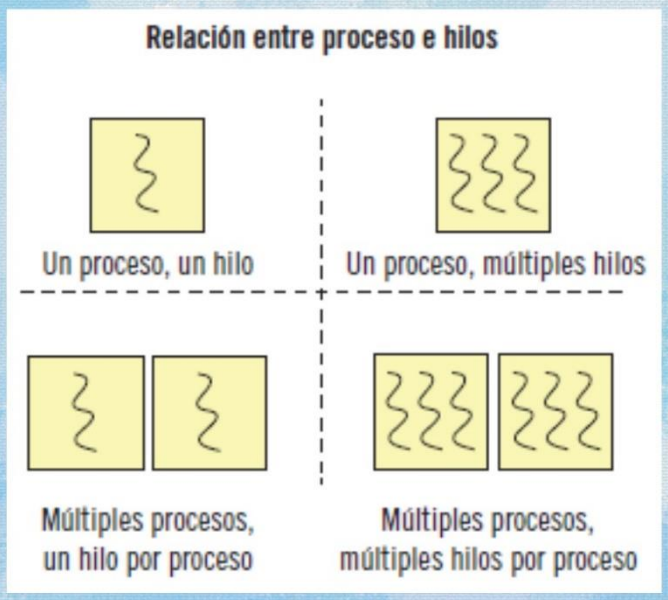
Realizado por el grupo #1:
Pablo Daniel Barillas Moreno, Carné No. 22193
José Antonio Mérida Castejón, Carné No. 201105

Cuadro comparativo de mecanismos de sincronización y control de hilos

CARACTERÍSTICA	MUTEX (EXCLUSIÓN MUTUA)	VARIABLES DE CONDICIÓN	SEMÁFOROS	BARRERAS
FUNCIÓN PRINCIPAL	Un Mutex es utilizado para controlar el acceso exclusivo a un recurso compartido, permitiendo que solo un hilo lo use a la vez (Crashreads, 2023).	Las **Variables de Condición** permiten que los hilos esperen / duerman hasta que se cumpla una condición específica, como la disponibilidad de un recurso (Baeldung, 2023). La ventaja sobre un Mutex es que el hilo no intenta acceder repetidamente al recurso.	Los **Semáforos** permiten la sincronización de varios hilos, limitando cuántos pueden acceder a un recurso a la vez (This vs. That, 2023).	Las **Barreras** sincronizan varios hilos, obligando a que todos lleguen a un punto de sincronización antes de continuar (Baeldung, 2023).
CUÁNDO SE USA	Se usa cuando se necesita asegurar que solo un hilo acceda a un recurso compartido o sección crítica a la vez (Crashreads, 2023).	Se usa cuando los hilos deben esperar hasta que una condición específica se cumpla antes de continuar (Crashreads, 2023).	Se usa para sincronizar el acceso a recursos limitados o coordinar el flujo de varios hilos (This vs. That, 2023).	Se usa cuando múltiples hilos necesitan llegar a un punto común de sincronización antes de continuar (Baeldung, 2023).
SINCRONIZACIÓN	Exclusión mutua: asegura que solo un hilo acceda al recurso (Crashreads, 2023). Los hilos esperan hasta que el hilo en la región mutuamente exclusiva termine.	Sincronización: permite que los hilos se coordinen con base en condiciones específicas (Crashreads, 2023). Esperan a que se cumpla una condición específica, pueden dormir hasta que se les señalice que se cumplió la condición o que pueden revisar nuevamente.	Sincronización y exclusión mutua: los semáforos binarios permiten un hilo, mientras que los contadores permiten varios (Baeldung, 2023). Esperan una condición / estado, sin embargo esta condición es controlada directamente por los hilos / sus tiempos de corrida.	Sincronización: asegura que todos los hilos lleguen a un punto específico antes de avanzar (This vs. That, 2023).
Número de hilos involucrados	Generalmente, solo un hilo puede acceder al recurso al mismo tiempo, mientras los demás esperan (Crashreads, 2023).	Varios hilos pueden estar involucrados, esperando una condición específica (Baeldung, 2023).	Los semáforos permiten varios hilos, dependiendo del valor del contador (This vs. That, 2023).	Involucra múltiples hilos, que deben llegar al punto de sincronización (Baeldung, 2023).
Bloqueo/Espera	Los hilos se bloquean si el mutex está bloqueado por otro hilo hasta que se libere (Crashreads, 2023).	Los hilos esperan hasta que se cumpla una condición y se desbloquee el mutex (Crashreads, 2023).	Los hilos se bloquean si el valor del semáforo es 0 o si el recurso está ocupado (Baeldung, 2023).	Los hilos se bloquean hasta que todos hayan llegado al punto de la barrera (This vs. That, 2023).
Reinicialización	No necesita reinicialización, se puede usar repetidamente (Crashreads, 2023). Sin embargo, luego de ejecutar una región crítica el Mutex debe ser desbloqueado.	Las condiciones deben ser reactivadas después de su uso (Baeldung, 2023).	Los semáforos no necesitan reinicialización, aunque puede ser necesario en semáforos contadores (Baeldung, 2023).	No es necesaria la reinicialización, aunque puede ser requerida para ciclos múltiples (This vs. That, 2023).

Referencias

- **Baeldung. (2023). Semaphores vs. Mutexes: A Deep Dive into Synchronization Primitives. Baeldung on Computer Science. https://www.baeldung.com**
- **Crashreads. (2023). What is a Mutex? Comparing Mutex and Semaphore. Crashreads. https://www.crashreads.com**
- **This vs. That. (2023). Mutex vs. Semaphore - What's the Difference?. This vs. That. https://www.thisvsthat.io**



Ejercicio 02

32 puntos: En un sistema de cajero automático (ATM), varios clientes intentan retirar dinero de una cuenta compartida, utilizando hilos y mecanismos de sincronización, para gestionar la concurrencia y el acceso al recurso compartido. Crearás un programa simulador que ayudará a un banco para determinar el comportamiento de acceso concurrente a cuentas bancarias, por lo que se debe garantizar que solo un cliente acceda al cajero a la vez.

Requisitos: el programa ejecuta los clientes de manera concurrente, asegurando que solo un cliente acceda al cajero en un momento dado.

- **5 pts. Saldo inicial**: el saldo de la cuenta bancaria comienza en Q 100 000.00.
- **5 pts. Cantidad de clientes (hilos)**: el programa solicita al usuario que ingrese la cantidad de clientes que van a utilizar el cajero (con el que generará la simulación). Cada cliente será representado por un hilo.
- **5 pts. Montos de retiro**: para cada cliente, el usuario debe ingresar el monto que el cliente intentará retirar del cajero.
- **5 pts. Semáforo**: se utiliza un semáforo para controlar el acceso al cajero automático, de forma que solo un cliente pueda retirar dinero en un momento dado.

Resultado: cada cliente intentará retirar su monto. Si el saldo es suficiente, el retiro se realiza, se actualiza e indica el saldo restante. Si el saldo es insuficiente, se notifica al cliente que no puede completar la transacción debido a saldo insuficiente.

Responde:

1. **3 pts. ¿Cuál es el recurso compartido y cómo podría afectarse por las condiciones de carrera?**

R// El recurso compartido en este caso es el saldo de la cuenta bancaria. Cuando múltiples clientes (hilos) intentan acceder y modificar el saldo de forma concurrente sin la adecuada sincronización, se produce lo que se conoce como una condición de carrera. Esto significa que el resultado de las operaciones puede depender del orden en que los hilos acceden al recurso compartido. Sin mecanismos de control, es posible que varios hilos lean el mismo saldo, intenten hacer retiros al mismo tiempo, y luego actualicen el saldo basándose en información desactualizada. Como resultado, se podrían realizar múltiples retiros simultáneamente sobre un saldo insuficiente, generando inconsistencias como permitir retiros por encima del saldo disponible real, lo que compromete la integridad del sistema.

Las condiciones de carrera son un problema típico en entornos concurrentes cuando varios procesos o hilos compiten por acceder y modificar un mismo recurso. Esto puede llevar a resultados incorrectos y difícilmente predecibles, especialmente en aplicaciones bancarias, donde la precisión y consistencia de los datos es crítica.

2. **3 pts. ¿Qué pasa si no se utiliza ninguna forma de sincronización al acceder al recurso compartido?**

R// Si no se utiliza ninguna forma de sincronización al acceder al recurso compartido (en este caso, el saldo de la cuenta bancaria), ocurren las condiciones de carrera, como mencioné anteriormente. Esto significa que varios hilos podrían estar accediendo al saldo de manera simultánea, y si no hay mecanismos que controlen el acceso exclusivo, podrían ocurrir situaciones donde:

- Varios hilos leen el saldo al mismo tiempo y calculan los retiros basados en el mismo saldo sin tener en cuenta los retiros de los demás.
- El saldo podría volverse inconsistente o negativo si varios hilos realizan transacciones al mismo tiempo sin actualizar el saldo correctamente.

Esto afectaría la integridad de los datos, ya que el saldo que queda después de una serie de operaciones concurrentes podría no ser el correcto. Un ejemplo sería cuando dos clientes intentan retirar una cantidad de dinero al mismo tiempo, y ambos ven un saldo suficiente. Sin embargo, cuando ambos retiran su monto, el saldo real puede haber sido afectado de

manera errónea, permitiendo que se retire más dinero del que realmente había disponible. Esto podría generar pérdidas financieras y conflictos en el sistema.

3. 3 pts. ¿Cómo sabrás si un cliente tiene suficiente saldo antes de hacer un retiro? ¿Cuál mecanismo de control implementarías?

R// Para determinar si un cliente tiene suficiente saldo antes de hacer un retiro, el sistema debe realizar una verificación previa del saldo. Es decir, antes de proceder con el retiro, el programa debe consultar el saldo actual de la cuenta bancaria y comparar ese saldo con el monto que el cliente desea retirar. Si el saldo es mayor o igual al monto solicitado, se puede proceder con el retiro; de lo contrario, se debe informar al cliente que no hay suficiente dinero en la cuenta para completar la transacción.

El mecanismo de control que implementaría es un **semáforo** o un **mutex** para garantizar que solo un cliente (hilo) pueda acceder y modificar el saldo a la vez. Esto asegura que, durante el proceso de verificación y retiro, ningún otro cliente acceda simultáneamente al saldo, evitando que múltiples hilos lean y modifiquen el saldo de manera incorrecta. La exclusión mutua es esencial para prevenir condiciones de carrera y asegurar que los datos financieros sean coherentes.

En este caso, el flujo sería:

1. Un cliente adquiere el semáforo o mutex, bloqueando a los demás clientes.
2. Se verifica si el saldo es suficiente.
3. Si el saldo es suficiente, se actualiza el saldo y el cliente realiza el retiro.
4. El semáforo o mutex se libera para que otros clientes puedan acceder.

4. 3 pts. Si utilizas semáforo, ¿cuál es el valor inicial del semáforo y por qué?

R// El valor inicial del semáforo debería ser 1, y la razón es que el semáforo está diseñado para controlar el acceso a un recurso compartido por varios hilos. Al inicializarlo con el valor 1, se permite que solo un hilo pueda adquirir el semáforo a la vez. Cuando un cliente (hilo) adquiere el semáforo, el valor del semáforo se reduce a 0, lo que impide que otros hilos accedan al cajero hasta que el semáforo sea liberado (cuando el cliente actual termine su transacción).

Una vez que el cliente termina, el semáforo se libera, incrementando su valor nuevamente a 1, permitiendo que el siguiente cliente acceda al cajero. Este comportamiento asegura que solo un cliente a la vez pueda realizar operaciones con el saldo de la cuenta bancaria, evitando así condiciones de carrera y garantizando la **exclusión mutua**. El semáforo actúa como un

"guardia" que regula el acceso al recurso compartido, en este caso, el saldo de la cuenta bancaria, y previene accesos simultáneos que podrían llevar a inconsistencias en los datos.

Ejercicio 03

34 puntos. Una fábrica está compuesta por dos tipos de empleados, donde **productores** fabrican piezas de **sillas** y las colocan en un almacén de tamaño limitado. A su vez, los **consumidores** son ensambladores que retiran las piezas del almacén para ensamblar las sillas. Para fabricar una silla completa, se deben utilizar 1 respaldo, 1 asiento y 4 patas. Los hilos representan tanto a los productores como a los consumidores, y usamos semáforos para controlar el acceso concurrente al almacén, asegurándonos de que no haya sobreproducción cuando el almacén está lleno, ni intentos de retirar piezas cuando no hay productos disponibles.

Analiza el programa Ejercicio03A.cpp para comprender la simulación generada a través de código.

Indica y resuelve. Responde (deja evidencia de código):

1. **10 pts.** Explica el funcionamiento del sistema ¿Se inician y finalizan correctamente los procesos? ¿Por qué?

R// El programa utiliza el patrón productor-consumidor para simular la producción y ensamblaje de sillas, empleando hilos para representar a los productores y consumidores.

Inicio de los procesos:

- Los **productores** comienzan fabricando piezas de sillas y colocándolas en un buffer compartido, mientras que los **consumidores** retiran piezas de ese buffer y ensamblan las sillas.
- Cada productor y consumidor es un hilo que se crea correctamente utilizando pthread_create(). Los semáforos (vacios y llenos) gestionan la cantidad de piezas disponibles y los espacios en el buffer, mientras que un mutex (pthread_mutex_t mutex) garantiza que solo un hilo acceda al buffer a la vez, evitando condiciones de carrera.

Ejecución:

- Los **consumidores** no consumen las piezas indicadas, no hay alguna verificación sobre las piezas que consume o la cantidad que tienen.

```
Productor 1 ha fabricado la pieza Respaldo y la coloco en la posicion 0
Productor 2 ha fabricado la pieza Respaldo y la coloco en la posicion 1
Consumidor 1 ha retirado la pieza Respaldo de la posicion 0
Consumidor 2 ha retirado la pieza Respaldo de la posicion 1
Productor 1 ha fabricado la pieza Asiento y la coloco en la posicion 2
Productor 2 ha fabricado la pieza Asiento y la coloco en la posicion 3
Consumidor 3 ha retirado la pieza Asiento de la posicion 2
Consumidor 4 ha retirado la pieza Asiento de la posicion 3
Productor 2 ha fabricado la pieza Pata y la coloco en la posicion 4
Productor 1 ha fabricado la pieza Pata y la coloco en la posicion 0
Consumidor 2 ha retirado la pieza Pata de la posicion 4
Consumidor 2 ha ensamblado una silla completa. Sillas ensambladas: 1/3
```

En este caso el consumidor 2 únicamente retira un respaldo, un asiento y una pata; ensamblando una silla "incompleta".

Finalización de los procesos:

- Los **consumidores** finalizan cuando se alcanzan las sillas necesarias (MAX_SILLAS), ya que se revisa esta condición en el bucle del consumidor. Una vez ensambladas todas las sillas, los consumidores finalizan su ejecución.
- Sin embargo, los **productores** no tienen una condición explícita para detenerse. Ellos siguen ejecutándose indefinidamente porque su bucle es infinito. Aunque los consumidores terminan correctamente, los productores continúan produciendo, lo que significa que **no finalizan correctamente**.

```
Consumidor 2 ha retirado la pieza Pata de la posicion 3
Consumidor 2 ha ensamblado una silla completa. Sillas ensambladas: 3/3
Productor 1 ha fabricado la pieza Pata y la coloco en la posicion 0
Productor 2 ha fabricado la pieza Pata y la coloco en la posicion 1
Productor 1 ha fabricado la pieza Pata y la coloco en la posicion 2
Productor 2 ha fabricado la pieza Pata y la coloco en la posicion 3
```

Los hilos productores necesitan una condición de parada para detenerse una vez alcanzado el límite de sillas, ya que, tal como está el programa, no finalizan correctamente. Los productores siguen produciendo piezas y se quedan esperando a que el buffer no se encuentre lleno nuevamente.

2. **10 pts.** ¿Qué modificaciones deben realizarse para que los productores dejen de producir cuando se alcance el límite de sillas, y los consumidores también terminen?

R// ¿Qué modificaciones deben realizarse para que los productores dejen de producir cuando se alcance el límite de sillas, y los consumidores también terminen?

R// Para detener los **productores** cuando se alcance el número máximo de sillas y finalizar correctamente todo el sistema, se deben hacer las siguientes modificaciones:

1. **Bandera global:**

   Se agrega una bandera global para parar la producción cuando ya se haya llegado a la producción máxima

2. **Modificación en los consumidores**:

   Los consumidores setean la bandera global al identificar si ya se alcanzó la cantidad máxima de sillas, también revisan si ya se alcanzó antes de seguir tomando mas piezas del buffer.

3. **Modificación en los productores**:

   Los productores verifican la bandera global para verificar si ya se alcanzó la cantidad máxima de sillas al producir una silla.

3. **14 pts.** Agrega el código necesario para que se genere un reporte antes de finalizar la ejecución. El reporte debe indicar cuántas sillas se fabricaron en totalidad, y cuántas piezas de cada tipo sobraron (quedarán en almacén).

   Agreagado en Ejercicio3BModificado.cpp en el repositorio

Referencias

- Baeldung. (2023). *Semaphores vs. Mutexes: A Deep Dive into Synchronization Primitives*. Baeldung on Computer Science. https://www.baeldung.com

- Crashreads. (2023). *What is a Mutex? Comparing Mutex and Semaphore*. Crashreads. https://www.crashreads.com

- This vs. That. (2023). *Mutex vs. Semaphore – What's the Difference?*. This vs. That. https://www.thisvsthat.io