Laboratorio 08: sincronización con semáforos

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?

Solución realizada en pdf llamado Cuadro Comparativo Ejercicio1.

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

Solución en archivo llamado lab8_ej2.

Responde:

1. **3 pts.** ¿Cuál es el recurso compartido y cómo podría afectarse por las condiciones de carrera?
   El recurso compartido es la variable saldo, pues es la que almacena la cantidad de dinero disponible en cada cuenta. Esta se ve afectada por las condiciones condiciones de carrera puesto que si en caso no hubiera control de acceso puede que dos clientes intentan ingresar a la variable al mismo.
2. **3 pts.** ¿Qué pasa si no se utiliza ninguna forma de sincronización al acceder al recurso compartido?
   Si no se utiliza ninguna forma de sincronización muchos clientes podrían ingresar a las variables al mismo tiempo causando inconsistencias en las operaciones e incluso que el saldo llegue a estar negativo o que no permita sacar dinero a pesar de tener suficiente saldo.
3. **3 pts.** ¿Cómo sabrás si un cliente tiene suficiente saldo antes de hacer un retiro? ¿Cuál mecanismo de control implementarías?
   Para saber si un cliente tiene suficiente saldo se puede realizar la comparación del monto que se desea retirar con el saldo disponible y si en caso no hay suficiente saldo, se le indica al cliente que el saldo es insuficiente y se muestra el saldo que si tiene disponible para retirar. De igual forma, ningún otro hilo puede retirar dinero hasta que el cliente termine su operación.
4. **3 pts.** Si utilizas semáforo, ¿cuál es el valor inicial del semáforo y por qué?

El valor inicial del semáforo sería 1, puesto que el semáforo es el que limita el acceso del hilo a la variable compartida y su valor inicial, al ser 1, indica que solamente un hilo (cliente) podra ingresar al cajero a la vez.

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 proceso? ¿Por qué?

El programa funciona de manera que busca simular el problema clásico del productor-consumidor usando hilos, semáforos y mutexes para gestionar la producción y el consumo de piezas de sillas. Cada productor crea una pieza de forma aleatoria y la coloca en un buffer compartido, mientras que los consumidores retiran las piezas del buffer para ensamblar sillas completas. La sincronización entre los hilos se logra a través de semáforos, que controlan el número de espacios vacíos y llenos en el buffer, y un mutex que protege el acceso a la memoria compartida, evitando condiciones de carrera. Los productores continúan generando piezas indefinidamente, ya que no tienen una condición de finalización, mientras que los consumidores ensamblan un número limitado de sillas y luego terminan su ejecución. Lo que provoca que el sistema funcione correctamente en cuanto a la sincronización y ensamblaje de sillas, pero los hilos productores no terminan correctamente, ya que no se les da una señal para detenerse una vez que los consumidores han completado su tarea.

Los hilos de productores y consumidores se inician correctamente, ejecutando su lógica en paralelo y sincronizando el acceso al buffer.

Ahora, no finalizan completamente bien. Los hilos consumidores finalizan correctamente una vez que han ensamblado el número máximo de sillas, pero los hilos productores continúan ejecutándose indefinidamente porque no tienen una condición de parada.

Para corregir eso, se podría implementar una condición de terminación basada en el número de sillas ensambladas o una señalización explícita desde el hilo principal cuando ya no sea necesario seguir produciendo piezas.

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?

Tal como he mencionado en el inciso anterior, se puede implementar una condición de terminación basada en el número de sillas ensambladas o una señalización explícita desde el hilo principal cuando ya no sea necesario seguir produciendo piezas. Esto consistiría en agregar una condición de terminación para los productores, asegurar que los consumidores también terminen y notificar a los productores de que se ha alcanzado el límite. Agregaré estas modificaciones a la versión final del programa junto con la implementación del reporte.

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).

Solución en código llamado ej3B. Incluye una variable global llamada produccionTerminada para indicar cuando los consumidores han ensamblado el número máx de sillas, una condición para notificar a los productores de que ya terminen, y que los productores chequeen esta condición porque si es true termina el ciclo y finaliza la ejecución. Luego para el reporte, un arreglo llamado piezasSobrantes que almacena la cantidad de piezas de cada tipo que quedan en el buffer cuando los consumidores terminan, la función generarReporte para recorrer el buffer e imprimir el reporte con sillasProducidas y piezasSobrantes, y justo antes de destruir los semáforos y mutex, la llamada a generarReportes.