Sistemas Operativos
Alvarez Zunino Andrea y Pan Jose Ignacio

Trabajo Practico N°2 : "Concurrencia"

1) ¿Que es el problema de la sección critica en concurrencia entre procesos?. Da un ejemplo de código o seudocódigo explicando los fragmentos relevantes

El método más sencillo de comunicación entre los procesos de un programa concurrente es el uso común de unas variables de datos. Esta forma tan sencilla de comunicación puede llevar, no obstante, a errores en el programa ya que el acceso concurrente puede hacer que la acción de un proceso interfiera en las acciones de otro de una forma no adecuada. Para evitar este tipo de errores se pueden identificar aquellas regiones de los procesos que acceden a variables compartidas y dotarlas de la posibilidad de ejecución como si fueran una única instrucción. Se denomina Sección Crítica a aquellas partes de los procesos concurrentes que no pueden ejecutarse de forma concurrente o, también, que desde otro proceso se ven como si fueran una única instrucción. Esto quiere decir que, si un proceso entra a ejecutar una sección crítica en la que se accede a unas variables compartidas, entonces otro proceso no puede entrar a ejecutar una región crítica en la que acceda a variables compartidas con el anterior. Las secciones críticas se pueden mutuo excluir. Para conseguir dicha exclusión se deben implementar protocolos software que impidan o el acceso a una sección crítica mientras está siendo utilizada por un proceso.
En el ej1.py vemos creamos tres hilos que intentan agregar elementos a una lista compartida shared_list. Utilizamos un semáforo (semaphore) para garantizar que solo un hilo pueda acceder a la lista a la vez. Cuando un hilo intenta agregar un elemento, adquiere el semáforo, realiza su operación y luego lo libera para que otros hilos puedan acceder.
Gracias al semáforo, aseguramos que no haya condiciones de carrera y que los elementos se agreguen de manera segura en la lista compartida, evitando problemas en la sección crítica.

2) ¿Que es un semáforo y para que se utilizan?. ¿Que tipos de semáforos existen y cuáles son sus diferencias?.

Un semáforo es una herramienta de sincronización utilizada en programación concurrente y sistemas operativos para controlar el acceso a recursos compartidos por múltiples procesos o hilos. Su función principal es evitar condiciones de carrera y garantizar la exclusión mutua, lo que significa que solo un proceso o hilo puede acceder a ciertas secciones críticas de código a la vez. Los semáforos también se utilizan para coordinar la ejecución de procesos o hilos y para evitar el bloqueo mutuo.
Existen dos tipos principales de semáforos: semáforos binarios y semáforos contadores.

- Semáforo Binario:
  - Un semáforo binario tiene dos estados posibles: 0 y 1.
  - Se utiliza principalmente para implementar la exclusión mutua, es decir, para garantizar que solo un proceso a la vez pueda acceder a una sección crítica.
  - Los semáforos binarios se utilizan comúnmente para resolver el problema de la sección crítica.

- Semáforo Contador:
  - Un semáforo contador tiene un rango de valores no negativos, generalmente más grande que 1.
  - Se utiliza para controlar el acceso a múltiples instancias de un recurso compartido. El valor del semáforo indica cuántas instancias del recurso están disponibles.

- Los semáforos contadores se utilizan para coordinar la ejecución de un número limitado de procesos que pueden acceder al recurso simultáneamente.

Diferencias clave entre los dos tipos de semáforos:

- ✓ Número de Estados: Los semáforos binarios tienen dos estados (0 o 1), mientras que los semáforos contadores pueden tener un rango de valores no negativos.
- ✓ Uso Principal: Los semáforos binarios se utilizan principalmente para la exclusión mutua, mientras que los semáforos contadores se utilizan para controlar el acceso a múltiples instancias de un recurso compartido.

Ejemplos de Uso

- Un ejemplo de uso de semáforos binarios es garantizar que solo un proceso escriba en un archivo a la vez.
- Un ejemplo de uso de semáforos contadores es controlar el acceso a un grupo de impresoras en una red, donde varios procesos pueden imprimir al mismo tiempo, pero se limita el número total de procesos que pueden imprimir simultáneamente.

En resumen, los semáforos son una herramienta poderosa para coordinar la concurrencia y garantizar el acceso seguro a recursos compartidos en sistemas concurrentes. Los semáforos binarios y los semáforos contadores se utilizan en diferentes situaciones según los requisitos específicos del problema.

3) Da un ejemplo con código de una posible utilización de semáforos (súbelo a GitHub y adjunta el enlace).

El ejercicio ej3.py pertenece a esta consigna.

4) Describe brevemente que son los monitores y como se utilizan. ¿Que diferencia tiene con los semáforos?.

Un monitor es un conjunto de múltiples rutinas que están protegidas por un bloqueo de exclusión mutua mientras que , Un semáforo es una construcción más simple que un monitor porque es sólo un candado que protege un recurso compartido - y no un conjunto de rutinas como un monitor. La aplicación debe adquirir el bloqueo antes de usar ese recurso compartido protegido por un semáforo.
Tanto los monitores como los semáforos se utilizan para el mismo propósito: la sincronización de los hilos. Pero los monitores son más sencillos de usar que los semáforos porque manejan todos los detalles de la adquisición y liberación de la cerradura.
Otra diferencia al utilizar semáforos es que toda rutina que acceda a un recurso compartido tiene que adquirir explícitamente un candado antes de utilizar el recurso. Esto puede olvidarse fácilmente cuando se codifican las rutinas que tratan con multihilo. Los monitores, a diferencia de los semáforos, adquieren automáticamente los bloqueos necesarios.

5) Explica con tus palabras:
a. Problema del búfer limitado
b. El problema de los lectores-escritores
c. El problema de los filósofos comensales

Realiza un ejemplo de código de cada uno de ellos utilizando monitores y semáforos (súbelo a GitHub y adjunta el enlace).

a. Problema del búfer limitado:
Este problema se refiere a la gestión de un búfer (una zona de almacenamiento temporal) compartido entre productores y consumidores. Los productores colocan elementos en el búfer, y los consumidores los retiran. El problema radica en asegurarse de que los productores no coloquen elementos en el búfer cuando esté lleno y de que los consumidores no intenten retirar elementos cuando esté vacío.

b. El problema de los lectores-escritores:
Este problema involucra múltiples procesos (lectores y escritores) que acceden a un recurso compartido (por ejemplo, una base de datos). Los lectores pueden leer el recurso simultáneamente, pero los escritores deben tener acceso exclusivo. El problema radica en permitir un acceso seguro y eficiente a los recursos compartidos.

c. El problema de los filósofos comensales:
Este problema modela la cena de cinco filósofos que alternan entre pensar y comer. Los filósofos deben tomar dos tenedores para comer y liberarlos cuando terminen. El problema radica en evitar el bloqueo mutuo y el hambre eterna de los filósofos.

Utilizamos ejemplos en Python que utiliza semáforos para resolverlos, ya que los monitores no son una característica nativa en Python, pero los semáforos pueden utilizarse para lograr la sincronización necesaria. Se pueden ver en ej5a.py , ej5b.py y ej5c.py.

6) ¿Pueden utilizarse los monitores junto a los semáforos?. Es caso a formativo da un ejemplo de cómo se haría.

Sí, es posible utilizar monitores junto con semáforos para lograr una sincronización más avanzada y segura en programas concurrentes. Un monitor es una estructura de datos que incluye tanto datos compartidos como procedimientos (o métodos) que operan en esos datos compartidos. Los monitores proporcionan un mecanismo de sincronización más alto nivel que los semáforos y se encargan de manera más eficiente de la administración de bloqueos y desbloqueos.
En el ej6.py, creamos un monitor llamado `Monitor` que tiene un semáforo y una variable `data` como datos compartidos. Los métodos `modificar_data` y `obtener_data` dentro del monitor se encargan de modificar y acceder a la variable `data`. Los hilos `hilo_modificar` modifican la `data`, y los hilos `hilo_leer` la leen. El semáforo dentro del monitor garantiza que solo un hilo pueda acceder a la `data` a la vez, lo que asegura la consistencia de los datos.

7) Investiga y analiza que sistema de solución de problemas de concurrencia proporciona el lenguaje Java, Python y PHP

Los lenguajes de programación Java, Python y PHP proporcionan diferentes mecanismos para abordar problemas de concurrencia y sincronización. A continuación, se detalla cómo cada uno de estos lenguajes aborda estos problemas:

- Java:

Java es conocido por proporcionar un sólido conjunto de herramientas para abordar problemas de concurrencia. Algunos de los mecanismos y clases clave en Java para la sincronización y manejo de concurrencia son:

  - Synchronized Blocks/Methods: Java utiliza la palabra clave `synchronized` para crear secciones críticas que garantizan la exclusión mutua. Puedes sincronizar bloques de código o métodos para asegurarte de que solo un hilo pueda acceder a ellos a la vez.
  - Clase `java.util.concurrent`: Java ofrece una amplia gama de clases en el paquete `java.util.concurrent` para gestionar hilos y recursos compartidos de manera segura. Ejemplos incluyen semáforos, cerrojos, barreras, colas concurrentes, y más.
  - Clase `java.util.concurrent.locks.ReentrantLock`: Proporciona una alternativa más flexible a `synchronized`. Puedes crear instancias de `ReentrantLock` para administrar bloqueos explícitamente.
  - Executor Framework: Java ofrece el framework de ejecución (`java.util.concurrent.Executor`) para gestionar la ejecución de hilos de manera más eficiente y escalable.
  - Clase `java.util.concurrent.Atomic`: Proporciona operaciones atómicas, como `AtomicInteger`, `AtomicLong`, que permiten la actualización segura de variables compartidas.

- Python:

Python ofrece varios mecanismos para abordar problemas de concurrencia, aunque no es tan conocido por ser tan robusto como Java en este aspecto. Algunos mecanismos y módulos útiles en Python son:

  - Módulo `threading`: Este módulo proporciona una forma de trabajar con hilos en Python. Puedes usarlo para crear y gestionar hilos, pero debido al GIL (Global Interpreter Lock), no es tan eficiente para la concurrencia en sistemas multi-núcleo.
  - Módulo `multiprocessing`**: Para abordar problemas de concurrencia en sistemas multi-núcleo, Python ofrece el módulo `multiprocessing`, que permite la creación de procesos en lugar de hilos. Cada proceso tiene su propio espacio de memoria y no se ve afectado por el GIL.
  - Módulo `asyncio`**: Este módulo proporciona soporte para programación asincrónica, lo que permite la ejecución concurrente de tareas sin crear hilos o procesos. Es útil para aplicaciones de E/S intensivas como servidores web.

- PHP:

PHP no es conocido por ser un lenguaje ideal para la programación concurrente, ya que la mayoría de las implementaciones de PHP utilizan un modelo de subprocesos (threads) limitado y, por lo tanto, pueden no ser adecuadas para aplicaciones de alta concurrencia.

  - Extensión pthreads (no estándar): Existe una extensión llamada pthreads que permite la programación multihilo en PHP. Sin embargo, no es parte de la distribución estándar de PHP y no está ampliamente adoptada.
  - Sesiones y recursos compartidos*: PHP puede utilizar sesiones y archivos compartidos para almacenar datos compartidos entre solicitudes, pero esto no es una solución completa para problemas de concurrencia en tiempo real.

En resumen, Java es conocido por proporcionar un conjunto sólido de herramientas para la sincronización y manejo de concurrencia, mientras que Python ofrece diferentes enfoques dependiendo de las necesidades específicas, y PHP es menos adecuado para aplicaciones de alta concurrencia debido a su

arquitectura típica de subprocesos. Sin embargo, en PHP, puedes utilizar técnicas como almacenamiento en caché o bases de datos para mitigar problemas de concurrencia.

8) ¿A que hace referencia el pasaje de mensajes entre procesos?. Explícalo brevemente y da un ejemplo

El "pasaje de mensajes entre procesos" se refiere a un mecanismo de comunicación utilizado en sistemas operativos y programación concurrente para permitir que procesos o hilos se comuniquen y compartan información de manera segura. En lugar de compartir memoria directamente, los procesos se comunican enviando y recibiendo mensajes a través de canales o colas de mensajes gestionadas por el sistema operativo o una biblioteca de programación.

Brevemente, en el pasaje de mensajes entre procesos:

- Los procesos o hilos envían mensajes a un canal o cola de mensajes.
- Otros procesos o hilos pueden leer esos mensajes de la cola y responder en consecuencia.
- El sistema operativo o la biblioteca de programación maneja la sincronización y la comunicación, lo que evita condiciones de carrera y problemas de concurrencia.

En el ej8.py vemos un ejemplo simple en Python utilizando el módulo `multiprocessing` para ilustrar el pasaje de mensajes entre procesos.
En este ejemplo, creamos un proceso hijo que recibe un mensaje como argumento. El proceso padre y el proceso hijo pueden comunicarse enviando mensajes a través de los argumentos o utilizando mecanismos más avanzados como tuberías o colas de mensajes. El pasaje de mensajes entre procesos es esencial en la programación concurrente para lograr la comunicación y sincronización entre procesos de manera segura y eficiente.

9) ¿Que es interbloqueo o bloqueo mutuo (DeadLock)?. Explícalo con un ejemplo.

El interbloqueo, también conocido como bloqueo mutuo o "deadlock" en inglés, es una situación en la que dos o más procesos (o hilos) quedan atrapados en un estado en el que ninguno de ellos puede continuar porque cada uno está esperando que el otro libere un recurso que necesita. En otras palabras, los procesos quedan atrapados en un bucle de espera indefinida, lo que impide que el sistema progrese.
Para comprender mejor el concepto de interbloqueo, aquí tienes un ejemplo simplificado:

Supongamos que tenemos dos recursos, A y B, y dos procesos, P1 y P2.
1. P1 adquiere el recurso A.
2. P2 adquiere el recurso B.
3. P1 intenta adquirir el recurso B, pero está ocupado por P2, por lo que P1 se bloquea y espera a que B esté disponible.
4. P2 intenta adquirir el recurso A, pero está ocupado por P1, por lo que P2 se bloquea y espera a que A esté disponible.

Ahora, ambos procesos están bloqueados, y ninguno puede continuar porque están esperando que el otro libere un recurso. Esto es un estado de interbloqueo, y el sistema no puede avanzar a menos que se resuelva de alguna manera, ya sea liberando uno de los recursos o utilizando algoritmos de detección y recuperación de interbloqueos.

Para evitar el interbloqueo, se pueden tomar medidas como:
1. Asignar recursos de manera exclusiva y ordenada (por ejemplo, siempre adquirir A antes que B) para evitar la posibilidad de interbloqueo.
2. Utilizar temporizadores para liberar recursos después de un período de tiempo si no se pueden adquirir todos los recursos necesarios.
3. Implementar algoritmos de detección de interbloqueos que identifiquen y resuelvan situaciones de interbloqueo automáticamente.

El interbloqueo es un problema importante en sistemas concurrentes y puede causar bloqueos completos del sistema si no se maneja adecuadamente. Por lo tanto, es esencial tomar medidas para prevenirlo o detectarlo y resolverlo de manera eficaz.

10) ¿Que diferencia existe entre LiveLock y DeadLock?. Da ejemplos

Deadlock y LiveLock son dos problemas relacionados con la concurrencia en sistemas informáticos, pero tienen diferencias clave en cómo se manifiestan y se solucionan. Aquí hay una explicación de cada uno junto con ejemplos:

1. Deadlock:
   - Definición: El deadlock, también conocido como interbloqueo, es una situación en la que dos o más procesos (o hilos) quedan atrapados en un estado de espera mutua indefinida, donde cada uno está esperando un recurso que el otro tiene y no puede liberar.
   - Ejemplo:

Supongamos que tenemos dos procesos, P1 y P2, y dos recursos, A y B.
- P1 adquiere A.
- P2 adquiere B.
- P1 intenta adquirir B, pero está ocupado por P2.
- P2 intenta adquirir A, pero está ocupado por P1.

Ambos procesos quedan bloqueados indefinidamente, lo que resulta en un deadlock.
Solución: Los deadlocks se pueden evitar o resolver utilizando técnicas como asignación ordenada de recursos, liberación automática de recursos en caso de espera prolongada y detección de deadlock.

2. LiveLock:
   - Definición: El livelock es una situación en la que dos o más procesos o hilos están activos y responden continuamente a las acciones del otro, pero ninguno puede avanzar porque están atrapados en un patrón de comportamiento circular.
   - Ejemplo: Imagina dos personas que se cruzan en un pasillo estrecho y se mueven repetidamente para dejar pasar al otro. Ambos siguen moviéndose, pero ninguno puede avanzar porque siempre están cediendo el paso al otro, creando un patrón de comportamiento circular.

Solución: Los livelocks son más difíciles de detectar y solucionar que los deadlocks. A menudo, se requiere una estrategia de retroceso aleatorio para romper el patrón de comportamiento circular y permitir que uno de los procesos avance.

En resumen, la diferencia principal entre deadlock y livelock es que en un deadlock los procesos están bloqueados en espera mutua, mientras que en un livelock los procesos están activos, pero no pueden avanzar debido a su patrón de comportamiento repetitivo. Ambos son problemas no deseados en sistemas concurrentes y requieren estrategias específicas para su prevención o resolución.