Clase 4

Defectos de la sincronización por busy waiting

- `Busy waiting` técnica en la que un proceso entra en un bucle continuo y verifica repetidamente una condición hasta que se cumpla.

Una de las consecuencias del busy waiting es que no hay una separación clara entre las variables usadas para la sincronización o para cálculos.

Por lo tanto, diseñar algoritmos busy waiting puede ser complejo porque

1. Hay que asegurar la sincronización.
2. Seguir el código.
3. Aumento de posibilidad de errores.

Semáforos

Mecanismo de sincronización usado para controlar el acceso a recursos compartidos y coordinar la ejecución de múltiples procesos.

Los semáforos son variables enteras que tienen 2 operaciones atómicas fundamentales `P` para decrementar y `V` para incrementar.

- `Operación P:` se usa para solicitar el acceso a un recurso compartido o para entrar a la SC. Si el valor actual del semáforo es mayor que cero, el proceso que ejecuta la operación P disminuye el valor del semáforo en uno y continúa su ejecución. Si el semáforo está en 0, significa que todos los recursos están siendo utilizados y el proceso que intenta adquirirlo se bloquea hasta que alguien libere el recurso.
- `Operación V:` se usa para liberar un recurso o indicar que una tarea se completó. Cuando se ejecuta la operación V, el valor del semáforo se incrementa en uno. Esto permite que otros procesos que estaban esperando continúen su ejecución. En la analogía del tráfico, sería como avanzar y dejar el semáforo en verde para que otros puedan pasar.

Tipos de semáforos

- Semáforo general: variable entera no negativa que se usa para controlar el acceso a un recurso compartido o para gestionar la sincronización entre procesos. Su valor puede ser cualquier número no negativo, lo que significa que puede representar la cantidad de recursos disponibles.
- Semáforo binario: variante del general el cual su único valor puede ser 0 o 1. Se usa principalmente para logar exclusión mutua, o sea para permitir que un solo proceso acceda a un recurso compartido a la vez.

Operaciones básicas

Hay dos tipos comúnes de semáforos: semáforo general y semáforo binario.

- Semáforo general
  - `P(s)`: esta operación decrementa el valor del semáforo `s`. Si `s` es mayor que 0, el proceso que ejecuta esta operación continúa sin bloquearse. Si `s` es igual a 0, el proceso se bloquea hasta que otro hilo ejecute la operación `V(s)` y aumente el valor de `s`.
  - `V(s)`: esta operación incrementa el valor del semáforo `s`. Si había procesos bloqueados debido a la operación `P(s)`, uno de ellos se desbloquea y continúa su ejecución.
- Semáforo binario
  - `P(b)`: esta operación decrementa el valor del semáforo `b`. Si `b` es mayor que 0, el proceso que ejecuta esta operación continúa sin bloquearse. Si `b` es igual que 0, el proceso se bloquea hasta que otro proceso ejecute la operación `V(b)` y aumente el valor de `b` a 1.
  - `V(b)`: esta operación incrementa el valor del semáforo `b` de 0 a 1. Si había procesos bloqueados debido a la operación `P(b)`, uno de ellos se desbloquea y continúa su ejecución.

Barreras: señalización de eventos

Usar una barrera sirve para asegurarse de que ambos procesos alcancen cierto punto en su ejecución antes de continuar.

La idea principal es usar un par de semáforos `llega1` y `llega2` para sincronizar dos procesos. Inicialmente, ambos semáforos se establecen en 0. Cada primero ejecuta

primero `V(llega1)` y `V(llega2)` para indicar que ha llegado a la barrera.

Luego, el proceso espera a que el otro proceso también llegue a la barrera ejecutando `P(llega2)` o `P(llega1)`

Pero habrá varios problemas asociados. Si los procesos primero ejecutan `P` y luego `V` se crearía un problema de bloqueo. Ambos procesos quedarían bloqueados permanentemente esperando que el otro realice la operación `V` primero, lo que se conoce como un bloqueo mutuo.

Productores y consumidores: semáforos binarios divididos

Un conjunto de semáforos binarios, denotados como `b1, b2, ..., bn`, forma un semáforo binario dividido si cumplen con una cierta invariante global, que se expresa como `0 <= b1 + b2 + ... + bn <= 1`.

O sea que el valor total de los semáforos binarios divididos siempre se mantiene en el rango de 0 a 1. Cada `bi` es un semáforo binario individual que puede tener un valor de 0 o 1 y la suma de estos valores no puede superar 1.

Uno de sus usos más comunes es lograr la exclusión mutua entre procesos. Significa que solo un proceso a la vez puede ejecutar una SC.

Los SBS son útiles porque permiten que la ejecución de procesos comience con un `P` en un semáforo y termine con un `V` en otro semáforo del conjunto.

Los SBS también pueden usarse para resolver el problema de productores y consumidores, donde múltiples productores generan datos y múltiples consumidores los consumen. Los productores pueden bloquear su acceso a un buffer compartido utilizando un semáforo binario dividido (`P(b1)`), mientras que los consumidores pueden liberar el acceso (`V(b2)`). Esto asegura que un productor y un consumidor no intenten acceder al buffer al mismo tiempo.

Contadores de recursos

Cada semáforo cuenta el número de unidades libres de un recurso determinado. Esta forma de utilización es adecuada cuando los procesos compiten por recursos de múltiples unidades.

El problema de buffers limitados es un escenario común en la programación concurrente en el que un productor y un consumidor trabajan con un buffer de tamaño

limitado. En este caso, el productor agrega elementos al buffer y el consumidor los retira.

- Buffer limitado: se tiene un buffer con una capacidad máxima limitada. Esto significa que solo se pueden almacenar un número finito de elementos a la vez.
- Productor: hay un proceso llamado productor que genera elementos y los coloca en el buffer. El productor no puede agregar elementos al buffer si está lleno.
- Consumidor: hay procesos llamados consumidor que toma elementos del buffer y los procesa. El consumidor no puede retirar elementos si el buffer está vacío.
- Objetivo: el objetivo es garantizar que el productor y el consumidor funcionen de manera sincronizada.

Varios procesos compitiendo por varios recursos compartidos

A veces la situación en la que varios procesos compitan por varios recursos compartidos puede producir situaciones de deadlock.

Hay varios procesos `P` y varios recursos `R` denotados como `P[1], P[2], ..., P[n]` y os recursos denotados como `R[1], R[2], ..., R[n].`

Cada proceso necesita adquirir el lock de uno o varios recursos para llevar a cabo su trabajo. Un proceso debe adquirir los locks de todos los recursos que necesita antes de poder continuar su ejecución. El problema principal será que varios procesos pueden competir por conjuntos superpuestos de recursos. Por ejemplo, si cada proceso `P[i]` necesita los recursos `R[i] y R[(i+1) mod n]` y varios procesos intentan adquirir estos recursos al mismo tiempo, puede entrar en el estado deadlock.

Problema de los filósofos: exclusión mutua selectiva

En este problema, se representa a los filósofos como procesos que alternan entre dos actividades principales: comer y pensar. Cada filósofo necesita dos tenedores para comer, y la tarea es garantizar que los filósofos puedan hacer estas actividades de manera segura sin entrar en un estado de deadlock.

Problema de lectores y escritores

Problema común que tiene dos tipos de procesos: lectores y escritores que comparten una base de datos. El objetivo es garantizar que el acceso a la BD sea seguro y eficiente, teniendo en cuenta:

- Exclusión mutua para escritores: los escritores deben tener acceso exclusivo a la BD para evitar que dos o más escritores modifiquen simultáneamente los datos, lo que podría resultar en inconsistencias o corrupción de la información.
- Lectores concurrentes: los lectores pueden acceder a la BD concurrentemente entre ellos, siempre y cuando no haya escritores realizando modificaciones en ese momento.
- Prioridades asimétricas: los procesos de lectura y escritura pueden tener prioridades asimétricas según el planificador del S.O. O sea que algunos procesos pueden tener prioridad sobre otros para acceder a la BD.

Habrá varias soluciones:

1. Exclusión mutua: se garantiza que un solo proceso puede acceder a la BD a la vez. Cuando un escritor tiene acceso exclusivo, los lectores deben esperar y viceversa. Esto se logra usando mecanismos de sincronización. La desventaja es que puede llevar a la inanición de los lectores si hay escritores continuamente solicitando acceso.
2. Sincronización por condición: este enfoque permite que múltiples lectores accedan concurrentemente a la BD, siempre y cuando no haya escritores en curso. Si un escritor solicita acceso, los lectores deben esperar a que se complete la escritura antes de continuar.

Técnica passing de Baton

Estrategia usada para implementar sentencias `await` eficientemente y controlar el acceso a SC o recursos compartidos. Esta técnica se basa en el concepto de un token que simboliza el permiso para ejecutar una SC.

- Uno de token: cuando un proceso desea acceder a una SC o recurso compartido, necesita adquirir un toquen que simboliza el permiso para entrar a la SC. El proceso que tiene el token puede ejecutar la SC y otros procesos deben esperar hasta que el token esté disponible.
- Entrando y saliendo de las SC: cuando un proceso está dentro de una SC mantiene el token en su posesión y cuando termina de ejecutar la SC y sale de ella, el token pasa a otro proceso que está esperando para entrar en la misma SC.

- Transferencia del token: la transferencia ocurre en el momento en que un proceso sale de la SC y hace una operación que permite que otro proceso adquiera el token y entre en la SC.
- Liberación del token: si ningún proceso está esperando para adquirir el token, el token se libera y se pone a disposición del próximo proceso que lo solicite.

La técnica de Baton se usa para la sincronización mediante sentencias atómicas y se basa en el uso de semáforos binarios divididos.

Componentes de la técnica

1. Sentencias atómicas: la sincronización se expresa mediante sentencias atómicas que siguen la estructura `F1: ⟪Si⟫ o F2: ⟪await (Bj) Sj⟫` significa que hay 2 tipos de sentencias `F1` que se ejecutan si la condición es `Si` es verdadera y `F2` que espera a que la condición `Bj` sea verdadera para ejecutar `Sj.`
2. Semáforo binario inicial: para implementar esta técnica se usa un semáforo binario inicializado en 1. Este semáforo controla la entrada a las sentencias atómicas, asegurando que a lo sumo un proceso pueda ejecutar una de estas sentencias a la vez.
3. Semáforos `Bj` y contadores `Dj` se usan múltiples semáforos `Bj` cada uno asociado con una condición `Bj`. Todos estos semáforos bj se inicializan en 0.

Funcionamiento de la técnica

La técnica de passing the baton se basa en la idea de que cada camino de ejecución empiece con una operación P (tomar) y termine con una operación V (liberar).

1. Cuando un proceso llega a una sentencia atómica F1 o F2, primero intenta adquirir el semáforo binario inicial mediante una operación P (bloqueante). Esto garantiza que a lo sumo un proceso pueda ejecutar una sentencia atómica en un proceso dado.
2. Una vez que un proceso adquiere el semáforo binario, verifica la condición Si o espera a que la condición Bj sea verdadera.
3. Si la condición es verdadera, el proceso ejecuta la sentencia Sj. Luego, libera el semáforo binario mediante una operación V, lo que permite que otro proceso pueda intentar adquirir el semáforo.

4. Si la condición no es verdadera, el proceso libera el semáforo binario y espera a que la condición cambie (en el caso de F2) o simplemente se bloquea hasta que pueda adquirir el semáforo binario nuevamente (en el caso de F1).
5. Los semáforos bj y los contadores dj se utilizan para implementar la espera y liberación de procesos que esperan que se cumplan las condiciones específicas antes de poder ejecutar la sentencia atómica.

Problema de alocación de recursos y scheduling

A veces varios procesos compiten por el acceso a unidades de un recurso específico y cada uno de estos puede estar libre o en uso.

Habrá solicitudes para acceder a estas unidades

- `await` un proceso solicita acceso a ciertas unidades del recurso. Antes de solicitar, el proceso debe verificar si las condiciones para obtener las unidades están presentes. Si es así, procede a tomar las unidades.
- Un proceso que ha tomado unidades del recurso las devuelve, lo que permite que otras solicitudes puedan ser satisfechas.

La técnica "Passing the Baton" se utiliza para implementar la gestión de recursos y el acceso a ellos de manera sincronizada y controlada. Aquí está cómo funciona en el contexto de las operaciones de solicitud y liberación:

- **Operación de Solicitud (request)**: Un proceso que desea solicitar unidades del recurso primero intenta adquirir un semáforo binario "e". Si no puede adquirirlo (lo cual significa que otro proceso ya está solicitando o utilizando el recurso), se bloquea en una operación de "DELAY" hasta que las condiciones sean adecuadas. Si puede adquirir el semáforo "e", procede a tomar las unidades del recurso y luego realiza una operación de "SIGNAL" para liberar el semáforo "e" y notificar a otros procesos que pueden intentar solicitar el recurso.
- **Operación de Liberación (release)**: Un proceso que ha tomado unidades del recurso primero adquiere el semáforo "e" para evitar que otros procesos accedan al recurso mientras lo está liberando. Luego, devuelve las unidades del recurso y realiza una operación de "SIGNAL" para liberar el semáforo "e" y notificar a otros procesos que pueden intentar solicitar el recurso.

Alocación shortest-job-next

Técnica usada para asignar recursos a varios procesos que compiten por su uso. Varios procesos compiten por el uso de una sola unidad de recurso compartido, cada solicitud de recurso se acompaña con 2 parámetros

- El tiempo estimado de ejecución del proceso.
- Un identificador único del proceso.

El objetivo principal de SJN es minimizar el tiempo promedio de ejecución, por lo tanto priorizará el acceso al recurso al proceso que se espera que termine más rápido.

Operaciones de solicitud

- Cuando un proceso desea solicitar el recurso, usa la operación de solicitud. Si el recurso está libre en ese momento, se le asigna inmediatamente al proceso identificado por `id`.
- Si el recurso no está libre, el proceso identificado por `id` se demora y debe esperar hasta que el recurso esté disponible. El recurso se asignará a este proceso en el futuro cuando esté libre nuevamente.

Operación de liberación

- Cuando un proceso termina de usar el recurso, realiza una operación de liberación. En este punto, el recurso se pone nuevamente en estado libre y está disponible para ser asignado a otro proceso.
- Si hay procesos en espera que desean usar el recurso, se asignará el recurso al proceso con el tiempo estimado de ejecución más corto, es decir, al proceso que se espera que termine más rápido.

Ventaja

SJN minimiza el tiempo promedio de ejecución, lo que generalmente resulta en una alta eficiencia en la ejecución de procesos cortos.

Desventaja

Puede ser injusto para los procesos largos o más complejos. Los procesos largos pueden ser continuamente retrasados por procesos cortos que obtienen acceso al

recurso con más frecuencia.

Mejora con Aging

Para abordar la desventaja de unfairness, SJN puede mejorarse mediante la técnica de aging. Con el aging, se da preferencia gradualmente a los procesos que han esperado más tiempo. O sea que, a medida que un proceso largo espera más, su prioridad para obtener el recurso aumenta, equilibrando mejor el acceso al recurso entre procesos cortos y largos.