Semáforos

Como se decía en la teoría anterior la sincronización por busy waiting tiene varios problemas, por lo que necesitamos herramientas para diseñar protocolos de sincronización.

Los **semáforos** son una instancia de un tipo de datos abstracto con solo 2 operaciones atómicas P y V. El valor de un semáforo internamente es un entero no negativo, donde V señala la ocurrencia de un evento (incrementa) y P se usa para demorar un proceso hasta que ocurra un evento (decrementa). Esto nos permite proteger Secciones críticas y para implementar sincronización por condición.

Texto

Descripción generada automáticamente

En un semáforo binario ambas operaciones son bloqueantes. En la materia se usan los generales.

Problemas básicos y técnicas

**Exclusión mutua**

Texto

Descripción generada automáticamente

Más simple que las soluciones **busy waiting.** Es importante que free sea 1 y no 0, así puede pasar el primer proceso que aparezca, sino todos se quedan esperando en V.

**Barreras**

Se usa un semáforo para cada flag de sincronización. Un proceso setea el flag ejecutando V, y espera a que un flag sea seteado y luego lo limpia ejecutando P.

**Semáforo de señalización:** Un semáforo generalmente inicializado en 0. Un proceso señala el evento con V(s); otros procesos esperan la ocurrencia del evento ejecutando P(s).

Barrera para dos procesos. Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Puede usarse esta barrera para implementar una **butterfly barrier** para n procesos, o sincronización con un coordinador central.

Hay que tener cuidado con el orden de P y V, porque si primero hacen P y luego V ambos quedan bloqueados.

**Productores y consumidores:**

Se usa un **semáforo binario dividido**, donde los semáforos binarios (b1..bn) lo forman si el siguiente es un invariante global: SPLIT: 0 <= b1, + .. + bn <= 1

Los b pueden verse como un único semáforo binario que fue dividido en n semáforos binarios. Las sentencias entre el P y el V ejecutan con exclusión mutua.

Los semáforos binarios en realidad son semáforos generales, pero su valor no va a ser más de 1.

Diagrama

Descripción generada automáticamente con confianza baja

**Buffers limitados: Contadores de recursos**

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

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Vacio** cuenta los lugares y **lleno** los ocupados. Depositar y retirar se pueden asumir atómicas porque hay un productor y un consumidor. Si hay más de un productor y un consumidor las operaciones para depositar y retirar son SC y se deben ejecutar con Exclusión mutua. Sino podrían retirarse dos veces el mismo dato o sobrescribir datos.

**Varios procesos compitiendo por varios recursos compartidos**

Problema de varios procesos P y varios recursos R cada uno protegido por un lock. Un proceso debe adquirir los locks que necesita, pero puede caerse en deadlock cuando varios compiten por conjuntos superpuestos de recursos.

De acá sale el **problema de los filósofos**.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Cada tenedor es una SC que puede ser tomado por un único filosofo a la vez. Pueden representarse los tenedores como un arreglo de semáforo.

El problema es que si todos hacen exactamente lo mismo se revienta todo ( hay deadlock).

Solución:

Alcanza con que uno solo de los filósofos tome primero el tenedor izquierdo y luego del derecho ( opuesto al resto ). Así seguro va a quedar un filosofo que no pueda tomar sus 2 tenedores, evitando así el deadlock.Texto

Descripción generada automáticamente con confianza media

**Lectores y escritores.**

Dos clases de procesos comparten una BD. El acceso de los escritores debe ser exclusivo para evitar interferencia entre transacciones. Los lectores pueden ejecutar concurrentemente entre ellos si no hay escritores actualizando.

Los escritores necesitan EM para acceder, y los lectores necesitan EM con respecto a cualquier escritor para acceder.

Texto

Descripción generada automáticamente con confianza media

Esta solución esta buena, pero le da preferencia a los lectores y esto no es fair. Otro enfoque es usar la técnica de passing the baton, que emplea el SBS para brindar exclusión y despertar procesos demorados. Puede usarse para implementar await arbitrarios controlando de forma precisa el orden en que los procesos son despertados.

**Passing the baton:** Se usa para implementar sentencias await. Cuando un proceso esta dentro de una SC mantiene el **batón.** Cuando el proceso sale de la SC pasa el control a otro proceso. Si ningún proceso está esperando por el baton, el baton se libera para que lo tome el próximo proceso que trata de entrar.

La sincronización se expresa con sentencias atómicas de forma F1: <Si> o F2: <await (Bj) Sj> , que puede hacerse con semáforos binarios divididos.

e = semáforo usado para manejar exclusión mutua inicialmente en 1

Se usan semáforos bj y un contador dj para cada uno con guarda diferente Bj, todos inicialmente en 0. bj se usa para demorar procesos esperando que Bj sea true, dj es un contador del numero de procesos demorados sobre bj.

e y los bj se usan para formar un SBS: a lo sumo uno a la vez es 1 y cada camino de ejecución empieza con un P y termina con un único V.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Es un kilombo básicamente.

Volviendo al problema de los lectores y escritores:

Técnica con sentencias await // Técnica con passing the baton

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

El rol de los signal es el de señalizar exactamente a uno de los semáforos, es decir los procesos se van pasando el baton. Algunos de los SIGNAL se pueden simplificar.

Problema simplificado:

Texto

Descripción generada automáticamente con confianza media

Da preferencia a los lectores, por lo que puede pasar que no se cumpla la eventual entrada para los escritores.

Alocación de recursos y scheduling

**Problema:** Decidir cuando se le puede dar a un proceso acceso a un recurso

**Recurso:** Cualquier objeto, elemento, SC, por la que un proceso puede ser demorado esperando adquirirlo

**Definición del problema:** Procesos que compiten por el uso de unidades de un recurso compartido.

Para esto puede usarse **passing the baton**

**Alocacion Shortest-job-next**

Varios procesos que compiten por el uso de un recurso compartido de una sola unidad. Para el request si el recurso está libre se aloca al proceso id, sino se demora. Para el reléase cuando se libera se aloca el proceso demorado con el mínimo valor de tiempo.

SJN minimiza el tiempo promedio de ejecución, aunque es unfair. Podría mejorarse con la técnica de aging, es decir darle preferencia a un proceso que espero mucho tiempo.