SINCRONIZACIÓN

La sincronización es la relación entre dos o más eventos, estos podrían ser antes, después o durante. Los tipos de modelo de ejecución son:

Monoprocesador monotarea: no tenemos ningún problema debido a que no tenemos dos eventos concurriendo simultáneamente entonces no tenemos problemas con la sincronización.

Monoprocesador multitarea: en el mismo se pueden tener multiprocesos o multihilos (son no determinístico: no se ejecuta siempre igual, son difícil de depurar y además tiene problemas de sincronización), en los dos casos el planificador decide quién sigue.

Multiprocesador multitarea: en este tipo de sistema no se conoce el orden de la sentencia que se va a ejecutar.

Relación Entre Eventos

Concurrencia: esto ocurre cuando tengo multitarea debido a que más de un evento puede ocurrir a la vez y mirando el programa no se sabe cuál evento ocurre primero (no determinístico).

Serialización: en la serialización un evento se debe producir antes que otro, pudiendo este utilizar recursos modificados del primero que se ejecutó y no los recursos antes que se ejecutara. Se deben imponer restricciones por software. Ej. cajero

Punto de encuentro: Un evento debe esperar por otro evento e inversamente en un punto de la ejecución, y ninguno de los dos puede continuar hasta que ambos hayan llegado. Si tenemos más de uno se llama barrera. Esto posibilita que ambos eventos muestren el mismo dato, requiere restricciones por software.

Exclusión mutua: si dos eventos modifican recursos compartidos estos no deben producirse al mismo tiempo (no concurrente), siendo el resultado dependiente del orden de corrida por lo tanto se deben usar restricciones por software. Ej. escritura en base de datos.

Herramientas Para Sincronizar

SEMAFOROS

Un semáforo es una variable, por lo tanto, hay que definirla. En el software, un semáforo es una estructura de datos útil para resolver diversos problemas de sincronización.

Un semáforo es un entero, pero con 3 diferencias:

1. Cuando se crea el semáforo, se puede inicializar su valor con cualquier número entero, pero después las únicas operaciones que se pueden realizar son el incremento (aumentar en uno) y el decremento (disminuir en uno). No se puede leer el valor actual del semáforo.

2. Cuando un hilo disminuye el semáforo, si el resultado es negativo, el hilo se bloquea y no puede continuar hasta que otro hilo incremente el semáforo. No podemos saber si ese decremento bloqueara al hilo.

3. Cuando un hilo incrementa el semáforo, si hay otros hilos esperando, uno de los hilos en espera se desbloquea.

Decir que un hilo se bloquea (o simplemente "se bloquea") es decir que notifica al planificador que no puede continuar. El programador impedirá que el hilo se ejecute hasta que ocurra un evento que haga que el hilo se desbloquee.

Ambas primitivas son atómicas usan TSL (TEST AND SET LOCK), es decir, el planificador no corta el proceso justo ahí.

MUTEX

Los mutex permiten a los hilos sincronizar su uso de un recurso compartido, de modo que, por ejemplo, un hilo no intente acceder a una variable compartida al mismo tiempo que otro hilo la está modificando. Solo puede tener como valor “0” o “1”.

BARRIER (BARRERA)

Básicamente es la primitiva para el punto de encuentro para N eventos. Las primitivas que se utilizan son atómicas y son utilizadas para esperar a todos los procesos hasta que llegan al punto de encuentro.

VARIABLE DE CONDICIÓN

Trabaja en simultaneo con un mutex y es para evitar esperas activas.

TSL: *Instrucciones que deshabilitan el bus de datos mientras estoy modificando una variable.*

Solución a problemas de Sincronización

Productor – Consumidor: si no hay elementos para quitar, el consumidor se bloquea, solución:

- Usar un mutex para el acceso exclusivo al buffer.
- Usar semáforos para que el productor señalice que coloco un elemento nuevo.
- Poner otro semáforo para realizar el seguimiento del espacio disponible en el buffer.
- Los productores y consumidores deben comprobar la disponibilidad antes de conseguir el mutex.

Lectores – Escritores: cualquier número de lectores puede acceder a los datos simultáneamente (concurrente) y los escritores deben tener acceso exclusivo a los datos (sección critica). La solución a este problema es aplicando semáforos.

Interbloqueo: interbloqueo es el conjunto de procesos que esperan un recurso que tiene un proceso del conjunto.

Poner bandera soluciona el problema de serialización, pero gasto mucho recurso de la CPU y no aseguro que sea una instrucción TSL es decir que no es atómica el cambio de bandera por lo tanto la relación de eventos se soluciona a través de:

Serialización

Hilo 1()
deposito plata
pthread_mutex_ulock(&mtx)

Hilo 2()
pthread_mutex_lock(&mtx)
saco plata

main()
Inicio el mutex (inicia en 1)
pthread_mutex_lock(&mtx)

Exclusión mutua

Hilo 1()
pthread_mutex_lock(&mtx)
zona critica (variable global)
pthread_mutex_ulock(&mtx)

Hilo 2()
pthread_mutex_lock(&mtx)
zona critica (variable global)
pthread_mutex_ulock(&mtx)

main()
inicio el mutex (inicia en 1)
pthread_mutex_lock(&mtx)

Punto de encuentro

Hilo 1()
encuentro
pthread_mutex_ulock(&mtx2)
pthread_mutex_lock(&mtx1)
Mismo dato H2

Hilo 2()
encuentro
pthread_mutex_ulock(&mtx1)
pthread_mutex_lock(&mtx2)
Mismo dato H1

main()
inicio los mutex (inician en 1)
pthread_mutex_lock(&mtx1)
pthread_mutex_lock(&mtx)

Interbloqueo

Hilo 1()
pthread_mutex_***lock***(&mtx2)
pthread_mutex_ulock(&mtx1)

Hilo 2()
pthread_mutex_***lock***(&mtx1)
pthread_mutex_ulock(&mtx2)

main()
Inicio los mutex (inician en 1)
pthread_mutex_lock(&mtx1)
pthread_mutex_lock(&mtx2)