TAREA 5

GRUPO: 2CM12
SEMESTRE: 22/2

EQUIPO :

BALDOVINOS GUTIÉRREZ KEVIN

BOCANEGRA HEZIQUIO YESTLANEZI

CASTAÑARES TORRES JORGE DAVID

HERNÁNDEZ HERNÁNDEZ RUT ESTHER

PROFESOR:

JORGE CORTÉS GALICIA

INFOGRAFÍA DE LA LECTURA 6 Y LA LECTURA 7 (TEMA 2.5)

Monitores

Estructura fundamental de sincronización de alto nivel

Utilización

Un tipo abstracto de datos, agrupa una serie de datos privados con un conjunto de métodos públicos que se utilizan para operar sobre dichos datos

Solución al problema de la cena de los filósofos

Impone la restricción de que un filósofo puede coger sus palillos sólo si ambos están disponibles. Para codificar esta solución, necesitamos diferenciar entre tres estados en los que puede hallarse un filósofo

Implementación utilizando semáforos

Para cada monitor se proporciona un semáforo mutex inicializado con el valor de 1. Un proceso debe ejecutar la operación wait (mutez) antes de entrar en el monitor y tiene que ejecutar una operación signal (mutex) después de salir del monitor

Reanudación de procesos

Si hay varios procesos suspendidos en la condición x y algún proceso ejecuta una operación x. signal(). Una solución sencilla consiste en usar el orden FCFS, de modo que el proceso que lleve más tiempo en espera se reanude en primer lugar.

Monitores

A fin de facilitar la escritura de programas correctos, Hoare y Brinch Hansen propusieron una primitiva de sincronización de nivel más alto llamada monitor

Un monitor es una colección de procedimientos, variables y estructuras de datos que se agrupan en un tipo especial de módulo o paquete

Los procesos pueden invocar los procedimientos de un monitor en el momento en que deseen, pero no pueden acceder directamente a las estructuras de datos internas del monitor desde procedimientos declarados afuera del monitor

Los monitores poseen una propiedad especial que los hace útiles para lograr la exclusión mutua: sólo un proceso puede estar activo en un monitor en un momento dado.

Por lo regular, cuando un proceso invoca un procedimiento de monitor, las primeras instrucciones del procedimiento verifican si hay algún otro proceso activo en ese momento dentro del monitor. Si así es, el proceso invocador se suspende hasta que el otro proceso abandona el monitor. Si ningún otro proceso está usando el monitor, el proceso invocador puede entrar

Es responsabilidad del compilador implementar la exclusión mutua en las entradas a monitores, pero una forma común es usar un semáforo binario

La solución está en la introducción de variables de condición, junto con dos operaciones que se realizan con ellas, WAIT y SIGNAL. Cuando un procedimiento de monitor descubre que no puede continuar (si encuentra lleno el buffer), ejecuta WAIT (esperar) con alguna variable de condición, digamos full (lleno). Esta acción hace que el proceso invocador se bloquee, y también permite la entrada de otro proceso al que antes se le había impedido entrar en el monitor.

INFOGRAFÍA DE LA LECTURA 8 (TEMA 2.6)

Transferencia de mensajes

Este método de comunicación entre procesos utiliza dos primitivas SEND y RECEIVE que, al igual que los semáforos y a diferencia de los monitores, son **llamadas al sistema** y no construcciones del lenguaje.

1 Transferencia

La primera llamada envía un mensaje a un destino dado, y la segunda recibe un mensaje de un origen dado. Si no hay un mensaje disponible, el receptor podría bloquearse hasta que uno llegue. Como alternativa, podría regresar de inmedia-to con un código de error.

2 Aspectos de diseño

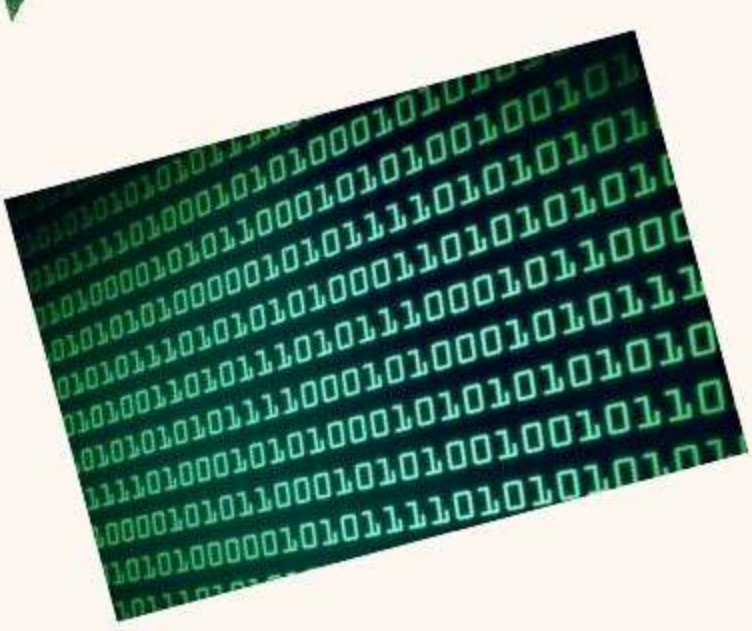
Los sistemas de mensajes también tienen que resolver la cuestión del nombre de los procesos, a fin de que el proceso especificado en una llamada SEND o RECEIVE no sea ambiguo. Hay aspectos de diseño que son importantes cuando el emisor y el receptor están en la misma máquina. Uno de éstos es el rendimiento. El copiado de mensajes de un proceso a otro siempre es más lento que efectuar una operación de semáforo o entrar en un monitor.

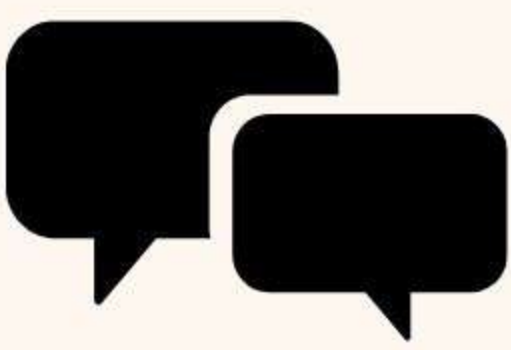
3 El problema de productor-consumidor

El consumidor inicia enviando N mensajes vacíos al productor. Cada vez que el productor tiene un elemento que entregar al consumidor, toma un mensaje vacío y devuelve uno lleno. De este modo, el número total de mensajes en el sistema permanece constante y pueden almacenarse en una cantidad de memoria que se conoce con antelación.

4 Conclusión

Cuando se adopta este enfoque, si el SEND se ejecuta antes que el RECEIVE, el proceso emisor queda bloqueado hasta que ocurre el RECEIVE, y en ese momento el mensaje podrá copiarse directamente del emisor al receptor, sin buffers intermedios. De forma similar, si el RECEIVE se ejecuta primero, el receptor se bloquea hasta que ocurre el SEND. Esta estrategia se conoce como **cita o rendezvous**





HISTORIETA DE LA LECTURA 9 Y LA LECTURA 10 (TEMA 2.7)

Problemas clásicos de la comunicación entre procesos

Capítulo 1
Problema del buffer limitado


Supongamos que la cola consta de n bufferes, siendo cada uno capaz de almacenar un elemento


El semáforo proporciona exclusión mutua para los accesos al conjunto de bufferes y se inicializa con el valor 1


Los semáfotos empty y full cuetan el número de bufferes vacios y llenos


El semáforo empty se inicializa con el valor n; el semáforo full se inicializa con el valor de 0


Podemos interpretar esto como un productor que genera los bufferes llenos para el consumidor


o como un consumidor que genera bufferes vacios para el productor

Capítulo 2
El problema de los lectores-escritores


Imagina que una base de datos debe ser compartida por varios procesos concurrentes


Algunos de estos procesos pueden querer leer datos, mientras que otros pueden querer actualizarla


Para asegurar que eso no pase, requerimos que los procesos escritores tengan acceso exclusivo a la base de datos compartida


Ningun lector debe esperar a que otros lectores terminen porque un proceso escritor esté esperando


Si un escritor está esperando para acceder al objeto, ningún lector nuevo puede iniciar a leer


Eso puede dar como resultado un problema de inanición


En algunos sistemas, sus soluciones se han generalizado para proporcionar bloqueos lector-escritor


Esto permite que múltiples procesos adquieran de forma concurrente un cerrojo


En aquellas aplicaciones en las que sea fácil identificar qué procesos sólo desean leer los datos compartidos


También en aquellas aplicaciones que tengan más procesos lectores que escritores

Capítulo 3
Problema de la cena de los filósofos

Se considera un problema clásico de sincronización, es un problema de concurrencia, donde trata de repartir recursos entre varios proceso

Una solución sencilla consiste en usar semáfors. Un filósofo intenta hacerse con un palillo ehecutando una operación wait()

En dicho semáforo y libera sus palillos ejecutando la operación signal() en los semáforos adecuados

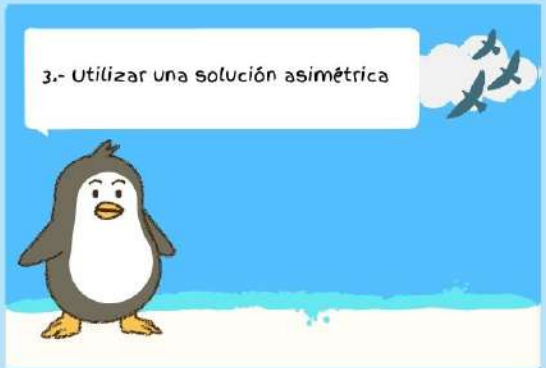
Supongamos que los 5 filosofos sienten hambre a la vez, tendrán que esperar eternamente

Pero no te preocupes, yo te diré las posibles soluciones a este problema de interbloqueo

1.- Permitir que como máximo haya 4 filósofos sentados a la mesa al mismo tiempo

2.- Pertitir a cada filósofo agarrar sus palillos sólo si ambos palillos están disponibles

3.- Utilizar una solución asimétrica



Es muy fácil, un filósofo impar agarra primero el palillo de su izquierda y luego el que está a su derecha


Mientras que un filósofo par agarra primero el palillo de su derecha y luego el de la izquierda


Recuerda... Toda solución al problema debe proteger de que uno de los filósofos muera por inanición


Una solucion libre de interbloqueos no necesariamente elimina la posibilidad de muerte por unanición

Capítulo 4
Problema del peluquero dormido

Esta peluquería tiene un peluquero, una silla de peluquero y n sillas donde pueden sentarse los clientes que esperan, si los hay

Si no hay clientes presentes, el peluquero se sienta en la silla de peluquero y se duerme

Cuando llega un cliente, tiene que despertar al peluquero dormido

Si llegan clientes adicionales mientras el peluquero está cortándole el pelo a un cliente, se sientan o bien salen del establecimiento

La solución utiliza tres semáforos: customers, que cuenta a los clientes en espera, barbers, el número de peluqueros que están ociosos

Esperando clientes, y mutex, que se usa para la exclusión mutua ademas necesitamos una variable, waiting

que también cuenta los clientes que están esperando, y que en esencia es una copia de customers

Cuando el peluquero llega a trabajar en la mañana, ejecuta el procedimiento barber que lo obliga a bloquearse

En espera de customers hasta que llegue alguien. Luego se duerme.


Cuando un cliente llega, ejecuta customer, cuya primera instrucción es adquirir mutex para entrar en una región crítica


que también cuenta los clientes que están esperando, y que en esencia es una copia de customers


Si otro cliente llega poco tiempo después, no podrá hacer nada hasta que el primero haya liberado mutex


Una vez terminado el corte de pelo, el cliente sale del procedimiento y de la peluquería