**Centro Universitario de Ciencias Exactas e Ingenerias**

**Departamento de Ciencias Computacionales**

**Seminario de Sistemas Operativos**



**Violeta del Rocio Becerra Velazquez**

**Edgar Agustin Martinez Gonzalez**

**220286695**

**Ingeneria en Computacion**

**D01**

**Tarea 5**

**02/10/2022**

**Problema de la cena de los filósofos**

**Enunciado del problema**

Cinco filósofos se sientan alrededor de una mesa y pasan su vida cenando y pensando. Cada filósofo tiene un plato de fideos y un tenedor a la izquierda de su plato. Para comer los fideos son necesarios dos tenedores y cada filósofo sólo puede tomar los que están a su izquierda y derecha. Si cualquier filósofo toma un tenedor y el otro está ocupado, se quedará esperando, con el tenedor en la mano, hasta que pueda tomar el otro tenedor, para luego empezar a comer.

Si dos filósofos adyacentes intentan tomar el mismo tenedor a una vez, se produce una condición de carrera: ambos compiten por tomar el mismo tenedor, y uno de ellos se queda sin comer.

Si todos los filósofos toman el tenedor que está a su derecha al mismo tiempo, entonces todos se quedarán esperando eternamente, porque alguien debe liberar el tenedor que les falta. Nadie lo hará porque todos se encuentran en la misma situación (esperando que alguno deje sus tenedores). Entonces los filósofos se morirán de hambre. Este bloqueo mutuo se denomina interbloqueo o deadlock.

El problema consiste en encontrar un algoritmo que permita que los filósofos nunca se mueran de hambre.

**Diversas soluciones posibles**

**Por turno cíclico**

Se empieza por un filósofo, que si quiere puede comer y después pasa su turno al de la derecha. Cada filósofo sólo puede comer en su turno. Problema: si el número de filósofos es muy alto, uno puede morir de hambre antes de su turno.

**Varios turnos**

Se establecen varios turnos. Para hacerlo más claro supongamos que cada filósofo que puede comer (es su turno) tiene una ficha que después pasa a la derecha. Si por ejemplo hay 7 comensales podemos poner 3 fichas en posiciones alternas (entre dos de las fichas quedarían dos filósofos).

Se establecen turnos de tiempo fijo. Por ejemplo cada 5 minutos se pasan las fichas (y los turnos) a la derecha.

Con base al tiempo que suelen tardar los filósofos en comer y en volver a tener hambre, el tiempo de turno establecido puede hacer que sea peor solución que la anterior. Si el tiempo de turno se aproxima al tiempo medio que tarda un filósofo en comer esta variante da muy buenos resultados. Si además el tiempo medio de comer es similar al tiempo medio en volver a tener hambre la solución se aproxima al óptimo.

**Colas de tenedores**

Cuando un filósofo quiere comer se pone en la cola de los dos tenedores que necesita. Cuando un tenedor está libre lo toma. Cuando toma los dos tenedores, come y deja libre los tenedores.

Visto desde el otro lado, cada tenedor sólo puede tener dos filósofos en cola, siempre los mismos.

Esto crea el problema comentado de que si todos quieren comer a la vez y todos empiezan tomando el tenedor de su derecha se bloquea el sistema (deadlock).

**Resolución de conflictos en colas de tenedores**

Cada vez que un filósofo tiene un tenedor espera un tiempo aleatorio para conseguir el segundo tenedor. Si en ese tiempo no queda libre el segundo tenedor, suelta el que tiene y vuelve a ponerse en cola para sus dos tenedores.

Si un filósofo A suelta un tenedor (porque ha comido o porque ha esperado demasiado tiempo con el tenedor en la mano) pero todavía desea comer, vuelve a ponerse en cola para ese tenedor. Si el filósofo adyacente B está ya en esa cola de tenedor (tiene hambre) lo toma y si no vuelve a cogerlo A.

Es importante que el tiempo de espera sea aleatorio o se mantendrá el bloqueo del sistema.

**El portero del comedor**

Se indica a los filósofos que abandonen la mesa cuando no tengan hambre y que no regresen a ella hasta que vuelvan a estar hambrientos (cada filósofo siempre se sienta en la misma silla). La misión del portero es controlar el número de filósofos en la sala, limitando su número a n-1, pues si hay n-1 comensales seguro que al menos uno puede comer con los dos tenedores.

**Problema productor-consumidor**

**El problema**

El programa describe dos procesos, productor y consumidor, ambos comparten un buffer de tamaño finito. La tarea del productor es generar un producto, almacenarlo y comenzar nuevamente; mientras que el consumidor toma (simultáneamente) productos uno a uno. El problema consiste en que el productor no añada más productos que la capacidad del buffer y que el consumidor no intente tomar un producto si el buffer está vacío.

**Soluciones**

La idea para la solución es la siguiente, ambos procesos (productor y consumidor) se ejecutan simultáneamente y se “despiertan” o “duermen” según el estado del buffer. Concretamente, el productor agrega productos mientras quede espacio y en el momento en que se llene el buffer se pone a “dormir”. Cuando el consumidor toma un producto notifica al productor que puede comenzar a trabajar nuevamente. En caso contrario, si el buffer se vacía, el consumidor se pone a dormir y en el momento en que el productor agrega un producto crea una señal para despertarlo. Se puede encontrar una solución usando mecanismos de comunicación de interprocesos, generalmente se usan semáforos.

**Problema de lectores y editores**

**El problema**

Suponga que una base de datos tiene lectores y escritores, y los lectores y escritores de esa base de datos deben estar programados.

Las restricciones son las siguientes:

* varios lectores deben poder leer la base de datos al mismo tiempo;
* si un editor está modificando la base de datos, ningún otro usuario (ni editor ni lector) debería poder acceder a ella.

Es bastante sencillo tener el editor en espera mientras todavía hay lectores.

Pero esta solución presenta grandes problemas, si el flujo de lectores es regular: el editor podría tener que esperar un tiempo infinito.

Por tanto, existe una segunda solución, que consiste en poner en espera a todos los lectores que han enviado su solicitud de acceso después de la de un editor.

Edsger Dijkstra , quien formuló este problema, propone resolverlo mediante semáforos .

**Solución con uso de semáforos y prioridad a los lectores**

La siguiente solución resuelve el problema de lectores y escritores al priorizar a los lectores. Esta solución requiere tres semáforos y una variable, a saber:

* Un semáforo *M\_Lec*t, inicializado a 1, que permite proteger la variable lect. Por tanto, es un mutex .
* Un semáforo *M\_Red*, inicializado a 1, que permite bloquear las tareas de escritura (dando prioridad a los lectores). Por tanto, es un mutex .
* Un semáforo rojo, inicializado a 1, que permite bloquear las tareas de lectura si se está escribiendo en curso.
* Variable *lect* que cuenta el número de lectores.

**Conteste las siguientes preguntas:**

1. ¿En qué consiste el problema de la concurrencia?

Consiste en las situaciones en las que dos o más procesos puedan coincidir en el acceso a un recurso compartido o, dicho de otra forma, que requieran coordinarse en su ejecución. Para evitar dicha coincidencia, el sistema operativo ofrece mecanismos de arbitraje que permiten coordinar la ejecución de los procesos.

2. ¿Cuáles son los procesos concurrentes cooperantes?

puede afectar o ser afectado por los demás procesos que se ejecutan en el sistema, colaboran entre sí buscando un objetivo común. Obviamente, cualquier proceso que comparte datos con otro proceso es cooperativo.

3. ¿En qué consiste la Exclusión mutua?

La exclusión mutua es la actividad que realiza el sistema operativo para evitar que dos o más procesos ingresen al  mismo tiempo a un área de datos compartidos o accedan a un mismo recurso.

4. Defina Interbloqueo.

Un interbloqueo se produce cuando dos subprocesos bloquean una variable diferente al mismo tiempo y, a continuación, intentan bloquear la variable que el otro subproceso ya ha bloqueado. Como resultado, cada subproceso deja de ejecutarse y espera a que el otro subproceso libere la variable.

5. Defina Inanición.

Es un problema relacionado con los sistemas multitarea, donde a un proceso o un hilo de ejecución se le deniega siempre el acceso a un recurso compartido. Sin este recurso, la tarea a ejecutar no puede ser nunca finalizada.

7. ¿Qué son los Hilos?

Es un medio que permite administrar las tareas de un procesador y de sus diferentes núcleos de una forma más eficiente. Gracias a los hilos, las unidades mínimas de asignación, que son las tareas o procesos de un programa, pueden dividirse en trozos para así optimizar los tiempos de espera de cada instrucción en la cola del proceso.

8. ¿Qué son los Semáforos?

Los semáforos son una herramienta de sincronización que ofrece una solución al problema de la sección crítica (porción de código de un programa de computador en la cual se accede a un recurso compartido que no debe ser accedido por mas de un proceso o hilo en ejecución). Un semáforo provee una simple pero útil abstracción para controlar el acceso de múltiples procesos a un recurso común en programación paralela, o entornos multiusuarios.

9. ¿Qué es lo que mejora el tener más de un núcleo?

La mayor ventaja de tener más núcleos en una CPU es que pueden trabajar paralelamente para ofrecer un mejor rendimiento