10-3-2024

Centro universitario de ciencias exactas e ingenierías

Sistemas operativos

Actividad

Temas:

Producto consumidor, filósofos comensales y lectores escritores

Alumnos:

Agosto Aceves Carolina

Briseida Yeilin Zepeda Núñez

Jessica Alexandra Magaña Salcedo

Garcia Herrera Juan Pablo

Maestra: Violeta del Roció Becerra Velázquez

Sección: D04

Contenido

ilustraciones

Modelos de productor/consumidor

El modelo de productor/consumidor (a veces denominado *pipelining*) se tipifica mediante una línea de producción. Un artículo procede de componentes en bruto a un artículo final en una serie de etapas.

Normalmente, un solo trabajador en cada etapa modifica el elemento y lo pasa a la siguiente etapa. En términos de software, un conducto de mandatos de AIX® , como el mandato **cpio** , es un ejemplo de este modelo.

Por ejemplo, una entidad de lector lee datos en bruto de la entrada estándar y los pasa a la entidad de procesador, que procesa los datos y los pasa a la entidad de grabador, que los graba en la salida estándar. La programación en paralelo permite que las actividades se realicen simultáneamente: la entidad de transcriptor puede generar algunos datos procesados mientras que la entidad de lector obtiene más datos en bruto.

En computación, el **problema del productor-consumidor** es un ejemplo clásico de problema de sincronización de multiprocesos. El programa describe dos procesos, productor y consumidor, ambos comparten un búfer 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.

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. Una inadecuada implementación del problema puede terminar en un deadlock, donde ambos procesos queden en espera de ser despertados.

El patrón de diseño productor/consumidor se basa en el patrón maestro/esclavo y está diseñado para mejorar el intercambio de datos entre múltiples ciclos que se ejecutan a

diferentes velocidades. El patrón productor/consumidor se utiliza para separar procesos que producen y consumen datos a diferentes velocidades. Los ciclos paralelos del patrón productor/consumidor se dividen en dos categorías; los que producen datos y los que consumen los datos producidos.

Comunicación en búfer

Cuando hay varios procesos que se ejecutan a diferentes velocidades, la comunicación en búfer entre procesos es extremadamente efectiva. Con un búfer lo suficientemente grande, el ciclo productor puede funcionar a velocidades mucho más altas que el ciclo consumidor sin pérdida de datos.

Por ejemplo, considere que una aplicación tiene dos procesos; el primer proceso realiza la adquisición de datos y el segundo proceso toma esos datos y los coloca en una red. El primer proceso opera a tres veces la velocidad del segundo proceso. Si se utiliza el patrón de diseño productor/consumidor para implementar esta aplicación, el proceso de adquisición de datos actuará como productor y el proceso de red como consumidor. Con una cola de comunicación (búfer) suficientemente grande, el proceso de red tendrá acceso a una gran cantidad de datos que adquiere el ciclo de adquisición. Esta capacidad de almacenar datos en búfer minimizará la pérdida de datos.

Adquisición de datos y procesamiento

El patrón productor/consumidor se usa comúnmente cuando se adquieren múltiples conjuntos de datos para procesarlos en orden.

Supongamos que desea escribir una aplicación que acepte datos mientras los procesa en el orden en que se recibieron. Debido a que poner en cola (producir) estos datos son mucho más rápido que el procesamiento actual (consumir), el patrón de diseño productor/consumidor es el más adecuado para esta aplicación.

Comunicación en red

La comunicación en red requiere dos procesos para operar al mismo tiempo y a diferentes velocidades: el primer proceso sondearía constantemente la línea de la red y recuperaría paquetes, y el segundo proceso tomaría estos paquetes recuperados por el primer proceso y los analizaría. En este ejemplo, el primer proceso actuará como productor porque está proporcionando datos al segundo proceso, que actuará como consumidor

El problema de la cena de los filósofos

En 1965 Dijkstra planteó y resolvió un problema de sincronización llamado el problema de la cena de los filósofos, que se puede enunciar como sigue. Cinco filósofos se sientan a la mesa, cada uno con un plato de espaghetti. El espaghetti es tan escurridizo que un filósofo necesita dos tenedores para comerlo. Entre cada dos platos hay un tenedor. En la figura 4.15 se muestra la mesa. La vida de un filósofo consta de periodos alternos de comer y pensar. Cuando un filósofo tiene hambre, intenta obtener un tenedor para su mano derecha, y otro para su mano izquierda, cogiendo uno a la vez y en cualquier orden. Si logra obtener los dos tenedores, come un rato y después deja los tenedores y continúa pensando. La pregunta clave es: ¿puede el lector escribir un programa para cada filósofo que permita comer equitativamente a los filósofos y no se interbloquee?

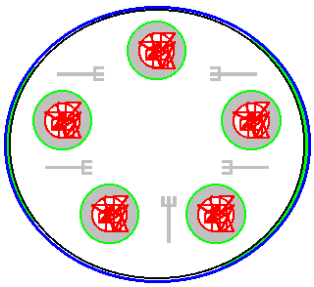

Ilustración 1mesa de comensales

```
#define N 5                                 /* número de filósofos */

void filósofo (int i)                       /* i: qué filósofo (desde 0 hasta N-1) */
{
while (1) {
    pensar ( );                            /* el filósofo está pensando */
    coger_tenedor (i);                      /* coge el tenedor izquierdo */
    coger_tenedor ((i + 1) % N); /* coge el tenedor derecho */
    comer ( );
    dejar_tenedor (i);                      /* deja el tenedor izquierdo en la mesa */
    dejar_tenedor ((i + 1) % N);  /* deja el tenedor derecho en la mesa */
}
}
```

Ilustración 2ejemplo de solucion en pseudocodigo

La figura muestra una solución obvia. El procedimiento coger_tenedor espera hasta que el tenedor especificado esté disponible y lo coge. Por desgracia la solución obvia es incorrecta. Supongamos que los cinco filósofos cogen sus tenedores izquierdos de forma simultánea. Ninguno podría coger su tenedor derecho, lo que produciría un interbloqueo.

```
#define N                5              /* número de filósofos */
#define IZQ          (i - 1) % N        /* número del vecino izq.  de i */
#define DER          (i + 1) % N        /* número del vecino der. de i */
#define PENSANDO     0                  /* el filósofo está pensando */
#define HAMBRIENTO   1                  /* el filósofo está hambriento */
#define COMIENDO     2                  /* el filósofo está comiento */

int estado[N];                          /* vector para llevar el estado de los filósofos */
semáforo exmut = 1;                     /* exclusión mutua para las secciones críticas */
semáforo s[N];                          /* un semáforo por filósofo */
```

```
void filósofo (int i)                   /* i: de qué filósofo se trata (desde 0 hasta N - 1) */
{
while (1) {                             /* se ejecuta eternamente */
    pensar ( );                         /* el filósofo está pensando */
    coger_tenedores (i);                /* obtiene dos tenedores, bloqueándose si no puede */
    comer ( );
    dejar_tenedores (i);                /* deja ambos tenedores en la mesa */
}
}
```

Ilustración 3 ejemplo de solucion en pseudocodigo 2

Se podría modificar el programa de forma que después de coger el tenedor izquierdo, el programa verificara si el tenedor derecho está disponible. Si no lo está, el filósofo deja el izquierdo, espera cierto tiempo y vuelve a repetir el proceso. Esta propuesta también falla, aunque por razones distintas. Con un poco de mala suerte todos los filósofos podrían empezar el algoritmo de forma simultánea, por lo que cogerían sus tenedores izquierdos, verían que los derechos no están disponibles, esperarían, volverían a coger sus tenedores izquierdos simultáneamente, etc. eternamente. Esto implica un *aplazamiento indefinido*.

El lector podría pensar: "si los filósofos esperaran un tiempo arbitrario, en vez del mismo tiempo, después de que no pudieran coger el tenedor derecho, la probabilidad de que todo quedara bloqueado, incluso una hora, sería muy pequeña". Esta observación es correcta, pero en ciertas aplicaciones se desea una solución que funcione siempre, y no que pueda

funcionar bien con gran probabilidad. (Piense en el control de seguridad de una planta nuclear).

Una mejora a la *figura 4.16*, que no tiene interbloqueos ni aplazamiento indefinido, es la protección de los cinco enunciados siguientes a la llamada al procedimiento *pensar* mediante un semáforo binario *exmut*. Antes de empezar a coger los tenedores, un filósofo haría un **wait** sobre *exmut*. Desde el punto de vista teórico esta solución es adecuada. Desde el punto de vista práctico presenta un error de eficiencia: en todo instante existirá a lo sumo un filósofo comiendo. Si se dispone de cinco tenedores, se debería permitir que dos filósofos comieran al mismo tiempo.

La solución que aparece en la *figura 4.17* es correcta, y permite el máximo paralelismo para un número arbitrario de filósofos. Utiliza un vector, *estado*, para llevar un registro de la actividad de un filósofo: si está comiento, pensando o hambriento (estado que indica que quiere coger los tenedores). Un filósofo puede comer únicamente si los vecinos no están comiendo. Los vecinos del i-ésimo filósofo se definen en las macros *IZQ* y *DER*. En otras palabras, si *i*= 2, entonces *IZQ* = 1, y *DER* = 3.

El programa utiliza un vector de ***semáforos***, uno por filósofo, de forma que los filósofos hambrientos puedan bloquearse si los tenedores necesarios están ocupados. Observe que cada proceso ejecuta el procedimiento *filósofo* como programa principal, pero los demás procedimientos, *coger_tenedores*, *dejar_tenedores* y *prueba*, son procedimientos ordinarios y no procesos separados.

El problema de la cena de los filósofos o problema de los filósofos cenando (dining philosophers problem)

es un problema clásico de las ciencias de la computación propuesto por Edsger Dijkstra en 1965 para representar el problema de la sincronización de procesos en un sistema operativo. Cabe aclarar que la interpretación está basada en pensadores chinos, quienes comían con dos palillos, donde es más lógico que se necesite el del comensal que se siente al lado para poder comer.

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.

- **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.

- **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 de lectores y escritores

Consiste de dos procesos que acceden el mismo bloque de memoria compartida al mismo tiempo tomando en cuenta la restricción de que dos acciones no pueden ser llevadas a cabo sobre el mismo bloque de memoria al mismo tiempo, es decir, si un proceso esta escribiendo un bloque, este no puede ser escrito ni leído por ningún otro proceso manteniendo así la exclusión mutua sobre una región critica

El «problema de lectores-escritores» es un dilema de programación que se crea cuando varios lectores y escritores necesitan acceder al mismo recurso. Si se les permitiera el acceso a todos a la vez, podrían surgir problemas como sobreescrituras, información incompleta y otros problemas. Por lo tanto, los programadores pueden restringir el acceso para controlar qué subprocesos de procesamiento ven el recurso y cuándo, considerando las necesidades del sistema y los usuarios. Hay varias formas de abordar el problema de lectores-escritores. Una de las soluciones más comunes implica el uso de semáforos para marcar el estado y controlar el acceso.

Desde una perspectiva, cualquier número de lectores podría acceder de forma segura a un recurso porque no están realizando cambios en el contenido. Una vez que un escritor entra en la ecuación, la situación se vuelve más complicada. Si un hilo está escribiendo mientras otros hilos están leyendo, es posible que los lectores no obtengan la información

correcta. Podrían recibir solo una parte del cambio, o podrían ver la información desactualizada y pensar que es precisa.

En una solución en la que los lectores tienen prioridad, el sistema asume que cualquier lector que solicite acceso debe poder ingresar primero, cuando el acceso esté disponible. Esto significa que los escritores que quieran acceder al recurso podrían tener que esperar. Por el contrario, el sistema podría asumir que debido a que los escritores necesitan hacer cambios que puedan afectar a los lectores, se les debe dar prioridad en el problema de lectores-escritores. Cuando un lector haya terminado con un recurso, un escritor podría intervenir para hacer un cambio. Esto se aplica no solo a las acciones del usuario, como intentar guardar un documento, sino a los procesos internos dentro de la computadora que mantienen el sistema en funcionamiento.

En conclusión, las problemáticas abordadas, como el problema de la cena de los filósofos y el dilema de lectores-escritores, destacan la complejidad de la sincronización de procesos en sistemas operativos. A través de diversas soluciones, se resalta la importancia de evitar interbloqueos y aplazamientos indefinidos para garantizar un funcionamiento constante y seguro.

El modelo de productor/consumidor, por su parte, ofrece una eficiente organización y sincronización de procesos en sistemas informáticos, destacando la relevancia de la programación en paralelo. La solución al problema del productor-consumidor, mediante la ejecución simultánea y sincronización, resuelve desafíos cruciales, aunque se enfatiza la importancia de una implementación cuidadosa para evitar situaciones de deadlock.

En resumen, tanto el modelo de productor/consumidor como las soluciones propuestas para los problemas mencionados subrayan la necesidad de una sincronización precisa de procesos en sistemas operativos, destacando la importancia de estrategias eficientes y seguras para garantizar un óptimo funcionamiento.

Referencias

desconocido. (desconocido de desconocido de desconocido). *desconocido*. Obtenido de desconocido: https://www3.uji.es/~redondo/so/capitulo3_IS11.pdf

IBM. (24 de 03 de 2023). *IBM*. Obtenido de IBM.com: https://www.ibm.com/docs/es/aix/7.3?topic=models-producerconsumer

lsi.vc.ehu.eus. (desconocido de desconocido de desconocido). *lsi.vc.ehu.eus*. Obtenido de lsi.vc.ehu.eus: https://lsi.vc.ehu.eus/pablogn/docencia/manuales/SO/TemasSOuJaen/CONCURRENCIA/4ProblemasClasicosdeComunicacionentreProcesos.htm

ni.com. (21 de 06 de 2023). *ni.com*. Obtenido de ni.com: https://www.ni.com/es/support/documentation/supplemental/21/producer-consumer-architecture-in-labview0.html

ricardogeek.com. (13 de 03 de 2014). *ricardogeek.com*. Obtenido de ricardogeek.com: https://ricardogeek.com/problemas-clasicos-de-sincronizacion/

spiegato. (desconocido de desconocido de desconocido). *spiegato*. Obtenido de spiegato: https://spiegato.com/es/cual-es-el-problema-de-lectores-escritores

WIKIPEDIA. (13 de 12 de 2022). *WIKIPEDIA*. Obtenido de WIKIPEDIA: https://es.wikipedia.org/wiki/Problema_productor-consumidor

WIKIPEDIA. (22 de 01 de 2024). *WIKIPEDIA*. Obtenido de WIKIPEDIA: https://es.wikipedia.org/wiki/Problema_de_la_cena_de_los_fil%C3%B3sofos#:~:text=El%20problema%20de%20la%20cena,procesos%20en%20un%20sistema%20operativo.