quinto intento: propiedades

Existen otros algoritmos que solamente con operaciones atómicas de `load` y `store` consiguen un acceso con exclusión mutua a secciones críticas:

- algoritmo de Peterson
- algoritmo de Lamport
- algoritmo de Eisenberg–McGuire

Operaciones en la memoria

- Existen instrucciones más potentes (que los simples `load` y `store`) en los microprocesadores actuales para la realización la exclusión mutua más fácil.
- (Casi) todos los procesadores implementan varios tipos de instrucciones atómicas que realizan algún cambio en la memoria al mismo tiempo que devuelve el contenido anterior de la memoria.
- La idea principal es implementar ciclos de *read-modify-write* de forma atómica directamente en memoria compartida.
- El hecho que tienen que actuar finalmente en memoria compartida suelen tener una eficiencia reducida ya que no pueden aprovechar tanto de la jerarquia de memoria (cachés).

TAS

La instrucción `test-and-set` (TAS) implementa

- una comprobación a cero del contenido de una variable en la memoria
- al mismo tiempo que varía su contenido
- en caso que la comprobación se realizó con el resultado verdadero.

TAS

```
Initially:  vi is equal false
            C  is equal true

a: loop
b:   non-critical section
c:   loop
d:     if C equals true                ; atomic
         set C to false and exit
e:   endloop
f:   set vi to true
g:   critical section
h:   set vi to false
i:   set C to true
j: endloop
```

TAS: propiedades

- En caso de un sistema multi-procesador hay que tener cuidado que la operación `test-and-set` esté realizada en la memoria compartida.
- Teniendo solamente una variable para la sincronización de varios procesos el algoritmo arriba no garantiza una espera limitada de todos los procesos participando.
  ¿Por qué?
- ¿Cómo se puede garantizar una espera limitada?
- Para conseguir una espera limitada se implementa un protocolo de paso de tal manera que un proceso saliendo de su sección crítica da de forma explícita paso a un proceso esperando (en caso que tal proceso exista).

CAS

- La instrucción `compare-and-swap` (CAS) es una generalización de la instrucción `test-and-set`.
- La instrucción trabaja con dos variables, les llamamos `C` (de *compare*) y `S` (de *swap*).
- Se intercambia el valor en la memoria por `S` si el valor en la memoria es igual que `C`.
- Se devuelve un booleano con un valor dependiendo si se ha realizado el intercambio o no.
- Es la operación que se usa por ejemplo en los procesadores de Intel y es la base para facilitar la concurrencia en la máquina virtual de Java 1.5 para dicha familia de procesadores.
- Con CAS se pueden realizar los protocolos de entrada y salida. ¿Cómo?

otras posibilidades

Existen otras operaciones hardware para conseguir sincronización entre procesos en la memoria, ejemplos son:

- EXCH (atomic exchange)
- F&A (fetch-and-add)
- DCAS (double compare-and-swap)
- LL/SC (link load / store conditional)
- y muchos más...

Uso de memoria común

- Como hemos visto todos los protocolos necesitan variables con acceso atómico en memoría compartida entre procesos.
- Tal hecho puede resultar en pérdidas de rendimiento, si existe una jerarquía de memoria con diferentes niveles de cachés.
- Muchos compiladores ofrecen acceso directo a las instrucciones de nivel bajo, por ejemplo la gama de compiladores GCC con sus *atomic builtins*, (https://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc/Atomic-Builtins.html)
- Nuevas versiones ya lo hacen diferente (https://gcc.gnu.org/onlinedocs/gcc-8.3.0/gcc/)
- Mirad por ejemplo: https://gcc.gnu.org/onlinedocs/gcc-8.3.0/libstdc++/manual/ y su cuaderno de bitácora que menciona los cambios a lo largo de los años

ABA problem

no se transmite information

- Los protocolos de entrada y salida como implementados hasta ahora no transmiten información de un hilo al otro,
- en el sentido que si un hilo entra en su sección crítica dos veces,
- no puede averiguar si otro hilo ha (o otros hilos han) entrado mientras tanto entre sus dos entradas.
- Este problema se conoce como ABA-problema (la secuencia *primero A, luego B, después A*, no se puede distinguir del simple hecho *solo A*).
- Un ejemplo, donde este problema puede ser relevante es, si se compara solo punteros o referencias para averiguar posibles cambios de estado,
- por ejemplo en acciones sobre listas compartidas (secuencias de borrar e insertar),
- puede ser que los punteros no se modificaron (ya que están protejidos con exclusión mutua) pero el lugar a donde apuntan sí.

problema

Un bloqueo se produce cuando un proceso está esperando algo que nunca se cumple.

Ejemplo:

Cuando dos procesos P_0 y P_1 quieren tener acceso simultaneamente a dos recursos r_0 y r_1, es posible que se produzca un bloqueo de ambos procesos. Si P_0 accede con éxito a r_1 y P_1 accede con éxito a r_0, ambos se quedan atrapados intentando tener acceso al otro recurso.

condiciones necesarias

Se tienen que cumplir cuatro condiciones para que sea posible que se produzca un bloqueo entre procesos:

1. los procesos tienen que compartir recursos con exclusión mutua
2. los procesos quieren acceder a un recurso más mientras ya tienen acceso exclusivo a otro
3. los recursos solo permiten ser usados por menos procesos que lo intentan al mismo tiempo
4. existe una cadena circular entre peticiones de procesos y asignaciónes de recursos

funcionamiento parcial

Un problema adicional con los bloqueos es que es posible que el programa siga funcionando correctamente según la definición, es decir, el resultado obtenido es el resultado deseado, pero algunos de sus procesos están bloqueados durante la ejecución (es decir, se produjo solamente un bloqueo parcial).

técnicas de actuar

Existen algunas técnicas que se pueden usar para que no se produzcan bloqueos:

- Detectar y actuar
- Evitar
- Prevenir

detectar y actuar

Se implementa un proceso adicional que vigila si los demás forman una cadena circular.
Más preciso, se define el grafo de asignación de recursos:

- Los procesos y los recursos representan los nodos de un grafo.
- Se añade cada vez una arista entre un nodo tipo recurso y un nodo tipo proceso cuando el proceso ha obtenido acceso exclusivo al recurso.
- Se añade cada vez una arista entre un nodo tipo recurso y un nodo tipo proceso cuando el proceso está pidiendo acceso exclusivo al recurso.
- Se eliminan las aristas entre proceso y recurso y al revés cuando el proceso ya no usa el recurso.

actuación

Cuando se detecta en el grafo resultante un ciclo, es decir, cuando ya no forma un grafo acíclico, se ha producido una posible situación de un bloqueo.
Se puede reaccionar de dos maneras si se ha encontrado un ciclo:

- No se da permiso al último proceso de obtener el recurso.
- Sí, se da permiso, pero una vez detectado el ciclo se aborta todos o algunos de los procesos involucrados.

desventaja

Sin embargo, las técnicas pueden dar como resultado que el programa no avance, incluso, el programa se puede quedar atrapado haciendo trabajo inútil: crear situaciones de bloqueo y abortar procesos continuamente.

evitar

El sistema no da permiso de acceso a recursos si es posible que el proceso se bloquee en el futuro.
Un método es el algoritmo del banquero (Dijkstra) que es un algoritmo centralizado y por eso posiblemente no muy practicable en muchas situaciones.
Se garantiza que todos los procesos actuan de la siguiente manera en dos fases:

1. primero se obtiene todos los cerrojos necesarios para realizar una tarea, eso se realiza solamente si se puede obtener todos a la vez,
2. después se realiza la tarea durante la cual posiblemente se liberan recursos que no son necesarias.

prevenir

Se puede prevenir el bloqueo siempre y cuando se consiga que alguna de las condiciones necesarias para la existencia de un bloqueo no se produzca.

1. los procesos tienen que compartir recursos con exclusión mutua
2. los procesos quieren acceder a un recurso más mientras ya tienen acceso exclusivo a otro
3. los recursos no permiten ser usados por más de un proceso al mismo tiempo
4. existe una cadena circular entre peticiones de procesos y alocación de recursos

prevenir exclusión mutua

los procesos tienen que compartir recursos con exclusión mutua:

- No se de a un proceso directamente acceso exclusivo al recurso, si no se usa otro proceso que realiza el trabajo de todos los demás manejando una cola de tareas (por ejemplo, un demonio para imprimir con su cola de documentos por imprimir).

prevenir accesos consecutivos

los procesos quieren acceder a un recurso más mientras ya tienen acceso exclusivo a otro:

- Se exige que un proceso pida todos los recursos que va a utilizar al comienzo de su trabajo

prevenir uso único

los recursos no permiten ser usados por más de un proceso al mismo tiempo:

- Se permite que un proceso aborte a otro proceso con el fin de obtener acceso exclusivo al recurso. Hay que tener cuidado de no caer en *livelock*
- (Separar lectores y escritores alivia este problema también.)

prevenir ciclos

existe una cadena circular entre peticiones de procesos y alocación de recursos:

- Se ordenan los recursos línealmente y se fuerza a los procesos que accedan a los recursos en el orden impuesto. Así es imposible que se produzca un ciclo.

Bloqueo en Java

Un ejemplo de un bloqueo en Java muestra el siguiente trozo de código, incluso si se asume que un hilo ya está durmiendo. ¿Por qué?

```
hilo0:

synchronized(A) {
  ...
  synchronized(B) {
    ...
    A.notify();
    B.wait();
  }
}
```

```
hilo1:

synchronized(B) {
  ...
  synchronized(A) {
    ...
    B.notify();
    A.wait();
  }
}
```