Finales resueltos sistemas Operativos

Maximiliano Toledo 2018

Final 1er llamado Diciembre 2017

1) Hilos (Threads):

a) Definición y diferencia con el enfoque de proceso clásico.

El hilo es la unidad básica de utilización de la CPU.

Para establecer una diferencia entre el concepto de proceso y de hilo, debemos distinguir entre la unidad de propiedad de recursos (el proceso) y la unidad de ejecución (el hilo).

Un hilo es la unidad de trabajo que se puede someter a ejecución. Se ejecuta secuencialmente y es interrumpible para que el procesador pase a otro hilo. Permite que el programador pueda ejercer la modularidad.

Un proceso puede verse como una colección de uno o más hilos.

Cuando un sistema operativo soporta varios hilos en ejecución dentro del mismo proceso se dice que es *multihilo*.

Diferencias con el concepto tradicional

Estructura

En un ambiente multihilo, un proceso es la unidad de protección y asignación de recursos.

Así, cada hilo dentro de un proceso contará con un estado de ejecución, un contexto de procesador, una pila en modo usuario y otra en modo supervisor, almacenamiento para variables locales y acceso a memoria y recursos del proceso (archivos abiertos, señales, además de la parte de código y datos) que compartirá con el resto de los hilos.

Todos juntos, los hilos de un proceso constituyen una tarea.

La estructura de un hilo está constituida por:

- un program counter
- un conjunto de registros
- un espacio de stack

Un proceso tradicional (*heavyweight process o monohilo*) es una tarea formada por un solo hilo. Cuenta con un PCB (process control block), espacio de direcciones de usuario y 2 pilas.

En un proceso multihilo, además de PCB, espacio de direcciones de usuario y 2 pilas, cada hilo tiene su propio TCB (thread control block) y 2 pilas.

Los hilos de un proceso comparten estado y recursos del sistema, residiendo en el mismo espacio de direcciones y accediendo a los mismos datos.

Context switch

Cuando un proceso tradicional gana CPU, el sistema operativo debe hacer un cambio de contexto (context switch) para hacer el correspondiente salvado del ambiente relacionado con el proceso anterior y cargar el ambiente del nuevo. Esta actividad la realiza el sistema operativo, en modo supervisor, y es la consecuencia de algún system call emitido por el proceso saliente o interrupción.

Entre los hilos que comparten una tarea, el cambio entre uno y otro es más simple pues el context switch es sólo a nivel del conjunto de registros y carga no hay cambios con respecto a espacios de direcciones.

Podemos resumir que en el switching de los hilos no se exige la participación del sistema operativo e interrupciones al kernel.

La creación de un hilo

La creación de un proceso involucra una operación tipo fork, creando un nuevo espacio de direcciones, PCB, PC, etc. El nuevo proceso es creado y controlado a través de system calls que exigen el mismo tratamiento que una interrupción.

La creación de un hilo involucra sólo la creación de la estructura detallada más arriba: un conjunto de registros, un PC y un espacio para stack, requiriendo un gasto mínimo de procesamiento.

Operación

Los hilos pueden ser una manera eficiente de permitir que un servidor pueda atender varios requerimientos.

En un ambiente monoprocesador, un hilo comparte la CPU con los otros hilos, y sólo uno está activo en un momento dado. Se ejecuta secuencialmente y tiene su propio PC y stack.

Puede tener hilos hijos y bloquearse a la espera de un system call. Si se bloquea un hilo, puede ejecutarse otro hilo.

Si se hace swapping del proceso, será acompañado por sus hilos. Al terminar un proceso, terminan todos sus hilos.

Los hilos de un proceso comparten estado y recursos del sistema, residiendo en el mismo espacio de direcciones y accediendo a los mismos datos.

Context switch

Cuando un proceso tradicional gana CPU, el sistema operativo debe hacer un cambio de contexto (context switch) para hacer el correspondiente salvado del ambiente relacionado con el proceso anterior y cargar el ambiente del nuevo. Esta actividad la realiza el sistema operativo, en modo supervisor, y es la consecuencia de algún system call emitido por el proceso saliente o interrupción.

Entre los hilos que comparten una tarea, el cambio entre uno y otro es más simple pues el context switch es sólo a nivel del conjunto de registros y carga no hay cambios con respecto a espacios de direcciones.

Podemos resumir que en el switching de los hilos no se exige la participación del sistema operativo e interrupciones al kernel.

La creación de un hilo

La creación de un proceso involucra una operación tipo fork, creando un nuevo espacio de direcciones, PCB, PC, etc. El nuevo proceso es creado y controlado a través de system calls que exigen el mismo tratamiento que una interrupción.

La creación de un hilo involucra sólo la creación de la estructura detallada más arriba: un conjunto de registros, un PC y un espacio para stack, requiriendo un gasto mínimo de procesamiento.

Operación

Los hilos pueden ser una manera eficiente de permitir que un servidor pueda atender varios requerimientos.

En un ambiente monoprocesador, un hilo comparte la CPU con los otros hilos, y sólo uno está activo en un momento dado. Se ejecuta secuencialmente y tiene su propio PC y stack.

Puede tener hilos hijos y bloquearse a la espera de un system call. Si se bloquea un hilo, puede ejecutarse otro hilo.

Si se hace swapping del proceso, será acompañado por sus hilos. Al terminar un proceso, terminan todos sus hilos.

Estados de un hilo

Un hilo puede estar en estado de ready, blocked, running ó exiting, como los procesos tradicionales.

Hay cuatro operaciones que provocan los cambios de estado: la creación, el bloqueo, el desbloqueo y la terminación.

Al crearse un proceso, se crea un hilo (que puede crear otros hilos, cada uno con su nuevo contexto y pilas) y pasará al estado de listo.

Cuando el hilo debe esperar por un evento se bloquea. Según las diferentes implementaciones, puede ocurrir que quede todo el proceso bloqueado, o solo ese hilo.

En este último caso se le puede dar el control a otro hilo de ese proceso.

Cuando se produce el evento por el que estaba esperando, el hilo bloqueado se desbloquea, pasando al estado de listo.

Al terminar, el hilo libera su contexto y sus pilas.

Beneficios del uso de los hilos

_ Es más rápido crear un nuevo hilo que un nuevo proceso.

_ Un hilo, al terminar, sólo libera el espacio asignado, siendo este procedimiento mucho más rápido que la terminación de un proceso.

_ Los hilos, además, mejoran la comunicación entre procesos. Mientras los procesos tradicionales necesitan la intervención del núcleo del SO para protección y administración del mecanismo de comunicación, la comunicación entre hilos del mismo proceso se puede hacer sin la intervención del núcleo, pues comparten memoria y archivos.

Tipos de hilos

Hay **hilos a nivel de usuario** (ULT, user level thread) e **hilos a nivel de núcleo** (KLT, kernel level thread).

En los ULT, la administración de los hilos lo hace la aplicación sin intervención del kernel. Es más: el kernel ni se entera de su existencia. El kernel no ve el hilo: ve un proceso haciendo un requerimiento. LA creación de los hilos y operaciones se hacen en a nivel de usuario y por lo tanto son más rápidos de crear y utilizar.

En estos casos se trabaja con una biblioteca de hilos que son funciones para implementar ULT invocadas desde la aplicación (crear y destruir hilos,

intercambio de mensajes y datos entre hilos, planificación de ejecución, salvado y restauración de contexto).

Todo esto se realiza dentro del mismo proceso, en su espacio de direcciones.

Ejemplos de bibliotecas de Hilos son POSIX Pthreads, Mach C-Threads y Solaris Threads.

Las ventajas de los ULT son:

_ No interviene el núcleo en el intercambio de hilos (las estructuras de datos están todas dentro del espacio de direcciones del proceso).

_ Se puede planificar la ejecución de los hilos dentro de cada proceso como se prefiera (prioridades, RR). O sea: cada proceso administra sus hilos como le convenga al programador, aunque use una planificación distinta de la que usa el kernel para los procesos.

_ Los ULT se pueden ejecutar en cualquier SO pues no dependen de él.

Los ULT tienen también, desventajas:

_ Los hilos no son independientes entre ellos, pues pueden acceder a cualquier dirección dentro de la tarea, y un hilo puede leer o escribir en el stack de otro***.***

No hay protección entre hilos.

_ La suspensión de un hilo, como puede ser por una llamada al sistema, bloquea a todo el proceso, o sea, todos los hilos de ese proceso.

_ En un ambiente multiprocesador, cada procesador ejecuta un proceso. Por lo tanto no voy a poder aprovechar la potencialidad de estos ambientes y los hilos de un proceso usarán un único procesador (no se puede implementar una concurrencia entre los hilos de un mismo proceso ejecutándose paralelamente entre los distintos procesadores).

En los KLT, el trabajo de gestión de hilos lo hace el núcleo. La aplicación gestiona el hilo a través de una API. Existen un conjunto de system calls similar a la de los procesos específicas para hilos). Esta metodología se usa en W2000, Linux, OS/2.

El kernel mantiene información del proceso en general y de cada hilo del proceso en particular. La planificación la hace el kernel en base a los hilos.

La ventaja es que en un ambiente multiprocesador, los hilos de un mismo proceso pueden estar ejecutándose concurrentemente en los distintos procesadores (la unidad de asignación del procesador ya no es el proceso).

La desventaja de los KLT es que el pase del control entre hilos de un mismo proceso necesita un cambio de modo. Esto hace prever que la creación y administración de los KLTs es más lenta que los ULTs.

Modelos multihilos

Como algunos sistemas proveen tanto KLT como ULT, tenemos diferentes modelos multihilo: varios a uno, uno a uno y varios a varios

Modelo varios a uno

Este modelo relaciona varios hilos a nivel de usuario con un hilo de kernel.

Si bien toda la operación entre hilos se hace en el espacio de usuario, si uno de ellos se bloquea, se bloquea todo el proceso. Por ejemplo, ante un system call que realice un hilo, se bloqueará todo el proceso.

Al kernel acceden de a un hilo, así que este modelo no es útil en ambiente multiprocesador.

Modelo uno a uno

En este modelo cada hilo de usuario se relaciona con uno de kernel.

En este caso, si un hilo se bloquea, puede ejecutarse otro hilo del mismo proceso. Lo que si garantiza este modelo es la concurrencia pues en un proceso formado por varios hilos, en un ambiente multiprocesador puede correr cada uno en un procesador distinto.

No obstante se debe tener en cuenta que cada vez que necesito un hilo de usuario se debe crear uno de kernel con el gasto que eso significa.

Modelo de varios a varios

Este modelo combina o “multiplexa” muchos hilos a nivel de usuario con un número menor o igual de hilos a nivel de kernel.

Se pueden crear tantos hilos de usuario como sea necesario y los hilos a nivel de kernel correspondientes se ejecutan en paralelo si es un ambiente multiprocesador.

En el modelo uno a uno si bien permite mayor concurrencia, hay que ser cuidadoso con no crear demasiados hilos, pues en algunos casos este número se limita.

Si es un modelo varios a varios se crearan un número de hilos kernel que se permita y se multiplexan los hilos de usuario en esos hilos de kernel.

b)En un esquema de ULT puro (modelo N:1) indique que funcionalidades mínimas debería incluir la biblioteca de hilos para operar.

Nivel Usuario

Denominado ULT (user-level threads), su implementación está a cargo totalmente de la aplicación, a través de una biblioteca de hilos. El sistema operativo administra el proceso como una unidad, y la aplicación migra de un subproceso a otro con su propia administración.

Así como un programador maneja el cambio de una rutina a otra, y esto es transparente al sistema operativo, así puede manejar el cambio de un subproceso a otro, si tiene implementada esta opción en su lenguaje. Las ventajas son evidentes, cuando un proceso perdería uso de la CPU por esperar un evento, aun cuando su quantum de tiempo no esté agotado, bajo este modelo, simplemente migra a otro subproceso. Toda aplicación inicia con un solo hilo de ejecución, y durante su ejecución, va creando y finalizando distintos hilos de acuerdo a sus necesidades. Toda esta actividad se desarrolla en el espacio de usuario y dentro de un solo proceso, el que ve el sistema operativo.

Dentro de las ventajas de uso de los ULT sobre los KLT podemos mencionar:

- Se eliminan las pérdidas de tiempo por Cambio de contexto ya que todo sucede dentro del mismo proceso.
- El tipo de planificación, como prioridad o FCFS, puede variar de aplicación a aplicación, de acuerdo a las necesidades de cada una.
- Pueden ser ejecutados en cualquier sistema operativo.

En las desventajas, podemos nombrar:

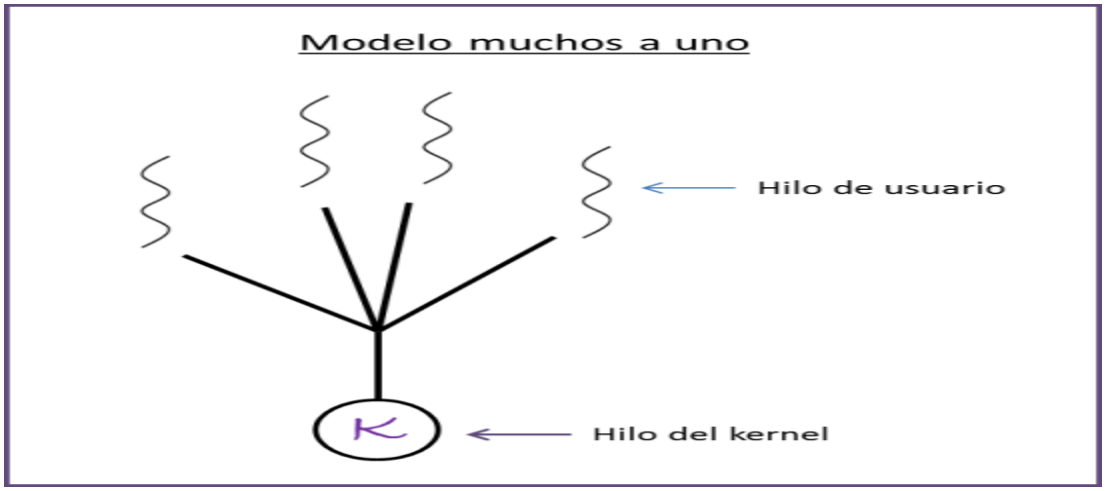
- Cuando un ULT realiza una llamada al sistema operativo, detiene todos los ULT, ya que el proceso pasa a espera, o Hold, o sleep.
- No puede hacer uso del paralelismo en caso de existir más de una CPU.

c) Indique que ventajas presenta el uso del modelo 1:1 sobre el modelo N:1

Modelos muchos a uno

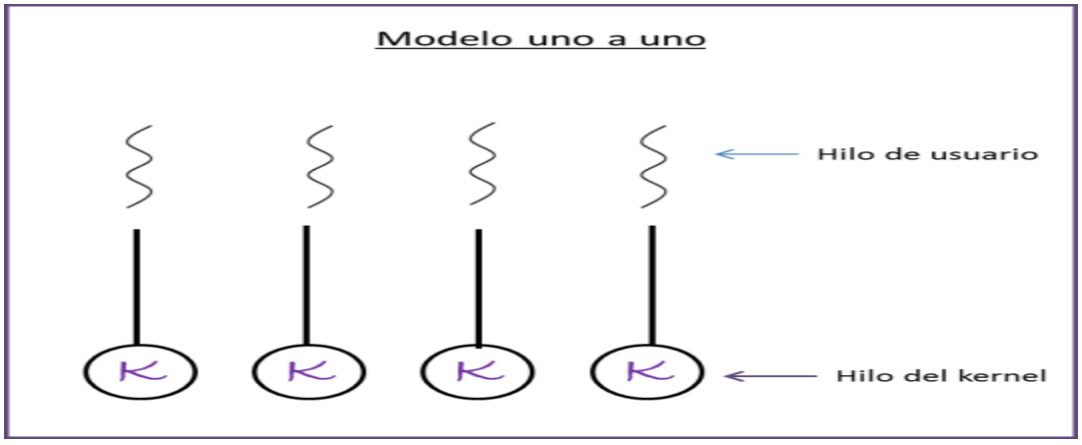
Asigna múltiples hilos del nivel usuario a un hilo del kernel. La gestión de hilos se hace mediante la biblioteca de hilos en el espacio de usuario, por lo que

resulta eficiente, pero el proceso completo se bloquea si un hilo realiza una llamada bloqueante al sistema. Dado que sólo un hilo puede acceder al kernel cada vez, no podrán ejecutarse varios hilos en paralelo sobre múltiples procesadores.



Modelos uno a uno

Asigna cada hilo de usuario a un hilo del kernel. Proporciona una mayor concurrencia que el modelo muchos a uno, permitiendo que se ejecute otro hilo mientras un hilo hace una llamada bloqueante al sistema; también permite que se ejecuten múltiples hilos en paralelo sobre varios procesadores. El único inconveniente de este modelo es que crear un hilo de usuario requiere crear el correspondiente hilo del kernel. Dado que la carga de trabajo administrativa para la creación de hilos del kernel puede repercutir en el rendimiento de una aplicación, la mayoría de las implementaciones de este modelo restringen el nro. de hilos soportados por el sistema. Linux junto con la flia Windows (Windows 95, 98, NT, 200 y XP) implementan el modelo uno a uno.

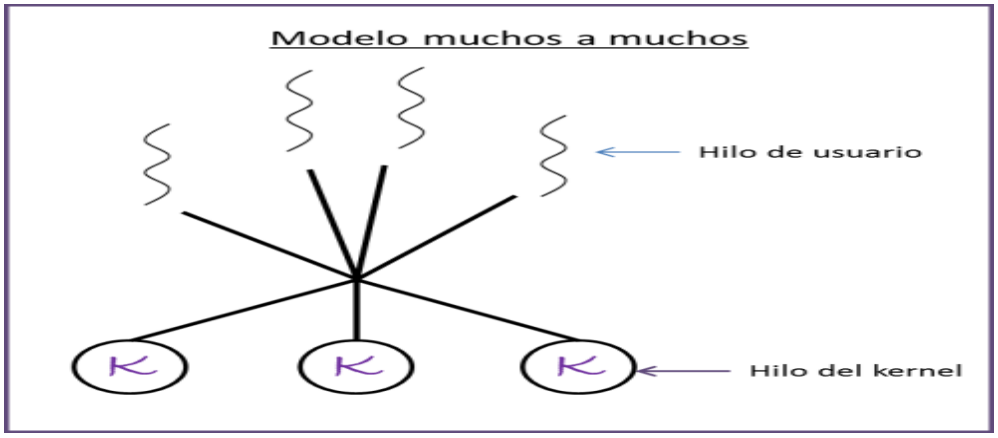


Modelo muchos a muchos

Multiplexa muchos hilos de usuario sobre un nro. menor o igual de hilos del kernel. El nro. de hilos del kernel puede ser específico de una determinada aplicación o de una determinada máquina (pueden asignarse más hilos del

kernel a una aplicación en un sistema multiprocesador que en uno de un solo procesa-dor). Mientras que el modelo muchos a uno permite al desarrollador crear tan-tos hilos de usuario como desee, no se consigue una concurrencia real, ya que el kernel sólo puede planificar la ejecución de un hilo cada vez. El modelo uno a uno permite una mayor concurrencia, pero el desarrollador debe tener cuidado de no crear demasiados hilos dentro de una aplicación. El modelo muchos a muchos no sufre ninguno de estos inconvenientes. Los desarrolladores pueden crear tantos hilos de usuario como sean necesarios y los correspondientes hilos del kernel pueden ejecutarse en paralelo en un multiprocesador. Así mismo, cuando un hilo realiza una llamada bloqueante al sistema, el kernel puede planificar otro hilo para su ejecución.

Una popular variación del modelo de muchos a muchos, Multiplexa muchos hilos del nivel de usuario sobre un nro. menor o igual de hilos del kernel, pero también permite acoplar un hilo de usuario a un hilo del kernel.



2) Virtualización:

a) Definición. Diferencias con Emulación

Es una capa de abstracción sobre el hw, para obtener una mejor utilización de los recursos y flexibilidad. Permite que haya múltiples máquinas virtuales (MV) o entornos virtuales (EV), con distintos (o iguales) sistemas operativos corriendo aisladamente.

Cada MV tiene su propio conjunto de hardware virtual (RAM, CPU, NIC, etc.) sobre el cual se carga el SO “guest”.

El SO “guest” ve un conjunto consistente de hw, no el hw real.

MV están encapsuladas en archivos.

Fácil de almacenar, copiar.

Sistemas completos (aplicaciones ya configuradas, so, hw virtual) pueden moverse de un servidor a otro, rápidamente.

Componentes: software host y guest

Hay un software host (que simula) y un software guess (lo que se quiere simular).

Guest puede ser un sistema operativo completo.

Características de una MV

Equivalencia / Fidelidad: un programa ejecutándose sobre un VMM debería que tener un comportamiento idéntico al que tendría ejecutándose directamente sobre el hardware subyacente.

Control de recursos / Seguridad: El VMM tiene que controlar completamente y en todo momento el conjunto de recursos virtualizados que proporciona a cada guest.

Eficiencia / Performance: Una fracción estadísticamente dominante de instrucciones tienen que ser ejecutadas sin la intervención del VMM, o en otras palabras, directamente por el hardware.

Emulación y virtualización

Algunas máquinas virtuales en realidad son emuladas.

Emulación: provee toda la funcionalidad del procesador deseado a través de software (ej: QUEMU).

Se puede emular un procesador sobre otro tipo de procesador

Tiende a ser lenta.

Virtualización: se trata de particionar un procesador físico en distintos contextos, donde cada uno de ellos corre sobre el mismo procesador.

Es más rápida que la emulación.

Emulación ¡= Virtualización

El primer concepto permite toda la funcionalidad del procesador deseado a través de software, pero es lento.

La virtualización trata de particionar un procesador físico en distintos contextos, donde cada uno de ellos corre sobre el mismo procesador, es más rápido.

En la virtualización, el guest emitirá una instrucción privilegiada que no debe ser ignorada.

Las instrucciones privilegiadas generan interrupción.

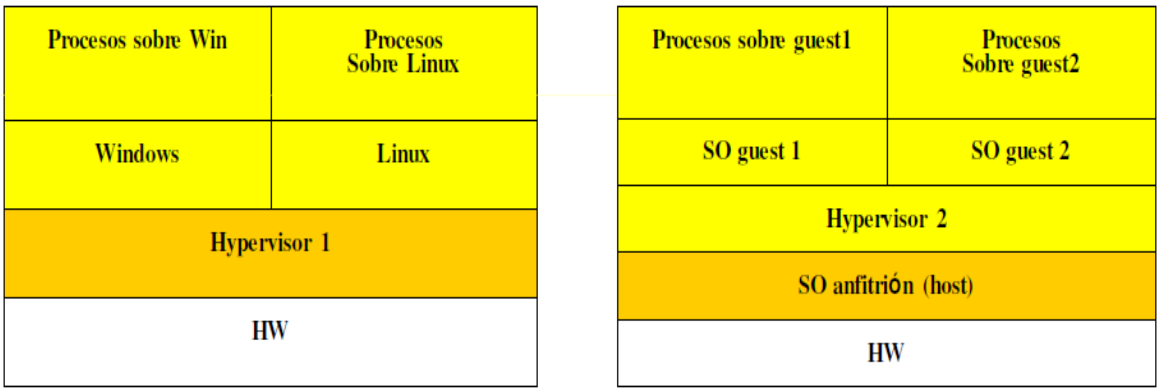
Las sensibles solo se pueden ejecutar en modo kernel.

b)Diferencias entre Hypervisor de Tipo I y de Tipo II

Hypervisor: Plataforma de virtualización que permite múltiples SOs corriendo en un host al mismo tiempo. Interactúa con el hardware y realiza la multiprogramación

Tipo I: Se ejecuta en modo kernel. Cada máquina virtual se ejecuta como un proceso de usuario en modo usuario, modo kernel virtual y modo usuario virtual, cuando la máquina virtual ejecuta una instrucción sensible se produce una trap que procesa el hypervisor. Se ejecuta sobre el hardware

Tipo II: Se ejecuta como un programa de usuario sobre un SO host, interpreta un conjunto de instrucciones de máquina. El SO host es quien se ejecuta sobre el hardware. Las instrucciones sensibles se sustituyen por llamadas a procedimiento que emulan las instrucciones.



3) Deadlocks. Definición. Diferencias entre Evitar y Prevenir los Deadlocks.

Definición

Un conjunto de procesos están en deadlock cuando cada uno de ellos está esperando por un recurso que está siendo usado por otro proceso del mismo conjunto.

Un estado de Deadlock puede involucrar recursos de diferentes tipos.

Ejemplos:

Un proceso A pide un scanner.

Un proceso B pide una grabadora de CD.

El proceso A pide ahora la grabadora de CD

El proceso B quiere el scanner.

En una BD:

Un proceso A bloquea el registro R1,

Un proceso B bloquea el registro R2.

Luego cada proceso trata de bloquear el registro que está usando el otro.

Quieren el Recurso que tiene otro Proceso

Clases de recursos: Conjunto de instancias de un recurso. El ciclo es solicitud, uso y liberación

Para que exista deadlock se tienen que dar 4 situaciones:

- **Exclusión mutua**: Uso exclusivo del recurso
- **Retención y espera**: El proceso tiene el recurso hasta obtener todos los que le faltan
- **No apropiación**: No se pueden sacar los recursos a un proceso
- **Espera circular**

Existen tres métodos para el tratamiento de deadlock:

- Usar protocolo que asegure que nunca se entrara en deadlock
    1. **Prevenir** que ocurra el Deadlock
    2. **Evitar** que ocurra el Deadlock
- Permitir el deadlock y luego recuperar
- Ignorar el problema y esperar que nunca ocurra un deadlock

1. Prevention (PREVENIR la formación del interbloqueo): que por lo menos 1 de las 4 condiciones no pueda mantenerse. Se imponen restricciones en la forma en que los procesos REQUIEREN los recursos.

2. Avoidance (EVITAR la formación del interbloqueo): asignar cuidadosamente los recursos, manteniendo información actualizada sobre requerimiento y uso de recursos. (NO ES BUENA SOLUCIÓN)

Prevenir: Por lo menos una de las cuatro condiciones no podrá mantenerse. Se logra poniendo restricciones en la forma en que los procesos requieren los recursos.

Evitar: Asignar cuidadosamente los recursos, manteniendo información actualizada sobre requerimiento y uso de recursos.

Prevención:

 Condición de exclusión mutua: Si no se asigna recursos de manera exclusiva, no habrá deadlock. Pero deberá tener en cuenta que hay recursos compartibles y no compartibles. Mantener la exclusión mutua para los no compartibles.

 Condición de retención y espera: Antes de comenzar la ejecución se debe asegurar que el proceso tenga todos los recursos requeridos y la única manera de poder requerir recursos es cuando no tiene ninguno. Las desventajas son que hay baja utilización de recursos y posibilidad de inanición de alguno de los procesos

 Condición de no apropiación: Tenemos un recurso que no puede asignarse a un proceso y queda en wait y se liberan el resto de los recursos. Se desbloqueará cuando el recurso pedido más todos los que tenía estén disponibles. Otra opción: Si P1 solicita R1 y R1 lo tiene P2 que se encuentra bloqueado a la espera de R2 entonces R1 se lo quita a R2 y se le asigna a P1, Sino se bloquea P1

 Condición de espera circular: Se define un ordenamiento de los recursos. Luego un proceso puede requerir recursos en un orden numérico ascendente

Síntesis

Condición	Método
Exclusión mutua	Evitar recurso se asigne exclusivo
Contención y espera	Solicitar todos los recursos al principio
No apropiativa	Quitar los recursos
Espera circular	Ordenar los recursos en forma numérica

Evitación: Hay mucho procesamiento lógico, dado que el SO tiene un detalle de cómo se requieren los recursos, en qué momento se requieren, la demanda máxima, etc.

Requiere que el SO tenga información ANTES

El SO cuenta con información sobre el uso de los recursos

-cómo se requieren

-en qué momento del sistema son requeridos

- la demanda máxima de recursos, etc.

Desventajas:

-puede producir una baja utilización de los recursos

-puede producir una baja performance del sistema

Sobre información de los recursos

Conocer la secuencia de solicitud, uso y liberación de cada recurso requerido por el proceso.

Requiere que cada proceso declare el número máximo de recursos de cada tipo que pueda necesitar.

El algoritmo de prevención de interbloqueo examina dinámicamente el estado de asignación de recursos para asegurar que nunca puede haber una condición de espera circular. (posibles consecuencias)

El estado de asignación de recursos se define por el número de recursos disponibles y asignados y las demandas máximas de los procesos.

Estado sano o seguro

Un sistema está en un estado seguro si el SO puede asignar recursos a cada proceso de un conjunto de alguna manera, evitando el deadlock.

Cuando un **proceso solicita** un **recurso disponible**, el **sistema DEBE DECIDIR** si la **asignación** inmediata **deja** el sistema en un **estado seguro**.

Debe haber una secuencia “cadena segura” de **TODOS** procesos <P0, P1 , , Pn >, que puedan ejecutarse con todos los recursos disponibles sin que haya deadlock.

Importante!

Un estado seguro garantiza que NO hay deadlock.

SI hay deadlock, estoy en estado inseguro.

En estado inseguro hay posibilidad que SI haya deadlock.

No todo estado inseguro es deadlock

EVITAR/AVOIDANCE: trata de nunca entrar a estado inseguro. Garantiza lo seguro entonces no hay deadlock.

Algoritmos para <u>evitar</u> el deadlock

1. Instancia única de un tipo de recurso

    Algoritmo que determina el estado seguro de un sistema.

    Utilizar un grafo de asignación de recursos. Encuentro secuencia

2. Múltiples instancias de un tipo de recurso

    Utilice el algoritmo del banquero

    Algoritmo teórico

1. Grafo asignación de recursos

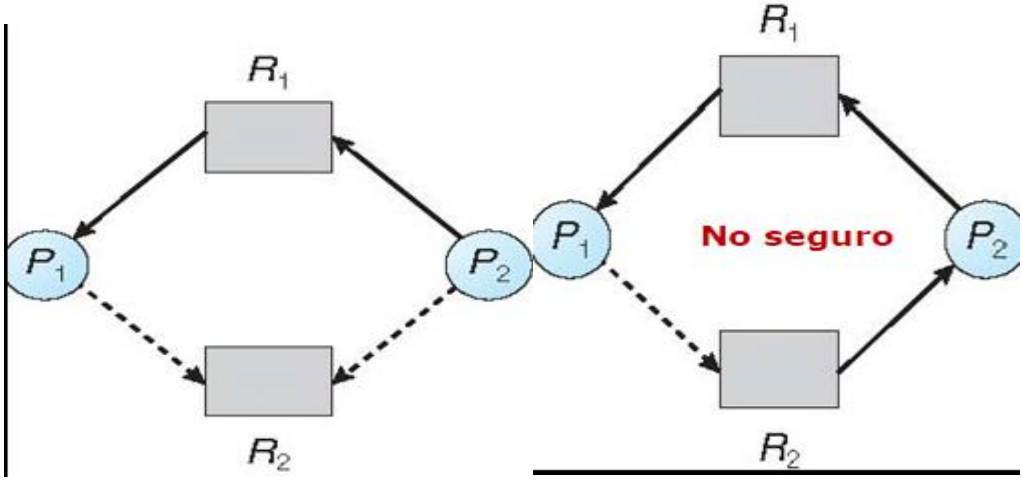
Los recursos deben ser reclamados ANTES en el sistema. Se arma un grafo

• La línea de reclamación Pi -> Rj indica que el proceso Pj puede solicitar el recurso Rj; (línea punteada)

• La línea de reclamación/punteada se convierte en línea de solicitud cuando un proceso solicita un recurso (línea continua)

• línea de solicitud convertida a línea de asignación cuando el recurso se asigna al proceso Rj -> Pi

• Cuando un recurso es liberado por un proceso, la línea de asignación se vuelve a convertir en una línea de reclamación/punteada.



2. Algoritmo del banquero: (Dijkstra) Se aplica para sistemas con múltiples instancias de cada recurso. Los procesos declaran el número máximo de instancias de cada recurso que necesitaría. Ese número no puede exceder el total de instancias de recursos de ese tipo en el sistema. El SO decidirá en que momento asignarlos, garantizando un estado seguro.

Cuando un proceso solicita un recurso puede tener que esperar

Cuando un proceso obtiene TODOS sus recursos, debe devolverlos en una cantidad de tiempo finita

Es difícil para el SO saber todo esto antes. Debería realizar simulaciones.

4)Sincronización:

a) Sección crítica: Definición. Cite ejemplos de secciones críticas que se pueden encontrar en un SO.

Se denomina **sección crítica** o región crítica, a la porción de código de un programa de ordenador en la que se accede a un recurso compartido (estructura de datos o dispositivo o variable) que no debe ser accedido por más de un proceso o hilo en ejecución. La sección crítica por lo general termina en un tiempo determinado y el hilo, proceso o tarea sólo tendrá que esperar un período determinado de tiempo para entrar. Se necesita un mecanismo de sincronización en la entrada y salida de la sección crítica para asegurar la utilización en exclusiva del recurso, por ejemplo un semáforo, monitores, el algoritmo de Dekker y Peterson, los candados.

Ejemplo1: Se tiene un Sistema Operativo que debe asignar un identificador de proceso (PID) a dos procesos en un sistema multicore. Cuando el SO realiza esta acción en dos procesadores de forma simultánea sin ningún tipo de control, se pueden producir errores, ya que se puede asignar el mismo PID a dos procesos distintos. Este problema se debe a que constituyen una sección crítica que debe ejecutarse en forma atómica, es decir, de forma completa e indivisible y ningún otro proceso podrá ejecutar dicho código mientras el primero no haya acabado su sección.

Ejemplo2: **Problema de los lectores-escritores**

El problema trata de dos perfiles de procesos que interactúan simultáneamente, es decir, imaginando una Base de Datos donde se tiene un proceso que desea leer los datos y otro proceso que desea actualizar los datos, a los cuales les llamaremos lector y escritor. Si los lectores acceden simultáneamente no habrá problema alguna, pero el problema surge cuando quiere interactuar con la base de datos el lector y escritor al mismo tiempo, generando así el problema.

Para asegurar que no ocurra este tipo de problemas, se debe asegurar que los procesos escritores tengan acceso exclusivo a la base de datos.

Ejemplo3: **Problema de los 5 filósofos**

El problema de los filósofos es una representación sencilla de la necesidad de repartir varios recursos entre varios procesos de una forma que no se produzcan interbloqueos ni bloqueos indefinidos.

Se consideran cinco filósofos en una mesa redonda, donde existe un solo plato de arroz al medio de la mesa, luego entre cada par de filósofos hay 1 solo palillo, teniendo un total de 5 palillos en la mesa (Al igual que la cantidad de filósofos). El problema surge cuando un filósofo quiere comenzar a comer, el cual toma los 2 palillos entre él y los filósofos de la izquierda y derecha. Los demás filósofos no podrán comer, ya que, un filósofo no tomará un palillo que está ocupado por otro. Ergo, no pueden existir dos filósofos contiguos comiendo arroz.

b) Comparar Spinlocks y Elevar nivel del procesador(también conocido como elevar el nivel de interrupciones) para implementar Sección Critica. Tenga en cuenta: Funcionamiento, implementación, ventajas y desventajas de cada una.

Semáforos

Es una herramienta para sincronizar procesos.

Para superar la complejidad de la utilización de las instrucciones mencionadas en el inciso anterior, se puedo optar por utilizar una herramienta de sincronización denominada semáforo. Un **semáforo** S es una variable de dominio entero (no es una variable entera) a la que solo se accede mediante dos operaciones atómicas estándar: *wait()* y *signal()*.

Todas las modificaciones del valor entero del semáforo en las operaciones wait() y signal() deben ejecutarse de forma indivisible. Es decir, cuando un proceso modifica el valor del semáforo, ningún otro proceso puede modificar simultá-neamente el valor de dicho semáforo.

Utilización

Los SO diferencian a menudo entre semáforos contadores y semáforos binarios. El valor de un **semáforo contador** puede variar en un dominio no restringido, mientras que el valor de un **semáforo binario** solo puede ser 0 o 1. Estos últimos, en algunos sistemas, se los conocen como cerrojos mútex, ya que son cerrojos que proporcionan exclusión mutua. Los semáforos contadores se pueden usar para controlar el acceso a un determinado recurso formado por un número finito de instancias. El semáforo se inicializa con el número de recursos disponibles. Cada proceso que desee usar un recurso ejecuta una operación wait() en el semáforo (decrementando la cuenta). Cuando un proceso libera un recurso, ejecuta una operación signal () (incrementando la cuenta). A su vez, se pueden utilizar semáforos para resolver diversos problemas de sincronización.

Implementación

Spinlock*: Toma el valor entero y tiene tres instrucciones: wait, signal (ambas limitan cada sección critica) e init, la cual se ejecuta una sola vez. Usamos una variable porque puede haber varias instancias del mismo recurso; por ejemplo tener cuatro buffers. Problemas (Dijstra): se perdía mucho tiempo, esperando nada; debido a que está en estado de espera hasta que se le agote el quantum.*

Se “bloquea” el uso del bus multiprocesador para proteger la ubicación de memoria que se está accediendo.

SPINLOCK: mecanismo que asocia una primitiva de locking a la estructura de datos que se quiere proteger. Los procesos deben “adquirir” el spinlock.

Se puede implementar por test-and-set

- Cuando está tratando de adquirir el spinlock se eleva nivel de procesador.
- Problemas:
    - si el código de spinlock provoca interrupciones de menor nivel (por ejemplo, si se invoca un handler de page fault).
    - Procesos Ready de Mayor Prioridad

-Problema con el Spin-lock (bloqueo de giro)

-Se desperdicia tiempo de la CPU que hace la petición

-Genera Carga en la memoria y el BUS

La principal desventaja de la definición de semáforo, es que requiere una **espera activa**. Mientras un proceso está en su sección crítica, cualquier proceso que intente entrar en su sección crítica debe ejecutar continuamente un bucle en el código de entrada. Este bucle continuo plantea un problema en un sistema real de multiprogramación, donde una sola CPU se comparte entre muchos procesos. La espera activa desperdicia ciclos de CPU que algunos otros procesos podrían usar de forma productiva. Este tipo de semáforo se denomina **cerrojo mediante bucle sin fin (spinlock),** ya que el proceso permanece en un bucle sin fin en espera de adquirir el cerrojo. Estos cerrojos tienen una ventaja y es que no requieren ningún cambio de contexto cuando un proceso tiene que esperar para adquirir un cerrojo. Los cambios de contexto pueden llevar un tiempo considerable, y teniendo en cuenta que los cerrojos mediante bucle sin fin son útiles, se emplean a menudo en los sistemas multiprocesador, donde un hilo puede ejecutar un bucle sobre un procesador mientras otro hilo ejecuta su sección crítica.

Para salvar la necesidad de la espera activa, podemos modificar la definición de wait () y signal (). Cuando un proceso ejecuta la operación wait() y determina que el valor del semáforo no es positivo, tiene que esperar. Sin embargo, en lugar de entrar en una espera activa, el proceso puede bloquearse a sí mismo.

Deshabilitar interrupciones

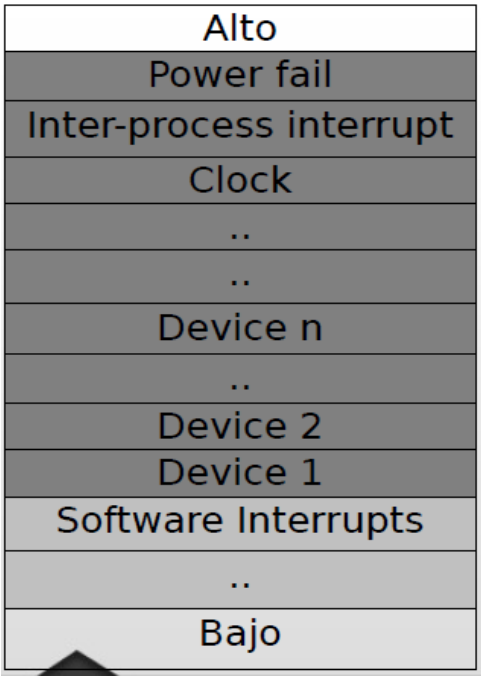
- Conocida también Como “elevar el nivel de procesador”

- Un proceso se ejecuta hasta que se invoca una System Call o es interrumpido

- No pueden ocurrir interrupciones de reloj

- Multiprocesadores: Deshabilitar las interrupciones en un procesador no garantiza Exclusión Mutua

- Peligroso su uso por parte de procesos de usuario




```
While (true)
  {
   /* deshabilitar interrupciones */;
   /* sección crítica */;
   /* habilitar interrupciones */;
   /* resto */;

   }
```

Ejemplo en el Kernel:

- Deshabilitar las interrupciones mientras se está trabajando con la lista de procesos.

En entornos uniprocesador basta con deshabilitar las interrupciones mientras se ejecuta la sección crítica.

En sistemas multiprocesadores, se deshabilitan las interrupciones de una CPU, pero otra podría generarlas…

Exclusión Mutua con espera ocupada

DESHABILITANDO INTERRUPCIONES: En un sistema con un solo procesador, la solución más simple es hacer que cada proceso deshabilite todas las interrupciones justo después de entrar a su región crítica y las rehabilite justo después de salir. ◦ Con las interrupciones deshabilitadas, no pueden ocurrir interrupciones de reloj. Después de todo, la CPU solo se conmuta de un proceso a otro como resultado de una interrupción del reloj o de otro tipo, y con las interrupciones desactivadas la CPU no se conmutara a otro proceso.

Por ende, una vez que un proceso ha deshabilitado las interrupciones, puede examinar y actualizar la memoria compartida sin temor de que algún otro proceso intervenga. ◦ La posibilidad de lograr la exclusión mutua al deshabilitar las interrupciones (incluso dentro del kernel) está disminuyendo día con día debido al creciente número de chips multinucleo que se encuentran hasta en las Pcs de bajo rendimiento. Ya es común que haya dos núcleos, cuatro y dentro de poco habrá 8, 16.

En un multinucleo(es decir, sistema con multiprocesadores) al deshabilitar las interrupciones de una CPU no se evita que las demás CPUs interfieran con las operaciones que la primera CPU está realizando, requiriendo sistemas más sofisticados.

En particular, si no se procesan las interrupciones no se producirán cambios de contexto que es precisamente lo que da lugar al problema de la sección crítica.

Inhibir_Interrupciones ();

Sección critica

Habilitar Interrupciones ();

Esta solución es adecuada únicamente cuando la duración de la sección crítica es pequeña, ya que la suspensión del servicio de interrupciones durante largos periodos de tiempo puede degradar el funcionamiento del sistema. Debido a lo peligroso que resulta que los usuarios tengan acceso al sistema de interrupciones, los sistemas operativos controlan y ocultan al usuario estas facilidades. Por consiguiente, este método de sincronización será solo aplicable a nivel de núcleo del sistema operativo.

VARIABLES DE CANDADO:

Es una solución de software. Considere tener una sola variable compartida (de candado), que al principio es 0. Cuando un proceso desea entrar a su región crítica primero evalúa el candado. Si este candado es 0, el proceso lo fija en 1 y entra a la región critica. Si el candado ya es 1 solo espera hasta que el

candado se haga 0. Por ende, un 0 significa que ningún proceso está en su región crítica y un 1 significa que algún proceso está en su región crítica.

c) Dada las dos técnicas indicadas en el punto anterior, compare su utilización en ambientes monoprocesador y multiprocesador.

Contestada en la respuesta anterior.

Final 8-02-2018

3) Indique y compare las diferentes formas de implementar una matriz de acceso, ¿Para qué se utiliza el permiso “switch” en una celda de una matriz? Agrego Protección y Seguridad, Objetos y Dominios

Protección!= Seguridad

Protección

Mecanismos específicos del SO para resguardar la información dentro de una computadora, para controlar el acceso de los procesos (o usuarios) a los recursos existentes.

Seguridad:

Medida de la confianza en que se pueden preservar la integridad de un sistema y sus datos.

La seguridad utiliza distintos mecanismos de protección con el fin de proteger y garantizar ante: **Amenazas**, Confidencialidad de los datos (Intercepción /Modificación), Integridad de los Datos (Modificación), Disponibilidad (Interrupción)

Intrusos

Perdida Accidental de Datos: Accidentes Naturales**,** Errores de HW o SW**,** Errores humanos.

Objetos y Dominios

Un sistema informático es una colección de procesos y objetos

Objetos: de HW (CPU, Memoria, etc.) o de SW (archivos, programas, semáforos).

Cada objeto debe tener un identificador único que permita referenciarlo

Los procesos pueden realizar un conjunto finito de operaciones sobre los objetos.

Un **dominio** es un conjunto de pares (objeto, derecho).

Cada par especifica un objeto y un subconjunto de operaciones que se pueden realizar con él.

Un **derecho** (right) significa autorización para efectuar esas operaciones.

Por ejemplo: el dominio D contiene la pareja (archivo A, {read,write}).

Un proceso que se ejecuta dentro del dominio D puede leer y grabar el archivo A.

Un dominio: Puede ser un usuario y define qué puede hacer ese usuario y que no.

Puede ser un proceso y el conjunto de objetos a los que podrá acceder dependerá de la identidad del proceso

Puede ser un procedimiento y definirá el conjunto de variables a las que puede acceder (variables locales, globales, etc…)

Un proceso se ejecuta dentro de un dominio. La asociación proceso-dominio puede ser:

- Estática: el conjunto de objetos disponibles para un proceso es fijo durante la vida del proceso.

- Dinámica: cambio de dominio durante la ejecución del proceso

Matriz de acceso

Controla la pertenencia de objetos a dominios y sus derechos.

Las filas representan dominios.

Las columnas representan objetos.

Cada elemento access(i,j) representa el conjunto de operaciones (derechos) que un proceso puede invocar en un objeto Oj dentro del dominio Di.

Implementa las políticas de protección/seguridad de un sistema.

El modelo de protección del sistema se puede ver en forma abstracta como una matriz, la *matriz de acceso.*

Una matriz de acceso es una representación abstracta del concepto de dominio de protección.

Este modelo fue propuesto por Lampson como una descripción generalizada de mecanismos de protección en sistemas operativos. Es el modelo más utilizado, del que existen numerosas variaciones, especialmente en su implementación.

Los elementos básicos del modelo son los siguientes:

• ***Sujeto:*** Una entidad capaz de acceder a los objetos. En general, el concepto de sujeto es equiparable con el de proceso. Cualquier usuario o aplicación consigue acceder en realidad a un objeto por medio de un proceso que representa al usuario o a la aplicación.

• ***Objeto:*** Cualquier cosa cuyo acceso debe controlarse. *Como ejemplo se incluyen los archivos, partes de archivos, programas y segmentos de memoria.*

• ***Derecho de acceso:*** la manera en que un sujeto accede a un objeto. *Como ejemplo está Leer, Escribir y Ejecutar.*

El modelo considera un conjunto de recursos, denominados objetos, cuyo acceso debe ser controlado y un conjunto de sujetos que acceden a dichos objetos. Existe también un conjunto de permisos de acceso que especifica los diferentes permisos que los sujetos pueden tener sobre los objetos (normalmente lectura, escritura, etc., aunque pueden ser diferentes, en general, dependiendo de las operaciones que puedan realizarse con el objeto).

Se trata de especificar para cada pareja (sujeto, objeto), los permisos de acceso que el sujeto tiene sobre el objeto. Esto se representa mediante una matriz de acceso M que enfrenta todos los sujetos con todos los objetos. En cada celda M [i, j] se indican los permisos de acceso concretos que tiene el sujeto i sobre el objeto j.

Ejemplo de matriz

Objeto / Dominio	File1	File2	File3	Printer
D1	Read	Read		Print
D2		Read	execute	print
D3	Read/write	Read/write		

Operación: switch

Un proceso puede cambiar de un dominio a otro.

Para poder cambiar de un dominio a otro se debe habilitar la operación switch sobre un objeto (dominio).

La matriz en sí es un objeto que posee tributos, de esta manera se define si sobre un dominio se realiza asignación estática o no:

> La conmutación del dominio Di al dominio Dj estará permitida si se encuentra definida en el switch access(i,j)

Dominio y matriz son objetos sobre los cuales se definen accesos.

Operación: Switch

Objeto / Dominio	File1	File2	File3	Printer	D1	D2	D3
D1	Read	Read		Print		switch	
D2		Read	execute	print			
D3	Read/write	Read/write			switch		switch

Operación copy

Es un derecho que se asocia a un elemento access(i,j) de la matriz.

Indica que un proceso ejecutándose en ese dominio puede copiar los derechos de acceso de ese objeto dentro de su columna

Se denota con *.

Aplicación de operación copy

Objeto / Dominio	File1	File2	File3	Printer
D1	Read	Read		Print
D2		Read *	execute	print
D3	Read/write			
Objeto / Dominio	File1	File2	File3	Printer
D1	Read	Read		Print
D2		Read *	execute	print
D3	Read/write	Read		

Operación owner

Permite agregar nuevos derechos y borrar ya existentes.

Si matriz(i,j) incluye el derecho de owner entonces un proceso ejecutándose en el dominio Di puede agregar y borrar cualquier entrada en la columna j.

Operación control

Indica que pueden modificarse y borrarse derechos dentro de una fila.

La operación control es aplicable sólo a dominios.

Si matriz (i,j) incluye el derecho de control, entonces un proceso ejecutándose en el dominio Di puede remover cualquier derecho de acceso dentro de la fila j.

Resumiendo:

Copy, owner y control son operaciones que se utilizan para controlar cambios al contenido de la matriz de acceso.

Switch y Control: son aplicables sólo a dominios.

Copy y owner: puede modificar derechos dentro de una columna.

Control: puede modificar derechos dentro de una fila.

Implementación de la matriz de acceso

El principal problema es que puede tener muchos elementos vacíos

Se debe optimizar el acceso para que sea rápido.

Alternativas:

Tabla Global: la más simple. Consiste en conjunto de tuplas

<dominio,objeto, derechos-acceso>.

Lista de Acceso para los objetos.

Tabla Global:

Es la más sencilla de implementar

Consiste en conjunto de tuplas <dominio,objeto, derechos-acceso>

Cada vez que se ejecuta una operación **M** sobre un objeto **Oj** sobre el dominio **Di**, se analiza la tabla y se verifica si se encuentra una terna < **Di , Oj , M** >

Si se encuentra se permite la operación

Si no se encuentra se deniega

Su principal desventaja es que el tamaño de la tabla hace que no se pueda almacenar toda en memoria.

Lista de Acceso para los objetos (ACL)

Cada columna de la matriz se puede ver como una lista de acceso a un objeto.

Se descartan elementos vacíos

Para cada objeto, hay una lista de pares ordenados

<dominio, derechos>

Se pueden definir listas + conjunto de derechos x defecto.

Cuando se intenta realizar una operación **M** sobre un objeto **Oj** en el dominio **Di**, se busca en la lista en el objeto **Oj** una entrada <**Di, Rk**>, donde **M** pertenece al conjunto **Rk**.

Comparación

La tabla global es una matriz dispersa, es ineficiente para su almacenamiento.

ACL → Cuando se accede a un objeto es fácil determinar si hay o no permiso para usarlo.

Capacidades → Las ACL están distribuidas, es difícil saber cuáles son los derechos de acceso para un proceso, cosa que si se puede hacer con la lista de capacidades.

Los sistemas reales suelen usar una mezcla de todos.

Ej. UNIX: Se abre un fichero, se verifica en la ACL si tiene permiso o no. Si lo tiene, se consigue un descriptor de fichero, es decir una capacidad que será lo que se use a partir de entonces.

Final 14/09/2015

2) Pasaje de Mensajes: Comunicación Directa e Indirecta

Comunicación entre procesos: IPC

- Es un mecanismo para comunicar y sincronizar procesos.
- Consta de:
  - Sistema de mensajes
  - Memoria Compartida (shared memory)
  - Semáforos

IPC – Mensajes

El paso de mensajes proporciona un mecanismo que permite a los procesos comunicarse y sincronizar sus acciones sin compartir el mismo espacio de direcciones, y es especialmente útil en un entorno distribuido, en el que los procesos que se comunican pueden residir en diferentes computadoras conectadas en red.

- Provee dos operaciones
  - send y receive.
- Se establece un link de comunicación entre los procesos que se quieren comunicar.
- Existen varios métodos para implementar lógicamente un enlace y las operaciones de envío y recepción:

  -Comunicación directa o indirecta.

  -Comunicación síncrona o asíncrona.

  -Almacenamiento en búfer explícito o automático.

La operación send puede ser por copia o por referencia; los mensajes de medida fija o variable.

Comunicación directa: cada proceso que quiere comunicarse con otro, explícitamente nombre quien recibe o manda la comunicación.

Send (P, mensaje) Envía un mensaje al proceso P

Receive (Q, mensaje) Recibe un mensaje desde el proceso Q.

Un enlace de comunicaciones, según este esquema, tiene las siguientes propiedades:

-Los enlaces se establecen de forma automática entre cada par de procesos que quieran comunicarse. Los procesos solo tienen que conoces la identidad del otro para comunicarse.

-Cada enlace se asocia con exactamente dos procesos.

-Entre cada para de procesos existe exactamente un enlace.

Este esquema presenta **simetría** en lo que se refiere al direccionamiento, es decir, tanto el proceso transmisor como el proceso receptor deben nombrar al otro para comunicarse. Existe una variante de este esquema que emplea **asimetría** en el direccionamiento. En este caso, solo el transmisor nombra al receptor; el receptor no tiene que nombrar al transmisor. En este caso, las primitivas se definen:

Send(P, mensaje)- envía un mensaje al proceso P

Receive(id , mensaje)- recibe un mensaje de cualquier proceso; a la variable id se le asigna el nombre del proceso con el que se ha llevado a cabo la comunicación.

La desventaja de estados dos esquemas (simétrico y asimétrico) es la limitada modularidad de las definiciones de procesos resultantes. Cambiar el identificador de un proceso puede requerir que se modifiquen todas las restantes definiciones de procesos. Deben localizarse todas las referencias al identificador antiguo, para poder sustituirlas por el nuevo identificador.

Con el **modelo de comunicación indirecta,** los mensajes se envían y reciben en **buzones de correo o puertos**. Un buzón de correo puede verse de forma abstracta como un objeto en el que los procesos pueden colocar mensajes y del que pueden eliminarse mensajes. Cada buzón de correo tiene asociada una identificación univoca. En este esquema, un proceso puede comunicarse con otros procesos a través de una serie de buzones de correo diferentes. Sim

embargo, dos procesos solo se pueden comunicar si tiene un buzón de correo compartido.

En este esquema, un enlace de comunicaciones tiene las siguientes propiedades:

-Puede establecerse un enlace entre un par de procesos solo si ambos tienen un buzón de correo compartido.

-Un enlace puede asociarse con más de dos procesos.

-Entre cada para de procesos en comunicación, puede haber una serie de enlaces diferentes, correspondiendo cada enlace a un buzón de correo.

Comunicación Indirecta: usa un mailbox o port.

- Un mailbox puede verse como un objeto donde se ponen y sacan mensajes.
- Cada mailbox tiene una identificación única

Send (A, mensaje) Envía un mensaje al mailbox A

Receive (A, mensaje) Recibe un mensaje desde el mailbox A

En un esquema de **comunicación indirecta** el sistema operativo debe proveer un mecanismo para que un proceso pueda:

- Crear un nuevo mailbox
- Compartir un mailbox
- Enviar y recibir mensajes a través del mailbox
- Destruir un mailbox

Capacidad del Link: ¿Cuántos mensajes puede mantener el link?

- Cero: no puede haber mensajes esperando. Es lo que se llama Rendezvous: el emisor debe esperar que el receptor reciba el mensaje para poder mandar otro. Hay sincronismo.
- Capacidad limitada: la cola tiene una longitud finita
- Capacidad ilimitada: tiene una longitud “infinita”. El emisor nunca espera.

Final Junio 2015

2) Comunicación y Sincronización.

a) Dos procesos intentan acceder a la misma estructura del kernel para modificarla. Por ej. La Ready Queue. Si se cuenta con un único procesador ¿Qué herramienta utilizaría para garantizar la exclusión mutua?¿Por qué?¿Cómo se implementaría si se contara con con multiprocesadores y una organización SMP?

Posibles soluciones al problema de Exclusión mutua:

Soluciones por Hardware:

- Deshabilitar interrupciones
    - o Cuando un proceso entra en una sección crítica desactiva las interrupciones para que no pueda ser interrumpido.
    - o Las reactiva cuando sale de la sección crítica.
    - o En sistemas multiprocesador desactivar las interrupciones de un procesador no garantiza exclusión mutua.
- Instrucción TSL/Test and set lock (Es una instrucción para lograr exclusión mutua en sistemas multiprocesadores, evalúa el estado de una variable lock hasta que esté libre, luego la marca como ocupada y permite el acceso a la sección crítica).
- Instrucción xchg/swap (Es una instrucción que intercambia dos registros sin necesidad de usar una variable temporal, convirtiendo la acción en atómica y evitando así el problema de la sección crítica. Al igual que Test and set, evalúa y asigna la variable lock evitando problemas de interferencia).

SMP – Multiprocesadores Simétricos

Soluciona el inconveniente de saturación de una única CPU

Única copia del SO en memoria y cualquier CPU puede ejecutarlo

Equilibro entre procesos y memoria, ya que solo hay un único conjunto de tablas del SO

No hay cuello de botella, ya que no hay una CPU master

Problemas:

-Dos o más CPUs ejecutando código del SO en un mismo instante de tiempo

-Dos CPUs seleccionando el mismo proceso para ejecutar, o seleccionan la misma página de la memoria libre!

Posibles soluciones a los problemas planteados:

1. Utilizar "locks" para las estructuras del SO:

Considerar a todo el SO como una gran sección critica. Cualquier CPU puede ejecutar el SO, pero solo una a la vez.

Se comportaría como el modelo maestro-esclavo

Es un modelo poco utilizado, debido a la mala performance que provee

2. Lock por estructura(s):

Existen varias secciones criticas independientes cada una protegida por su propio mutex

Mejora el rendimiento

Dificultad para determinar cada sección critica

Ciertas estructuras pueden pertenecer a más de una sección crítica, lo cual ante bloqueos podría generar Deadlocks

Es el esquema que generalmente se utiliza

Es necesario que las CPU de un multiprocesador se encuentren sincronizadas (acceso a regiones críticas, estructuras, etc.).

En entornos uniprocesador basta con deshabilitar las interrupciones mientras se ejecuta la sección crítica

En sistemas multiprocesadores, se deshabilitan las interrupciones de una CPU, pero otra podría generarlas…

Surge la necesidad de contar con un protocolo de mutex apropiado para garantizar la exclusión mutua.

Una posibilidad para garantizar la exclusión mutua es el uso de **TSL (Probar y establecer bloqueo):**

Lee la palabra de memoria y la almacena en un registro. Al mismo tiempo escribe un 1 en la memoria para hacer el lock (2 accesos al BUS). Cuando termina libera (escribe 0)

El problema surge en entornos multiprocesadores:

Ambas CPU obtuvieron un 0 de la instrucción TSL, por lo que ambas tienen acceso a la sección crítica

La solución al problema anterior, es que en multiprocesadores TSL se bloquee el acceso al BUS

Se necesita soporte de hardware para poder implementarlo

Problema con el Spin-lock (bloqueo de giro)

Se desperdicia tiempo de la CPU que hace la petición

Genera Carga en la memoria y el BUS

No es lo más eficiente…

Surge una nueva solución que es el **uso de cache** (para evitar el bus), pero también genera problemas:

Al leer la palabra en la cache, generalmente se realizan modificaciones

Como se modifica, se deben invalidar todas las copias de las otras caches (trashing), lo que causa que se deba escribir el valor a la memoria y las otras CPUs lo relean.

Esto genera mayor uso del BUS

Además al tener un lockeo establecido por una CPU, las otras están continuamente consultando por la liberación

Otras soluciones

Cada CPU tiene su propia variable de lockeo en cache

La CPU que no puede obtener el bloqueo se agrega a una lista y espera en su propio lock

Agregar “delays” entre cada intento de TSL

b) En el modelo Threading N:1 se necesita un semáforo para sincronizar los threads de un proceso ¿El semáforo se debe implementar en el SO(Kernel) o en la librería de hilos? Justifique.

Semáforos

- Es una herramienta de sincronización
- Sirve para solucionar el problema de la sección critica.
- Sirve para solucionar problemas de sincronización.

Funcionamiento

- Es una variable entera
- Inicializada en un valor no negativo
- Dos operaciones:
- wait (down, p)
    - Decrementa el valor. El proceso no puede continuar ante un valor negativo, se bloquea.
- signal (up, v)
    - Incrementa el valor. Desbloqueo de un proceso
- Operaciones atómicas
    - Cuando un proceso modifica el valor del semáforo, otros procesos no pueden modificarlo simultáneamente.

Mutex

- Podemos ver el mutex como una versión simplificada del semáforo.
- Son buenos SOLO para garantizar exclusión mutua.
- Variable que está en estado abierto (desbloqueado) o cerrado (bloqueado).
- Se pueden implementar en espacio de usuario.
- Útil para sincronización de ULTs.

Procedimientos del mutex

・Mutex_lock y mutex_unlock

・Cuando un hilo/proceso necesita acceder a la SC invoca a mutex_lock

・Si el mutex está abierto, puede entrar a la SC.

・Si está cerrado, se bloquea hasta que el hilo que está usando libere la SC invocando a mutex_unlock.

・Si hay varios hilos bloqueados por el mutex, se selecciona uno y se le da el mutex.

3) Deadlocks.

b) Modelo "detectar y recuperar "¿Cómo funciona?

Método: Detección y Recuperación

Si no puedo asegurar que el deadlock no ocurrirá necesito usar este esquema (el NUNCA)

Permitir que el sistema entre en estado de deadlock.(Ceder libremente los recursos).

Mantener información de recursos asignados y pedidos

Algoritmo que examine si ocurrió un deadlock (lo detecte)

Algoritmo para recuperación del deadlock. (Intentar romper alguna condición)

Detección

Con recursos con 1 sola instancia

análisis del grafo de asignación

Con recursos de varias instancias

Algoritmo de Shoshani y Coffman

Algoritmo del Banquero

Instancia única de cada tipo de recurso

Mantener un grafo wait-for (Grafo de espera)

• Nodes are processes (quita los recursos)

• Pi ® Pj if Pi is waiting for Pj libere recurso Rq

Invoca periódicamente un algoritmo que busca un ciclo en el gráfico.

Si hay un **ciclo**, existe un **bloqueo**

Un algoritmo para detectar un ciclo en una gráfica requiere un orden de n2 operaciones, donde n es el número de vértices de Procesos en el gráfico.

Múltiples instancias de cada tipo de recursos

• Disponible: Un vector de longitud m indica el número de recursos disponibles de cada tipo

• Asignación: Una matriz n x m define el número de recursos de cada tipo asignado actualmente a cada proceso

• Solicitud: Una matriz n x m indica la petición actual de cada proceso. Si Request [i][j] = k , entonces el proceso Pi está solicitando k más instancias del tipo de recurso Rj. (Proceso se encuentra en estado de Espera)

Cuando Uso del Algoritmo de Detección

Cuando, y con qué **frecuencia**, invocar depende de:

• ¿Con qué **frecuencia** es probable que ocurra un deadlock?

• ¿Cuántos procesos tendrán que ser revertidos (rolled back)?

> Un aspecto **importante** es la **frecuencia** con la que se **debe ejecutar** el **algoritmo** de **detección de interbloqueos**, que suele ser un parámetro del sistema.

• Una posibilidad extrema es comprobar el estado cada vez que se solicita un recurso y éste no puede ser asignado. Consume mucha CPU
• Si el algoritmo de detección se invoca arbitrariamente, puede haber muchos ciclos en el gráfico de recursos y, por lo tanto, no podríamos decir cuál de los muchos procesos bloqueados "causó" el bloqueo.
• Otra alternativa es activar el algoritmo ocasionalmente, a intervalos regulares o cuando uno o más procesos queden bloqueados durante un tiempo sospechosamente largo.
• Por ejemplo 1 hora, o cuando la utilización del Procesador desciende su actividad por debajo de un determinado porcentaje de utilización, pues un 'deadlock' eventualmente disminuye el uso de procesos en actividad.

Recuperación frente al deadlock
Cuando un algoritmo de detección determina que existe un interbloqueo, existen varias alternativas para tratar de eliminarlo:
_Caso 1: El Operador lo puede resolver manualmente. (Informar al operador del SO)

_Caso 2: Esperar que el sistema se recupere automáticamente del deadlock. El Sistema rompe el deadlock:

- Abortar 1 o más Procesos para romper ciclo
- Expropiar recursos a 1 o más Procesos del ciclo

Cómo Recuperarse frente al deadlock

_Violar la exclusión mutua y asignar el recurso a varios procesos
_Cancelar los procesos suficientes para romper la espera circular
_Desalojar recursos de los procesos en deadlock.

Recuperación frente al deadlock

Para eliminar el deadlock matando procesos pueden usarse 2 métodos:
_ **Matar todos** los procesos en estado de deadlock. Simple pero a un muy alto costo.
_ **Matar de a un proceso por vez** hasta eliminar el ciclo. Este método requiere considerable overhead ya que por cada proceso que vamos eliminando se debe re-ejecutar el **Algoritmo de Detección** para verificar si el deadlock efectivamente desapareció o no.

TERMINACIÓN DE PROCESOS

_Puede no ser fácil terminar un proceso

- (p.e. si se encuentra actualizando un fichero).

_Cuando se utiliza el método incremental, se presenta un nuevo problema de política o decisión: selección de la víctima
_Se debe seleccionar aquel proceso cuya terminación represente el coste mínimo para el sistema.

Recuperación: Criterios para elegir proceso "víctima"
¿En qué orden debemos optar por abortar/terminar?

_Prioridad más baja.
_Menor cantidad de tiempo de CPU hasta el momento.
_Mayor tiempo restante estimado para terminar.
_Menor cantidad de recursos asignados hasta ahora.
_Hay recursos del proceso que deben completarse?
_¿Cuántos procesos tendrán que ser terminados?
_¿El proceso es interactivo o batch? Puede volver atrás?
Ideal: elegir un proceso que se pueda volver a ejecutar sin problemas (ej. una compilación).

EXPROPIACIÓNDE RECURSOS

Quitar sucesivamente los recursos de los procesos que los tienen y asignarlos a otros que los solicitan hasta conseguir romper el interbloqueo.

Hay que considerar tres aspectos:
1. **Apropiación/Selección de Víctima** - minimizar el costo
2. **Rollback** - regresar a algún estado seguro, reiniciar el proceso para ese estado
3**. Inanición** - el mismo proceso siempre se puede escoger como víctima. Asegurar que se elige un número finito de veces, Incluir el número de retrocesos como factor de coste.

Recuperación Por apropiación

Selección de Víctima

_A qué proceso le saco los recursos?
_A cuáles recursos?
_A qué proceso se lo asigno?

• Elegir la de mínimo costo

Cómo?
_Puede haber intervención “manual” del operador.
_Se puede aplicar según la naturaleza del recurso.

Retroceso (Rollback)

_Volver hacia una instancia segura del proceso que le sacamos los recursos.
_Luego relanzar el proceso
_Establecer puntos de comprobación (check points)
_Información asociada a esos puntos:
_Imagen de la memoria
_Recursos asignados en ese momento
_No sobreescribir check points anteriores.

Pasos en recuperación con rollback

1. Detectar el interbloqueo
2. Detectar recursos que se solicitan
3. Detectar qué procesos tienen esos recursos
4. Volver atrás antes de la adquisición del recurso (check points anteriores)
5. Asignación del recurso a otro proceso

Inanición

1. el mismo proceso siempre se puede escoger como víctima. (ej. Si se asignan prioridades)

Solución:

1. Asegurar que se elige un número finito de veces
2. Incluir el número de retrocesos como factor de coste.

Resumen de Detección y Recuperación

• La Detección y Recuperación de interbloqueos proporciona un mayor grado potencial de concurrencia que las técnicas de Prevención o de Evitación.
• Además, hay sobrecarga en tiempo de ejecución de la Detección
• El precio a pagar es la sobrecarga debida a la recuperación una vez que se han detectado los interbloqueos y también una reducción en el aprovechamiento de los recursos del sistema debido a aquellos procesos que son reiniciados o rollback.
• La recuperación de interbloqueos puede ser atractiva en sistemas con una baja probabilidad de interbloqueos. En cambio, en sistemas con elevada carga, se sabe que la concesión sin restricciones de peticiones de recursos puede conducir a frecuentes interbloqueos.

TEORIA DEL LIBRO

Trata a los deadlock como interbloqueos

DETECCIÓN Y RECUPERACIÓN
DE UN INTERBLOQUEO

Una segunda técnica es la detección y recuperación. Cuando se utiliza esta técnica, el sistema no trata de evitar los interbloqueos. En vez de ello, intenta detectarlos cuando ocurran y luego realiza cierta acción para recuperarse después del hecho. En esta sección analizaremos algunas de las formas en que se pueden detectar los interbloqueos, y ciertas maneras en que se puede llevar a cabo una recuperación de los mismos.

Detección de interbloqueos con un recurso de cada tipo

Vamos a empezar con el caso más simple: sólo existe un recurso de cada tipo. Dicho sistema podría tener un escáner, un grabador de CD, un trazador (plotter) y una unidad de cinta, pero no más que un recurso de cada clase. En otras palabras, estamos excluyendo los sistemas con dos impresoras por el momento. Más adelante lidiaremos con ellos, usando un método distinto.

Para un sistema así, podemos construir un gráfico de recursos del tipo ilustrado en la figura 6-3.

Si este gráfico contiene uno o más ciclos, existe un interbloqueo. Cualquier proceso que forme parte de un ciclo está en interbloqueo. Si no existen ciclos, el sistema no está en interbloqueo.

Como ejemplo de un sistema más complejo que los que hemos analizado hasta ahora, considere un sistema con siete procesos, *A* a *G*, y seis recursos, *R* a *W*. El estado de cuáles recursos están contenidos por algún proceso y cuáles están siendo solicitados es el siguiente:

1. El proceso *A* contiene a *R* y quiere a *S*.
2. El proceso *B* no contiene ningún recurso pero quiere a *T*.
3. El proceso *C* no contiene ningún recurso pero quiere a *S*.
4. El proceso *D* contiene a *U* y quiere a *S* y a *T*.
5. El proceso *E* contiene a *T* y quiere a *V*.
6. El proceso *F* contiene a *W* y quiere a *S*.
7. El proceso *G* contiene a *V* y quiere a *U*.

La pregunta es: “¿Está este sistema en interbloqueo y, de ser así, cuáles procesos están involucrados?”.

Para responder a esta pregunta podemos construir el gráfico de recursos de la figura 6-5(a). Este gráfico contiene un ciclo, que se puede ver mediante una inspección visual. El ciclo se muestra en la figura 6-5(b). De este ciclo podemos ver que los procesos *D*, *E* y *G* están en interbloqueo. Los procesos *A*, *C* y *F* no están en interbloqueo debido a que *S* puede asignarse a cualquiera de ellos, que a su vez termina y lo devuelve. Después los otros dos pueden tomarlo en turno y completarse también (observe que para hacer este ejemplo más interesante, hemos permitido que algunos procesos, como *D*, pidan dos recursos a la vez).

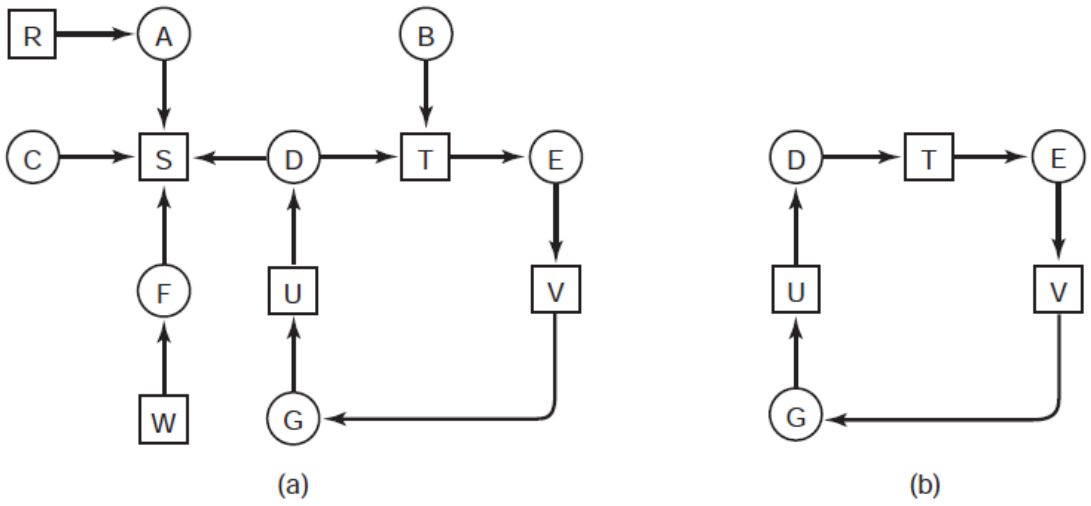


Figura 6-5. (a) Un gráfico de recursos. (b) Un ciclo extraído de (a).

Aunque es relativamente fácil distinguir los procesos en interbloqueo a simple vista a partir de un sencillo gráfico, para usar este método en sistemas reales necesitamos un algoritmo formal para la detección de interbloqueos. Hay muchos algoritmos conocidos para detectar ciclos en gráficos dirigidos. A continuación veremos un algoritmo simple que inspecciona un gráfico y termina al haber encontrado un ciclo, o cuando ha demostrado que no existe ninguno. Utiliza cierta estructura dinámica de datos: *L*, una lista de nodos, así como la lista de arcos. Durante el algoritmo, los arcos se marcarán para indicar que ya han sido inspeccionados, para evitar repetir inspecciones.

El algoritmo opera al llevar a cabo los siguientes pasos, según lo especificado:

1. Para cada nodo *N* en el gráfico, realizar los siguientes cinco pasos con *N* como el nodo inicial.
2. Inicializar *L* con la lista vacía y designar todos los arcos como desmarcados.
3. Agregar el nodo actual al final de *L* y comprobar si el nodo ahora aparece dos veces en *L*.

Si lo hace, el gráfico contiene un ciclo (listado en *L*) y el algoritmo termina.

4. Del nodo dado, ver si hay arcos salientes desmarcados. De ser así, ir al paso 5; en caso contrario, ir al paso 6.
5. Elegir un arco saliente desmarcado al azar y marcarlo. Después seguirlo hasta el nuevo nodo actual e ir al paso 3.
6. Si este nodo es el inicial, el gráfico no contiene ciclos y el algoritmo termina.

En caso contrario, ahora hemos llegado a un punto muerto. Eliminarlo y regresar al nodo anterior; es decir, el que estaba justo antes de éste, hacerlo el nodo actual e ir al paso 3.

Lo que hace este algoritmo es tomar cada nodo en turno, como la raíz de lo que espera sea un árbol, y realiza una búsqueda de un nivel de profundidad en él. Si alguna vez regresa a un nodo que ya se haya encontrado, entonces ha encontrado un ciclo. Si agota todos los arcos desde cualquier nodo dado, regresa al nodo anterior. Si regresa hasta la raíz y no puede avanzar más, el subgráfico que se puede alcanzar desde el nodo actual no contiene ciclos. Si esta propiedad se aplica para todos los nodos, el gráfico completo está libre de ciclos, por lo que el sistema no está en interbloqueo.

Para ver cómo funciona el algoritmo en la práctica, vamos a utilizarlo con el gráfico de la figura 6-5(a). El orden de procesamiento de los nodos es

arbitrario, por lo que sólo los inspeccionaremos de izquierda a derecha, de arriba hacia abajo, ejecutando primero el algoritmo empezando en *R*, después sucesivamente en *A*, *B*, *C*, *S*, *D*, *T*, *E*, *F*, y así en lo sucesivo. Si llegamos a un ciclo, el algoritmo se detiene.
Empezamos en *R* e inicializamos *L* con la lista vacía. Después agregamos *R* a la lista y avanzamos a la única posibilidad, *A*, y lo agregamos a *L*, con lo cual tenemos que *L* = [*R*, *A*]. De *A* pasamos a *S*, para obtener *L* = [*R*, *A*, *S*]. *S* no tiene arcos salientes, por lo que es un punto muerto, lo cual nos obliga a regresar a *A*. Como *A* no tiene arcos salientes desmarcados, regresamos a *R*, completando nuestra inspección de *R*.
Ahora reiniciamos el algoritmo empezando en *A*, y restablecemos *L* a la lista vacía. Esta búsqueda también se detiene rápidamente, por lo que empezamos de nuevo en *B*. De *B* continuamos siguiendo los arcos salientes hasta llegar a *D*, donde *L* = [*B*, *T*, *E*, *V*, *G*, *U*, *D*]. Ahora debemos realizar una elección (al azar). Si elegimos *S* llegamos a un punto muerto, y regresamos a *D*. La segunda vez que elegimos *T* y actualizamos *L* para que sea [*B*, *T*, *E*, *V*, *G*, *U*, *D*, *T*], en donde descubrimos el ciclo y detenemos el algoritmo.
Este algoritmo está muy lejos de ser óptimo. Sin embargo, sirve para demostrar que existe un algoritmo para la detección del interbloqueo.

Detección del interbloqueo con varios recursos de cada tipo
Cuando existen varias copias de algunos de los recursos, se necesita un método distinto para detectar interbloqueos. Ahora presentaremos un algoritmo basado en matrices para detectar interbloqueos entre *n* procesos, de *P1* a *Pn*. Hagamos que el número de clases de recursos sea *m*, con *E1* recursos de la clase 1, *E2* recursos de la clase 2 y en general, *Ei* recursos de la clase *i* ($1 \leq i \leq m$). *E* es el **vector de recursos existentes**. Este vector proporciona el número total de instancias de cada recurso en existencia. Por ejemplo, si la clase 1 son las unidades de cinta, entonces *E1* = 2 significa que el sistema tiene dos unidades de cinta.
En cualquier instante, algunos de los recursos están asignados y no están disponibles. Hagamos que *A* sea el **vector de recursos disponibles**, donde *Ai* proporciona el número de instancias del recurso *i* que están disponibles en un momento dado (es decir, sin asignar). Si ambas de nuestras unidades de cinta están asignadas, *A1* será 0.
Ahora necesitamos dos arreglos: *C*, la **matriz de asignaciones actuales** y *R*, la **matriz de peticiones**.
La *i*-ésima fila de *C* nos indica cuántas instancias de cada clase de recurso contiene *Pi* en un momento dado. Así *Cij* es el número de instancias del recurso *j* que están contenidas por el proceso
i. De manera similar, *Rij* es el número de instancias del recurso *j* que desea *Pi*. Estas cuatro estructuras de datos se muestran en la figura 6-6.

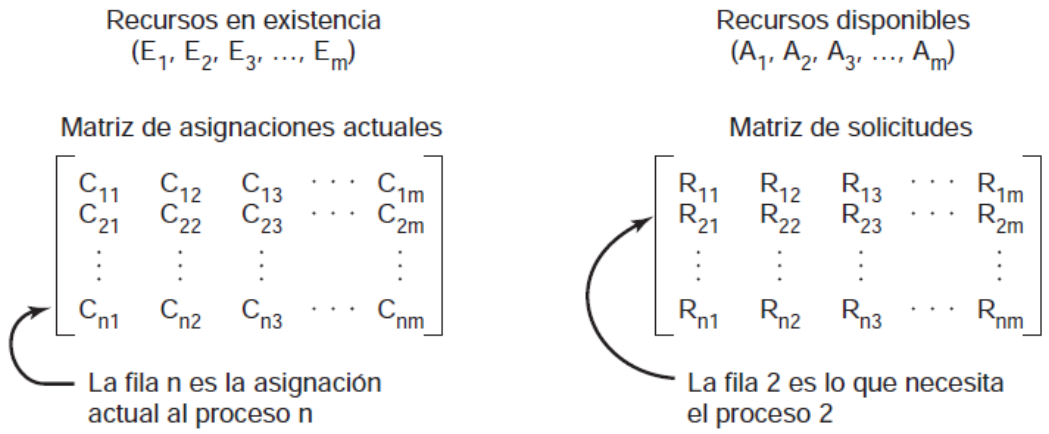


Figura 6-6. Las cuatro estructuras de datos que necesita el algoritmo de detección de interbloqueos.

Si agregamos todas las instancias del recurso *j* que se han asignado, y para ello agregamos todas las instancias disponibles, el resultado es el número de instancias existentes de esa clase de recursos.

El algoritmo de detección de interbloqueos se basa en la comparación de vectores. Vamos a definir la relación $A \leq B$ en dos vectores A y B para indicar que cada elemento de A es menor o igual que el elemento correspondiente de B. En sentido matemático $A \leq B$ se aplica sí, y sólo si $Ai \leq Bi$ para $1 \leq i \leq m$.

Al principio, se dice que cada proceso está desmarcado. A medida que el algoritmo progrese se marcarán los procesos, indicando que pueden completarse y, por ende, no están en interbloqueo.

Cuando el algoritmo termine, se sabe que cualquier proceso desmarcado está en interbloqueo. Este algoritmo supone un escenario del peor caso: todos los procesos mantienen todos los recursos adquiridos hasta que terminan.

El algoritmo de detección de interbloqueos se muestra a continuación.

1. Buscar un proceso desmarcado, *Pi*, para el que la *i*-ésima fila de *R* sea menor o igual que *A*.
2. Si se encuentra dicho proceso, agregar la *i*-ésima fila de *C* a *A*, marcar el proceso y regresar al paso 1.
3. Si no existe dicho proceso, el algoritmo termina.

Cuando el algoritmo termina, todos los procesos desmarcados (si los hay) están en interbloqueo.

Lo que hace el algoritmo en el paso 1 es buscar un proceso que se pueda ejecutar hasta completarse.

Dicho proceso se caracteriza por tener demandas de recursos que se pueden satisfacer con base en los recursos disponibles actuales. Entonces, el proceso seleccionado se ejecuta hasta que termina, momento en el que devuelve los recursos que contiene a la reserva de recursos disponibles.

Después se marca como completado. Si todos los procesos pueden en última instancia ejecutarse hasta completarse, ninguno de ellos está en interbloqueo. Si algunos de ellos nunca pueden terminar, están en interbloqueo. Aunque el algoritmo no es determinístico (debido a que puede ejecutar los procesos en cualquier orden posible), el resultado siempre es el mismo.

Como ejemplo acerca de cómo funciona el algoritmo de detección de interbloqueos, considere la figura 6-7. Aquí tenemos tres procesos y cuatro clases de recursos, que hemos etiquetado en forma arbitraria como unidades

de cinta, trazadores, escáner y unidad de CD-ROM. El proceso 1 tiene un escáner. El proceso 2 tiene dos unidades de cinta y una unidad de CD-ROM. El proceso 3 tiene un trazador y dos escáneres. Cada proceso necesita recursos adicionales, como se muestra con base en la matriz *R*.

Figura 6-7. Un ejemplo para el algoritmo de detección de interbloqueos.

Para ejecutar el algoritmo de detección de interbloqueos, buscamos un proceso cuya petición de recursos se pueda satisfacer. La primera no se puede satisfacer debido a que no hay unidad de CD-ROM disponible. La segunda tampoco se puede satisfacer, ya que no hay escáner libre. Por fortuna la tercera se puede satisfacer, por lo que el proceso 3 se ejecuta y en un momento dado devuelve todos sus recursos, para obtener

A = (2 2 2 0)

En este punto se puede ejecutar el proceso 2 y devolver sus recursos, con lo cual obtenemos

A = (4 2 2 1)

Ahora se puede ejecutar el proceso restante. No hay interbloqueo en el sistema.

Consideremos ahora una variación menor de la situación de la figura 6-7. Suponga que el proceso 2 necesita una unidad de CD-ROM, así como las dos unidades de cinta y el trazador. Ninguna de las peticiones se puede satisfacer, por lo que todo el sistema está en interbloqueo.

Ahora que sabemos cómo detectar interbloqueos (por lo menos cuando se conocen de antemano las peticiones de recursos estáticas), surge la pregunta sobre cuándo buscarlos. Una posibilidad es comprobar cada vez que se realiza una petición de un recurso. Esto sin duda los detectará lo más pronto posible, pero es potencialmente costoso en términos de tiempo de la CPU. Una estrategia alternativa es comprobar cada *k* minutos, o tal vez sólo cuando haya disminuido el uso de la CPU por debajo de algún valor de umbral. La razón de esta consideración es que si hay suficientes procesos en

interbloqueo, habrá pocos procesos ejecutables y la CPU estará inactiva con frecuencia.

Recuperación de un interbloqueo

Suponga que nuestro algoritmo de detección de interbloqueos ha tenido éxito y detectó un interbloqueo.

¿Qué debemos hacer ahora? Se necesita alguna forma de recuperarse y hacer funcionar el sistema otra vez. En esta sección analizaremos varias formas de recuperarse de un interbloqueo. Sin embargo, ninguna de ellas es en especial atractiva.

Recuperación por medio de apropiación

En algunos casos puede ser posible quitar temporalmente un recurso a su propietario actual y otorgarlo a otro proceso. En muchos casos se puede requerir intervención manual, en especial en los sistemas operativos de procesamiento por lotes que se ejecutan en mainframes.

Por ejemplo, para quitar una impresora láser a su propietario, el operador puede recolectar todas las hojas ya impresas y colocarlas en una pila. Después se puede suspender el proceso (se marca como no ejecutable). En este punto, la impresora se puede asignar a otro proceso. Cuando ese proceso termina, la pila de hojas impresas se puede colocar de vuelta en la bandeja de salida de la impresora y se puede reiniciar el proceso original.

La habilidad de quitar un recurso a un proceso, hacer que otro proceso lo utilice y después regresarlo sin que el proceso lo note, depende en gran parte de la naturaleza del recurso. Con frecuencia es difícil o imposible recuperarse de esta manera. Elegir el proceso a suspender depende en gran parte de cuáles procesos tienen recursos que se pueden quitar con facilidad.

Recuperación a través del retroceso

Si los diseñadores de sistemas y operadores de máquinas saben que es probable que haya interbloqueos, pueden hacer que los procesos realicen **puntos de comprobación** en forma periódica. Realizar puntos de comprobación en un proceso significa que su estado se escribe en un archivo para poder reiniciarlo más tarde. El punto de comprobación no sólo contiene la imagen de la memoria, sino también el estado del recurso; en otras palabras, qué recursos están asignados al proceso en un momento dado. Para que sean más efectivos, los nuevos puntos de comprobación no deben sobrescribir a los anteriores, sino que deben escribirse en nuevos archivos, para que se acumule una secuencia completa a medida que el proceso se ejecute.

Cuando se detecta un interbloqueo, es fácil ver cuáles recursos se necesitan. Para realizar la recuperación, un proceso que posee un recurso necesario se revierte a un punto en el tiempo antes de que haya adquirido ese recurso, para lo cual se inicia uno de sus puntos de comprobación anteriores.

Se pierde todo el trabajo realizado desde el punto de comprobación (por ejemplo, la salida impresa desde el punto de comprobación se debe descartar, ya que se volverá a imprimir). En efecto, el proceso se restablece a un momento anterior en el que no tenía el recurso, que ahora se asigna a uno de los procesos en interbloqueo. Si el proceso reiniciado trata de adquirir el recurso de nuevo, tendrá que esperar hasta que vuelva a estar disponible.

Recuperación a través de la eliminación de procesos

La forma más cruda y simple de romper un interbloqueo es eliminar uno o más procesos. Una posibilidad es eliminar a uno de los procesos en el ciclo. Con un

poco de suerte, los demás procesos podrán continuar. Si esto no ayuda, se puede repetir hasta que se rompa el ciclo.
De manera alternativa, se puede elegir como víctima a un proceso que no esté en el ciclo, para poder liberar sus recursos. En este método, el proceso a eliminar se elige con cuidado, debido a que está conteniendo recursos que necesita cierto proceso en el ciclo. Por ejemplo, un proceso podría contener una impresora y querer un trazador, en donde otro proceso podría contener un trazador y querer una impresora. Estos dos procesos están en interbloqueo. Un tercer proceso podría contener otra impresora idéntica y otro trazador idéntico, y estar felizmente en ejecución. Al eliminar el tercer proceso se liberarán estos recursos y se romperá el interbloqueo que involucra a los primeros dos.
Donde sea posible, es mejor eliminar un proceso que se pueda volver a ejecutar desde el principio sin efectos dañinos. Por ejemplo, una compilación siempre podrá volver a ejecutarse, ya que todo lo que hace es leer un archivo de código fuente y producir un archivo de código objeto. Si se elimina a mitad del proceso, la primera ejecución no tiene influencia sobre la segunda.
Por otra parte, un proceso que actualiza una base de datos no siempre se puede ejecutar una segunda vez en forma segura. Si el proceso agrega 1 a algún campo de una tabla en la base de datos, al ejecutarlo una vez, después eliminarlo y volverlo a ejecutar de nuevo, se agregará 2 al campo, lo cual es incorrecto.

Final Noviembre 2016

1) Threads. Si tuviera que implementar un Servidor de Archivos, ¿Cuál de los tipos de threads conviene utilizar, por qué?

Ejemplo teorías:

Supongamos un ambiente de servidor de archivos. Cada vez que llega una solicitud para acceder a un nuevo archivo, se debe crear una unidad de atención. Pero esta unidad de atención tendrá un ciclo de vida corto y, además, puede haber una gran demanda de estas unidades que se creerán y destruirán en un corto período.
Si es un ambiente multiprocesador estas unidades pueden ejecutarse simultáneamente en los diferentes procesadores. Que estas unidades de atención tengan estructura de hilos mejorará la productividad del sistema.
Para ejemplificar, en el modelo productor-consumidor se comparte un buffer donde el productor almacena lo producido y el consumidor lo usa. Es un espacio compartido.
Podríamos implementar este modelo como una tarea con dos hilos (el productor y el consumidor). De esa manera, el switching entre el hilo productor y el hilo consumidor se hace dentro del mismo espacio de direcciones, siendo el buffer compartido parte de ese espacio, y según la implementación, puede ser sin intervención del SO.
También pensemos en un servidor de red que acepta solicitudes de páginas WEB por parte de los clientes. SI este servidor se ejecutara como un proceso tradicional de un solo hilo, atendería a un cliente por vez. También podría haber un proceso receptor de solicitudes que por cada una que llegue cree un proceso hijo para que la atienda, con toda la actividad (y gasto de recursos) que significa la creación de nuevos procesos. Se obtendrá una mayor performance si tengo un proceso que atiende que crea hilos en su espacio.

Ejemplo de una página de internet:
Un ejemplo de aplicación que podría hacer uso de hilos es un servidor, como puede ser un servidor de archivos de una red de área local. Cada vez que llegue una solicitud de un nuevo archivo, se puede generar un nuevo hilo para el programa de gestión de archivos. Puesto que el servidor debe manejar muchas solicitudes, se crearan y destruirán muchos hilos en un corto periodo de tiempo. Si el servidor es un multiprocesador, se pueden ejecutar varios hilos de una misma tarea simultáneamente y en diferentes procesadores. Los hilos son también útiles en los monoprocesadores para simplificar la estructura de los programas que lleven a cabo diversas funciones.
Otro ejemplo de internet:
Usos de Hilos:
Los hilos permiten la combinación del paralelismo con la ejecución secuencial y el bloqueo de las llamadas al sistema.
Consideramos el ejemplo del *servidor de archivos* con sus posibles organizaciones para muchos hilos de ejecución.
Iniciamos con el *modelo servidor / trabajador:*

- Un hilo, el *servidor*, lee las solicitudes de trabajo en el buzón del sistema.
- Elige a un hilo *trabajador* inactivo (bloqueado) y le envía la solicitud, despertándolo.
- El hilo trabajador verifica si puede satisfacer la solicitud por medio del bloque caché compartido, al que tienen acceso todos los hilos.
- Si no envía un mensaje al disco para obtener el bloque necesario y se duerme esperando el fin de la operación.
- Se llama:
    - Al planificador y se inicializa otro hilo, que tal vez sea el servidor, para pedir más trabajo; o.
    - A otro trabajador listo para realizar un trabajo.

Los hilos ganan un desempeño considerable pero cada uno de ellos se programa en forma secuencial.
Otro *modelo* es el *de equipo:*

- Todos los *hilos son iguales* y cada uno obtiene y procesa sus propias solicitudes.
- *No hay servidor.*
- Se utiliza una cola de trabajo que contiene todos los trabajos pendientes, que son trabajos que los hilos no han podido manejar.
- Un hilo debe verificar primero la cola de trabajo antes de buscar en el buzón del sistema.

Un tercer *modelo* es el *de entubamiento:*

- El primer hilo genera ciertos datos y los transfiere al siguiente para su procesamiento.
- Los datos pasan de hilo en hilo y en cada etapa se lleva a cabo cierto procesamiento.

Un programa diseñado adecuadamente y que utilice hilos debe funcionar *bien:*

- En una *única cpu* con hilos compartidos.
- En un verdadero *multiprocesador.*