Índice

1. Planificacin de la carga (FaQ)

2. File system (Matías)

2.1. Introducción

El sistema de archivos poseé 2 partes:

- una colección de **archivos**: cada uno con datos
- una **estructura de directorios**: para organizarlos

algunos además tienen particiones.

2.2. Archivo

Es una colección de información relacionada. Es lo más pequeo que podemos escribir en memoria secundaria.

Representan programas o datos, pero en realidad son tiras de biy, bytes o registros que son interpretados como una u otra cosa (estructura del archivo).

2.2.1. Atributos

- **nombre**: única información entendible por humanos así como está
- **tipo**
- **ubicación**: puntero a dispositivo y a una posición en él
- **tamaño**
- **protección**
- **hora, fecha e identificación del usuario**: datos útiles para protección, seguridad y control del uso.

La información de todos los archivos se guarda la info de todos los archivos.

2.2.2. Operaciones

Operaciones que realiza un sistema con archivos:

- **Crear un archivo** Se debe encontrar el lugar donde estará y luego agregarlo al directorio.
- **Escribir un archivo** Se hace una llamada al sistema con el nombre del archivo y lo que hay que escribir. Este lo busca y lo escribe. Interviene un puntero a escritura, para saber por donde va escribiendo.
- **Leer un archivo** Se hace una llamada con el nombre y el lugar donde debe colocarse el siguiente bloque del archivo. También tiene un puntero a la próxima sección a leer. Se suele usar un único puntero para las 2 op (r/w).
- **Reubicarse dentro de un archivo** No implica E/S. Es como una búsqueda.
- **Eliminar un archivo** se libera el espacio y se borra la entrada del directorio.
- **Truncar un archivo** Se usa para borrar un archivo pero no los atributos (poner la long en 0)

Con este set de instrucciones se pueden hacer todas las operaciones.

Para evitar tener que buscar varias veces un archivo abierto en el directorio, los SO tienen una **Tabla de Archivos Abiertos**.

En sistemas complejos, ej multiusuario, hay además una Tabla de Archivos Abiertos por cada proceso que figura en la Tabla de Procesos. En estos casos hay un contador de aperturas para cada archivo.

Los archivos pueden tener extensiones para determinar de que tipo son, o tener atributos extras.

2.2.3. Métodos de acceso

Secuencial: lineal, en orden, con un puntero que va avanzando.

Directo: Accede a bloques, ya que los discos permiten acceso aleatorio a cualquier bloque. Sin orden. Las bases de datos usan este tipo.

Mediante índices: con una tabla, como si fuera **realmente un índice**. Puede haber 2 niveles de índices.

2.3. Estructura de directorios

Los discos se organizan en particiones, que forman estructuras lógicas (de mayor tama no o menor que un disco) que el usuario trata como un dispositivo. Cada partición tiene un **directorio** en el que se registran los datos de todos los archivos de la misma. El directorio nos debe prestar las siguientes operaciones:

- **Buscar un archivo** por nombre o por patrón

- **Crear un archivo**
- **Eliminar un archivo**
- **Listar un directorio**
- **Cambiar el nombre de un archivo**
- **Recorrer el sistema de archivos**

2.3.1. Niveles de un directorio

Un nivel

La más sencillas. No es escalable. Más de un usuario podría querer crear un archivo llamado "prueba" o "puto" y se violaría la regla de nombres únicos.

Dos niveles

Aca se puede crear un directorio por usuario y listo. Así, cada usuario tiene su directorio de archivos de usuario (UFD user file directory). Hay un directorio de archivos maestro (MFD master file...) que sabe donde están los UFD de cada usuario.

Se debe agregar un directorio de usuario al cual se debe poder acceder desde el MFD.

Para que un usuario pueda acceder a un archivo de otro usuario debe proveer el nombre del due no del mismo y el nombre del archivo, formando un camino (path) entre la raíz (MFD) y la hoja (archivo) del árbol que es el sistema de archivos.

Se agrega un "usuario especial" que tiene todos los programas, entonces cuando un usuario quiere correr uno, primero se busca en su directorio local y después en el del usuario especial. De esta manera se evita tener que copiar todos los archivos del sistema a los directorios de cada usuario. (Camino de busqueda)

Árboles o más de dos niveles

Permite crear subdirectorios. Cada entrada del directorio se define con un bit que lo identifica como archivo (0) o como subdirectorio(1). Aparece el concepto de directorio actual para un usuario, antes era su único directorio, ahora es alguno de sus subdirectorios. Se agrega la llamada al sistema de **cambiar directorio (cd)**. Hay path absolutos (desde la raíz) o relativos (desde el directorio actual)

Grafo acíclico

Es para compartir archivos y/o directorios, no tiene estructuras cíclicas. La forma más facil de implementar esto es poner un link (puntero) al mismo archivo. También se puede duplicar todo para compartir, pero es un garrón para mantener.

Hay que tener cuidado de que si queremos contar los archivos que tenemos, no contar 2 veces el mismo por 2 ramas de nuestro directorio.

También hay que tener cuidado cuando se borra. Si se borra el archivo, quedan colgando los links (punteros), sera costoso buscarlos todos, a menos que guardemos una lista de links al archivo en nuestro archivo o un conteo de referencias que nos deje borrar sólo cuando está en cero.

DOS no permite grafos acíclicos porque es demasiado complicado.

Grafo general - General Graph, companiero de General Electric

Nada, bardo, la busqueda mal dise nada da facilmente un algoritmo infinito.

Es dificil saber si al agregar un link no estamos cayendo en un grafo con ciclos.

2.4. Protección

Contra da nos físicos (confiabilidad) se logra con backups, contra acceso indebido (protección) se logra en un sistema monousuario retirando físicamente los discos flexibles y guardándolos bajo llave en un cajón del escritorio o en un archivero (sic).

2.4.1. Tipos de acceso

En los que no se permite el acceso a los archivos de otros usuario no se necesita protección. Los tipos de acceso se separan por la operación que se quiere realizar en cada uno (de los accesos):

- Leer
- Escribir
- Ejecutar
- Anexar (append) agregar información al final del archivo.
- Eliminar, borrar el archivo y liberar el espacio
- Listar el nombre y los atributos de un archivo.

Las demás operaciones se pueden hacer con estas, ej copiar un archivo es leer muchas veces y escribir un archivo.

2.4.2. Listas y grupos de acceso

Lo mejor es que el accedo dependa de quién es el que quiera acceder. Para eso se tiene una lista de acceso que especifica el nombre de un usuario y sus permisos. Cuando un usuario solicita eacceso a un archivo, el sistema mira la lista de acceso asociada a ese archivo. Si el usuario está autorizado para el acceso que pidió, entonces se lo permite, sino... no, ovbio.

El problema de estas tablas es el tama no. Una tabla que permita el acceso a TODOS los usuario puede ser muy grande, antes la entrada de un directorio tenía tama no fijo. Para solucionar esto hay 3 categorioas en relación con cada archivo:

- Propietario
- Grupo (cjto de usuarios que comparten el archivo o que necesitan acceso similar)
- Universo (los demás, el resto, la plebe, la gilada, etc)

Si usamos un bit para read R, uno para write W y otro para execute X, nos queda que con 3 campos de 3 bits tenemos todas las combinaciones de RWX con los grupos anteriores. La opción es tener una Lista de Control de Acceso por Archivo, con la anterior clasificación es bastante fácil y peque.

Falta lo de LCU, que dice a que archivos puede acceder cada usuario. PODEMOS COMPLETAR ESTO CON EL APUNTE MAANA EN LA FACU.

Otra opción es ponerle contrase na a los archivos, pero es demasiado pesado para el usuario recordar muchas, con lo cual es una pass a "todo o nada".

2.5. Estructura

Las transferencias entre la memoria se realizan en bloques. Cada bloque ocupa uno o más sectores. Dependiendo el disco el sector varía de tama no entre 32 y 4096 bytes, en gral son de 512

ME QUEDAN UNAS 6 HOJAS, PERO YA NO ENTIENDO NADA.

3. Deadlock, o abrazo de la muerte (Tomi)

3.1. Introducción

En un entorno de multiprogramación varios procesos pueden competir por un numero finito de recursos. Un procesos solicita recursos y si estos no estan disponibles, el proceso queda en estado de espera. Puede pasar que los procesos en espera nunca cambien de estado porque los recursos que necesitan los tienen otros procesos que tambien están en espera, a esta situación se la llama Dreadlock.

3.2. Modelo del sistema

Los recursos se dividen en varios tipos, cada uno de los cuales consiste en ejemplares idénticos. En el modo de operación normal, un proceso solo puede utilizar un recurso en la secuencia siguiente:

- Solicitud: Si la solicitud no se puede atender de inmediato entonces el proceso solicitante debe esperar a que pueda adquirir el recurso.
- Utilización
- Liberación

La solicitud y liberación de recursos son llamadas al sistema, estas pueden lograrse a travez de las operación wait y signal sobre semáforos. Una tabla del sistema registra si cada uno de los recursos esta libre o asignado, y si esta asignado a que proceso. Si un proceso solicita un recurso que ya esta asigando entoces lo puede encolar en la cola de espera de ese proceso. Un conjunto de procesos se encuentra en Deadlock si cada uno espera un suceso que solo puede originar otro proceso del mismo conjunto.

3.3. Caracterización de los bloqueos mutuos

3.3.1. Condiciones necesarias

Una situacion de deadlock puede surgir si y solo si en un sistema se presentan las siguientes cuatro condiciones:

- Exclusion mutua: Por lo menos un recurso debe reternerse en modo no compartido; es decir, solo un proceso puede usar el recurso a la vez. Si otro proceso solicita el recurso debera esperar a que sea liberado.
- Retención y espera: debe haber por lo menos un proceso que retenga un recurso y espere adquirir otros recursos retenidos por otros procesos.
- No apropiación: Los recursos no se pueden quitar. Es decir, un recurso solo puede ser liberado por el proceso que lo tiene despues de haber cumplido su tarea.
- Espera circular: Debe haber un conjunto $P0, ..., Pn$ de procesos en espera, tales que $P0$ espera un recurso retenido por $P1$, $P1$ espera un recurso retenido por $P2$,..., $Pn-1$ espera un recurso retenido por Pn y Pn espera un recurso retenido por $P0$.

La condición de espera circular implica la de retención y espera, sin embargo es util verlas por separado.

3.3.2. Grafo de asignación de recursos

Es un grafo dirigido donde los nodos son procesos o recursos. Una arista de un recurso a un proceso indica que ese recurso esta siendo utilizado por ese proceso (arista de asignación). Una arista de un proceso a un recurso indica que ese proceso solicito ese recurso (arista de solicitud). A los procesos los dibujamos con redondeles y a los recursos con cajitas. Como pueden haber muchas instancias de un mismo tipo de recurso en las cajitas dibujamos puntitos que representan cuantas instancias de ese recurso hay. Ejemplo: si tenemos dos impresoras dibujaremos una cajita con dos puntos dentro, en este caso podrán salir dos flechas de este nodo (una por impresora). IMPORTANTE: Si el grafo no contiene ciclos entonces ningun proceso esta en deadlock, por otra parte, si hay ciclo entonces PUEDE haber un deadlock. Si cada tipo de recurso tiene un solo ejemplar, entonces un ciclo representa que ha ocurrido un bloqueo mutuo y todos los procesos en el ciclo estan en deadlock. Si el ciclo comprende solo un conjunto de tipos de recursos, cada uno con un solo ejemplar, entonces ha ocurrido un bloqueo mutuo entre todos los procesos en el ciclo.

3.3.3. Métodos para manejar Deadlocks

Existen dos métodos principales para tratar el tema del deadlock. Podemos usar un protocolo para que el sistema nunca entre en deadlock o podemos permitir que entre y se recupere. Primero consideraremos métodos para que el sistema nunca entre en deadlock. Para ello contamos con dos métodos comunes: la prevención y la evitación.

3.4. Prevención de bloqueos mutuos

Para que haya deadlock deben presentarse cada una de las cuatro condiciones necesarias. Veremos distintas maneras de evitar cada una de estas condiciones, previniendo el deadlock.

3.4.1. Exclusión Mutua

Los recursos compartibles no pueden participar en un deadlock. Por lo general no es posible evitar el deadlock negando la condición de exclusión mutua. Por su propia condición algunos recursos no son compartibles.

3.4.2. Retención y espera

Para asegurar que la condición de retención y espera nunca se de, debemos garantizar que cuando un proceso solicita un recurso, no retenga otros.

- Un protocolo que puede usarse es que cada proceso solicite todos los recursos antes de empezar su ejecución.

- Otro similar es que un proceso solo pueda pedir recursos si no tiene ninguno asignado. Si necesita más recursos que los que tiene en un momento dado, debe liberar todo para volver a pedir lo que necesita.

La desventaja de estas políticas es que puede pasar que se pidan recursos que nunca serán usados por el proceso que los solicito. Además es posible un bloqueo indefinido, si un proceso pide varios recursos populares, es posible que nunca le sean adjudicados porque estan en uso por otros procesos (es una especie de inanición, pero el libro no lo nombra asi).

3.4.3. No apropiación

La tercer condicion es que no haya apropiación de recursos: para evitar esto podemos permitir la apropiacíon ;P

- Si un proceso que tiene un recurso solicita otro que no se le puede dar de inmediato, entonces se le sacan todos los recursos que tiene y volverá a ejecutar cuando se le puedan asignar todos los recursos que pidió mas los que tenía.

- Si un proceso solicita recursos que no estan disponibles comprobamos si estan asignados a otro proceso que espera mas recursos, en cuyo caso expropiamos los recursos deseados del proceso en espera y los asignamos al proceso solicitante. Si los recursos no están disponibles ni asignados a un proceso en espera, el proceso debera esperar.

Este protocolo se usa cuando se puede guardar con facilidad el estado de los recursos, como CPU y memoria principal. Es un garron para impresora o cintas.

3.4.4. Espera circular

Una forma de asegurar que no se presente la condición de espera circular es imponer una ordenación total de todos los tipos de recursos y requerir que cada proceso solicite los recursos en orden ascendente.

3.5. Evitación de bloqueos mutuos

Otro método para evitar los bloqueos mutuos consiste en requerir información adicional sobre como se solicitarán los recursos. Cada solicitud requiere que el sistema considere los recursos disponibles en ese momento, los actualmente asignados a cada proceso, y las futuras solicitudes y liberaciones de cada proceso, para decidir si puede satisfacer la solicitud presente o debe esperar para evitar un posible deadlock futuro. Los diversos algoritmos difieren en la cantidad y tipo de información que requieren. El modelo mas sencillo y útil requiere que el proceso declare la cantidad máxima de

recursos que usará para cada tipo de recurso. Un algoritmo de evitacion de deadlock examina dinámicamente el estado de asignación de recursos para asegurar que no pueda presentar una condición de espera circular. El estado de asignación de recursos viene definido por el número de recursos disponibles y asignados, y por la demanda máxima de los procesos. Un estado es seguro si el sistema puede asignar recursos a cada proceso (hasta el máximo) siguiendo algun orden y aun asi avitar el deadlock. Mas formalmente un sistema se encuentra en estado seguro sólo si existe una secuencia segura. Una secuencia de procesos $P_1, .., P_n$ es segura para el estado actual de asignaciones si, para cada P_i, los recursos que aun puede solicitar P_i pueden satisfacerse con los recursos actualmente disponibles mas los retenidos por todos los P_j, donde $j < i$. En esta situación si los recursos que necesita P_i no estan inmediatamente disponibles, entonces P_i puede esperar a que terminen todos los procesos P_j. Una vez terminados, P_i puede obtener todos los recursos necesarios, completar la tarea, devolver los recursos que se le han asignado y terminar. Cuando P_i termina, P_{i+1} puede obtener todos los recursos necesarios, etc. Si no existe esta secuencia, se dice que el estado es *inseguro*. Un estado seguro NO ES un estado de bloqueo mutuo, y un estado de bloqueo mutuo ES un estado inseguro, pero no todos los estados inseguros son de deadlock. Un estado inseguro puede llevar a un estado de deadlock. Ya establecido el concepto de estado seguro podemos ahora definir algoritmos de evitación que aseguren que el sistema nunca caerá en un estado de deadlock. La idea es asegurar que el sistema nunca caerá en un estado inseguro. Al principio el sistema está en un estado seguro, cuando un proceso solicita un recurso que en ese momento esta disponible, el sistema debe decidir si en ese momento le puede asignar ese recurso inmediatamente o si el proceso tiene que esperar. La solicitud se atiende solo si deja al sistema en un estado seguro. Observe que en este esquema si un proceso solicita un recurso que esta disponible puede llegar a tener que esperar, por esto la utilización de recursos puede ser menor que sin la utilización de este algoritmo.

3.5.1. Varios ejemplares de un mismo tipo de recurso

Describiremos el algoritmo del banquero. Cuando un proceso entra en el sistema debe declarar la cantidad máxima de recursos que puede llegar a necesitar, esta cantidad no puede exceder la cantidad que hay en el sistema. Cuando un proceso solicita un conjunto de recursos, el sistema debe determinar si esta asignación deja al sistema en un estado seguro. Si es asi, los recursos se asignan, sino el proceso debe esperar hasta que otro libere los recursos que el necesita. Para mayor detalle en el algoritmo mirar el libro, no se puede resumir y no tiene sentido que lo copie.

3.5.2. Un ejemplar único de cada tipo de recurso

Este algoritmo utiliza una variante del grafo de asignación de recursos, además de las aristas de solicitud y asignación, aadimos un nuevo tipo de arista, llamada *arista de reserva*. Una arista de reserva $P_i \rightarrow R_j$ indica que en el futuro el proceso P_i puede pedir el recurso R_j. Esta linea semaja a una linea de reserva en cuanto a dirección, pero la linea es punteada. Cuando el proceso P_i solicita el recurso R_j, la arista de reserva $P_i \rightarrow R_j$ se convierte en una arista de solicitud. De manera parecida cuando P_i libera un recurso R_j, la arista de asignación $R_j \rightarrow P_i$ vuelve a ser una arista de reserva $P_i \rightarrow R_j$. Observemos que en el sistema todos los recursos deben reservarse a priori; es decir antes de que el proceso P_i comienze su ejecución, todas sus aristas de reserva ya deben aparecer en el grafo de asignación de recursos. Podemos hacer mas flexible permitiendo que una arista de reserva $P_i \rightarrow R_j$ se agregue al grafo si todas las aristas relacionada con P_i son de reserva. Suponga que el proceso P_i solicita el recurso R_j. Esta solicitud puede atenderse solo si al convertir la arista de solicitud $P_i \rightarrow R_j$ en una de asignación $R_j \rightarrow P_i$ no se forma un ciclo en el grafo de asignación de recursos. Si no hay ningún ciclo, entonces la asignación del recurso dejará al sistema en un estado seguro. Si se detecta un ciclo, entonces la asignación colocará al sistema en un estado inseguro, por lo que el proceso P_i tendrá que esperar a que se satisfagan sus solicitudes.

3.6. Detección de Deadlocks

Si un sistema no emplea un algoritmo de prevención o evitación de deadlocks, entonces puede ocurrir una situación de deadlock. En este entorno el sistema debe ofrecer:

- Un algoritmo que examine el estado del sistema para determinar si ha ocurrido un deadlock.
- Un algoritmo para recuperarse del deadlock.

3.6.1. Varios ejemplares de un tipo de recurso

El algoritmo de detección emplea estructuras de datos que varian en el tiempo, similares a las utilizadas en el banquero.

- Disponible: Un vector de longitud m que indica el número de recursos disponibles de cada tipo.
- Asignacion: Una matriz de mxn que define el numero de recursos de cada tipo actualmente asignados a cada proceso.
- Solicitud: Una matriz de nxm que indica la solicitud actual de cada proceso.

El algoritmo de detección descrito aqui se limita a investigar cada una de las posibles secuencias de asignación para los procesos que quedan por terminar. (ES PARECIDO AL DEL BANQUERO, OTRA VEZ MIREN ESTE ALGORITMO EN EL LIBRO PORQUE NO TIENE SENTIDO QUE LO COPIE Y ADEMAS: LOS ALGORITMOS, SI SON BUENOS, SON NO COMPRESIBLES ;P).

3.6.2. Un único ejemplar de cada tipo de recurso

Una vez mas usaremos una variante del grafo de asignación, llamada *grafo de espera*. Podemos obtener este grafo sacando los nodos correspondientes a recursos del grafo de asignación de recursos y uniendo las aristas respectivas. Como antes, hay un deadlock si y solo si, hay un ciclo. Para detectar deadlock el sistema tiene que mantener actualizado este grafo y cada una cierta cantidad de tiempo correr un algoritmo de detección de ciclos $(O(n^2), nprocesos)$.

3.6.3. Utilización de los algoritmos de detección

Cuando debemos invocar el algoritmo de detección? La respuesta depende de dos factores:

- Con qué frecuencia es probable que ocurra un deadlock?
- Cuántos procesos se verán afectados al ocurrir el deadlock?

Tener en cuenta: Frecuencia de llamada al algoritmo de detección (Intevalo de tiempo): costo de overhead de este procesamiento. Si existe un deadlock, hasta que se solucione se puede .^{a}grandar.^{el} deadlock porque otros procesos pueden pedir recursos que estan trabados por estos giles. Los bloqueos mutuos solo pueden aparecer cuando una solicitud no se puede atender de inmediato, es posible que esta solicitud sea la que cierra un ciclo en el grafo de asignación. Por un lado podemos invocar el algoritmo de detección de deadlock cada vez que no pueda atenderse inmediatamente un pedido de asignación. En este caso podemos identificar los procesos que estan en el ciclo, pero mejor aun, podemos saber cual de ellos es el que cerro el ciclo. Si existen muchos tipos de recursos, una solicitud puede ocasionar varios ciclos en el grafo de recursos, cada uno de ellos completado por la solicitud mas reciente y "provocado"por un proceso identificable (si lo boleteamos nos sacamos flor de quilombo de encima). No podemos llamar al algoritmo de detección cada vez que se efectua una solicitud porque es demasiado costoso, en vez de esto lo que se puede hacer es llamarlo cada tanto (una hora por ejemplo), o cuando el uso de CPU es menor a algún porcentaje (%40 ponele), el problema de esto es que cuando lo llames te podes encontrar con banda de ciclos y no sabes cual de los procesos fue el que cerro los ciclos.

3.7. Recuperación después de bloqueos mutuos

Cuando un algoritmo de detección determina que hay un deadlock, tenemos dos alternativas: el sistema lo deja en manos del usuario informandole la existencia del deadlock o el sistema se recupera solo de esta situación. Hay dos opciones para romper el deadlock: kill -9 o sacarle recursos a los procesos.

3.7.1. Terminación de procesos: kill -9

Hay dos maneras de matar procesos:(cuando se mata un proceso, este libera todos los recursos que tenia asignados)

- Abortar todos los procesos que estan en el bloqueo mutuo: esto lo malo que tiene es que tiras a la basura bocha de laburo ya calculado (todo lo que habian hecho los procesos que fueron abortados).
- Abortar un proceso en cada ocasión hasta eliminar el ciclo de bloqueo: lo malo de esta política es que cada vez que se mata un proceso hay que correr el algoritmo de detección de deadlock, esto provoca un overhead no despreciable.

Observe que quizá no es fácil abortar un proceso, si se esta escribiendo un archivo, este podría quedar en un estado inconsistente. Si se utiliza el método de terminación parcial debemos elegir al proceso víctima, intentando abortar el proceso mas económico posible. Muchos factores determinan el proceso que se seleccionará:

- La prioridad del proceso
- Hace cuánto que se esta ejecutando y cuánto le falta
- Cuántos recursos y de qué tipo uso el proceso (por ejemplo si es fácil expropiar los recursos)
- Cuántos recursos mas necesita el proceso para poder concluir
- Cuántos procesos habrá que terminar
- Si el proceso es por lotes o interactivo

3.7.2. Expropiación de recursos

Se expropian recursos y se los asigna a otros procesos hasta que desaparezca el deadlock. Se debe tener en cuenta los siguientes aspectos:

- *Seleccion de la víctima:*Los factores de costo pueden incluir parámetros como el número de recursos que tiene un proceso en deadlock y la cantidad de tiempo que ha consumido ese proceso durante su ejecución.

- *Retroceso:* Si le expropiamos un recurso a un proceso, qué hacemos con el proceso?, claramente no puede seguir con su ejecución, debemos retrocederlo hasta un estado seguro y reiniciarlo desde ahi. Como es muy dificil determinar si un estado es seguro, la solución mas sencilla es reiniciarlo desde el comienzo: abortar y reiniciar. Sin embargo es mas efectivo retroceder el proceso unicamente lo necesario para romper el deadlock, pero esta alternativa requiere que el sistema guarde mas información sobre el estado de todos los procesos en ejecución.

- *Bloqueo indefinido:*Cómo podemos asegurar que no ocurrirá un bloqueo indefinido?Cómo podemos garantizar que los recursos expropiados no serán siempre del mismo proceso? Esto puede pasar y para evitarlo se incluye el número de retrocesos en el factor de costo con el cual se selecciona la víctima.

3.8. Estrategia combinada para el manejo de deadlocks

En la vida un poco mas real se usa un mix de todas estas politicas.

4. Proteccin y seguridad (Cristian)

Los mecanismos de proteccin proveen el acceso controlado y restringido de los recursos a los procesos que obtienen la autorizacin adecuada. El sistema de seguridad previene el acceso sin autorizacin al sistema y sus recursos.

4.1. Proteccin

El rol de la proteccin en un sistema es proveer los "mecanismos" encargados de hacer cumplir las "polticas" de utilizacin de los recursos.

Las polticas de utilizacin de los recursos pueden cambiar, por esta razn el sistema operativo lo que tiene que proveer son los mecanismos de proteccin al diseador de las aplicaciones para poder manipularlas de forma dinmica.

Es muy importante separar las polticas de los mecanismos, las polticas dicen que se tiene que hacer y los mecanismos como. Con robustos mecanismos uno puede llegar a tener que cambiar solo algunos parametros en caso de que cambien las polticas de uso.

4.2. Dominio

Un sistema es una coleccin de objetos y procesos. Un objeto puede ser software o hardware y se identifica univocamente. Estos son esencialmente un tipo abstracto de dato. A estos objetos se le pueden asignar operaciones que dependen del objeto (leer, escribir, ejecutar, etc).

Entonces, los procesos pueden realizar estas operaciones sobre los objetos siempre y cuando tengan la autorizacin adecuada. Adems, los procesos

solo deberan tener autorizacin necesaria para realizar su tarea (objetos y operaciones).

Los procesos ejecutan sobre un dominio. Los dominios son los que tienen estos objetos con sus respectivas operaciones. Un dominio es una coleccin de permisos de acceso, un par objeto y conjunto de operaciones permitidas (<archivo 1,[leer, ejecutar] >). Estos dominios pueden compartir accesos, no necesariamente son disjuntos.

Esta asociacin entre procesos y dominios no puede ser fija, ya que al cambiar alguna poltica un proceso podra necesitar mas autorizacin o perder algunos accesos. Por esto, se deben proveer mecanismos de modificacin o resignacin de dominios.

Formas de realizar los dominios:

- Los usuarios pueden ser dominios, los procesos acceden a los objetos con la identidad del usuario. Un switch se hace con un loggeo de un distinto usuario.
- Los procesos pueden ser dominios, cada proceso tiene su conjunto de accesos y un switch se hace con un mensaje a otro proceso.
- Los procedimientos pueden ser dominios, los objetos que pueden ser accedidos corresponden a las variables locales del procedimiento. Un switch es la llamada de un procedimiento.

Ejemplos

El modelo dual estndar (monitor-usuario) cuando se ejecuta en modo monitor se pueden ejecutar las rutinas del sistema mientras que en modo usuario solo se tienen instrucciones no privilegiadas.

Ejemplos - Unix

Unix utiliza la asociacin de dominios a usuarios. El cambio de dominio se realiza con un cambio temporal de usuario. Unix utiliza el modelo de file system, a cada archivo se le asocia un dueo y un bit de dominio. Cuando se ejecuta un archivo cuyo dueo es otro usuario y tiene el bit de dominio apagado el proceso se inicia con el usuario que lo ejecuta y sus permisos, sin embargo, si el bit esta encendido se le asigna el usuario dueo al procesos y sus permisos correspondientes hasta la finalizacin del proceso. El problema con este mecanismo es que si un usuario logra crear un archivo con dueo a rootü el bit de dominio encendido puede tener acceso sin restricciones al sistema. Un mtodo de mitigacin de este problema es asignar los permisos por directorio y realizar el cambio de usuario solo al dueo del directorio. Esto seria menos flexible. Algo incluso mas restrictivo es no permitir el cambio de usuario y realizar una estrategia de proceso "daemon.ªl cual invocar para acceder a instrucciones privilegiadas.

Multics

Multics organiza los dominios como anillos gerrquicos, 0 al 7 con sus dominios D0 a D7. si $i < j$ entonces Dj es un subconjunto de Di.

Multics usa direccin de memoria segmentada, cada segmento es un archivo asociado a un anillo. El descriptor incluye el numero del dominio y 3 bits de control, escritura, lectura y ejecucin. Cada proceso tiene asociado un numero de dominio y este puede acceder a los segmentos mayor o igual a este numero y respetando los bit de control.

Para el intercambio de dominio se tienen en cuenta los siguientes cambios en los descriptores de segmento:

- Accesos: Dos enteros, $b1 <= b2$
- Limite: Un entero $b3 > b2$
- Gates: Puntos de entrada que puede llamar el segmento.

Si un proceso del anillo i llama a un procedimiento (segmento) con accesos b1, b2 tal que $b1 <= i <= b2$ el proceso continua con nivel i. Sino:

Si $i < b1$ el procedimiento se ejecuta, sin embargo, los datos deben ser copiados a un segmento con acceso de i.

Si $b2 < i$ la llamada es permitida solo si $b3 <= i$ y se utiliza un gate.

El principal defecto de esta tcnica es que no se restringen los permisos solo a los necesarios.

4.3. Matriz de accesos

Los permisos pueden verse como una matriz abstracta donde las filas son los dominios y las columnas son los objetos (ver figura 19.3). Los elementos de esta matriz son los conjuntos de operaciones permitidas. Un proceso que se ejecuta en el dominio i solo tiene los permisos que figuran en la fila i. De esta forma podemos implementar las polticas de acceso con esta matriz. Para realizar un cambio de dominio se utiliza una estrategia de colocar a los dominios como objetos tambin (como columnas) y utilizar una operacin "switch". Si switch esta en (i,j) entonces el dominio de fila i puede cambiar al dominio de la columna j. Para permitir el cambio de esta matriz de forma dinmica tenemos que agregar los siguentes operaciones: copiar, dueo y control. La posibilidad de copiar se denota como un (*) al lado de la operacin de acceso. hay 3 tipos de permisos de copia: copia idntica (con el * tambin), copia limitada (solo la operacin de acceso sin el *) o transferencia (mover el permiso). Estas operaciones son siempre en el mismo objeto, es decir, en la misma columna. Un proceso con el permiso de dueo (denotado owner) puede crear nuevas operaciones de acceso o eliminar en cualquier campo de la misma columna. Estos 2 permisos nos proveen mecanismos para limitar la propagacin de accesos, sin embargo, no la propagacin de la informacin de los objetos fuera del entorno de ejecucin. Este problema se llama problema de confinamiento y en general no es resuelto. La operacin de control es solo aplicable a objetos de tipo dominio (dominio como columna). Control en el elemento (i,j) nos dice que el dominio i puede modificar los accesos de la fila

dominio j. Recordar que la importancia de estas 3 operaciones es permitir la modificacin dinamica de la matriz de accesos.

4.4. Implementaciones de la matriz de accesos

Tabla global:
Conjunto de tuplas <dominio, objeto, conjunto de permisos>.

Lista de accesos por Objeto: Por objeto se tiene una lista de duplas ¡dominio, conjunto de permisos¿(dominios con conjunto de permisos no vacos).

Lista de capacidad por dominios: dem anterior pero usando las filas en lugar de las columnas. La lista de capacidades es en si un objeto protegido y mantenido por el sistema operativo. Estos objetos se diferencian de los datos por tener un tag que los identifica o utilizando una divisin de la memoria entre datos y capacidades (memoria segmentada).

Candado-llave: Este utiliza lista de accesos (con candados) y lista de capacidades (con llaves). Un proceso en un determinado dominio debe tener la llave para abrir el candado si quiere acceder a un objeto.

4.5. Comparacin entre implementaciones

Lista de accesos corresponde directamente a las necesidades del usuario asignando los permisos por objeto. Sin embargo, hay que checkear los accesos al procesar en todo momento. Una lista larga de accesos puede demorar esta tarea.

Lista de capacidades se corresponde mas a los procesos, es mas fcil identificar a la hora de ejecucin. Sin embargo, el quitar permisos es complicado a la hora de redefinir permisos.

Candado -llave, al ser un mix de las 2 puede ser efectiva y flexible dependiendo del largo de las llaves.

Lo mas usado es un mix de los 2 primeros (ej. tener lista de accesos y en cada ejecucin ir generando una lista de capacidades que se elimina al finalizar el proceso).

4.6. Revocar permisos de acceso

- inmediatamente vs demorado
- selectivo vs general
- parcial vs general
- temporal vs permanente

Si se utiliza lista de accesos es sencillo y se puede realizar de la forma que se desee. Con lista de capacidades se plantean varias estrategias:

Readquisicin: La lista se borra peridicamente de los dominios y hay que actualizarla.

Punteros: Una lista de punteros se mantiene en el objeto a las capacidades y se revoca utilizando estos punteros (es costoso, se usa en multics).

Indireccin: Las capacidades apuntan a una entrada en una tabla global de puntero a los objetos, cuando se quita el objeto de la tabla se pierde el acceso por fallo en la indireccin (puntero no valido). No permite mtodo selectivo, se revoca a todos los dominios.

Llaves: Una llave maestra se le asigna a los accesos del objeto, cuando se crea una capacidad en un dominio se copia esta llave. El cambio de llave revoca de forma general el acceso a todos los dominios. Este mtodo tambin puede implementarse con varias llaves para tener mas flexibilidad y proveer una revocacin mas selectiva.

4.7. Sistemas basados en capacidades

Hydra (descripcin no relevante, ir al libro)
Cambridge CAP (descripcin no relevante, ir al libro)

4.8. Proteccin basada en lenguaje

La proteccin de los recursos puede variar por aplicacin o cambios de polticas, por lo que se comenz a considerar un problema no solo de los diseadores de sistemas operativos, sino tambin, de los diseadores de aplicaciones. Por esto, los lenguages de programacin se comenzaron a utilizar para brindar proteccin. Cuando la "declaracin" de la proteccin se adjunta a la declaracin de los tipos de datos se permite especificar los requerimientos de proteccin al diseador del subsistema. Ventajas:

- Las protecciones son simplemente declaradas, en lugar de llamar a subrutinas del sistema operativo.
- La proteccin se independiza de las facilidades provistas por el sistema operativo.
- Los medios de ejecucin no se proveen por parte del diseador del subsistema.
- Una notacin declarativa es mas natural ya que los privilegios de acceso estn relacionados a los conceptos lingisticos de los tipos de dato.

(lase tipos de datos como los objetos a restringir, al menos es lo que yo entend).

Todos los metodos de proteccin dependen en cierto grado de la implementacin de la proteccin del sistema operativo. La principal distincin entre tener una buena implementacin en el sistema operativo es la seguridad de la

proteccin brindada por el kernel, ya que los mecanismos se basan en como opera el sistema (un compilador puede asegurar que no hay huecos en el soft pero no que el archivo en ejecucin no mute estas protecciones).

Seguridad: Mas relevante en el kernel, a nivel compilacin se basa en la correcta traduccin, mecanismos de proteccin de segmentos para el manejo del almacenamiento y la seguridad de los archivos de los que se carga el programa.

Flexibilidad: Hay muchos limites a nivel kernel para implementar polticas especificas de usuarios y es muy costoso la modificacin del kernel. En cambio, a nivel lenguaje la modificacin afecta menos al sistema general.

Eficiencia: La ejecucin de proteccin por hardware es la mas eficiente. Sin embargo, soporte del software en necesario y un compilador inteligente puede evitar sobreprocesamiento por parte del kernel.

4.9. Resumen

La especificacin de la proteccin en un lenguaje provee una descripcin de las polticas en un alto nivel. la implementacin provee proteccin no brindada por el hardware y se pueden interpretar llamadas a rutinas de proteccin en el kernel.

4.10. Seguridad

Seguridad, a diferencia de proteccin, no solo requiere una proteccin adecuada del sistema, sino que tambin se deben considerar factores externos. Se debe protejer de uso sin autorizacin, destruccin o alteracin intencional o inconsistencias accidentales.

4.11. El problema de la seguridad

Un sistema se dice seguro si sus recursos se acceden y utilizan como corresponde bajo cualquier circunstancia. Violaciones de seguridad pueden ser maliciosas o accidentales. las maliciosas se categorizan como lectura, modificacin o destruccin de informacin (desautorizada).

No es posible una proteccin absoluta, solo aumentar la dificultad de acceso lo mas posible. Para proteger el sistema se deben tomas 2 medidas de seguridad fsica (el lugar donde esta el hardware) y humana (que una persona de accesos a otro sin autorizacin). La seguridad que vamos a estudiar es a nivel operativo, la cual queda insignificante ante una falla de los dos aspectos mencionados.

4.12. Autentificacin

La habilidad de identificar un usuario. Hay 3 aspectos: posesin (llave o tarjeta), conocimiento (identificador o contrasea) y atributos (huella digital,

firma). Password (contrasea)

Es muy flexible, se puede asignar a un usuario o a recursos especficos. Es muy comn y de concepto sencillo. Las vulneravilidades pueden ser varias. Un password sencillo puede ser adivinado o encontrado por fuerza bruta. Un password complicado (maysculas, minsculas, nmeros y smbolos) y largo puede ser escrito por el usuario. Visualizacin del password, ya sea mirando el teclado o un mtodo espa por medio de una red. Se pueden proponer cambios de password peridicos, incluso sin dejar repetir los N ltimos como metodos de seguridad. Si los password se comparten puede ser complicado identificar el origen de la infraccin. Los password pueden ser elegidos por los usuarios o generados automticamente (mas difcil de recordar).

4.13. Password encriptados

Se utiliza una funcin fcil de ejecutar pero difcil de invertir. La tabla con los password no necesita ser protegida, sin embargo, teniendo la funcin se puede realizar fuerza bruta para comparar los resultados con la tabla.

Password de una vez: Este mtodo utiliza un secreto y una semilla. El sistema realiza una pregunta al usuario utilizando la semilla y este contesta utilizando el secreto. El sistema conoce tambin el secreto y puede calcular el resultado para compararlo con el que dio el usuario. Este mtodo es muy efectivo para evitar capturas de password, ya que todas las veces se genera un nuevo password al cambiar la semilla(ej: secreto: fn cuadrado, semilla: 2, respuesta: 4). Una variante puede ser una lista de password que se utilizan en orden.

Utilizacin de programas: Un programa escrito por una persona puede ser utilizado por otra.

Caballo de Troya: Un programa que trata de ejecutarse en un dominio con mayor permiso causando quiebres de seguridad. Una variante es aparentar una ventana de logueo haciendo que el usuario ingrese su contrasea.

Puerta trampa (trap door): Huecos dejados de forma intencionada por el productor de un software ara utilizarlos mas tarde. Una puerta de entrada mus inteligente incluso puede estar en un compilador.

Utilizacin del sistema: Los sistemas operativos proveen la posibilidad de que un proceso ejecute otro proceso, esto posibilita dos tipos de mala utilizacin.

Gusanos: Los gusanos utilizan estos metodos para reproducirse copiandose a si mismo tomando los recursos del sistema y no permitiendo el uso de ellos por el usuario. Estos son muy peligrosos ya que pueden reproducirse incluso a travez de la red.

Virus: Un virus es un fragmento de codigo que se agrega a un programa con la finalidad de daar o destruir informacin. La diferencia principal con el gusano es que el virus no es una entidad autosuficiente. Los principales afectados son los usuarios de microcomputadoras. Los antivirus son una

forma efectiva para controlarlos, estos buscan patrones de ejecucin utilizados por los virus en los programas y quitan esos fragmentos de codigo.

Un metodo mas efectivo es la prevencin o utilizacin de mecanismos para verificar la consistencia de los programas (ej. contrar el tamao de los ejecutables y notificar los cambios)

4.13.1. Utilizacin de monitoreo

Pueden verse 2 aspectos, revisar patrones sospechosos (fallas repetidas de password) o llevar una bitacora (log) de los sucesos.

Otro metodo es realizar revisiones frecuentes y reparar automaticamente o avisar sobre aspectos irregulares:

1. Password cortos
2. Seteo de id sin autorizacin a un programa
3. Programas nuevos sin autorizacin
4. Procesos largos que no se esperaban
5. Protecciones inadecuadas en directorios de usuario y sistema
6. Protecciones inadecuadas en archivos privilegiados (lista de password, kernel)
7. Entradas peligrosas, como troyanos
8. Cambios en programas del sistema encontrados por checkeo de suma (checksum)

Otro problema es la conexion remota a un sistema. Una proteccin de esto es la utilizacion de un firewall, es un interlocutor entre 2 sistemas filtrando la informacin que intercambian.

4.14. Encriptacin

TODO...

5. Monitores (FaQ)

6. Procesos concurrentes (Fran)