Sistemas Operativos

UNIVERSIDAD NACIONAL DEL SUR

::: ACORDE A SISTEMAS OPERATIVOS 2DO CUATRIMESTRE 2012 :::

NOTAS:

0) ESTE RESUMEN PERMITE ALCANZAR UNA APROBACIÓN SEGURA DEL EXAMEN FINAL REGULAR **– PARTE TEÓRICA --** *SIEMPRE Y CUANDO SE LO ESTUDIE EN SU TOTALIDAD, LO CUAL QUEDA A TOTAL DISCRECIÓN DEL LECTOR*

1) SI BIEN FALTAN ILUSTRACIONES SE LAS PUEDE ENCONTRAR EN LA BIBLIOGRAFÍA DE LA CÁTEDRA.

2) SE BRINDA TOTAL DERECHO DE MODIFICACIÓN, AUNQUE SE PROVEE UNA VERSIÓN EN PDF COMO COPIA DE SEGURIDAD.

3) FUENTE: Silberschatz, A. y Galvin, P.B. Operating System Concepts, Addison-Wesley, 2001, 6ta Edición

Introducción - Módulo 1

¿Qué es un sistema operativo?

Un sistema operativo es un programa que administra el hardware de la computadora. Entre sus funciones principales se puede destacar que provee una base para los programas de aplicación y actúa como intermediario entre un usuario de una computadora y el hardware de la misma, simplificando el uso de la computadora y permitiendo la solución de problemas del usuario más fácilmente.

Los sistemas operativos se pueden clasificar de varias maneras, entre otras a través de los siguientes criterios: de código fuente abierto, de código cerrado, gratuitos, pagos, en función del tipo de kernel, de la plataforma a la cual brindan soporte (32 o 64 bits).

Componentes del sistema de Cómputo

Un sistema informático puede dividirse a grandes rasgos en cuatro componentes: el hardware, el sistema operativo, los programas de aplicación y los usuarios.

El hardware, la unidad central de procesamiento, UCP o CPU (central processing unit), la memoria y los dispositivos de E/S (entrada/salida), proporcionan los recursos básicos de cómputo al sistema. El sistema operativo controla y coordina el uso del hardware entre los diversos grupos de programas de aplicación por parte de los distintos usuarios. Los programas de aplicación, por caso, los procesadores de texto, definen el modo en el cual los recursos del sistema son usados para resolver los problemas de computación de los usuarios.

Cabe aclarar que la visión de usuario de la computadora varía de acuerdo a la interfaz que se utilice. La mayoría de los usuarios que utilizan una PC disponen de un monitor, un teclado, un mouse y un disco rígido. Un sistema así se diseña para que un usuario monopolice sus recursos. El objetivo es maximizar el trabajo (o el juego) que el usuario realice. En este caso, el sistema operativo se diseña principalmente para que sea de fácil uso, prestando cierta atención al rendimiento y ninguna a la utilización de los recursos. En otros casos, un usuario se encuentra frente a un terminal conectado a un mainframe. En este último entorno, otros usuarios acceden simultáneamente a través de otros terminales a dicha supercomputadora. Estos usuarios comparten recursos y pueden intercambiar información. En tales casos, el sistema operativo se diseña para maximizar la utilización de los recursos, asegurar que todo el tiempo de CPU, memoria y E/S disponibles se usen de manera eficiente y que todo usuario disponga sólo de la parte equitativa que le corresponde.

En otros casos, los usuarios usan estaciones de trabajo conectadas a redes de otras estaciones de trabajo y servidores. Estos usuarios tienen recursos dedicados a su disposición, pero también tienen recursos compartidos como la red y los servidores (servidores de archivos, de impresión). Por tanto, su sistema operativo está diseñado para llegar a un compromiso entre la usabilidad individual y la utilización de los recursos.

Por otro lado, algunas computadoras tienen poca o ninguna interacción con el usuario. Por ejemplo, las computadoras incorporadas en los electrodomésticos y los automóviles pueden disponer de teclados numéricos e indicadores luminosos que se encienden y apagan para mostrar el estado, pero tanto estos equipos como sus sistemas operativos están diseñados fundamentalmente para funcionar sin intervención del usuario.

Definiciones de Sistemas Operativos

Desde el punto de vista de la computadora, el sistema operativo es el programa más íntimamente relacionado con el hardware. En este contexto, podemos ver un sistema operativo como un alocador de recursos. Un sistema informático cuenta con muchos recursos que pueden ser necesarios para solucionar un problema: tiempo de CPU, espacio de memoria, espacio de almacenamiento de archivos, dispositivos de E/S, etc. El sistema operativo actúa como el

administrador de estos recursos. Al enfrentarse a numerosas y posiblemente conflictivas solicitudes de recursos, el sistema operativo debe decidir cómo asignarlos a programas y usuarios específicos, de modo que la computadora pueda operar en forma eficiente y equitativa.

Un punto de vista ligeramente diferente de un sistema operativo hace hincapié en la necesidad de controlar los distintos dispositivos de E/S y programas de usuario. Un sistema operativo es un programa de control. Como programa de control, gestiona la ejecución de los programas de usuario para evitar errores y evitar el uso impropio de la computadora.

Definiciones de Sistemas Operativos (cont)

Si bien no existe una definición universalmente aceptada, la más común entiende al sistema operativo como un programa que se ejecuta continuamente en la computadora (usualmente denominado kernel), siendo todo lo demás programas del sistema y programas de aplicación.

Tipos de sistemas operativos

Casi todos los sistemas operativos tienen una interfaz de usuario. Está puede ser implementada de diferentes maneras. Una forma es a través de una línea de comandos, la cual usa comandos de texto y un método para ingresarlos. En una interfaz batch, los comandos y las directivas para controlar dichos comandos son ingresados en archivos que luego son ejecutados.

Por otro lado, un sistema operativo interactivo provee comunicación directa entre el usuario de la computadora y el sistema. El primero le da instrucciones al sistema operativo o al programa directamente, mediante la utilización de un dispositivo de entrada como un teclado o un mouse y queda a la espera de una respuesta inmediata en un periférico de salida. Convenientemente el tiempo de respuesta debe ser corto, típicamente inferior a un segundo.

Un sistema operativo de tiempo compartido permite que múltiples usuarios compartan una misma computadora simultáneamente. Dado que cada acción o comando en un sistema de tiempo compartido tiende a ser breve, solo es requisito asignar un pequeño tiempo de CPU a cada usuario, para lograr esto último se utiliza la planificación y multiprogramación del CPU.

En un sistema de tiempo compartido, se debe asignar un tiempo de respuesta razonable, para esto se recurre al swapping, donde los procesos son intercambiados desde la memoria principal al disco rígido o viceversa. Un método común para lograr este objetivo es recurrir al uso de la memoria virtual.

Además en este tipo de sistemas se provee un sistema de archivos que reside en una colección de discos, por lo que se debe brindar una administración de discos.

Los sistemas embebidos casi siempre corren sistemas operativos de tiempo real. Este tipo de sistemas operativos son usados cuando existen estrictos requerimientos de tiempo para el procesamiento de una operación o para el flujo de datos. Algunos ejemplos de sistemas de tiempo real son sistemas que controlan experimentos científicos, sistemas de diagnostico por imágenes de la medicina, sistemas de control industrial entre otros.

Un sistema distribuido está compuesto por una colección de computadoras separadas, posiblemente heterogéneas, que están interconectadas para permitir a los usuarios acceder a los múltiples recursos que el sistema mantiene. La utilización de recursos compartidos incrementa la velocidad de cómputo, la funcionalidad, la disponibilidad de datos y la fiabilidad.

Los sistemas distribuidos dependen en su capacidad de interconexión para su propia funcionalidad ya que se comunican mutuamente para “formar” una especie de sistema único que controla la red y sus recursos.

Kernel

El núcleo o kernel del un sistema operativo es un programa que ejecuta todo el tiempo en la computadora y es el encargado de gestionar recursos, a través de llamadas al sistema, por caso, proveyendo de servicios relacionados con la entrada/salida.

Inicio de la computadora

Para que una computadora comience a funcionar, por ejemplo cuando se enciende o reinicia, es necesario que tenga un programa que ejecutar. Este programa de bootstrap, o programa de arranque, suele ser simple. Normalmente, se almacena en la memoria ROM o EEPROM y se conoce con el término general de firmware. Se inicializan todos los aspectos del sistema, desde los registros de la CPU hasta las controladoras de dispositivos y el contenido de la memoria. El programa de arranque debe saber cómo cargar el sistema operativo e iniciar la ejecución de dicho sistema. Para conseguir este objetivo, el programa de arranque debe localizar y cargar en memoria el kernel del sistema operativo. A veces es un proceso de dos pasos donde el boot block en una locación fija carga el bootstrap loader. Después, el sistema operativo comienza ejecutando el primer proceso, como por ejemplo init, y espera que se produzca algún suceso.

Funcionamiento de las interrupciones y traps

La ocurrencia de un suceso normalmente se indica mediante una interrupción o bien de hardware o bien de software. El hardware puede activar una interrupción en cualquier instante enviando una señal a la CPU, normalmente a través del bus del sistema. El software puede activar una interrupción ejecutando una operación especial denominada llamada al sistema, llamada de monitor o syscall.

Cuando se interrumpe a la CPU, deja lo que está haciendo e inmediatamente transfiere el control a la rutina de servicio de la misma, generalmente por medio del vector de interrupción, que contiene las direcciones de todas las rutinas de servicio. La rutina de servicio a la interrupción se ejecuta y, cuando ha terminado, la CPU reanuda la operación que estuviera haciendo. Cabe aclarar que la arquitectura de la interrupción debe salvar la dirección de la instrucción interrumpida y además las interrupciones entrantes son deshabilitadas mientras otra interrupción está siendo procesada para prevenir una pérdida de interrupción.

El método más simple para tratar esta transferencia consiste en invocar una rutina genérica para examinar la información de la interrupción; esa rutina genérica, a su vez, debe llamar a la rutina específica de tratamiento de la interrupción. Sin embargo, las interrupciones deben tratarse rápidamente y este método es algo lento. En consecuencia, como sólo es posible un número predefinido de interrupciones, puede utilizarse otro sistema, consistente en disponer una tabla de punteros a las rutinas de interrupción, con el fin de proporcionar la velocidad necesaria. De este modo, se llama a la rutina de interrupción de forma indirecta a través de la tabla, sin necesidad de una rutina intermedia. Esta matriz, o vector de interrupciones, de direcciones se indexa mediante un número de dispositivo univoco que se proporciona con la solicitud de interrupción, para obtener la dirección de la rutina de servicio la interrupción para el dispositivo correspondiente.

Por su lado, un trap o una excepción es una interrupción generada por el software que puede ser causada por un error (división por cero) o por una solicitud especifica proveniente de un programa del usuario solicitando un servicio del sistema operativo.

Otro aspecto de la interfaz de llamadas al sistema es relativo a la elección entre E/S bloqueante y no bloqueante. Cuando una aplicación ejecuta una llamada al sistema bloqueante, se suspende la ejecución de la aplicación. En ese instante, la aplicación se pasa de la cola de ejecución del sistema operativo a una cola de espera. Después de que se complete la llamada al sistema, vuelve a colocarse la aplicación en la cola de listos, donde pasará a ser elegible para reanudar su ejecución, en cuyo momento se le entregaran los valores devueltos por la llamada al sistema. Las acciones físicas realizadas por los dispositivos de E/S son generalmente asíncronas, consumiendo una cantidad variable o

impredecible de tiempo. De todos modos, la mayoría de los sistemas operativos utilizan llamadas al sistema bloqueantes para la interfaz de aplicaciones, porque el código de aplicación bloqueante es más fácil de entender que el no bloqueante.

Algunos procesos a nivel de usuario necesitan una E/S no bloqueante. Un ejemplo seria la interfaz de usuario que recibe una entrada de teclado y de ratón mientras que está procesando y mostrando datos en la pantalla.

Una forma en que el desarrollador de una aplicación puede solapar la ejecución con las operaciones de E/S consiste en escribir una aplicación multihilo. Algunos hilos pueden realizar llamadas al sistema bloqueantes mientras que otros continúan ejecutándose.

Una alternativa a las llamadas al sistema no bloqueantes consiste en utilizar llamadas al sistema asíncronas. Una llamada al sistema asíncrona vuelve inmediatamente, sin esperar a que la operación de E/S se complete. La aplicación continúa entonces ejecutando su código. La terminación de la operación de E/S se comunica a la aplicación en algún instante.

Organización y Operación del Sistema de Computo

Una computadora de propósito general consta de una o más CPU y de múltiples controladoras de dispositivo que se conectan a través de un bus común. Cada controladora de dispositivo se encarga de un tipo específico de dispositivo. Dependiendo de la controladora, puede haber más de un dispositivo conectado. Una controladora de dispositivo mantiene algunos búferes locales y un conjunto de registros de propósito especial. La controladora de dispositivo es responsable de transferir los datos entre los dispositivos periféricos que controla y su búfer local. Normalmente, los sistemas operativos tienen un controlador de dispositivo para cada controladora de dispositivo. Este software controlador del dispositivo se encarga de entenderse con la controladora de hardware y presenta al resto del sistema operativo una interfaz uniforme mediante la cual comunicarse con el dispositivo.

Al iniciar una operación de E/S, el controlador del dispositivo carga los registros apropiados de la controladora. Ésta, a su vez, examina el contenido de estos registros para determinar la acción a realizar. La controladora inicia entonces la transferencia de datos desde el dispositivo a su búfer local. Una vez completada la transferencia de datos, la controladora informa al controlador del dispositivo, a través de una interrupción, de que ha terminado la operación. El controlador devuelve entonces el control al sistema operativo, devolviendo posiblemente los datos, o un puntero a los datos, si la operación ha sido lectura. Para otras operaciones, el controlador del dispositivo devuelve información de estado.

Esta forma de E/S controlada por interrupción resulta inadecuada para transferir cantidades grandes de datos, como en la E/S de disco. Para resolver este problema, se usa acceso directo a memoria (DMA, direct memory access). Después de configurar búferes, punteros y controladores para el dispositivo de E/S, la controladora de hardware transfiere un bloque entero de datos entre su propio búfer y la memoria, sin que intervenga el CPU. Sólo se genera una interrupción por cada bloque, para decir al controlador software del dispositivo que la operación se ha completado, en lugar de la interrupción por byte generada en los dispositivos de baja velocidad. Mientras la controladora hardware realizas estas operaciones, la CPU está disponible para llevar a cabo otros trabajos.

Algunos sistemas de gama alta emplean una arquitectura basada en conmutador, en lugar de un bus. En estos sistemas, los diversos componentes pueden comunicarse con otros componentes en forma concurrente, en lugar de competir por ciclos de un bus compartido. En este caso, el acceso directo a memoria es incluso más eficaz.

Tabla de estado de dispositivos

Los desarrolladores de sistemas operativos implementan los mecanismos de planificación manteniendo una cola de espera de solicitudes para cada dispositivo. Cuando una aplicación ejecuta una llamada al sistema de E/S bloqueante, la

solicitud se coloca en la cola correspondiente a dicho dispositivo. El planificador de E/S reordena la cola para mejorar la eficiencia global del sistema y el tiempo medio de respuesta experimentado por las aplicaciones.

Cuando un kernel soporta mecanismos de E/S asíncrona, debe ser capaz de controlar múltiples solicitudes de E/S al mismo tiempo. Con este fin, el sistema operativo puede asociar la cola de espera con una tabla de estado de dispositivo. El kernel se encarga de gestionar esta tabla, que contiene una entrada por cada dispositivo de E/S. Cada entrada en la tabla indica el tipo, la dirección y el estado (no funciona, inactivo u ocupado) del dispositivo. Si el dispositivo está ocupado con una solicitud, se almacenaran en la entrada de la tabla correspondiente a dicho dispositivo el tipo de la solicitud y otros parámetros.

Una forma en que el sistema de E/S mejora la eficiencia de la computadora es planificando las operaciones de E/S. Otra forma es utilizando espacio de almacenamiento en la memoria principal o en el disco mediante técnicas denominadas almacenamiento en búfer, almacenamiento en caché y gestión de colas.

Estructura de almacenaje

La amplia variedad de sistemas de almacenamiento en un sistema informático puede organizarse en una jerarquía según la velocidad y el coste. Los niveles superiores son caros, pero rápidos. A medida que se desciende por la jerarquía, el coste por bit generalmente disminuye, mientas que el tiempo de acceso habitualmente aumenta.

Los programas de la computadora deben hallarse en la memoria principal para ser ejecutados.

Para que un programa pueda ejecutarse, debe estar asignado a direcciones absolutas y cargado en memoria. Mientras el programa se está ejecutando, accede a instrucciones y a los datos de la memoria generando dichas direcciones absolutas. Finalmente, el programa termina, el espacio de memoria se declara disponible y el siguiente programa puede ser cargado y ejecutado.

Para mejorar tanto la utilización de la CPU como la velocidad de respuesta de la computadora frente a los usuarios, las computadoras de propósito general pueden mantener varios programas en memoria, lo que crea la necesidad de mecanismos de gestión de memoria.

El sistema operativo es responsable de las siguientes actividades en lo que se refiere a la gestión de memoria:

- Controlar que partes de la memoria están actualmente en uso y por parte de quién
- Decidir qué datos y procesos (o parte de procesos) añadir o extraer de memoria
- Asignar y liberar la asignación de espacio de memoria según sea necesario.

La memoria principal es la única área de almacenamiento de gran tamaño a la que el procesador puede acceder directamente. Está formada por palabras, las cuales cada una tiene su propia dirección.

Idealmente, es deseable que los programas y los datos residan en la memoria principal de forma permanente. Usualmente, esta situación no es posible por las dos razones siguientes:

1. Normalmente, la memoria principal es demasiado pequeña como para almacenar todos los programas y datos necesarios de forma permanente.

2. La memoria principal es un dispositivo de almacenamiento volátil que pierde su contenido cuando se quita la alimentación.

Por tanto, la mayor parte de los sistemas informáticos proporcionan un segundo nivel de la jerarquía, el almacenamiento secundario como una extensión de la memoria principal. El requerimiento fundamente de este almacenamiento secundario es que se tienen que poder almacenar grandes cantidades de datos de forma permanente.

El dispositivo de almacenamiento secundario más común es el disco magnético, que proporciona un sistema de almacenamiento tanto para programas como para datos. La superficie del disco está lógicamente divida en pistas, las cuales están subdivididas en sectores. El controlador de disco determina la interacción lógica entre el dispositivo y la computadora.

Caching

Este es un principio importante que es llevado a cabo por varios niveles en una computadora (léase vía hardware y software). La idea básica consiste en copiar la información en uso desde un almacenaje lento a uno más rápido temporariamente. El almacenaje más rápido (cache) es verificado primero para determinar si la información está allí. Si está, es usada directamente del cache, lo cual aumenta la velocidad sino el dato es copiado a cache y es utilizado allí.

El cache es más pequeño que el almacenaje donde se ha traído la información. La administración del cache es un problema importante de diseño así como también su tamaño y las políticas de reemplazo.

Estructura de un Sistema Operativo

Uno de los aspectos más importantes de los sistemas operativos es la capacidad de soportar multiprogramación. La multiprogramación incrementa la utilización del CPU mediante la organización de las tareas (código o datos) de forma tal que el CPU siempre tenga que ejecutar. El sistema operativo conserva varias tareas en la memoria simultáneamente. Este conjunto de tareas puede ser un subconjunto de las tareas almacenadas en el pool de tareas, el cual contiene las tareas de todo el sistema.

El sistema operativo selecciona a través de un planificador de tareas y comienza a ejecutar una de las tareas en memoria. Eventualmente la tarea deberá esperar por otra, por caso una operación de E/S, pero en este modelo el CPU no se detiene, conmuta a otra tarea.

Los sistemas multiprogramados proveen un marco en el cual varios recursos de sistema son utilizados efectivamente, pero no son provistos para la interacción entre el humano y el sistema. El tiempo compartido es una extensión lógica de la multiprogramación. En los sistemas de tiempo compartido, el CPU ejecuta múltiples tareas intercambiando entre ellas, aunque dicho intercambio ocurre tan rápidamente que los usuarios pueden interactuar con cada programa mientras estén corriendo.

Problemas en entornos de multiprogramación y tiempo compartido

En los entornos de multiprogramación y tiempo compartido los usuarios comparten el sistema simultáneamente, esto puede conducir a varios problemas de seguridad, entre otros, uso inapropiado de recursos, ejecución fuera de orden y trabas en un deadlock. Es posible lograr un alto grado de seguridad mediante la utilización de mecanismos para la sincronización de trabajos y la comunicación. Además para evitar el uso inapropiado de los recursos, por ejemplo, que un usuario controle la CPU y la E/S todo el tiempo se puede acceder a la planificación de las tareas, asignándole un quantum q a cada usuario del sistema de tiempo compartido.

Operaciones del Sistema Operativo

Dado que tanto el sistema operativo como los usuarios comparten los mismos recursos del hardware y software del sistema, debemos asegurarnos que un error en un programa del usuario puede causar problemas solo en él. En los sistemas multitarea, varios procesos pueden resultar afectados por una falla en un programa. Por ejemplo, si un proceso queda detenido en un ciclo infinito, este ciclo puede afectar a la correcta ejecución de muchos otros procesos. Errores más sutiles pueden ocurrir en un sistema multiprogramado, donde un programa erróneo puede modificar a otros programas, los datos del otro programa o incluso puede afectar al mismo sistema operativo.

Un sistema operativo correctamente diseñado debe asegurar que un programa incorrecto no pueda causar que otros programas ejecuten incorrectamente.

Con objeto de asegurar una correcta ejecución del sistema operativo, debemos ser capaces de distinguir entre la ejecución del código del sistema operativo y aquel que es definido por el usuario. La propuesta que se brinda en la mayoría de los sistemas de computación consiste en proveer soporte por hardware que nos permite diferenciar entre los varios modos de ejecución.

Como mínimo, se requieren dos modos separados de operación: el modo usuario y el modo kernel.

Un bit, llamado bit de modo, es añadido al hardware de la computadora para indicar el modo actual: kernel (0) o usuario (1). Cuando el sistema está ejecutando en nombre de una aplicación del usuario, el mismo se encuentra en modo usuario. Sin embargo, cuando un programa del usuario requiere un servicio del sistema operativo, debe cambiar del modo usuario al modo kernel para satisfacer el requerimiento y luego retorna al modo usuario al finalizar el servicio.

Algunas instrucciones son privilegiadas, sólo se ejecutan en modo kernel.

La falta de soporte por hardware del modo dual puede provocar serios problemas en un sistema operativo. Por ejemplo, MS-DOS fue escrito para la arquitectura Intel 8088, el cual no tenía bit de modo y por lo tanto no tenía un modo dual. Un programa de usuario podría destruir el sistema operativo escribiéndolo con datos, además múltiples programas eran capaces de escribir en un dispositivo al mismo tiempo con posibles resultados desastrosos. Las versiones recientes de los CPU de Intel, como el Pentium, proveen un modo dual de operación. Consistentemente, la mayoría de los sistemas operativos actuales como Windows 2000, Windows XP y Linux toman ventaja de esta iniciativa y ofrecen mayor protección para el sistema operativo.

Una vez que la protección por hardware está instalada, los errores que violan los modos son detectados por el hardware. Estos errores son comúnmente manejados por el sistema operativo.

Si un programa de usuario falla de alguna manera, como por ejemplo, intenta ejecutar una instrucción ilegal o acceder a un espacio de la memoria que no está contenido en el espacio de direccionado del usuario entonces el hardware transferirá el aviso al sistema operativo. Este aviso es una excepción que transfiere el control a través del vector de interrupciones al SO, tal como lo haría una interrupción. Cuando ocurre un error de estas características, el sistema operativo debe abortar al programa de usuario el programa del usuario anormalmente. A continuación se muestra un cartel informando del error y un volcado de la memoria del programa. Este mecanismo ayuda a prevenir lazos infinitos.

Administración de procesos

Un proceso es un programa en ejecución. Es una unidad de trabajo dentro del sistema. Mientras que un programa es una entidad pasiva, el proceso es una entidad activa.

Un proceso necesita para llevar a cabo su tarea ciertos recursos, entre los que se incluyen tiempo de CPU, memoria, archivos y dispositivos de E/S. Estos recursos se proporcionan al proceso en el momento de crearlo o se le asignan mientras se está ejecutando. Además de los diversos recursos físicos y lógicos que un proceso obtiene en el momento de su creación, pueden pasársele diversos datos de inicialización. Por ejemplo, consideremos un proceso cuya función sea la de mostrar el estado de un archivo en la pantalla de un terminal. A ese proceso le proporcionamos como entrada el nombre del archivo y el proceso ejecutaría las apropiadas instrucciones y llamadas al sistema obtener y mostrar en el terminal la información deseada. Cuando el proceso termina, el sistema operativo reclama todos los recursos reutilizables.

Los procesos de hilo siempre tienen un contador de programa especificando la locación de la próxima instrucción a ejecutar. Los procesos multihilados tienen un contador de programa por hilo. La ejecución de un proceso debe ser secuencial: la CPU ejecuta una instrucción del proceso después de otra, hasta completarlo. Además, en cualquier instante, se estará ejecutando como mucho una instrucción en nombre del proceso. Por lo tanto, aunque pueda haber

dos procesos asociados con el mismo programa, se considerarían no obstante como dos secuencias de ejecución separadas. Cada sistema consta de una colección de procesos, siendo algunos de ellos procesos del sistema operativo y el resto procesos del usuario. Todos estos pueden, potencialmente, ejecutarse en forma concurrente, por ejemplo multiplexando la CPU cuando sólo se disponga de una.

Actividades de la Administración de Procesos

El sistema operativo es responsable de las siguientes actividades en lo que se refiere a la gestión de procesos:

- Creación y eliminación de procesos
- Suspensión y reactivación de procesos
- Provisión de mecanismos para:
    - ✓ Sincronización de procesos
    - ✓ Comunicación de procesos
    - ✓ Manejo de interbloqueos

Administración del almacenaje

Para que el sistema sea cómodo para los usuarios, el sistema operativo proporciona una visión lógica y uniforme del sistema de almacenamiento de la información. El sistema operativo abstrae las propiedades físicas de los dispositivos de almacenamiento y define una unidad de almacenamiento lógico, el archivo. El sistema operativo asigna los archivos a los soportes físicos y accede a dichos archivos a través de los dispositivos de almacenamiento. Las computadoras pueden almacenar la información en diferentes tipos de medios físicos. Cada uno de estos medios tiene sus propias características y organización física. Cada medio se controla mediante un dispositivo, tal como una unidad de disco o unidad de cinta, que también tiene sus propias características. Estas propiedades incluyen la velocidad de acceso, la capacidad, la velocidad de transferencia de datos y el método de acceso (secuencial o aleatorio).

Los archivos, usualmente están organizados en directorios, nuevamente, para hacer más fácil su uso. Por último, cuando varios usuarios tienen acceso a los archivos, puede ser deseable controlar quién y en qué forma accede a los archivos. El sistema operativo es el responsable de las siguientes actividades en lo que se refiere a la gestión de archivos.

- Creación y destrucción de archivos y directorios
- Soporte de primitivas para el manejo de archivos y directorios
- Mapeo de archivos sobre el almacenaje secundario
- Respaldo sobre medios de almacenaje estables.

Administración del Almacenaje Secundario

Los discos son usados para almacenar datos que no entran en memoria principal o para datos que tienen que ser guardados un largo periodo de tiempo. El sistema operativo es responsable de las siguientes actividades en lo que se refiere a gestión de discos:

- Administración del espacio libre
- Alocación del almacenaje
- Planificación del disco, aunque ya no la realiza el sistema operativo hoy en día.

Dado que el almacenamiento secundario se lo usa con frecuencia, debe emplearse de forma eficiente. La velocidad final de operación de una computadora puede depender de las velocidades del subsistema de disco y de los algoritmos que manipulan dicho subsistema.

Sin embargo, hay muchos usos para otros sistemas de almacenamiento más lentos y más baratos que el almacenamiento secundario. La realización de copias de seguridad de los datos del disco, el almacenamiento de datos raramente utilizados y el almacenamiento definitivo a largo plazo son algunos ejemplos.
Las unidades de cinta magnética y sus citas y las unidades de CD y DVD y sus discos son dispositivos de almacenamiento terciario. Este nivel de almacenamiento no es crucial para el rendimiento del sistema aunque también es necesario gestionarlo.

Subsistema de E/S

Uno de los propósitos del Sistema Operativo es ocultar las peculiaridades de los dispositivos de hardware de los usuarios.

Los subsistemas de E/S son responsables de:

- Administración de memoria de las E/S incluyendo el buffering (almacena los datos temporariamente mientras son transferidos), caching (almacena partes de datos en almacenamiento rápido por rendimiento), spooling (el solapado de la salida de un job con la entrada a otros).
- Brindar una interfaz general de drivers de dispositivos
- Drivers específicos para dispositivos de hardware

Estructura del Sistema Operativo - Módulo 2

Servicios del Sistema Operativo

El sistema presta ciertos servicios a los programas y a los usuarios de dichos programas. Por supuesto, los servicios específicos que se suministran difieren de un sistema operativo a otro, pero podemos identificar perfectamente una serie de clases comunes. Estos servicios del sistema operativo se proporcionan para comodidad del programador, con el fin de facilitar la tarea de desarrollo.

Un cierto conjunto de servicios del sistema operativo proporciona funciones que resultan útiles al usuario:

Interfaz de usuario: Casi todos los sistemas operativos disponen de una interfaz de usuario (UI, user interface), que puede tomar diferentes formas.

Uno de los tipos existentes es la interfaz de línea de comandos, que usa comandos de texto y algún tipo de método para introducirlos, es decir un programa que permite introducir y editar los comandos.

Algunos sistemas operativos incluyen el intérprete de comandos en el kernel. Otros, como Windows XP y Unix, tratan al intérprete como un programa especial que ejecuta cuando un trabajo es inicializado o cuando un usuario inicia sesión por primera vez por lo tanto lo separan del kernel.

Además lo separan del kernel para facilitar la actualización del intérprete. En los sistemas que disponen de varios intérpretes de comandos entre los que elegir, los intérpretes se conocen como shells.

La función principal del intérprete de comandos es obtener y ejecutar el siguiente comando especificado por el usuario. Muchos de los comandos que se proporcionan en este nivel se utilizan para manipular archivos: creación, borrado, listado, impresión, copia, ejecución, etc.

Otro tipo destacable es la interfaz de proceso por lotes, en la que los comandos y las directivas para controlar dichos comandos se introducen en archivos, y luego dichos archivos se ejecutan.

Habitualmente, se utiliza una interfaz grafica de usuario; en este caso, la interfaz del sistema es un sistema de ventanas, con un dispositivo señalador para dirigir la E/S, para elegir opciones en los menús y para realizar otras selecciones, y con un teclado para introducir texto.

Ejecución de programas: El sistema tiene que poder cargar un programa en la memoria y ejecutar dicho programa. Todo programa debe poder terminar su ejecución, de forma normal o anormal (indicando un error).

Operaciones de E/S: Un programa en ejecución puede necesitar llevar a cabo operaciones de E/S dirigidas a un archivo o a un dispositivo de E/S. Por cuestiones de eficiencia y protección, los usuarios no pueden, normalmente, controlar de modo directo los dispositivos de E/S; por tanto, el sistema operativo debe proporcionar medios para realizar la E/S.

Manipulación del sistema de archivos: El sistema de archivos tiene una importancia especial. Obviamente, los programas necesitan leer y escribir en archivos y directorios. También necesitan crearlos y borrarlos usando su nombre, realizar búsquedas en un determinado archivo o presentar la información contenida en un archivo. Por último, algunos programas incluyen mecanismos de gestión de permisos para conceder o denegar el acceso a los archivos basándose en quién sea el propietario del archivo.

Comunicaciones: Hay muchas circunstancias en las que un proceso necesita intercambiar información con otro. Dicha comunicación puede tener lugar entre procesos que estén ejecutándose en la misma computadora o entre procesos que se ejecuten en computadoras diferentes conectadas a través de una red. Las comunicaciones se pueden implementar utilizando memoria compartida o mediante paso de mensajes, procedimiento éste en el que el sistema operativo transfiere paquetes de información entre unos procesos y otros.

Detección de errores: El sistema operativo necesita detectar constantemente los posibles errores. Estos errores pueden producirse en el hardware del procesador y de memoria (como, por ejemplo, un error de memoria o un fallo de la alimentación) en un dispositivo de E/S. Para cada tipo de error, el sistema operativo debe llevar a cabo la acción apropiada para asegurar un funcionamiento correcto y coherente.

Hay disponible otro conjunto de funciones del sistema de operativo que no están pensadas para ayudar al usuario, sino para garantizar la eficiencia del propio sistema vía recursos compartidos.

Alocación de recursos: Cuando hay varios usuarios, o hay varios trabajos ejecutándose al mismo tiempo, deben asignarse a cada uno de ellos los recursos necesarios. El sistema operativo gestiona muchos tipos diferentes de recursos, algunos (como los ciclos de CPU, la memoria principal y el espacio de almacenamiento de archivos) pueden disponer de código software especial que gestione su asignación, mientras que otros (como los dispositivos de E/S) pueden tener código que gestión de forma mucho más general su solicitud y liberación. Por ejemplo, para poder determinar cuál es el mejor modo de usar la CPU, el sistema operativo dispone de rutinas de planificación de la CPU que tienen en cuenta la velocidad del procesador, los trabajos que tienen que ejecutarse, el número de registros disponibles y otros factores. También puede haber rutinas para asignar impresoras, módems, unidades de almacenamiento USB y otros dispositivos periféricos.

Contabilidad: Normalmente conviene hacer un seguimiento de qué usuarios emplean que clase de recursos de la computadora y en qué cantidad. Este registro se puede usar para propósitos contables o simplemente para acumular estadísticas de uso. Estas estadísticas pueden ser una herramienta valiosa para aquellos investigadores que deseen reconfigurar el sistema con el fin de mejorar los servicios informáticos.

Protección y seguridad:

La protección es cualquier mecanismo que controle el acceso de procesos y usuarios a los recursos definidos por un sistema informático. Este mecanismo debe proporcionar los medios para la especificación de los controles que hay que imponer y para la aplicación de dichos controles.

Y la seguridad es la defensa del sistema contra ataques internos y externos incluyendo la defensa de los dispositivos de E/S. Esto abarca un amplio rango de categorías, como DoS, worms, virus, robo de identidad, y robo de servicios.

Llamadas al sistema

Las llamadas al sistema proporcionan una interfaz con la que poder invocar los servicios que el sistema operativo ofrece. Estas llamadas, generalmente, están disponibles como rutinas escritas en C y C++, aunque determinadas tareas de bajo nivel, como por ejemplo aquellas en las que se tiene que acceder directamente al hardware, pueden necesitar escribir con instrucciones de lenguaje ensamblador.

Considerando que copiar los datos de un archivo a otro requiere de una gran cantidad de llamadas al sistema y estas y otras llamadas al sistema varían entre los sistemas impidiendo la portabilidad, normalmente, los desarrolladores de aplicaciones se abstraen utilizando una API (Application Program Interface). La API especifica un conjunto de funciones que el programador de aplicaciones puede usar, indicándose los parámetros que hay que pasar cada función y los valores de retorno que el programador debe esperar. Las tres API más comunes son Win32 para Windows, POSIX API para sistemas POSIX (incluyendo virtualmente todas las versiones de UNIX, Linux y Mac OS X), y Java Api para la máquina virtual Java.

Pasaje de parámetros en Llamadas a Sistema

Las llamadas al sistema se llevan a cabo de diferentes formas, dependiendo de la computadora que se utilice. A menudo, se requiere más información que simplemente identificar la llamada a sistema deseada. El tipo exacto y la cantidad de información necesaria varían según el sistema operativo y la llamada concreta que se efectué.

Para pasar parámetros al sistema operativo se emplean tres métodos generales. El más sencillo de ellos consiste en pasar los parámetros en una serie de registros.

Sin embargo, en algunos casos, puede haber más parámetros que registros disponibles. En estos casos, generalmente, los parámetros se almacenan en un bloque o tabla, en memoria, y la dirección del bloque se pasa como parámetro en un registro. Este es el método que utilizan Linux y Solaris. En programa también puede colocar, o insertar, los parámetros en la pila y el sistema operativo se encargara de extraer de la pila esos parámetros. Algunos sistemas operativos prefieren el método del bloque o el de la pila, porque no limitan el número o la longitud de los parámetros que se pueden pasar.

Tipos Llamadas a Sistema

Las llamadas al sistema pueden agruparse de forma muy general en las siguientes categorías:

- Control de procesos
- Administración de archivos
- Administración de dispositivos
- Mantenimiento de la información
- Comunicaciones
- Protección

Programas de Sistema

Los programas de sistema proporcionan un cómodo entorno para desarrollar y ejecutar programas. Algunos de ellos son, simplemente interfaces de usuario para las llamadas al sistema; otros son considerablemente más complejos. Pueden dividirse en las siguientes categorías:

Administración de archivos: Estos programas crean, borrar, copian, imprimen, vuelcan, listan y, de forma general, manipulan archivos y directorios.

Información de estado: Algunos programas simplemente solicitan al sistema la fecha, la hora, la cantidad de memoria o de espacio de disco disponible, el número de usuarios o información de estado similar. Otros son más complejos y proporcionan información detallada sobre el rendimiento, los inicios de sesión y los mecanismos de depuración e inclusive algunos sistemas también cuentan con un registro, que se usa para almacenar y recuperar información de configuración.

Modificación de archivos: Puede disponerse de varios editores de texto para crear y modificar el contenido de los archivos almacenados en el disco o en otros dispositivos de almacenamiento. También puede haber comandos especiales para explorar el contenido de los archivos en busca de un determinado dato o para realizar cambios en el texto.

Soporte de lenguajes de programación: Con frecuencia, con el sistema operativo se proporcionan al usuario compiladores, ensambladores, depuradores e intérpretes para los lenguajes de programación habituales.

Carga y ejecución de programas: Una vez que el programa se ha ensamblado o compilado, debe cargarse en memoria para poder ejecutarlo. El sistema puede proporcionar cargadores absolutos, cargadores reubicables, editores de montaje y cargadores de sustitución.

Comunicaciones: Estos programas proporcionan los mecanismos para crear conexiones virtuales entre procesos, usuarios y computadoras. Permiten a los usuarios enviar mensajes a las pantallas de otros, explorar páginas web, enviar mensajes de correo electrónico, iniciar una sesión de forma remota o transferir archivos de una máquina a otra.

Cabe aclarar, que la visión que tienen la mayoría de los usuarios del sistema operativo está dada por los programas de sistema y no por las llamadas al sistema.

Diseño e Implementación de un Sistema Operativo

El diseño e implementación de un SO no está estructurado pero hay algunos enfoques que han resultado exitosos. La estructura interna de los diferentes SOs puede variar ampliamente.

El primer problema al diseñar un sistema es el de definir los objetivos y especificaciones. En el nivel más alto, el diseño del sistema se verá afectado por la elección del hardware y el tipo de sistema.

Más allá de este nivel superior de diseño, se pueden dividir los objetivos en dos grupos básicos: objetivos de los usuarios y objetivos del sistema. Los usuarios desean ciertas propiedades obvias en un sistema: debe ser cómodo de utilizar, fácil de aprender y de usar, fiable, seguro, rápido. Por supuesto, estas especificaciones no son particularmente útiles para el diseño del sistema, ya que no existe un acuerdo general sobre cómo llevarlas a la práctica.

Un conjunto de requisitos similares puede ser definido por aquellas personas que tienen que diseñar, crear, mantener y operar el sistema. El sistema debería ser fácil de diseñar, implementar y mantener; debería ser flexible, fiable, libre de errores y eficiente. De nuevo, estos requisitos son vagos y puede interpretarse de diversas formas.

Un principio importante es el de separar las políticas de los mecanismos. Los mecanismos determinan como hacer algo; las políticas determinan qué hacer.

La separación de políticas y mecanismos es importante por cuestiones de flexibilidad. Las políticas probablemente cambien de un sitio a otro con el paso del tiempo. En el caso peor, cada cambio en una política requerirá un cambio en el mecanismo subyacente, sería, por tanto, deseable un mecanismo general insensible a los cambios de política. Un cambio de política requeriría entonces la redefinición de sólo determinados parámetros del sistema. Por ejemplo, considere un mecanismo para dar prioridad a ciertos tipos de programas: si el mecanismo está apropiadamente separado de la política, puede utilizarse para dar soporte a una decisión política que establezca que los programas que hacen un uso intensivo de la E/S tenga prioridad sobre los que hacen un uso intensivo de la CPU, o para dar soporte a la política contraria.

Estructura del sistema operativo

La ingeniería de un sistema tan grande y complejo como un sistema operativo moderno debe hacerse para que el sistema funcione apropiadamente y pueda modificarse con facilidad.

Estructura simple

Muchos sistemas comerciales no tienen una estructura bien definida. Frecuentemente, tales sistemas operativos comienzan siendo sistemas pequeños, simples y limitados y luego crecen más allá de su ámbito original; MS-DOS es un ejemplo de un sistema así. Fue escrito para proporcionar la máxima funcionalidad en el menor espacio posible, por lo que no fue dividido en módulos de forma cuidadosa.

En MS-DOS, las interfaces y niveles de funcionalidad no están separados. Por ejemplo, los programas de aplicación pueden acceder a las rutinas básicas de E/S para escribir directamente en la pantalla y en las unidades de disco. Tal libertad hace que MS-DOS sea vulnerable a programas erróneos (o maliciosos), lo que hace que el sistema completo falle cuando los programas de usuario fallan.

Otro ejemplo de estructuración limitada es el sistema operativo UNIX original. UNIX es otro sistema que inicialmente estaba limitado por la funcionalidad del hardware. Consta de dos partes separadas: el kernel y los programas del sistema. El kernel se divide en una serie de interfaces y controladores de dispositivo, que se han ido añadiendo y ampliando a lo largo de los años, a medida que UNIX ha ido evolucionando. El mismo consiste de todo lo que está debajo de la interfaz de los system calls y encima del hardware. Por caso, contiene el sistema de archivos, la planificación del CPU, manejo de memoria, y otras funciones del sistema operativo; un gran número de funciones en un solo nivel. En resumen, un estructura monolítica difícil de implementar y mantener.

Enfoque por capas

En este enfoque el sistema operativo está dividido en un número de capas (niveles), cada una construida sobre el tope de otra. La capa inferior (nivel 0), es el hardware; la más alta (cana N) es la interfaz de usuario. En forma modular, las capas son seleccionadas de manera que cada una usa funciones y servicios de las capas inferiores. Los implementadores tienen más libertad cambiar el funcionamiento interno del sistema y crear sistemas operativos modulares. Con el método de diseño top-down, se determinan las características y la funcionalidad globales y se separan en componentes. La ocultación de los detalles a ojos de los niveles superiores también es importante, dado que deja libres a los programadores para implementar las rutinas de bajo nivel como prefieran, siempre que la interfaz externa de la rutina permanezca invariable y la propia rutina realice la tarea anunciada. El sistema operativo puede entonces mantener un control mucho mayor sobre la computadora y sobre las aplicaciones que hacen uso de dicha computadora.

La principal dificultad con el método de niveles es la definir apropiadamente los diferentes niveles. Dado que un nivel sólo puede usar los servicios de niveles inferiores, es necesario realizar una planificación cuidadosa. Un último problema

con las implementaciones por niveles es que tienden a ser menos eficientes que otros tipos de implementación. Por ejemplo, cuando un programa de usuario ejecuta una operación de E/S, realiza una llamada al sistema que será capturada por el nivel de E/S, el cual llamara al nivel de gestión de memoria, el cual a su vez llamara al nivel de planificación del CPU, que pasará a continuación la llamada al hardware. Cada nivel añade un retardo por lo que el resultado neto es una llamada al sistema que tarda más en ejecutarse que en un sistema sin niveles.

Microkernel

Este modelo estructura el sistema operativo eliminando todos los componentes no esenciales del kernel e implementándolos como programas del sistema y de nivel de usuario; el resultado es un kernel más pequeño. No hay consenso en lo que se refiere a qué servicios deberían permanecer en el kernel y cuáles deberían implementarse en el espacio de usuario. Sin embargo, los microkernels proporcionan una gestión de la memoria y de los procesos mínima, además de un mecanismo de comunicaciones.

La función principal del microkernel es proporcionar un mecanismo de comunicaciones entre el programa cliente y los distintos servicios que se ejecutan también en el espacio de usuario. La comunicación se proporciona mediante paso de mensajes. Por ejemplo, si el programa cliente desea acceder a un archivo, debe interactuar con el servidor de archivos. El programa cliente y el servicio nunca interactúan directamente, sino que se comunican de forma indirecta intercambiando mensajes con el microkernel.

Otra ventaja del método es la facilidad para ampliar el sistema operativo. Todos los servicios nuevos se añaden al espacio de usuario y, en consecuencia, no requiere que se modifique el kernel. El microkernel también proporciona más seguridad y facilidad, dado que la mayor parte de los servicios se ejecutan como procesos de usuario, en lugar de procesos del kernel. Si un servicio falla, el resto del sistema operativo no se ve afectado. Por último es más fácil de portar el sistema operativo a otras arquitecturas.

Desafortunadamente, los microkernels pueden sufrir deficiencias en su performance debido a la sobrecarga de funciones del sistema operativo.

Módulos

Quizá la mejor metodología actual para diseñar sistemas operativos es la que usa las técnicas de programación orientada a objetos para crear un kernel modular. En este caso, el kernel dispone de un conjunto de componentes fundamentales y enlaza dinámicamente los servicios adicionales, bien durante el arranque o en tiempo de ejecución.

Un diseño así permite proporcionar servicios básicos y también permite implementar ciertas características dinámicamente. Por ejemplo, se pueden añadir al kernel controladores de bus y de dispositivos para hardware específico y puede agregarse como módulo cargable el soporte para diferentes sistemas de archivos. El resultado global es similar a un sistema de niveles, en el sentido de que cada sección del kernel tiene interfaces bien definidas y protegidas, pero es más flexible que un sistema de niveles, porque cualquier módulo puede llamar a cualquier otro módulo. Además, el método es similar a la utilización de un microkernel, ya que el módulo principal sólo dispone de las funciones esenciales y de los conocimientos sobre como cargar y comunicar con otros módulos; sin embargo, es más eficiente que un microkernel, ya que los módulos no necesitan invocar un mecanismo de pasaje de mensajes para comunicarse.

El sistema operativo Mac OS X de las computadoras Apple utiliza una estructura hibrida. Mac OS X estructura el sistema operativo por niveles en la que uno de esos niveles es el microkernel Mach.

Máquinas Virtuales

La idea fundamental que subyace a una máquina virtual es la de abstraer el hardware de la computadora, formando varios entornos de ejecución diferentes, creando así la ilusión de que cada entorno de ejecución está operando en su propia computadora privada.

Con los mecanismos de planificación del CPU y las técnicas de memoria virtual, un sistema operativo puede crear la ilusión de que un proceso tiene su propio procesador con su propia memoria virtual. Normalmente, un proceso utiliza características adicionales, tales como llamadas al sistema y un sistema de archivos, que el hardware básico no proporciona. El método de maquina virtual no proporciona ninguna de estas funcionalidad adicionales, sino que proporciona una interfaz que es idéntica al hardware básico subyacente. De esta forma, cada invitado es provisto con una copia virtual de la computadora.

Beneficios de las máquinas virtuales

En primer lugar tanto al usuario como al diseñador de los sistemas operativos la utilización de maquinas virtuales les permiten compartir el mismo hardware entre múltiples sistemas operativos que ejecutan concurrentemente.

Por otro lado, el concepto de máquina virtual tiene múltiples ventajas. En primer lugar, existe una protección completa de los múltiples recursos del sistema. Cada máquina virtual se encuentra complemente aislada de las otras por lo que desaparecen los problemas de protección.

Las máquinas virtuales son un medio perfecto para la investigación y el desarrollo de sistemas operativos. Normalmente, cambiar un sistema operativo es una tarea dificultosa. Los SO están compuestos por una extensa y compleja colección de programas y es difícil asegurarse que los cambios en una parte no provoquen fallas en otras. Debido a que el SO ejecuta en modo kernel, un cambio incorrecto realizado en un puntero puede provocar que se destruya completamente todo el sistema de archivos. Debido a esto, es requisito testear todos los cambios realizados a un SO cuidadosamente.

El sistema operativo no obstante, ejecuta y controla la máquina completa. Es por esto, que el sistema debe ser detenido mientras los cambios son diseñados y testeados. Debido a esto provoca que el sistema no esté disponible para el usuario, el tiempo de desarrollo se agendaba para la noche o los fines de semana.

Con la invención de las maquinas virtuales, los problemas descriptos fueron eliminados. Los programadores de sistemas reciben su propia maquina virtual, y el desarrollo se realiza en una máquina virtual en lugar de una física.

Generación del sistema operativo

Es posible diseñar, programar e implementar un sistema operativo específicamente para una maquina situada en un único lugar. Más frecuentemente, los sistemas operativos son diseñados para funcionar en una clase de computadoras, en una multiplicidad de sitios y con diversas configuraciones de periféricos. El sistema luego debe ser configurado o generado para cada lugar especifico, un proceso conocido como generación del sistema.

El sistema operativo es normalmente distribuido en un disco o CDROM. Para generar el sistema, usamos un programa especial. El sysgen lee el contenido de un archivo dado, o consulta al operador del sistema sobre la configuración de hardware, o prueba directamente el hardware para determinar cuáles son sus componentes.

Procesos - Módulo 3

Concepto de Proceso

Un proceso es un programa en ejecución. Además del código, un proceso incluye también la actividad actual, que queda representada por el valor del contador de programa y por los contenidos de los registros del procesador. Generalmente, un proceso incluye también la pila del proceso, que contiene datos temporales, y una sección de datos, que contiene

variables globales. El proceso puede incluir asimismo, un cúmulo de memoria, que es la memoria que se le asigna dinámicamente al proceso en tiempo de ejecución.

Estado de los Procesos

A medida que se ejecuta un proceso, el proceso va cambiando de estado. El estado de un proceso se define, en parte, según la actividad actual de dicho proceso. Cada proceso puede estar en uno de los siguientes estados:

- Nuevo: El proceso es creado
- Corriendo: Las instrucciones están siendo ejecutadas
- Espera: El proceso está esperando que ocurra algún evento.
- Listo: El proceso está esperando ser asignado a la CPU.
- Terminado: El proceso ha finalizado su ejecución.

Es importante remarcar que sólo puede haber un proceso ejecutándose en cualquier procesador en cada instante de tiempo concreto. Sin embargo, puede haber muchos procesos en los estados de espera, listo y terminado.

Bloque de Control de Procesos (PCB)

Cada proceso es representado en el SO por el bloque de control de proceso o process control block. Este contiene diversos datos asociados con un proceso específico, tales como:

- Estado del proceso: El estado puede ser nuevo, listo, CPU, I/O, terminado.
- Program counter: Este contador indica la dirección de la próxima instrucción a ser ejecutada por el proceso.
- Registros del CPU: Los registros pueden variar en número y tipo en función de la arquitectura. Estos incluyen acumuladores, registros índices, stack pointers, registros de propósito general e información sobre sentencias condicionales.
- Información de planificación del CPU: Este registro incluye información tal como el valor de los registros base y limite, las tablas de páginas o las tablas de segmento dependiendo de la memoria usada por el SO.
- Información de conteo: Este campo incluye la cantidad de CPU y tiempo real utilizado, los límites de tiempo, los números de cuenta, los números de proceso, etc.
- Información de E/S: Este conserva la lista de los dispositivos de E/S alocados por el proceso, la lista de los archivos abiertos, etc.

En resumen, el PCB sirve como un repositorio de información que varía de proceso en proceso.

Conmutación de CPU de Proceso a Proceso

Una interrupción provoca que el sistema operativo cambie la tarea que actualmente se encuentra en el CPU y ejecuta una rutina del kernel. Cuando esto ocurre, el sistema necesita salvar el contexto del proceso que actualmente se encuentra en ejecución de forma tal que pueda restablecerlo cuando el otro proceso haya terminado. El contexto del proceso está representado por su PCB.

Esta tarea es conocida como cambio de contexto. Cuando ocurre este evento, el kernel salva el contexto del proceso antiguo en el PCB y carga el contexto del nuevo proceso a correr. El cambio de contexto representa un overhead en tiempo ya que el sistema no realiza nada útil mientras este transcurre.

La velocidad con la que se realiza varía de computadora en computadora, dependiendo de la velocidad de la memoria, el número de registros que deben ser copiados y la existencia de instrucciones especiales para tal fin.

Colas de Planificación de Procesos

Supongamos que el proceso realiza una petición de E/S a un dispositivo compartido como un disco. Debido a que existen diversos procesos en el sistema, el disco podría estar ocupado procesando una petición de alguno de ellos. En este caso,

el proceso deberá esperar al disco. La lista de procesos que esperan a un dispositivo de E/S particular recibe el nombre de cola de dispositivos.

Una representación común para el problema de la planificación de los procesos es el diagrama de colas.

Un proceso nuevo es inicialmente alojado en la cola de listos. Luego queda a la espera hasta que es elegido para entrar en ejecución o es despachado. Una vez que el proceso está en el CPU y está ejecutando puede ocurrir una de las siguientes situaciones:

- El proceso puede solicitar una operación de E/S y por lo tanto es alojado en la cola de E/S.
- El proceso puede crear un nuevo subproceso y esperar a su terminación.
- El proceso puede ser sacado forzosamente de la CPU, debido a una interrupción y es colocado nuevamente en la cola de listos.

Un proceso continua en este ciclo hasta que finaliza, momento en el cual es removido de todas las colas y su PCB y recursos alocados son liberados.
El sistema operativo debe seleccionar, por motivos de planificación, los procesos de esas colas de alguna forma. La selección es realizada por el planificador adecuado.

Planificadores de Procesos

Con frecuencia, en un sistema batch, se presentan más procesos que aquellos que pueden ser ejecutados inmediatamente.

Estos procesos son encolados en un dispositivo de almacenamiento masivo para su futura ejecución. El planificador de largo término o planificador de trabajos, selecciona que procesos deberían ser puestos en la cola de listos. El planificador de corto plazo o planificador de CPU, en cambio, selecciona entre los procesos que están en la cola de listos y les adjudica el CPU a uno de ellos.

La principal diferencia entre estos dos planificadores versa en torno a la frecuencia de ejecución. El planificador de corto término debe seleccionar un nuevo proceso para el CPU con frecuencia. Dado que un proceso puede ejecutar por unos pocos milisegundos y luego quedar a la espera de una operación de E/S el planificador debe ser rápido.

Por otro lado, el planificador de largo término se ejecuta con mucha menos frecuencia, pueden existir minutos, entre la creación de un proceso y el siguiente.

El planificador de largo término controla el grado de multiprogramación, esto es, el número de procesos en la memoria. Es muy importante que este último realice una selección cuidadosa del proceso a ejecutar. En general, los procesos pueden ser clasificados como aquellos que están vinculados a la E/S o al CPU. Un proceso que está ligado a la E/S o limitado por E/S es aquel que consume mayor tiempo realizando operaciones de E/S que computando en el CPU. De manera opuesta se define a un proceso ligado a la CPU o limitado por la CPU.

De esto se deduce que el planificador de largo término debe realizar una selección bien balanceada entre los procesos ligados del CPU y aquellos ligados a la E/S, como sigue, si todos los procesos elegidos por el planificador son aquellos que están vinculados a la E/S, entonces la cola de listos estará siempre vacía, y el planificador tendrá poco para hacer. Si por el contrario la mayoría de los procesos son aquellos ligados al CPU la cola de E/S estará siempre vacía y los dispositivos caerán en desuso.

Creación de Procesos

Los sucesos que normalmente llevan a la creación de un proceso son provenientes del accionar del usuario, por ejemplo, al ejecutar un programa o solicitar alguna acción sobre los dispositivos de E/S.

Un proceso puede dar origen a diversos procesos nuevos, a través de la llamada al sistema crear-proceso durante la ejecución. El proceso generador es llamado proceso padre, y los nuevos procesos se denominan hijos de este último. Cada uno de estos nuevos procesos podrá oportunamente crear nuevos procesos formando un árbol de procesos.

La mayoría de los sistemas operativos identifican los procesos mediante un identificador de proceso unívoco o pid, que normalmente es un número entero.

En general, un proceso requerirá ciertos recursos (tiempo de CPU, memoria, archivos y dispositivos de E/S) para realizar su tarea. Cuando un proceso crea un subproceso, este último puede ser capaz de obtener sus recursos directamente desde el SO, o puede estar restringido a un subconjunto de recursos del proceso padre.

El padre puede tener que particionar sus recursos entre sus hijos, o puede ser capaz de compartir algunos recursos (como la memoria o los archivos) entre varios hijos.

Restringir a un proceso hijo a un subconjunto de los recursos del proceso padre previene que cualquier proceso sobrecargue al sistema cuando existan demasiados subprocesos.

En conjunto con los recursos físicos y lógicos que un proceso adquiere cuando es creado, datos de inicialización son enviados a los procesos hijos a través de sus padres.

Cuando un proceso crea un nuevo proceso, existen dos alternativas en cuanto a lo que ejecución respecta:

- El padre continúa su ejecución concurrentemente con sus hijos.
- El padre queda a la espera de que algunos o todos sus hijos terminen

También existen dos posibilidades en cuanto al espacio de direccionado de los nuevos procesos:

- El proceso hijo es un duplicado del proceso padre (tiene los mismos datos y programa que su padre)
- El proceso hijo tiene un nuevo programa cargado en él.

La primera posibilidad permite que el proceso padre se comunique fácilmente con el proceso hijo.

En UNIX por ejemplo la llamada al sistema que crea un nuevo proceso es fork. Y la llamada al sistema execve es usada después de fork para reemplazar el espacio de memoria del proceso con un nuevo programa.

Terminación de los Procesos

Un proceso termina cuando ejecuta su última instrucción y pide al sistema operativo que lo elimine usando la llamada al sistema exit (). En este momento, el proceso puede devolver un valor de estado a su proceso padre a través de la llamada al sistema wait (). El sistema operativo libera la asignación de todos los recursos del proceso, incluyendo la memoria física, los archivos abiertos y los búferes de E/S.

La terminación puede producirse también en otras circunstancias. Un padre puede terminar la ejecución de uno de sus hijos por diversas razones, como por ejemplo, las siguientes:

- El proceso hijo ha excedido el uso de algunos de los recursos que se le han asignado. Para determinar si tal cosa ha ocurrido, el padre debe disponer de un mecanismo para inspeccionar el estado de sus hijos.
- La tarea asignada al proceso hijo no es más requerida.

- El padre abandona el sistema, y el sistema operativo no permite que un proceso hijo continúe si su padre ha terminado. En tales sistemas, si un proceso termina (sea normal o anormalmente), entonces todos sus hijos también deben terminarse. Este fenómeno, conocido como terminación en cascada, normalmente lo inicia el sistema operativo.

Procesos Cooperativos

Los procesos que se ejecutan concurrentemente pueden ser procesos independientes o procesos cooperativos. Un proceso es independiente si no puede afectar o verse afectado por los restantes procesos que se ejecutan en el sistema. Cualquier proceso que no comparte datos con ningún otro proceso es un proceso independiente. Un proceso es cooperativo si puede afectar o verse afectado por los demás procesos que se ejecutan en el sistema. Evidentemente, cualquier proceso que comparte datos con otros procesos es un proceso cooperativo.

Hay varias razones para proporcionar un entorno que permita la cooperación entre procesos, entre otros, información compartida, aceleración de la computación, modularidad y conveniencia.

Problema del Productor-Consumidor

El problema del productor consumidor, es un paradigma comúnmente utilizando para los procesos cooperativos. Un proceso productor genera información que consume un proceso consumidor.

Generalmente, pensamos en un servidor como en un productor y en un cliente en un consumidor.

Una solución para el problema del productor-consumidor es utilizar mecanismos de memoria compartida. Para permitir que los procesos de productor y consumidor se ejecuten en forma concurrente, debemos tener disponible un buffer de elementos que pueda rellenar el productor y vaciar el consumidor. Este búfer claramente residirá en una región de la memoria que será compartida por ambos procesos, consumidor y productor. El productor y el consumidor deben estar sincronizados, de modo que el consumidor no intente consumidor un elemento que todavía no haya sido producido.

Pueden emplearse dos tipos de búferes. El sistema de búfer no limitado no pone límites al tamaño de esa memoria compartida. El consumidor puede tener que esperar para obtener elementos nuevos, pero el productor siempre puede generar nuevos elementos. El sistema de buffer limitado establece un tamaño de búfer fijo. En este caso, el consumidor tiene que esperar si el búfer está vacío y el productor tiene que esperar si el búfer esta lleno.

Modelos de Comunicación

La cooperación entre procesos requiere mecanismos de comunicación interprocesos (IPC, interprocess comunication) que les permitan intercambiar datos e información. Existen dos modelos fundamentales de comunicación interprocesos: memoria compartida y pasaje de mensajes. En el modelo de memoria compartida, se establece una región de la memoria para que sea compartida por los procesos cooperativos. De este modo, los procesos pueden intercambiar información leyendo y escribiendo datos en la zona compartida. En el modelo de paso de mensajes, la comunicación tiene lugar mediante el intercambio de mensajes entre los procesos cooperativos.

El paso de mensajes resulta útil para intercambiar pequeñas cantidades de datos, ya que no existe la necesidad de evitar conflictos. El paso de mensajes también es más fácil de implementar que el modelo de memoria compartida como mecanismo de comunicación entre computadoras. La memoria compartida permite una velocidad máxima y una mejor comunicación, ya que puede realizarse a velocidad de memoria cuando se hace en una misma computadora. La memoria compartida es más rápida que el paso de mensajes, ya que este último método se implementa normalmente usando llamadas al sistema y, por tanto, requiere que intervenga el kernel, lo que consume más tiempo. Por el contrario, en los sistemas de memoria compartida, las llamadas al sistema sólo son necesarias para establecer las zonas

de memoria compartida. Una vez establecida la memoria compartida, todos los accesos se tratan como accesos a memoria rutinarios y no se precisa la ayuda del kernel.

Sistemas de Memoria Compartida

La comunicación interprocesos que emplea memoria compartida requiere que los procesos que se estén comunicando establezcan una región de memoria compartida. Normalmente, una región de memoria compartida reside en el espacio de direcciones del proceso que crea el segmento de memoria compartida. Otros procesos que deseen comunicarse usando este segmento de memoria deben conectarse a su espacio de direcciones. Debemos recordar que, habitualmente, el sistema operativo intenta evitar que un proceso acceda a la memoria de otro proceso. La memoria compartida requiere que dos o más procesos acuerden eliminar esa restricción. Entonces podrán intercambiar leyendo y escribiendo datos en las áreas compartidas. El formato de los datos y su ubicación están determinados por estos procesos, y no se encuentran bajo el control del sistema operativo.

Comunicación Directa

En el caso de comunicación directa, cada proceso que desea establecer una comunicación debe nombrar de forma explícita al receptor o emisor de la comunicación. En este esquema, las primitivas send () y receive () se definen del siguiente modo:

- send(P, mensaje) – envía un mensaje al proceso P
- receive(Q, mensaje) – recibe un mensaje del proceso Q

Los enlaces se establecen de forma automática entre cada par de procesos que quieran comunicarse. Cada enlace se asocia con exactamente dos procesos y entre cada par de procesos existe exactamente un enlace. El vínculo puede ser unidireccional, pero es usualmente bi-direccional.

Comunicación indirecta

Con el modelo de comunicación indirecta, los mensajes se envían y reciben en buzones de correo o puertos. Cada buzón de correo tiene asociada una identificación univoca. Por ejemplo, las colas de mensajes de POSIX usan un valor entero para identificar cada buzón de correo.
En este esquema, puede establecerse un enlace entre un par de procesos si ambos tienen un buzón de correo compartido. Un enlace puede asociarse con más de dos procesos. Entre cada par de procesos en comunicación, puede haber una serie de enlaces diferentes, correspondiendo cada enlace a un buzón de correo. Y por último, los vínculos pueden ser unidireccionales o bi-direccionales.
El sistema operativo debe proporcionar un mecanismo que permita a un proceso hacer lo siguiente:

- Crear un buzón de correo nuevo
- Enviar y recibir mensajes a través del buzón de correo.
- Eliminar un buzón de correo.

Las primitivas send () y receive () se definen del siguiente modo:

- send(A, mensaje) – envía un mensaje al buzón de correo A.
- receive (A, mensaje) – Recibe un mensaje del buzón de correo A.

Sincronización

El pasaje de mensajes puede ser bloqueante o no bloqueante, mecanismos también conocidos como sincrónico y asincrónico.

- Send bloqueante: El proceso que envía se bloquea hasta que el proceso receptor o el buzón de correo reciben el mensaje.

- Receive bloqueante: El receptor se bloque hasta que hay un mensaje disponible.
- Send no bloqueante: El proceso transmisor envía el mensaje continúa operando.
- Receive no bloqueante: El receptor extrae un mensaje válido o no válido.

Buffering

Sea la comunicación directa o indirecta, los mensajes intercambiados por los procesos que se están comunicando residen en una cola temporal. Básicamente, tales colas se pueden implementar de tres maneras:

- Capacidad cero. La cola tiene una longitud máxima de cero; por lo tanto, no puede haber ningún mensaje esperando en el enlace. En este caso, el emisor debe bloquearse hasta que el receptor reciba un mensaje.
- Capacidad limitada. La cola tiene una longitud finita n; por tanto, puede haber en ella n mensajes como máximo. Si la cola no está llena cuando se envía un mensaje, el mensaje se introduce en la cola, y el transmisor puede continuar la ejecución sin esperar. Sin embargo, la capacidad del enlace es finita. Si el enlace esta lleno, el transmisor debe bloquearse hasta que haya espacio disponible en la cola.
- Capacidad ilimitada: La longitud de la cola es potencialmente infinita; por tanto, puede haber cualquier cantidad de mensajes esperando en ella. El transmisor nunca se bloquea.

Comunicación Cliente-Servidor

Existen otras dos estrategias para comunicación en los sistemas cliente-servidor, llamadas a procedimientos remotos (RPC) e invocación de métodos remotos de Java (RMI).

Sockets

Un socket se define como un punto terminal de una comunicación. Una pareja de procesos que se comunican a través de una red emplea una pareja de sockets, uno para cada proceso. Cada socket se identifica mediante una dirección IP concatenada con un número de puerto. En general, los sockets usan una arquitectura cliente-servidor; el servidor espera a que entren solicitudes del cliente, poniéndose a la escucha en un determinado puerto. Una vez que se recibe una solicitud, el servidor acepta una conexión del socket cliente y la conexión queda establecida.

Llamadas a Procedimientos Remotos

Una de las formas más comunes de prestar servicios remotos es el uso de llamadas a procedimiento remoto (RPC). Las RPC se diseñaron como un método para abstraer los mecanismos de llamada a procedimientos, con el fin de utilizarlos entre sistemas conectados en red.

Cada mensaje se dirige a un demonio RPC que escucha en un puerto del sistema remoto, y cada uno contiene un identificador de la función que se va a ejecutar y los parámetros que hay que pasar a dicha función. La función se ejecuta entonces de la forma solicitada y se devuelven los posibles datos de salida a quien haya efectuado la solicitud usando un mensaje diferente.

Un puerto es simplemente un número incluido al principio del paquete de mensaje.

Invocación de Métodos Remotos (RMI)

La invocación a métodos remotos (RMI, remote method invocation) es una funcionalidad Java similar a las llamadas a procedimientos remotos. RMI permite a un programa Java, sobre una máquina, invocar un método sobre un objeto remoto. Los objetos se consideran remotos si residen en una maquina virtual Java diferente.

Comparaciones entre RMI y RPC

Los sistemas RMI y RPC difieren en dos aspectos fundamentales. En primer lugar, el mecanismo RPC soporta la programación procedimental, por lo que sólo se puede llamar a procedimientos o funciones remotas. Por el contrario, el

mecanismo RMI se basa en objetos, permite la invocación de métodos correspondientes a objetos remotos. En segundo lugar, los parámetros para los procedimientos remotos en RPC son estructuras de datos ordinarias; con RMI, es posible pasar objetos como parámetros a los métodos remotos.

Para hacer que los métodos remotos sean transparentes tanto para el cliente como para el servidor, RMI implementa el objeto remoto utilizando stubs y esqueletos. Un stub es un proxy para el objeto remoto y reside en el cliente. Cuando un cliente invoca a un método remoto, se llama al stub correspondiente al objeto remoto. Este stub del lado del cliente es responsable de crear un paquete, que consta del nombre del método que se va a invocar en el servidor y de los parámetros del método debidamente envueltos. El stub envía entonces ese paquete al servidor, donde el esqueleto correspondiente al objeto remoto lo recibe. El esqueleto es el responsable de desenvolver los parámetros y de invocar el método deseado en el servidor. El esqueleto envuelve entonces el valor de retorno en un paquete y lo devuelve al cliente. El stub desenvuelve el valor de retorno y se lo pasa al cliente.

Hilos – Módulo 4

Hilos

Un thread (o proceso de peso liviano) es una unidad básica de utilización de la CPU; comprende un contador de programa, un conjunto de registros y una pila. Un thread comparte con sus threads compañeros (colectivamente conocidos como tarea o task) su sección de código, sección de datos y otros recursos del sistema operativo, como los archivos abiertos y las señales. Un proceso tradicional o proceso de peso pesado es igual a una tarea con un solo thread.

Beneficios

Las ventajas de la programación multihilo pueden dividirse en cinco categorías principales.

Capacidad de respuesta: El uso de múltiples hilos en una aplicación interactiva permite que un programa continúe ejecutándose incluso aunque parte de él esté bloqueado o realizando una operación muy larga, lo que incrementa la capacidad de respuesta al usuario.

Compartición de recursos: Por omisión, los hilos comparten la memoria y los recursos del proceso al que pertenecen. La ventaja de compartir código y los datos es que permite que una aplicación tenga varios hilos de actividad diferentes dentro de un mismo espacio de direcciones

Economía: La asignación de memoria y recursos para la creación de procesos es costosa. Dado que los hilos comparten recursos del proceso al que pertenecen, es más económico crear y realizar cambios de contexto entre unos y otros.

Utilización sobre Arquitecturas Multiprocesador: Las ventajas de usar configuraciones multihilo pueden verse incrementadas significativamente en una arquitectura multiprocesador, donde los hilos pueden ejecutarse en paralelo en los diferentes procesadores. Los mecanismos multihilo en una máquina con varios CPU incrementan el grado de concurrencia. No obstante, los sistemas multicore ponen desafíos a los programadores tales como: dividir actividades, balance, partición de datos, dependencia de los datos y verificación y depuración.

Escalabilidad.

Modelos multihilados

Existen tres tipos de relaciones entre los hilos de usuario y las de kernel:

- Muchos-a-Uno: Este modelo asigna múltiples hilos del nivel de usuario a un hilo del kernel. La gestión de hilos se realiza mediante la biblioteca de hilos a nivel de usuario, por lo que resulta eficiente, pero el proceso completo se bloquea si un hilo realiza una llamada bloqueante al sistema. También, dado que sólo un hilo puede acceder

al kernel cada vez, no podría ejecutarse varios hilos en paralelo sobre múltiples procesadores. Se lo utiliza en sistemas que no soporta hilos a nivel de kernel.

- Uno-a-Uno: Bajo este modelo, cada thread a nivel de usuario mapea aun thread a nivel de kernel. Si bien permite mayor concurrencia, se deben evitar crear muchos hilos dentro de una misma aplicación.
- Muchos-a-Muchos: Este modelo multiplexa muchos hilos de usuario sobre un número menor o igual de hilos de kernel. El número de hilos de kernel puede ser específico de una determinada aplicación o de una determinada máquina, por caso, pueden asignarse más hilos de kernel a una aplicación en un sistema multiprocesador que en uno de un solo procesador. Aquí los desarrolladores pueden crear tantos hilos de usuario como sean necesarios y los correspondientes hilos de kernel pueden ejecutarse en paralelo en un multiprocesador. Asimismo, cuando un hilo realiza una llamada bloqueante al sistema, el kernel puede planificar otro hilo para su ejecución. Se lo implemento en Solaris antes de la versión 9 y en Windows NT/2000 en adelante con el paquete ThreadFiber.

Librerías de Hilos

Las librerías de hilos proporcionan al programador una API para crear y administrar hilos. Existen dos formas principales de implementar una biblioteca de hilos. El primer método consiste en proporcionar una librería enteramente en el espacio de usuario, sin ningún soporte del kernel. Esto implica que invocar a una función de la librería es como realizar una llamada a una función local en el espacio de usuario y no una llamada al sistema.

El segundo método consiste en implementar una biblioteca en el nivel del kernel, soportada directamente por el sistema operativo. Invocar una función de la API de la biblioteca normalmente da lugar a que se produzca una llamada al sistema dirigida al kernel.

Cancelación de Hilos

La cancelación de un hilo es la acción de terminar un hilo antes de que se haya completado. Por ejemplo, si varios hilos están realizando una búsqueda en forma concurrente en una base de datos y un hilo devuelve el resultado, los restantes hilos deben ser cancelados.

Un hilo que vaya a ser cancelado se denomina a menudo hilo objetivo. La cancelación de un hilo puede ocurrir en dos escenarios diferentes:

1. Cancelación asincrónica: termina el hilo objetivo inmediatamente

2. Cancelación diferida: Permite al hilo objetivo verificar periódicamente si debería ser cancelado.

Manejo de Signal

Una señal se usa en los sistemas UNIX para notificar a un proceso que se ha producido un determinado suceso. Una señal puede recibirse sincrónica o asincrónicamente, dependiendo del origen y de la razón por la que el suceso deba ser señalado. Todas las señales siguen el mismo patrón:

1. Una señal se genera debido a que se produce un determinado suceso

2. La señal generada se suministra a un proceso

3. Una vez suministrada, la señal debe ser tratada.

Como ejemplo de señales síncronas podemos citar los accesos ilegales a memoria y la división por cero. Las señales síncronas se proporcionan al mismo proceso que realizo la operación que causo la señal, esta es la razón por la que se consideran síncronas. Cuando una señal se genera por un suceso externo a un proceso en ejecución, dicho proceso

recibe la señal en modo asíncrono. Como ejemplos de tales podemos citar la terminación de un proceso mediante la pulsación de teclas especificas.

Cada señal puede ser tratada o bien por una rutina de tratamiento predeterminada o bien por una definida por el usuario.

El tratamiento de señales en programas de un solo hilo resulta sencillo; las señales siempre se suministran a un proceso. Sin embargo, suministrar las señales en los programas multihilo es más complicado, ya que un proceso puede tener varios hilos. ¿A quién entonces, debe suministrarse la señal?

En general, hay disponibles las siguientes opciones:

1. Enviar la señal al hilo sobre el cual el signal se aplica

2. Enviar el signal a cada hilo del proceso

3. Enviar el signal a ciertos hilos en el proceso

4. Asignar un hilo específico para recibir todos los signals al proceso.

El método para suministrar una señal depende del tipo de señal generada. Por ejemplo, las señales síncronas tienen que suministrarse al hilo que causó la señal y no a los restantes hilos del proceso. Sin embargo, la situación con señales asíncronas no es tan clara. Algunas señales asíncronas, como una señal de terminación de un proceso, debería enviarse a todas las hebras.

Pool de Hilos

En esta situación, cuando el servidor recibe una solicitud, crea un hilo nuevo para dar servicio a la solicitud. Aunque crear un nuevo hilo es, indudablemente, mucho menos costo que crear un proceso nuevo, los servidores multihilo no están exentos de problemas potenciales. El primero concierte a la cantidad de tiempo necesario para crear el hilo antes de dar servicio a la solicitud y el segundo se basa en que si todas las solicitudes concurrentes son servidas mediante un nuevo hilo, no ponemos limite al número de hilos activos de forma concurrente en el sistema. Un número ilimitado de hilos podría agotar determinados recursos del sistema, como el tiempo de CPU o la memoria. Una solución para este problema consiste en usar lo que se denomina Pool de Hilos.

La idea general subyacente al pool de hilos consiste en crear una serie de hilos al principio del proceso y colocarlos en un conjunto compartido (pool), donde los hilos queden a la espera de que se les asigne un trabajo. Cuando un servidor recibe una solicitud, despierta a un hilo del conjunto, si hay uno disponible, y le pasa la solicitud para que la hebra se encargue de dar servicio. Una vez que el hilo completa su tarea, vuelve al pool y espera a que haya más trabajo. Si no hay ningún hilo libre, el servidor espera hasta que quede uno libre.

Esta metodología provee de las siguientes ventajas:

- Usualmente es ligeramente más rápido servir un nuevo requerimiento con un hilo existente que crear uno nuevo
- Permite que el número de hilos de la aplicación sean limitado al tamaño del pool

Hilos en Linux

Linux proporciona la capacidad de crear hilos usando la llamada al sistema clone (). Sin embargo, Linux no diferente entre procesos e hilos. De hecho, generalmente, Linux utiliza el término tarea en lugar de proceso o hilo, para hacer

referencia al flujo de control dentro de un programa. Cuando se invoca a clone (), se pasa un conjunto de indicadores que determina el nivel de compartición entre las tareas padre e hijo. Por caso, el indicador CLONE_VM permite compartir el espacio de direcciones de la tarea-padre (proceso).

Planificación de Procesos – Módulo 5

Conceptos básicos

El objetivo de la multiprogramación es tener continuamente varios procesos en ejecución, con el fin de maximizar el uso de la CPU. La idea es bastante simple: un proceso se ejecuta hasta que tenga que esperar. En un sistema informático simple, la CPU permanece entonces inactiva y todo el tiempo de espera se desperdicia; no se realiza ningún trabajo útil. Con la multiprogramación, se intenta usar ese tiempo de forma productiva. En este caso, se mantienen varios procesos en memoria a la vez. Cuando un proceso tiene que esperar, el sistema operativo retira el uso de la CPU a ese proceso y se lo cede a otro proceso. Este patrón se repite continuamente y cada vez que un proceso tiene que esperar, otro proceso puede hacer uso de la CPU.

La adecuada planificación de la CPU depende de una propiedad observada de los procesos: la ejecución de un proceso consta de un ciclo de ejecución en la CPU, seguido de una espera de E/S; los procesos alternan entre estos dos estados. La ejecución de un proceso comienza con una ráfaga de CPU. Ésta va seguida de una ráfaga de E/S, a la cual sigue otra ráfaga de CPU, luego otra ráfaga de E/S, etc. Finalmente, la ráfaga final de CPU concluye con una solicitud al sistema para terminar la ejecución.

La duración de las ráfagas de CPU se ha medido exhaustivamente en la práctica. Aunque varían enormemente de un proceso a otro y de una computadora a otra, tienden a presentar una curva de tipo exponencial o hiperexponencial con un gran número de ráfagas de CPU cortas y un número menor de ráfagas de CPU largas. Esta distribución puede ser importante en la sección de un algoritmo apropiado para la planificación del CPU.

Planificador de CPU

Cuando la CPU queda inactiva, el sistema operativo debe seleccionar uno de los procesos que se encuentran en la cola de procesos preparados para ejecución. El planificador a corto plazo (o planificador de CPU) lleva a cabo esa selección de proceso. El planificador elige uno de los procesos que están en memoria preparados para ejecutarse y asigna a la CPU dicho proceso.

La decisión de planificar el CPU puede tener lugar cuando un proceso:

1. Conmuta de ejecutado a estado de espera

2. Conmuta de ejecutando a estado de listo

3. Conmuta de espera a listo

4. Termina.

En las situaciones 1 y 4, n hay más que una opción en términos de planificación: debe seleccionar un nuevo proceso para su ejecución (si hay alguno). Sin embargo, en las situaciones 2 y 3 si que existe la opción de planificar un nuevo proceso o no.

Cuando las decisiones de planificación sólo tienen lugar en las circunstancias 1 y 4, decimos que el esquema de planificación es sin desalojo o no apropiativo, en caso contrario, se trata de un esquema apropiativo. En la planificación

no apropiativa, una vez que se ha asignado el CPU a un proceso, el proceso se mantiene en la CPU hasta que ésta es liberada bien por la terminación del proceso o bien por la conmutación al estado de espera.

La planificación apropiativa requiere de hardware especial, por ejemplo un temporizador. Lamentablemente, además, la planificación apropiativa tiene un coste adicional asociado con el acceso a los datos compartidos. Consideremos el caso de dos procesos que compartan datos y supongamos que, mientras que uno está actualizando los datos, resulta despojado con el fin de que el segundo proceso pueda ejecutarse. El segundo proceso podría intentar entonces leer los datos, que se encuentran en un estado incoherente.

La técnica de planificación apropiativa también afecta al diseño del kernel del sistema operativo. Durante el procesamiento de una llamada, el kernel puede estar ocupado realizando una actividad en nombre del proceso. ¿Qué ocurre si el proceso se desaloja en la mitad de estos cambios y el kernel necesita leer o modificar la misma estructura? El resultado será un caos. Ciertos sistemas operativos, resuelven este problema esperando que se complete la llamada al sistema o se transfiera un bloque de E/S antes de hacer un cambio de contexto. Lamentablemente, si bien permite obtener una estructura del kernel simple, no resulta adecuado para permitir la realización de cálculos en tiempo real y el multiprocesamiento.

Despachador

Otro componente implicado en la función de planificación de la CPU es el despachador. El despachador es el módulo que proporciona el control de la CPU a los procesos seleccionados por el planificador de corto plazo. Esta función implica lo siguiente:

1. Cambio de contexto

2. Conmutación al modo usuario

Salta a la dirección apropiada en el programa de usuario para reiniciarlo.

El despachador debe ser lo más rápido posible, ya que se invoca en cada conmutación de un proceso. El tiempo que tarda el despachador en detener un proceso e iniciar la ejecución de otro se conoce como latencia de despacho.

Criterios de Planificación

Los diferentes algoritmos de planificación de la CPU tienen distintas propiedades, y la elección de un algoritmo en particular puede favorecer una clase de procesos sobre otros.

Se han sugerido los siguientes criterios para comparar los distintos algoritmos de planificación:

- Utilización de la CPU: Deseamos mantener la CPU tan ocupada como sea posible.
- Procesamiento total (Throughput): Es el número de procesos que completan su ejecución por unidad de tiempo.
- Tiempo de retorno: Cantidad de tiempo para ejecutar un determinado proceso
- Tiempo de Espera: Cantidad de tiempo que un proceso ha estado esperando en las colas.
- Tiempo de Respuesta: Cantidad de tiempo que transcurre desde que fue hecho un requerimiento hasta que se produce la primera respuesta, no salida.

Criterios de Optimización

El objetivo consiste en maximizar la utilización de la CPU y el procesamiento total, y minimizar el tiempo de retorno, el tiempo de espera y el tiempo de respuesta.

Planificación Primero-Entrar, Primero-Servido (FCFS)

El algoritmo más simple de la planificación de la CPU es, con mucho, el algoritmo FCFS. Con este esquema, se asigna primero la CPU al proceso que primero la solicite. La implementación de la política FCFS se gestiona fácilmente con una cola FIFO. Cuando un proceso entra en la cola de procesos listos, su PCB se coloca al final de la cola. Cuando la CPU queda libre, se asigna al proceso que este al principio de la cola y ese proceso que pasa a ejecutarse se elimina de la cola. El código de este algoritmo es fácil de escribir y comprender.

Sin embargo, el tiempo medio de espera con el algoritmo FCFS es a menudo bastante largo.

Cabe aclarar que el tiempo medio de espera con una política FCFS no es, generalmente, mínimo y puede variar significativamente si la duración de las ráfagas de CPU de los procesos es muy variable.

Además esta planificación da lugar al efecto convoy que se describe a continuación. Supongamos que tenemos un proceso limitado por la CPU y muchos procesos limitados por la E/S. A medida que los procesos fluyen en el sistema, puede llegarse al siguiente escenario: el proceso limitado por la CPU obtendrá y mantendrá la CPU; durante ese tiempo, los demás procesos terminarán sus operaciones de E/S y pasarán a la cola de procesos listos, esperando acceder a la CPU. Mientras que los procesos esperan en la cola de listos los dispositivos de E/S están inactivos. Finalmente, el proceso limitado por la CPU termina su ráfaga de CPU y pasa a esperar por un dispositivo de E/S. Todos los procesos limitados por E/S que tienen ráfagas de CPU cortas, se ejecutan rápidamente y vuelven a las colas de E/S. En este punto, la CPU permanecerá inactiva. El proceso limitado por CPU volverá en algún momento a la cola de procesos preparados y se le asignara la CPU. De nuevo se repite la espera de los procesos limitados a la E/S esperando por el proceso limitado al CPU. Esto se lo conoce como efecto convoy, y se produce a medida que todos los procesos restantes esperan a que un único proceso de larga duración deje de usar la CPU.

El algoritmo de planificación FCFS es no apropiativo, por lo tanto, es especialmente problemático en los sistemas de tiempo compartido, donde es importante que el CPU mantenga una cuota de la CPU a intervalos regulares.

Planificación Job-Más Corto Primero (SJF)

Este algoritmo asocia con cada proceso la duración de la siguiente ráfaga de CPU del proceso. Cuando la CPU está disponible, se asigna el proceso que tiene la ráfaga de CPU más corta. Si las siguientes ráfagas de CPU de dos procesos son iguales, se usa la planificación FCFS en la decisión.

El algoritmo de planificación SJF es probablemente óptimo, en el sentido de que proporciona el tiempo medio de espera mínimo para un conjunto de procesos dados. La dificultad real del algoritmo SJF es conocer la duración de la siguiente solicitud de CPU. En una planificación a largo plazo de trabajos en un sistema de procesamiento por lotes, podemos usar como duración el límite de tiempo del proceso que el usuario especifique en el momento de enviar el trabajo. Aunque SJF es óptimo, no se puede implementar en el nivel de planificación de la CPU a corto plazo, ya que no hay forma de conocer la duración de la siguiente ráfaga de CPU. No obstante, un método consiste en predecir su valor, confiando que la siguiente será similar en duración a las anteriores.

Hay dos esquemas: no apropiativo y apropiativo. Bajo el primero una vez que la CPU es dada a un proceso, no puede ser apropiada hasta que el mismo complete su ráfaga de CPU. Bajo el segundo, si un nuevo proceso llega con una longitud de ráfaga de CPU menor que el resto del tiempo de ejecución que le queda al proceso que esta ejecutando entonces se apropia de la CPU. Este esquema es conocido como el tiempo remanente más corto primero (SRTF).

Planificación por Prioridad

El algoritmo SJF es un caso especial del algoritmo de planificación por prioridades general. A cada proceso se le asigna una prioridad y la CPU se asigna al proceso que tenga la prioridad más alta. Los procesos con la misma prioridad se clasifican en orden FCFS. Cuanto más larga sea la ráfaga de CPU, menor será la prioridad y viceversa.

Debe observarse que al hablar de planificación pensamos en términos de alta prioridad y baja prioridad. Generalmente, las prioridades se indican mediante un número entero y no existe un acuerdo general sobre si 0 es la prioridad más alta o más baja.

La planificación por prioridades puede ser apropiativa o no apropiativa.

Un problema importante de los algoritmos de planificación por prioridades es el bloqueo indefinido o la muerte por inanición. Un algoritmo de planificación por prioridades puede dejar a algunos procesos de baja prioridad esperando indefinidamente. Una solución al problema del bloque indefinido de los procesos de baja prioridad consiste en aplicar mecanismos de envejecimiento. Esta técnica consiste en aumentar gradualmente la prioridad de los procesos que estén esperando en el sistema durante mucho tiempo.

Planificación Round Robin (RR)

El algoritmo de planificación por turnos (RR; round robin) está diseñado especialmente para los sistemas de tiempo compartido. Es similar a la planificación FCFS, pero se añade la técnica de desalojo para conmutar entre procesos. En este tipo de sistema se define una pequeña unidad de tiempo, denominada quantum, o franja temporal. Para implementar la planificación por turnos, mantenemos la cola de procesos listos como una cola FIFO de procesos. El planificador de la CPU recorre la cola de procesos listos asignando la CPU a cada proceso durante un intervalo de tiempo de hasta 1 quantum. Luego pueden ocurrir una de las siguientes dos cosas. El proceso puede tener una ráfaga de CPU cuya duración sea menor que un quantum de tiempo; en este caso, el propio proceso liberara voluntariamente la CPU. El planificador continuará entonces con el siguiente proceso de la cola de procesos preparados. En caso contrario, si la ráfaga de CPU del proceso actualmente en ejecución tiene una duración mayor que 1 quantum de tiempo, se producirá un fin de cuenta del temporizador y éste enviara una interrupción al sistema operativo; entonces se ejecutará un cambio de contexto y el proceso se colocará al final de la cola de procesos listos.

El tiempo medio de espera en los sistemas RR es, con frecuencia, largo.

El rendimiento de este algoritmo depende enormemente del tamaño del quantum de tiempo. Por un lado, si el quantum de tiempo es extremadamente largo, la planificación por turnos es igual que la FCFS y la rotación de procesos se reduce disminuyendo así el tiempo de respuesta. Si el quantum es muy pequeño, el método por turnos se denomina compartición del procesador y crea la apariencia de que cada uno de los n procesos tiene su propio procesador ejecutándose a 1/n de la velocidad del procesador real ya que se agrega un overhead considerable producto del incremento en los cambios de contexto.

Por tanto, conviene que el quantum de tiempo sea grande con respecto al tiempo requerido por un cambio de contexto.

El tiempo de ejecución también depende del valor del quantum de tiempo. En general, el tiempo medio de ejecución puede mejorar si la mayor parte de los procesos termina en su siguiente ráfaga de CPU en un solo quantum de tiempo.

Colas Multinivel

Bajo este algoritmo la cola de listos se particiona en dos colas separadas. Por ejemplo, una clasificación habitual consiste en diferenciar entre procesos de primer plano (interactivos) y procesos de segundo plano (batch). Estos dos tipos de procesos tienen requisitos diferentes de tiempo de respuesta y, por tanto, pueden tener distintas necesidades de planificación, por caso, podría planificarse la cola de primer plano con RR y la de segundo con FCFS. Los procesos se asignan permanentemente a una cola, generalmente en función de alguna propiedad del proceso, como por ejemplo el tamaño de la memoria, la prioridad o el tipo de proceso.

Además, la planificación debe ser hecha entre las colas. Por caso, los procesos de primer plano pueden tener prioridad absoluta sobre los procesos de segundo plano. Esto podría dar lugar a inanición.

Otra posibilidad consiste en repartir el tiempo entre las colas. En este caso, cada cola obtiene una cierta porción del CPU, con la que pueda entonces planificar sus distintos procesos. Por ejemplo, 80% en foregroud en RR y 20% en background en FCFS.

Colas Multinivel Realimentadas

Normalmente, cuando se usa el algoritmo de planificación mediante colas multinivel, los procesos se asignan de forma permanente a una cola cuando entran en el sistema y no se pueden mover de una cola a otra porque no pueden cambiar su naturaleza de proceso de primer o segundo plano. Esta configuración presenta la ventaja de una baja carga de trabajo de planificación, pero resulta poco flexible.

Por el contrario, el algoritmo de planificación mediante colas multinivel realimentadas permite mover un proceso de una cola a otra. La idea es separar los procesos en función de las características de sus ráfagas de CPU. Si un proceso utiliza demasiado tiempo de CPU, se pasa a una cola de prioridad más baja. Este esquema deja los procesos limitados por E/S y los procesos interactivos en las colas de prioridad más alta. Además, un proceso que esté esperando demasiado tiempo en una cola de baja prioridad puede pasarse a una cola de prioridad más alta. Este mecanismo de envejecimiento evita el bloqueo indefinido.

En general, un planificador mediante colas multinivel se define mediante los parámetros siguientes:

- Número de colas
- Algoritmos de planificación para cada cola
- Método usado para determinar cuándo mejorar un proceso
- Método usado para determinar cuándo degradar un proceso
- Método usado para determinar en qué cola entra un proceso cuando necesita servicio.

Multiprocesamiento simétrico y asimétrico

Un enfoque en lo que a planificación del CPU respecta en un sistema multiprocesador consiste en delegar todas las decisiones de planificación, procesamiento de E/S y demás actividades del sistema a un solo procesador, llamado servidor maestro.

En este esquema los otros procesadores ejecutan solo código del usuario. Esta organización se la conoce como multiprocesamiento asimétrico y es simple ya que un único procesador accede a las estructuras de datos del sistema, reduciendo así la necesidad de compartir información.

Por otro lado, se puede emplear multiprocesamiento simétrico.

En esta última configuración cada procesador realiza su propia planificación. Todos los procesos pueden estar en una única cola de listos o bien cada procesador puede tener su propia cola privada. De todas formas, la planificación permite que el planificador de cada procesador examine la cola de procesos listos y seleccione un proceso para ejecutar.

Como principal ventaja se tiene que en un sistema con multiprocesadores se puede ejecutar más de un proceso en un mismo quantum de tiempo, sin embargo, hay desventajas como veremos a continuación.

Cuando un proceso está siendo ejecutado en un procesador especifico, los datos accedidos más recientemente se alojan en la cache del procesador y como resultado los accesos a memoria consecutivos requeridos por el proceso son compartidos a nivel de la memoria cache. Ahora consideremos lo que sucede si el proceso migra a otro procesador. En esta situación, el contenido de la memoria cache del procesador del cual proviene el proceso debe ser invalidado y la cache del procesador hacia el cuál migro el proceso debe ser rellenada con la información anteriormente invalidada.

Esta operación es muy costosa, por lo que los sistemas con multiprocesamiento simétrico tratan de evitarla y en cambio buscan que los procesos siempre corran en el mismo procesador. Esto es conocido como afinidad de un procesador, indicando que un proceso tiene afinidad con el procesador que lo ejecuta.

Otro problema que se presenta en los sistemas de multiprocesamiento simétrico es el balanceo de la carga. Es importante conservar la carga de trabajos balanceada a lo largo de todos los procesadores para utilizar completamente los beneficios de un esquema con multiprocesadores. De otra manera, uno o más procesadores podrían quedar en espera mientras otros experimentan grandes cargas de tareas. Es importante notar que el balanceo de la carga es típicamente necesario en sistemas en los cuales cada procesador tiene su propia cola de procesos para ejecutar.

En cambio, en aquellos sistemas en los cuales se tiene una cola común, generalmente esta política es innecesaria, debido a que cuando un procesador se queda en espera, inmediatamente extrae un proceso de la cola común de procesos listos.

El balanceo de la carga aporta un beneficio a favor de la afinidad de un procesador, al tener un proceso corriendo en el mismo núcleo, el proceso puede tomar ventaja utilizando los datos alojados en la memoria cache de dicho procesador.

Planificación Tiempo Real

Los sistemas de tiempo real duro requieren completar tareas críticas en una cantidad de tiempo garantizado. Por el contrario, los sistemas de computación de tiempo real blando requieren que los procesos críticos reciban prioridad sobre otros.

Planificación de Hilos

Una de las diferencias entre los hilos de nivel de usuario y de nivel de kernel radica en la forma en que se planifican unos y otras.

En los sistemas que implementan los modelos muchos-a-uno y muchos-a-muchos, la librería de hilos planifica los hilos de nivel de usuario para que se ejecuten sobre un proceso LWP (Light-Weight-Process) disponible, lo cual es un esquema conocido con el nombre de planificación local o ámbito de contienda del proceso.

Para decidir que hilo del kernel planificar en una CPU, el kernel usa la planificación global o ámbito de contienda del sistema, donde la competición por la CPU con esta ultima planificación tiene lugar entre todos los hilos del sistema.

Planificación en UNIX y Linux

El planificador de Linux es un algoritmo basado en prioridades y es apropiativo, con dos rangos distintos de prioridades: un rango de tiempo real de 0 a 99 y un valor normal en el rango comprendo entre 100 y 140.

Bajo tiempo compartido, se priorizan los procesos basados en créditos y aquellos con más créditos son planificados primero. Dichos créditos se restan cuando la interrupción del timer ocurre. Si el crédito es cero, es elegido otro proceso. Cuando todos los procesos tienen crédito cero, entonces ocurre una re acreditación basada factores que incluyen prioridad e historia.

Linux implementa la planificación en tiempo real blando tal como se define en POSIX 1b. Bajo esta policita se aplica FCFS y RR y aquellos procesos de más alta prioridad corren primero.

Históricamente, la planificación tradicional de UNIX emplea colas multinivel (los niveles se definen en bandas de propiedades) usando Round Robin en cada una de ellas.

La prioridad de cada proceso es computada cada segundo (en los primeros UNIX, hoy es cada quantum). El propósito de la prioridad base es dividir todos los procesos en bandas de niveles de prioridad.

Los componentes CPU (que mide la utilización del proceso j en el intervalo de tiempo i) y nice (valor normal) se utilizan para prevenir que los procesos migren fuera de su banda asignada (dada por la prioridad base).

Estas bandas son utilizadas para optimizar el acceso a los dispositivos que se manejan con bloques de información (discos, cintas, CD, etc) y permitir al sistema operativo responder rápidamente a las llamadas al sistema.

Dentro de la banda de procesos de usuario, el uso de la historia de ejecución tiene a penalizar a los procesos limitados por procesador a expensas de los procesos limitados por E/S.

Sincronización de Procesos – Módulo 6

Condición de carrera

La condición de carrera es la situación donde varios procesos acceden y manejan datos compartidos concurrentemente. El valor fin de los datos compartidos depende de que proceso termine último. Para prevenir las condiciones de carrera, los procesos concurrentes deben ser sincronizados. Estas situaciones se producen frecuentemente en los sistemas operativos, cuando las diferentes partes del sistema manipulan los recursos.

Problema de la Sección Crítica

Considere un sistema que consta de n procesos (P_0, P_1, …, P_{n-1}) todos compitiendo para usar datos compartidos. Cada proceso tiene un segmento de código, llamado sección critica, en la cual los datos compartidos son accedidos. La característica importante del sistema es que, cuando un proceso está ejecutando su sección critica, ningún otro proceso puede ejecutar su correspondiente sección crítica. El problema de la sección crítica consiste en diseñar un protocolo que los procesos puedan usar para cooperar de esta forma. Cada proceso debe solicitar permisos para entrar en su sección crítica; la sección de código que implementa esta solicitud es la sección de entrada. La sección crítica puede ir seguida de una sección de salida. El código restante se encuentra en la sección restante.

Cualquier solución al problema de la sección crítica deberá satisfacer los tres requisitos siguientes:

1. Exclusión mutua: Si el proceso P_j está ejecutando su sección crítica, los demás procesos no pueden estar ejecutando sus secciones críticas.
2. Progreso. Si ningún proceso está ejecutando su sección critica y algunos procesos desean entrar en sus correspondientes secciones críticas, sólo aquellos que no estén ejecutando sus acciones restantes pueden participar de cuál será el siguiente que entre en su sección critica, y está selección no se puede posponer indefinidamente.
3. Espera limitada. Existe un límite en el número de veces que se permite que otros procesos entren en sus secciones criticas después de que un proceso haya hecho una solicitud para entrar en su sección critica y antes de que la misma haya sido concedida.

   Estamos suponiendo que cada proceso se ejecuta a una velocidad distinta de cero. Sin embargo, no podemos hacer ninguna suposición sobre la velocidad relativa de los n procesos.

Solución de Peterson

Se trata de una solución de software.

Algoritmo

Proceso P_i

```
repeat
flag[i] := true;
```

```
turn := j;
while(flag [j] and turn = j) do no-op;
      //Sección critica
flag [i] := false;
      //Resto de sección
until false
```

Esta solución cumple los tres requerimientos por lo que resuelve el problema de la sección crítica para dos procesos. Estos dos procesos van alternando la ejecución de sus secciones críticas y de sus secciones restantes.

La solución requiere que los dos procesos compartan dos elementos de datos:

```
int turn;
boolean flag [2];
```

La variable turn indica qué proceso va a entrar en su sección crítica. Es decir, si turn == i, entonces el proceso P_i puede ejecutar su sección crítica. La matriz flag se usa para indicar si un proceso está preparado para entrar en su sección crítica. Por ejemplo, si flag[i] es verdadera, este valor indica que P_i está preparado para entrar en su sección crítica.

Para entrar en la sección critica, el proceso P_i primero asigna el valor verdadero a flag[i] y luego asigna a turn el valor j, confirmando así que, si el otro proceso desea entrar en la sección critica, puede hacerlo. Si ambos procesos intenta entrar al mismo tiempo, a la variable turn se le asignarán tanto i como j aproximadamente al mismo tiempo. Sólo una de estas asignaciones permanecerá; la otra tendrá lugar, pero será sobrescrita inmediatamente. El valor que adopte turn decidirá cuál de los dos procesos podrá entrar primero en la sección crítica.

Algoritmo para N Procesos

También conocido como Algoritmo del Panadero.

Consideremos una sección crítica para n procesos.

Antes de entrar en su sección crítica, el proceso recibe un número. El poseedor del número más chico entra en su sección crítica. Si los procesos P_i y P_j reciben el mismo número, si i<j entonces P_i es servido primero; si no lo es P_j. El esquema de numeración siempre genera números en orden incremental de numeración. P.E: 1,2,3,3,3,3,4,5…

Datos compartidos:

```
var choosing: array [0..n-1] of boolean;
number: array [0..n-1] of integer;
```

Estas estructuras de datos son inicializadas a false y 0 respectivamente.

Algoritmo

```
repeat
      choosing [i] := true;
      number [i] := max(number[0], number [1], …, number [n-1])+1;
      choosing [i] := false;
```

```
      for j:=0 to n-1
            do begin
                  while choosing [j] do no-op;
                  while number [j] ≠ 0 and (number [j], j) < (number[i], i) do no-op;
      end;
      //sección critica
      number [i] := 0;
      //resto de sección
until false;
```

Sincronización por hardware

Muchos sistemas proveen soporte por hardware para el código de la sección crítica. El soporte por hardware puede facilitar cualquier tarea de programación y mejorar la eficiencia del sistema.

El problema de la sección crítica podría resolverse de forma simple en un entorno de un solo procesador si pudiéramos impedir que se produjeran interrupciones mientras se está modificando una variable compartida. De este modo, podríamos asegurar que la secuencia actual de instrucciones se ejecute sin apropiación. Ninguna otra instrucción se ejecutará, por lo que no se producirán modificaciones inesperadas de la variable compartida. Lamentablemente, está solución no resulta tan adecuada en un sistema multiprocesador puede consumir mucho tiempo, ya que hay que pasar el mensaje a todos los procesadores. Los SOs que usan esto no son ampliamente escalables.

Para esta implementación, muchos sistemas informáticos modernos proporcionan instrucciones hardware especiales que nos permiten consultar y modificar el contenido de una palabra o intercambiar los contenidos de dos palabras atómicamente, es decir, como una unidad de trabajo ininterrumpible.

La función TestAndSet se ejecuta atómicamente. Por tanto, si dos instrucciones TestAndSet se ejecutan simultáneamente se ejecutaran secuencialmente en un orden arbitrario.

```
Función Test-and-Set (var target: boolean): boolean;
      begin
            Test-and-Set := target;
            target := true;
      end;
```

Las estructuras de datos comunes son

```
var lock: boolean (inicialmente false)
```

Algoritmo

Proceso P_i

```
      repeat
```

```
while Test-and-Set (lock) do no-op;
      //sección critica
lock := false;
      //resto de la sección
until false;
```

Lamentablemente para los diseñadores de hardware, la implementación de instrucciones TestAndSet en los sistemas multiprocesador no es una tarea trivial.

Solución usando Swap

La función se define de la siguiente manera:

```
void Swap (boolean *a, boolean *b)
{
      boolean temp := *a;
      *a := *b;
      *b := temp;
}
```

La variable compartida es inicializada en FALSE. Cada proceso tiene una variable local booleana key.

Algoritmo

```
While (true) {
      key := true;
      while (key == true) {
            swap(&lock,&key);
            //sección critica
      }
      lock := false;
      //sección restante
}
```

Semáforos

Un semáforo S es una variable entera a la que, dejando aparte la inicialización, sólo se accede mediante dos operaciones atómicas estándar: wait() y signal(). Son atómicas ya que cuando un proceso modifica el valor de un semáforo, ningún otro proceso puede modificar simultáneamente el valor de dicho semáforo. Es decir se debe garantizar que dos procesos no puedan ejecutar wait() y signal() sobre el mismo semáforo al mismo tiempo. Entonces, la implementación se convierte en el problema de la sección crítica donde el código del wait y el signal son la sección crítica. En un entorno de un solo procesador, podemos solucionar el problema de forma sencilla inhibiendo las interrupciones durante el tiempo en que se ejecutan las operaciones de wait() y signal(). Una vez que se inhiben las interrupciones, las instrucciones de los diferentes procesos no pueden intercalarse: sólo se ejecuta el proceso actual hasta que se reactivan

las interrupciones y el planificador puede tomar de nuevo el control. En un entorno multiprocesador, hay que deshabilitar las interrupciones en cada procesador. Esta es una tarea compleja y, además, puede disminuir seriamente el rendimiento. Por tanto, en estos sistemas se deben proporcionar técnicas alternativas de bloqueo, como la espera ocupada.

Las operaciones se definen de la siguiente manera:

```
wait (S): while S≤0 do no-op;
      S := S-1;
signal(S): S := S + 1;
```

El semáforo como Herramienta General de Sincronización

Los sistemas operativos 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 sólo puede ser 0 o 1. También se los conoce como mutex lock.

Podemos usar semáforos binarios para abordar el problema de la sección crítica en el caso de múltiples procesos. Los n procesos comparten un semáforo, mutex, inicializado con el valor 1.

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 recursos ejecuta una operación wait() en el semáforo decrementando la cuenta. Cuando un proceso libera un recurso, ejecuta una operación signal() incrementando la cuenta. Cuando la cuenta del semáforo llega a 0, todos los recursos estarán en uso. Después, los procesos que deseen usar un recurso se bloquearán hasta que la cuenta sea mayor que 0.

Implementación de S con Semáforos Binarios

La principal desventaja de la definición de semáforo es que requiere una espera ocupada. Mientras que un proceso está en su sección crítica, cualquier otro proceso que intente entrar en su sección crítica debe ejecutar continuamente un bucle en el código de entrada. Este bucle continuo plantea claramente un problema real en un sistema de multiprogramación, donde una sola CPU se comparte entre muchos procesos. La espera ocupada desperdicia ciclos de CPU que algunos otros procesos podrían usar de forma productiva.

Para salvar la necesidad de la espera ocupada, podemos modificar la definición de las operaciones de semáforo 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. La operación de bloqueo coloca al proceso en una cola de espera asociada con el semáforo y el estado del proceso pasa al estado de espera. A continuación, el control se transfiere al planificador de la CPU, que selecciona otro proceso para su ejecución. Un proceso bloqueado, que está esperando en un semáforo S, debe reiniciarse cuando algún otro proceso ejecuta una operación signal(). El proceso se reinicia mediante una operación wakeup(), que cambia al proceso de espera al estado de listo y lo ubica en la cola de listos.

Las operaciones del semáforo se definen como:

```
wait(S): S.value := S.value -1;
      if S.value < 0
      then begin
```

```
            agregue este proceso a S.L;
            block;
      end;
signal(S): S.value := S.value + 1;
      if S.value ≤ 0
      then begin
            remueva un proceso P de S.L;
            wakeup(P);
      end;
```

Interbloqueo e Inanición

La implementación de un semáforo con una cola de espera puede dar lugar a una situación en la que dos o más procesos están esperando indefinidamente a que se produzca un suceso que solo puede producirse como consecuencia de las operaciones efectuadas por otro de los procesos en espera. El suceso en cuestión es la ejecución de la operación signal(). Cuando se llega a un estado así, se dice que estos procesos se han interbloqueado.

Para ilustrar el concepto, consideremos un sistema que consta de dos procesos, P_0 y P_1, con acceso cada uno de ellos a dos semáforos, S y Q, inicializados con el valor 1.

P_0	P_1
wait(S)	*wait(Q)*
wait(Q)	*wait(S)*
.	.
.	.
.	.
signal(S)	*signal(Q)*
signal(Q)	*signal(S)*

Otro problema relacionado con los interbloqueos es el del bloqueo indefinido o muerte por inanición, una situación en la que algunos procesos esperan de forma indefinida dentro del sistema. En esta situación un proceso no puede ser removido nunca de la cola del semáforo en el que fue suspendido.

Monitores

Dado que existen distintos tipos de errores que se pueden generar fácilmente cuando los programadores emplean incorrectamente los semáforos, los investigadores han desarrollado una estructura de datos fundamental de sincronización de alto nivel, el tipo monitor.

Un tipo monitor tiene un conjunto de operaciones definidas por el programador que gozan de la característica de exclusión mutua dentro del monitor. El tipo monitor también contiene la declaración de una serie de variables cuyos valores definen el estado de una instancia de dicho tipo, junto con los cuerpos de los procedimientos o funciones que operan sobre dichas variables.

La estructura del monitor asegura que sólo un proceso esté activo cada vez dentro del monitor. En consecuencia, el programador no tiene que codificar explícitamente esta restricción de sincronización. Sin embargo, para que la estructura del monitor sea lo suficientemente potente para modelar todos los esquemas de sincronización se le deben agregar variables de tipo condición.

Las únicas operaciones que se pueden invocar en una variable de condición sobre wait() y signal(). La operación x.wait() indica que el proceso que invoca esta operación queda suspendido hasta que otro proceso invoque la operación x.signal(). La operación x.signal() hace que se reanude exactamente uno de los procesos suspendidos. Si no había ningún proceso suspendido, entonces la operación signal() no tiene efecto, es decir, el estado de x será el mismo que si la operación nunca se hubiera ejecutado.

Cuando un proceso invoca a la operación x.signal(), hay un proceso Q en estado de suspendido asociado a la condición x. Evidentemente, si se permite al proceso suspendido Q reanudar su ejecución, el proceso P que ha efectuado la señalización deberá esperar, en caso contrario, P y Q se activarían simultáneamente dentro del monitor. Sin embargo, debemos observar que conceptualmente ambos procesos pueden continuar su ejecución. Existen dos posibilidades:

1. Señalizar y esperar. P espera hasta que Q salga del monitor o espere a que se produzca otra condición

2. Señalizar y continuar: Q espera hasta que P salga del monitor o espere a que se produzca otra condición.

Hay argumentos razonables a favor de adoptar cualquiera de estas opciones. Por un lado, puesto que P ya estaba ejecutándose en el monitor, el método de señalizar y continuar parece el más razonable. Por otro lado, si permitimos que el hilo P continúe, para cuando se reanude la ejecución de Q, es posible que ya no se cumpla la condición lógica por la que Q estaba esperando.

Interbloqueos – Módulo 7

El problema del interbloqueo

Una tabla del sistema registra si cada recurso está libre o ha sido asignado; para cada recurso asignado, la tabla también registra el proceso al que está asignado actualmente. Si un proceso solicita un recurso que en ese momento está asignado a otro proceso, puede añadirse a la cola de procesos en espera para ese recurso.

Un conjunto de procesos estará en un estado de interbloqueo cuando todos los procesos del conjunto estén esperando a que se produzca un suceso que sólo puede producirse como resultado de la actividad de otro proceso del conjunto. Estos sucesos comprenden por caso la adquisición y liberación de recursos. El recurso puede ser un recurso físico o un recurso lógico. Cada recurso tiene de tipo R_i tiene W_i instancias.

Ejemplo:

- Un sistema tiene 2 discos. P_1 y P_2 cada uno tiene un disco y necesita otro.

Caracterización de Interbloqueos

En un interbloqueo, los procesos nunca terminan de ejecutarse y los recursos del sistema están ocupados, lo que impide que se inicien nuevos trabajos. El interbloqueo puede alcanzarse si se cumplen las cuatro condiciones simultáneamente.

- Exclusión mutua: Solo un proceso a la vez puede usar un recurso. Si otro proceso solicita el recurso, el proceso solicitante tendrá que esperar hasta que el recurso sea liberado.
- Retener y Esperar: Un proceso mantiene al menos un recurso y está esperando adquirir recursos adicionales tenidos por otros procesos.
- No Apropiación: Un recurso puede ser liberado solo voluntariamente por el proceso que lo tiene, después que el proceso ha completado su tarea.
- Espera circular: Existe un conjunto $\{P_0, P_1, \ldots, P_n\}$ de procesos esperando tal que P_0 está esperando por un recurso que es retenido por P_1, P_1 está esperando por un recurso que es retenido por P_2, …, P_{n-1} está esperando por un recurso que es retenido por P_n, y P_n está esperando por un recurso que es retenido por P_0.

Grafo de Alocación de Recursos

Los interbloqueos pueden definirse de forma más precisa mediante un grafo dirigido, que se llama grafo de alocación de recursos del sistema. Este grafo consta de un conjunto de vértices V y de un conjunto de aristas E. El conjunto de vértices V se divide en dos tipos diferentes de nodos: $P = \{P_1,P_2,\ldots,P_n\}$, el conjunto formado por todos los procesos activos del sistema, y $R = \{R_1, R_2,\ldots, R_n\}$, el conjunto formado por todos los tipos de recursos del sistema.

Gráficamente, representamos cada proceso P_i con un círculo y cada tipo de recurso R_j con un rectángulo. Puesto que el tipo de recurso R_j puede tener más de una instancia, representamos cada instancia mediante un punto dentro del rectángulo.

Una arista dirigida desde el proceso P_i al tipo de recurso R_j se indica mediante $P_i \rightarrow R_j$ y significa que el proceso P_i ha solicitado una instancia del tipo de recurso R_j y que actualmente está esperando dicho recurso. Una arista dirigida desde el tipo de recurso R_j al proceso P_i se indica mediante $R_j \rightarrow P_i$ y quiere decir que se ha asignado una instancia del tipo de recurso R_j al proceso P_i.

Si un grafo no contiene ciclos entonces no hay interbloqueo.

Si un grafo contiene un ciclo, entonces:

- Si hay una sola instancia por tipo de recurso, entonces hay interbloqueo.
- SI hay varias instancias por tipo de recurso, hay posibilidad de interbloqueo.

Métodos para Manejo de Interbloqueos

En general, podemos abordar el problema de los interbloqueos de una de tres formas:

- Podemos emplear un protocolo para impedir o evitar los interbloqueos, asegurando que el sistema nunca entre en estado de interbloqueo.
- Podemos permitir que el sistema entre en estado de interbloqueo, detectarlo y realizar una recuperación
- Podemos ignorar el problema y actuar como si nunca se produjeran interbloqueos en el sistema.

La tercera solución es la que usan la mayoría de los sistemas operativos, incluyendo UNIX y Windows; entonces, es problema del desarrollador de aplicaciones el escribir programas que resuelvan los posibles interbloqueos.
Las estrategias utilizadas para atacar el problema de interbloqueos establecen tres niveles de acción: antes de que sucede, cuando los procesos están corriendo o con hechos consumados. Estas estrategias son prevención, evasión y detección.

Prevención de Interbloqueos

La prevención de interbloqueos proporciona un conjunto de métodos para asegurar que al menos una de las condiciones necesarias no pueda cumplirse. Estos métodos evitan los interbloqueos restringiendo el modo en que se pueden realizar las solicitudes.

- Exclusión mutua: La condición de exclusión mutua se aplica a los recursos que no pueden ser compartidos. Por ejemplo, varios procesos no pueden compartir simultáneamente una impresora. Por el contrario, los recursos que sí pueden compartirse no requieren acceso mutuamente excluyente y, por tanto, no pueden verse implicados en un interbloqueo. Los archivos de solo lectura son un buen ejemplo de un recurso que puede compartirse.
- Mantener y Esperar: Para asegurar que la condición de mantener y esperar nunca se produzca en el sistema, debemos garantizar que, cuando un proceso solicite un recurso, el proceso no esté reteniendo ningún otro recurso. Un posible protocolo consiste en exigir que cada proceso solicite todos sus recursos antes de comenzar su ejecución. Un proceso puede solicitar algunos recursos y utilizarlos. Sin embargo, antes de solicitar cualquier recurso adicional, tiene que liberar todos los recursos que tenga asignados actualmente.
  En ambas alternativas existen dos desventajas importantes. Primero, la tasa de utilización de los recursos puede ser baja, dado que los recursos pueden asignarse pero no utilizarse durante un periodo largo de tiempo. La segunda desventaja es que puede producirse el fenómeno de inanición: un proceso que necesite varios recursos muy solicitados puede tener que esperar de forma indefinida, debido a que al menos uno de los recursos que necesita está siempre asignado a algún otro proceso.
- No apropiación: Para impedir que se cumpla está condición, podemos usar el protocolo siguiente: si un proceso está reteniendo varios recursos y solicita otro recurso que no se le puede asignar en forma inmediata, entonces todos los recursos actualmente retenidos se liberan. Los recursos liberados son agregados a la lista de recursos que el proceso está esperando. El proceso sólo se reiniciará cuando pueda recuperar sus antiguos recursos, junto con los nuevos que está solicitando.
- Espera circular: Una forma de garantizar que está condición nunca se produzca es imponer una ordenación total de todos los tipos de recursos y requerir que cada proceso solicite sus recursos en un orden creciente de numeración. Hay que tener presente que definir una ordenación o jerarquía no impide, por sí solo, la aparición de interbloqueos. Es responsabilidad de los desarrolladores de aplicaciones escribir programas que respeten esa ordenación.

Evasión de Interbloqueos

Un método alternativo para evitar los interbloqueos consiste en requerir información adicional sobre cómo van a ser solicitados los recursos.

El modelo más simple y útil requiere que cada proceso declare el número máximo de recursos de cada tipo que puede necesitar. Proporcionando esta información de antemano, es posible construir un algoritmo que garantice que el sistema nunca entrara en estado de interbloqueo. El algoritmo de evasión de interbloqueos examina dinámicamente el estado de asignación de cada recurso con el fin de asegurar que nunca se produzca una condición de espera circular. El estado de asignación del recurso está definido por el número de recursos disponibles y asignados y la demanda máxima de los procesos.

Estado seguro

Cuando un proceso requiere un recurso disponible, el algoritmo debe decidir si su alocación inmediata deja al sistema en un estado seguro.

El sistema está en un estado seguro si existe una secuencia segura de ejecución de todos los procesos. La secuencia $<P_1,P_2,\ldots,P_n>$ es segura para el estado de asignación actual si, por cada P_i, los recursos que P_i puede aún requerir pueden ser satisfechos por recursos disponibles corrientes más los recursos mantenidos por todos los P_j, con $j<i$. En esta situación, si los recursos que P_i necesita no están inmediatamente disponibles, entonces P_i puede esperar hasta que todos los P_j hayan finalizado. Cuando P_j finaliza, P_i obtiene los recursos necesitados, ejecuta, retorna los recursos alocados y termina. Cuando P_i termina, P_{i+1} puede obtener los recursos que necesita, y así sucesivamente. Si tal secuencia no existe, entonces se dice que el estado del sistema es inseguro.

Si un sistema está en un estado seguro entonces no hay interbloqueo. Por el contrario, si un sistema está en un estado inseguro entonces hay posibilidad de interbloqueo ya que no todos los estados inseguros conducen a interbloqueos.

La evasión asegura que el sistema nunca va a entrar en un estado inseguro.

Algoritmo de Grafo de Alocación de Recursos

Si tenemos un sistema de asignación de recursos con sólo una instancia de cada tipo de recurso, puede utilizarse esta variante del grafo de asignación de recursos para evitar interbloqueos.

Se agrega en este grafo un nuevo tipo de arista, la arista de reclamo. Una arista de reclamo $P_i \rightarrow R_j$ indica que el proceso P_j puede requerir el recurso R_j; representado por una línea de puntos. La arista de reclamo se convierte en arista de requerimiento cuando el proceso requiere un recurso. De forma similar, cuando un recurso es liberado por un proceso, la arista de requerimiento se reconvierte a arista de reclamo.

Debemos observar que los recursos deben declararse de antemano al sistema, es decir, antes de que el proceso P_i inicie su ejecución, todas sus aristas de reclamo deben estar indicadas en el grafo de asignación de recursos.

Supongamos que el proceso P_i solicita el recurso R_j. La solicitud se puede conceder sólo si la conversión de la arista de requerimiento $P_i \rightarrow R_j$ en una arista de asignación $R_j \rightarrow P_i$ no da lugar a la formación de un ciclo en el grafo de asignación de recursos. Si no se crea un ciclo, entonces la asignación del recurso dejará al sistema en un estado seguro. Por el contrario, si se encuentra un ciclo, entonces la asignación colocaría al sistema en un estado inseguro. Por tanto, el proceso P_i tendrá que esperar para que su solicitud sea satisfecha.

Algoritmo del Banquero

Si tenemos un sistema de asignación de recursos con múltiples instancias de cada tipo de recursos, debe utilizarse este algoritmo para evitar interbloqueos.

Cuando entra en el sistema un proceso nuevo, debe declarar el número máximo de instancias de cada tipo de recurso que puede necesitar. Este número no puede exceder el número total de recursos del sistema. Cuando un usuario solicita un conjunto de recursos, el sistema debe determinar si la asignación de dichos recursos dejará al sistema en un estado seguro. En caso afirmativo, los recursos se asignarán; en caso contrario, el proceso tendrá que esperar hasta que los otros procesos liberen los suficientes recursos.

Deben utilizarse varias estructuras de datos para implementar el algoritmo del banquero. Estas estructuras de datos codifican el estado del sistema de asignación de recursos. Sea n el número de procesos en el sistema y m el número de tipos de recursos. Necesitamos las siguientes estructuras:

- Disponible: Un vector de longitud m que indica el número de recursos disponibles de cada tipo. Si disponible [j] = k, entonces hay k instancias del tipo de recurso R_j disponibles

- Max: Una matriz de n x m que indica la demanda máxima de cada proceso. Si Max [i,j] = k, entonces el proceso P_i puede requerir a lo sumo k instancias del recurso de tipo R_j.
- Alocación: Una matriz de n x m que define el número de recursos de cada tipo actualmente asignados a cada proceso. Si Alocación [i,j] = k entonces P_i tiene alocadas k instancias de R_j.
- Necesidad: Una matriz de n x m que indica la necesidad restante de recursos de cada proceso. Si Necesidad [i,j] = k, entonces P_i puede necesitar k instancias más de R_j para completar su tarea.

Luego se tiene la siguiente relación: Necesidad [i,j] = Max [i,j] – Alocación [i,j]

Algoritmo de Seguridad

El algoritmo de seguridad permite averiguar si un sistema se encuentra en estado seguro o no.

1. Sean Trab y Final vectores de longitud m y n, respectivamente. Se inicializan:

Trab := Disponible

Final [i] := falso para i = 1,2,3,…,n

2. Encuentre un i tal que cumplan:

a) Final [i] = = falso

b) $Necesidad_i \leq Trab$

Si tal i no existe, vaya al paso 4.

3. $Trab := Trab + Alocación_i$

Final [i] := verdadero vaya al paso 2

4. Si Final [i] == verdadero para todo i, entonces el sistema está en un estado seguro.

Algoritmo de Requerimiento de Recursos para el Proceso P_i

El algoritmo de solicitud de recursos determina si las solicitudes pueden concederse de forma segura.

Sea $Request_i$ el vector de requerimiento para el proceso P_i.

Si $Request_i[j] = = k$ entonces el proceso P_i quiere k instancias del tipo de recurso R_j.

1. Si $Request_i \leq Necesidad_i$ vaya al paso 2. Sino condición de error, dado que el proceso ha excedido su máximo.

2. Si $Request_i \leq Disponible_i$, vaya al paso 3. Sino P_i debe esperar, dado que los recursos no están disponibles.

3. Se simula alocar los recursos requeridos P_i modificando el estado como sigue:

$Disponible := Disponible – Request_i$

$Alocación_i := Alocación_i + Request_i$

$Necesidad_i := Necesidad_i - Request_i$

Si el estado resultante de la asignación de los recursos es seguro -> los recursos son alocados a P_i

Si el estado resultante de la asignación de los recursos es inseguro -> P_i debe esperar, y es restaurado el viejo estado de alocación de recursos.

Detección de Interbloqueos

Si un sistema no emplea ni algoritmos de prevención ni de evasión de interbloqueos, entonces puede producirse una situación de interbloqueo en el sistema. En este caso, el sistema debe proporcionar:

- Un algoritmo que examine el estado del sistema para determinar si se ha producir un interbloqueo
- Un algoritmo para recuperarse del interbloqueo

Es importante resaltar que los esquemas de detección y recuperación tiene un coste significativo asociado, que incluye no sólo el coste (de tiempo de ejecución) asociado con el mantenimiento de la información necesaria y la ejecución del algoritmo de detección, sino también las potenciales perdidas inherentes al proceso de recuperación de un interbloqueo.

Instancia simple de Cada Tipo de Recurso

Si todos los recursos tienen una única instancia, entonces podemos definir un algoritmo de detección de interbloqueos que utilice una variante del grafo de asignación de recursos, denominada grafo wait-for.

Los nodos son procesos. Una arista de P_i a P_j en un grafo de espera implica que el proceso P_i está esperando a que el proceso P_j libere un recurso que P_i necesita.

Como antes, existirá un interbloqueo en el sistema si y sólo si el grafo de espera contiene un ciclo. Para detectar los interbloqueos, el sistema necesita mantener el grafo de espera e invocar un algoritmo periódicamente que compruebe si existe un ciclo en el grafo. Un algoritmo para detectar un ciclo en un grafo requiere un orden de n^2 operaciones, donde n es el número de vértices del grafo.

Varias Instancias de un Tipo de Recurso

El algoritmo que da soporte a recursos con múltiples instancias de cada tipo de recursos, emplea varias estructuras de datos variables con el tiempo, que son similares a las utilizadas en el algoritmo del banquero:

- Disponible: Un vector de longitud m que indica el número de recursos disponibles de cada tipo.
- Alocación: Una matriz de n x m que define el número de recursos de cada tipo que están asignados actualmente asignados a cada proceso.
- Request: Una matriz de n x m que indica el requerimiento corriente de cada proceso. Si Request [i,j] = k, entonces el proceso P_i está requiriendo k instancias más del recurso de tipo R_j.

Algoritmo de Detección

1. Sean Trab y Final vectores de longitud m y n, respectivamente inicializados

a) Trab := Disponible

b) Para i= 1,2,…,n

si Alocación$_i$≠0,

entonces

Final[i] := falso;

sino Final[i]:=Verdadero

2. Encuentro un índice i tal que:

a) Final [i] = falso

b) Request$_i$ ≤ Trab

Sino existe tal i, vaya al paso 4.

3. Trab := Trab + Alocación$_i$

Final [i] := Verdadero

Vaya al paso 2

4. Si Final [i] == falso, para algún i, 1 ≤ i ≤ n, entonces el sistema está en un estado de interbloqueo. Es más, si Final[i] == falso, entonces P_i está interbloqueado.

Este algoritmo requiere del orden de mxn^2 operaciones para detectar si el sistema se encuentra en un estado de interbloqueo.

Uso del Algoritmo de detección

¿Cuándo debemos invocar el algoritmo de detección y con qué frecuencia? La respuesta depende de dos factores:

- ¿Con que frecuencia se producirá probablemente un interbloqueo?
- ¿Cuántos procesos se verán afectados por el interbloqueo cuando este se produzca?

Por un lado, si se invoca el algoritmo de detección de interbloqueos por cada solicitud de recursos, esto dará lugar a una considerable sobrecarga en el tiempo de uso del procesador. Una alternativa más barata consiste, simplemente, en invocar el algoritmo a intervalos menos frecuentes, por ejemplo, una vez por hora o cuando la utilización del CPU caiga por debajo del 40 por ciento. Si el algoritmo de detección se invoca en instantes de tiempo arbitrarios, pueden aparecer muchos ciclos en el grafo de recursos; en este caso, generalmente, no podremos saber cuál de los muchos procesos en el interbloqueados causo el interbloqueo.

Recuperación de Interbloqueos: Terminación de Procesos

Para eliminar los interbloqueos interrumpiendo un proceso; se utiliza uno de los dos posibles métodos. En ambos métodos, el sistema reclama todos los recursos asignados a los procesos terminados:

- Aborto todos los procesos interbloqueados: Claramente, este método interrumpirá el ciclo de interbloqueo, pero a un precio muy alto: los procesos en interbloqueo pueden haber consumido un largo periodo de tiempo y los resultados de estos cálculos parciales deben descartarse y, probablemente, tendrán que repetirse más tarde.
- Aborto un proceso a la vez hasta que el ciclo de interbloqueo haya sido eliminado: Este método requiere una gran cantidad de trabajo adicional, ya que después de haber interrumpido cada proceso hay que invocar un algoritmo de detección de interbloqueos para determinar si todavía hay procesos en interbloqueo.

Si se emplea el método de terminación parcial, entonces tenemos que determinar qué proceso o procesos en interbloqueo deben cancelarse. La cuestión es básicamente de carácter económico: debemos interrumpir aquellos procesos cuya terminación tenga un coste mínimo. Lamentablemente, el termino coste mínimo no es demasiado preciso. Hay muchos factores que influyen en qué procesos seleccionar, entre los que se incluyen:

1. La prioridad del proceso

2. Durante cuánto tiempo ha estado ejecutándose y cuánto tiempo de adicional necesita el proceso para completar sus tareas.

3. Cuántos y qué tipo de recursos ha utilizado el proceso.

4. Cuántos más recursos necesita el proceso para completarse

5. Cuantos procesos hará falta terminar.

6. Si se trata de un proceso interactivo o batch.

Recuperación de Interbloqueos: Apropiación de Recursos

Para eliminar los interbloqueos utilizando el método de apropiación de recursos, desalojamos de forma sucesiva los recursos de los procesos y asignamos dichos recursos a otros procesos hasta que el ciclo de interbloqueo se interrumpa.

Si se necesita la apropiación para tratar los interbloqueos, entonces debemos abordar tres cuestiones:

1. Selección de una víctima: Al igual que en la terminación de los procesos, es necesario determinar el orden de apropiación de forma tal que se minimicen los costos.

2. Rollback: Si nos apropiamos de un recurso de un proceso, evidentemente no puede continuar su ejecución normal, ya que no dispone de algún recurso que necesita. Debemos devolver el proceso a un estado seguro y reiniciarlo a partir de dicho estado.

Como principal ventaja este mecanismo permite en el mejor de los casos llevar al proceso unos instantes hacia atrás en lo que a tiempo de cómputo respecta y evita en dicho caso comenzar la computación nuevamente. Sin embargo, cuando retornamos a un proceso hacia atrás en el tiempo, debemos asegurarnos que este sea un estado seguro. Debido a que en general, es difícil determinar que es un estado seguro, la solución más simple termina siendo un rollback completo que equivale a abortar el proceso perdiendo todo el tiempo de cómputo.

3. Inanición: En un sistema en el que la selección de la víctima se basa fundamentalmente en factores de costo, puede ocurrir que siempre se elija el mismo proceso como víctima. Como resultado, este proceso nunca completará sus tareas, dando lugar a una situación de inanición que debe abordarse a la hora de implementar cualquier sistema real. La solución más habitual consiste en incluir el número de rollbacks en el factor de costo.

Administración de Memoria – Módulo 8

Base

La memoria está compuesta de una gran matriz de palabras, cada una con su propia dirección.

La memoria principal y los registros integrados dentro del propio procesador son las únicas áreas de almacenamiento a las que la CPU puede acceder directamente. Hay instrucciones de máquina que toman como argumentos direcciones de memoria, pero no existe ninguna instrucción que acepte direcciones de disco. Por tanto, todas las instrucciones en ejecución y los datos utilizados por esas instrucciones deberán encontrarse almacenados en uno de esos dispositivos de almacenamiento directo. Si los datos no se encuentran en memoria, deberán llevarse hasta allí antes de que la CPU pueda operar con ellos.

Generalmente, puede accederse a los registros integrados en la CPU en un único ciclo de reloj del procesador. El acceso a memoria puede requerir muchos ciclos del reloj para poderse completar, en cuyo caso el procesador necesitara normalmente detenerse, ya que no dispondrá de los datos requeridos para completar la instrucción que esté ejecutando. Esta situación es intolerable, el remedio consiste en añadir una memoria rápida entre el CPU y la memoria principal, tal memoria rápida se la conoce como memoria cache.

Requisitos de la gestión de la memoria

Reubicación

El programador no sabe en qué parte de la memoria principal se situará el programa cuando se ejecute. Mientras el programa se está ejecutando, éste puede llevarse al disco y traerse de nuevo a la memoria principal en un área diferente, en tal caso se habla de reubicación. Además deben traducirse las referencias de memoria encontradas en el código del programa en direcciones de memoria físicas.

Protección

Para asegurar una correcta operación se requiere cierta protección de memoria. Primero debemos asegurarnos de que cada proceso disponga de un espacio de memoria separado. Para hacer esto, debemos poder determinar el rango de direcciones legales a las que el proceso pueda acceder y garantizar también que el proceso sólo acceda a esas direcciones legales. Podemos proporcionar esta protección utilizando dos registros, usualmente una base y un límite. El registro base almacena la dirección de memoria física legal más pequeña, mientras que el registro limite especifica el tamaño del rango.

Es imposible comprobar las direcciones absolutas en tiempo de compilación, por lo que, deben comprobarse en el tiempo de ejecución. Es el procesador, en lugar del sistema operativo, el que debe satisfacer el requisito de protección de memoria debido a que el sistema operativo no puede anticipar todas las referencias de memoria que un programa hará.

Cualquier intento, por parte de un programa que se esté ejecutando en modo usuario, de acceder a la memoria del sistema operativo o a la memoria de los otros usuarios hará que se produzca una interrupción hacia el sistema operativo, que tratará dicho intento como un error fatal. Este esquema evita que un programa de usuario modifique el código y las estructuras de datos del sistema operativo o de otros usuarios

Compartición

Esta característica permite a varios procesos acceder a la misma porción de memoria principal. Es mejor permitir que cada proceso pueda acceder a la misma copia del programa en lugar de tener su propia copia separada.

Organización lógica

Los programas están escritos en módulos. Estos modulos se pueden escribir y compilar independientemente. Además se pueden proporcionar diferentes grados de protección a los módulos (sólo lectura, sólo ejecución) y se pueden compartir los módulos entre procesos.

Organización Física

La memoria principal disponible para un programa más sus datos podría ser insuficiente. La superposición (overlaying) permite asignar la misma región de memoria a varios módulos. Dada esta técnica el programador no conoce cuánto espacio estará disponible.

Mapeo de Instrucciones y Datos a Direcciones de Memoria

Clásicamente, la reasignación de instrucciones y datos a direcciones de memoria puede ocurrir en tres etapas diferentes.

- Tiempo de Compilación: Si sabemos en el momento de realizar la compilación dónde va a residir el proceso en memoria, podremos generar código absoluto. SI la ubicación inicial cambiase en algún instante posterior, entonces sería necesario recompilar ese código.
- Tiempo de Carga: Si no conocemos en tiempo de compilación dónde va a residir el proceso en memoria, el compilador deberá generar código reubicable. En este caso, se retarda la reasignación final hasta el momento de la carga. Si cambia la dirección inicial, tan sólo es necesario volver a cargar el código de usuario para incorporar el valor modificado.
- Tiempo de ejecución: Si el proceso puede desplazarse durante su ejecución desde un segmento de memoria a otro, entonces es necesario retardar la reasignación hasta el instante de ejecución. Para que este esquema pueda funcionar, será preciso disponer de hardware especial para el mapeo de direcciones (p.e., registros base-limite).

Carga dinámica

Con la carga dinámica, una rutina no se carga hasta que se la invoca; todas las rutinas se mantienen en disco en un formato de carga reubicable.

La ventaja del mecanismo de carga dinámica es que una rutina no utilizada no se cargará nunca en memoria. Este método resulta particularmente útil cuando se necesitan grandes cantidades de código para gestionar casos que sólo ocurren de manera infrecuente, como por ejemplo rutinas de error. En este caso, aunque el tamaño total del programa puede ser grande, la porción se que utilice (y que por tanto se cargue) puede ser mucho más pequeña.

El mecanismo de carga dinámica no requiere ningún soporte especial por parte del sistema operativo. Es responsabilidad de los usuarios diseñar programas que puedan aprovechar este método.

Enlace dinámico

El concepto de enlace dinámico es similar al de la carga dinámica, aunque en este caso lo que se pospone hasta el momento de la ejecución es el montaje, en lugar de la carga. Esta funcionalidad suele emplearse con las bibliotecas del sistema, como por ejemplo las bibliotecas de subrutinas del lenguaje.

Con el enlace dinámico, se incluye un stub dentro de la imagen binaria para cada referencia a una rutina de biblioteca. El stub es un pequeño fragmento de código que indica como localizar la rutina adecuada de biblioteca residente en memoria o como cargar la biblioteca si esa rutina no está todavía presente. Cuando se ejecuta el stub, éste comprueba si la rutina necesaria ya se encuentra en memoria; si no es así, el programa carga en memoria la rutina. En cualquier de los casos, el stub se sustituye así mismo por la dirección de la rutina y ejecuta la rutina.

A diferencia de la carga dinámica, el montaje dinámico suele requerir algo de ayuda por parte del sistema operativo. Si los procesos de la memoria están protegidos unos de otros, entonces el sistema operativo será la única entidad que pueda comprobar si la rutina necesaria se encuentra dentro del espacio de memoria de otro proceso y será también la única entidad que pueda permitir a múltiples procesos acceder a las mismas direcciones de memoria.

Espacio de Direcciones Lógico vs. Físico

El concepto de espacio de direcciones lógico que está limitado a un espacio de direcciones físicas es central a la administración de la memoria. Una dirección generada por la CPU se denomina dirección lógica, mientras que una dirección vista por la unidad de memoria se denomina dirección física.

Las direcciones lógicas y físicas son las mismas en tiempo de compilación y en los esquemas de mapeo de direcciones en tiempo de carga; las direcciones lógicas y físicas difieren en el esquema de mapeo de direcciones en tiempo de ejecución.

Unidad de Administración de Memoria (MMU)

La correspondencia entre direcciones virtuales y físicas en tiempo de ejecución es establecida por un dispositivo de hardware que se denomina unidad de administración de memoria (MMU, memory management unit).

En el esquema MMU, el valor en el registro de locación es agregado a cada dirección generada por el proceso del usuario en el momento que es presentada a la memoria. Los programas de usuario ven direcciones lógicas, nunca ven direcciones físicas reales.

Intercambio (Swapping)

Un proceso debe estar en memoria para ser ejecutado. Sin embargo, los procesos pueden ser intercambiados temporalmente, sacándolos de la memoria y almacenándolos en un almacenamiento de respaldo (backing store), y luego ser vuelto a la misma para continuar su ejecución.

Esta estrategia se emplea por ejemplo en un entorno de multiprogramación con el algoritmo de planificación de CPU round robin.

Los mecanismos de intercambio requieren un backing store, que normalmente será un disco suficientemente rápido. El disco debe ser también lo suficientemente grande como para poder albergar copias de todas las imágenes de memoria para todos los usuarios, y debe proporcionar un acceso directo a estas imágenes de memoria.

Hay una metodología que se denomina roll out, roll in, la cual es una variante del intercambio usado en algoritmo de planificaciones por prioridades; procesos con baja prioridad son intercambiados con procesos de alta prioridad que pueden ser cargados y ejecutados.

De observarse que la mayor parte del tiempo de intercambio es tiempo de transferencia. El tiempo de transferencia total es directamente proporcional a la cantidad de memoria intercambiada.

Además el intercambio está restringido también por otros factores. Si queremos intercambiar un proceso, debemos asegurarnos que esté completamente inactivo.

Actualmente, estos mecanismos estándar de intercambio se utilizan en muy pocos sistemas. Dicho mecanismo requiere de un tiempo muy alto y proporciona un tiempo de ejecución demasiado pequeño como para constituir una solución razonable de gestión de memoria. Sin embargo, lo que si podemos encontrar en muchos sistemas son versiones modificadas de este mecanismo de intercambio.

Alocación Contigua

La memoria principal está usualmente dividida en dos particiones: una para el sistema operativo residente y otra para los procesos de usuario. Podemos situar el sistema operativo en la zona baja o en la zona alta de la memoria. El principal factor que afecta a esta decisión es la ubicación del vector de interrupciones. Puesto que el vector de interrupciones se encuentra a menudo en la parte baja de la memoria, los programadores tienen a situar también el sistema operativo en dicha zona.

Bajo la alocación en partición simple, se usa un esquema de registro de realocación para proteger los procesos de usuario uno de otro, y cambios en el código y datos del SO.

El registro de realocación contiene el valor de la dirección física más pequeña; el registro limite contiene el rango de las direcciones lógicas, cada dirección lógica debe ser menor que el registro limite; la MMU convertirá la dirección lógica dinámicamente sumándole el valor contenido en el registro de reubicación. Esta dirección es la que se envía a la memoria.

Uno de los métodos más simples para asignar la memoria consiste en dividirla en particiones de tamaño fijo. Cada partición puede contener exactamente un proceso, de modo que el grado de multiprogramación estará limitado por el número de particiones disponibles. En este método de particiones múltiples, cuando una partición esta libre, se selecciona un proceso de la cola de entrada y se lo carga en dicha partición. Cuando el proceso termina, la partición pasa a estar disponible para otro proceso.

En el esquema de particiones fijas, el sistema operativo mantiene una tabla que indica qué partes de la memoria están disponibles y cuales están ocupadas. Inicialmente, toda la memoria está disponible para los procesos de usuario y se considera como un único bloque de gran tamaño de memoria disponible, al que se denomina agujero. Cuando llega un proceso y necesita memoria, busquemos un agujero lo suficientemente grande como para albergar este proceso. Si lo encontramos, sólo se asigna la memoria justa necesaria, manteniendo el resto de la memoria disponible para satisfacer futuras solicitudes.

En cualquier momento determinado, tendremos una lista de tamaños de bloque disponibles y una cola de entrada de procesos.

La asignación dinámica de espacio de almacenamiento, se ocupa de cómo satisfacer una solicitud de tamaño n a partir de una lista de agujeros libres. Hay muchas soluciones a este problema, y las estrategias más comúnmente utilizadas para seleccionar un agujero libre entre el conjunto de agujeros disponibles son primer lugar, próximo lugar, mejor lugar y peor lugar.

- Primer lugar. Aloca en el primer agujero lo suficientemente grande.
- Próximo lugar. Aloca en el próximo lugar luego de haber usado el primero.
- Mejor lugar. Aloca en el agujero más chico que lo puede alojar; debe buscar en la lista completa, salvo que sea ordenada por tamaño.
- Peor lugar. Aloca el agujero más grande; debe buscar en la lista completa. Produce el agujero restante más grande.

Las simulaciones muestran que tanto la estrategia de primer lugar como la de mejor lugar son mejores que la de peor lugar en términos de velocidad y utilización del almacenamiento.

Buddy System (Alocación)

Este es un sistema de alocación de memoria utilizado por algunos sistemas operativos. Aquí el espacio completo disponible es tratado como un bloque de tamaño 2^U.

Bajo este esquema, si surge un requerimiento de tamaño s tal que $2^{U-1} < s \leq 2^U$, es alocado el bloque completo. De otra manera el bloque es dividido en dos "compañeros" iguales. El proceso continua hasta que es generado el bloque más pequeño mayor o igual que s.

Fragmentación

El problema de la fragmentación externa aparece cuando hay un espacio de memoria total suficiente como para satisfacer una solicitud, pero esos espacios disponibles no son contiguos; el espacio de almacenamiento está fragmentado en un gran número de pequeños agujeros.

Dependiendo de la cantidad total de espacio de memoria y del tamaño medio de los procesos, esa fragmentación externa puede ser un problema grave o no.

La técnica general para evitar este problema consiste en descomponer la memoria física en bloques de tamaño fijo y asignar la memoria en unidades basadas en el tamaño del bloque. Con esta técnica, la memoria asignada a un proceso puede ser ligeramente superior a la memoria solicitada. La diferencia entre estos dos valores será la fragmentación interna, es decir, la memoria que es interna a una partición pero que no está siendo utilizada.

Otra solución al problema de la fragmentación externa consiste en la compactación. El objetivo es mover el contenido de la memoria con el fin de situar toda la memoria libre de manera contigua, para formar un único bloque de gran tamaño. Sin embargo, la compactación no es siempre posible. La compactación solo es posible si la reubicación es dinámica y se lleva a cabo en tiempo de ejecución. Además pueden surgir problemas de E/S, por lo que se aconseja dejar el trabajo en memoria mientras está involucrando en una E/S y hacer E/S solo en los buffers del SO.

El algoritmo de compactación más simple consiste en mover todos los procesos hacia uno de los extremos de la memoria; de esta forma, todos los agujeros se moverán en la otra dirección, produciendo un único agujero de memoria disponible de gran tamaño. Sin embargo, este esquema puede ser muy caro de implementar.

Otra posible solución al problema de la fragmentación externa consiste en permitir que el espacio de direcciones lógicas de los procesos no sea contiguo, lo que hace que podamos asignar memoria física a un proceso con independencia de dónde esté situada dicha memoria. Hay dos técnicas complementarias que permiten implementar esta solución: la paginación y la segmentación y pueden combinarse.

Paginado

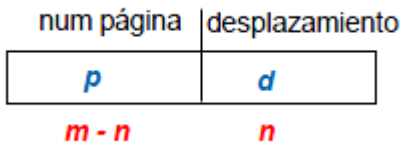
La paginación es un esquema de gestión de memoria que permite que el espacio de direcciones físicas de un proceso no sea contiguo. La paginación evita el considerable problema de encajar los fragmentos de memoria de tamaño variable en el backing store.

El método básico para implementar la paginación implica descomponer la memoria física en una serie de bloques de tamaño fijo denominados cuadros y descomponer la memoria lógica en bloques del mismo tamaño denominados páginas. Cuando hay que ejecutar un proceso, sus páginas se cargan desde el backing store en los cuadros de memoria disponibles. El backing store está dividido en bloques de tamaño fijo que tienen el mismo tamaño que los cuadros de memoria.

Se guarda información de todos los cuadros libres y se implementa una tabla de páginas que permite traducir las direcciones lógicas a físicas.

Toda dirección generada por la CPU está dividida en dos partes: un número de página (p) y un desplazamiento de página (d). El número de página se utiliza como índice para una tabla de páginas. La tabla de páginas contiene la dirección base de cada página en memoria física; esta dirección base se combina con el desplazamiento de página para definir la dirección física que se envía la unidad de memoria.

El tamaño de página al igual que el del cuadro, está definido por el hardware. El tamaño de página es, normalmente, una potencia de 2, variando entre 512 bytes y 16 MB por página, dependiendo de la arquitectura de la computadora. La selección de una potencia de 2 como tamaño de página hace que la traducción de una dirección lógica a un número de página y a un desplazamiento de página resulte particularmente fácil. Si el tamaño del espacio de direcciones lógicas es 2^m y el tamaño de página 2^n unidades de direccionamiento, entonces los m-n bits superiores de cada dirección lógica designaran el número de página, mientras que los n bits inferiores indicarán el desplazamiento de página. Por tanto, la dirección lógica tiene la estructura siguiente:



Donde p es un índice de la tabla de páginas y d es el desplazamiento dentro de la página.

Cuando usamos un esquema de paginación, no tenemos fragmentación externa: todos los cuadros libres podrán ser asignados a un proceso que los necesite. Sin embargo, lo que si podemos tener es un cierto grado de fragmentación interna, ya que los cuadros se asignan como unidades y, si los requisitos de memoria de un proceso no coinciden exactamente con las fronteras de página, el último cuadro asignado puede no estar completamente lleno.

Cuando llega un proceso al sistema para ejecutarlo, se examina el tamaño expresado en páginas. Cada página del proceso necesitara un cuadro. Por tanto, si el proceso requiere n páginas, deberá haber disponibles al menos n cuadros en memoria.

Hardware de Paginado

La tabla de páginas se mantiene en memoria principal, utilizándose un registro base de tabla de páginas (PTBR, page table base register) para apuntar a la tabla de páginas. Para cambiar las tablas de páginas, sólo hace falta cambiar este único registro, reduciéndose así sustancialmente el tiempo de cambio de contexto. El registro de longitud de la tabla de páginas (PTLR) indica el tamaño de la tabla de páginas.

En este esquema cada acceso a instrucción/dato requiere dos accesos a memoria. Uno para la tabla de páginas y otro para la instrucción/dato.

El problema del doble acceso puede ser resuelto por el uso de una cache hardware especial de adelantamiento llamado registro asociativo o translation look-aside buffers (TLBs). El búfer TLB es una pequeña memoria asociativa de alta velocidad. El búfer TLB se utiliza con las tablas de página de la forma siguiente: el búfer TLB contiene sólo unas cuantas entradas de la tabla de páginas; cuando la CPU genera una dirección lógica, se presenta el número de página al TLB y si se encuentra ese número de página, su número de cuadro correspondiente estará inmediatamente disponible y se utilizara para acceder a la memoria. Si el número de página no se encuentra en el búfer TLB, es necesario realizar una referencia a memoria para consultar la tabla de páginas. Una vez obtenido el número de cuadro, podemos emplearlo para acceder a la memoria. Además, podemos añadir el número de página y el número de marco al TLB, para poder encontrar los correspondientes valores más rápidamente en la siguiente referencia que se realice. Si el TLB ya está lleno, el sistema operativo deberá seleccionar una de las entradas para sustituirla.

Tiempo Efectivo de Acceso

Supongamos que el ciclo de memoria es de 1 microsegundo y la búsqueda asociativa demanda una unidad de tiempo ε, luego la relación de acierto es el porcentaje de veces que la página es encontrada en los registros asociativos; y está relacionado con el número de registros asociativos.

Si la relación de acierto es α entonces el tiempo efectivo de acceso (TEA) es

$$TEA = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)\,(1 - \alpha) = 2 + \varepsilon - \alpha$$

Protección de Memoria

La protección de memoria en un entorno paginado se consigue mediante una serie de bits de protección asociados con cada cuadro. Normalmente, estos bits se mantienen en la tabla de páginas.

Uno de los bits puede definir una página como de lectura-escritura o de sólo lectura. Podemos ampliar fácilmente este enfoque con el fin de proporcionar un nivel más fino de protección brindando protección de sólo lectura, de lectura escritura o de sólo ejecución, o podemos permitir cualquier combinación de estos accesos, proporcionando bits de protección separados para cada tipo de acceso. Los intentos ilegales provocaran una interrupción hardware hacia el sistema operativo.

Generalmente, se suele asociar un bit adicional con cada entrada de la tabla de páginas, un bit de válido-inválido. Cuando se configura este bit como “valido”, la página asociada se encontrara en el espacio de direcciones lógicas del proceso y será, por tanto, una página legal. Cuando se configura el bit como “inválido”, la página no se encuentra en el espacio de direcciones lógicas del proceso. Las direcciones ilegales se capturan utilizando este bit. El sistema operativo configura este bit para cada página, con el fin de permitir o prohibir el acceso a dicha página.

También se utiliza el registro PTLR como mecanismo de protección. Este se compara con cada dirección lógica para verificar que esa dirección se encuentre dentro del rango de direcciones válidas definidas para el proceso. Si esta comprobación fallara, se produciría una interrupción de error dirigida al sistema operativo.

Tablas de Páginas Multinivel

La mayoría de los sistemas informáticos modernos soportan un gran espacio de direcciones lógicas. En este tipo de entorno, la propia tabla de páginas llega a ser excesivamente grande. Obviamente, no conviene asignar la tabla de páginas de forma contigua en memoria principal. Una solución simple a este problema consiste en dividir la tabla de páginas en fragmentos más pequeños y podemos llevar a cabo esta división en formas distintas.

Una de esas formas consiste en utilizar un algoritmo de paginación en dos niveles, en el que la propia tabla de páginas esta también paginada. Puesto que la traducción de las direcciones se realiza comenzando por la tabla de páginas

externa y continuando luego por la interna, este esquema también se conoce con el nombre de tabla de páginas con mapeo directo. También se pueden establecer configuraciones en más niveles.

Tablas de Páginas con Hash

Una técnica común en espacios de direcciones superiores a 32 bits consiste en utilizar una tabla hash de páginas, donde el valor hash es el número de página virtual. Cada entrada de la tabla hash contiene una lista enlazada de elementos que tiene como valor hash una misma ubicación. Cada elemento está compuesto de tres campos: el número de página virtual, el valor del cuadro de página mapeado y un punto del siguiente elemento en la lista enlazada.

El algoritmo funciona de la forma siguiente: al número de página virtual de la dirección virtual se le aplica una función hash para obtener un valor que se utiliza como índice para la tabla hash. El número de página virtual se compara con el campo 1 del primer elemento de la lista enlazada. Si hay una correspondencia, se utiliza el cuadro de página correspondiente para formar la dirección física deseada. Si no se produce una correspondencia, se exploran las subsiguientes entradas de la lista enlazada hasta encontrar el correspondiente número de página virtual.

Tablas de Páginas Invertida

La tabla de páginas incluye una entrada por cada página que el proceso esté utilizando.

El sistema operativo debe entonces traducir cada referencia a una dirección física de memoria. Puesto que la tabla esta ordenada según la dirección virtual, el sistema operativo podrá calcular dónde se encuentra en la tabla la entrada de la dirección física asociada y podrá utilizar dicho valor directamente. Pero una de las desventajas de este método es que cada tabla de página puede estar compuesta por millones de entradas. Estas tablas pueden ocupar una gran cantidad de memoria física, simplemente para controlar el modo en que se están utilizando otras partes de la memoria física.

Para resolver este problema, podemos utilizar una tabla de páginas invertida. Las tablas de páginas invertidas tienen una entrada por cada página real o cuadro de la memoria. Cada entrada está compuesta de la dirección virtual de la página almacenada en dicha ubicación de memoria real, e incluye información acerca del proceso que posee dicha página. Por tanto, en el sistema sólo habrá una única tabla de páginas y esa tabla sólo tendrá una entrada por cada página de la memoria física.

Aunque este esquema permite reducir la cantidad de memoria necesaria para almacenar cada tabla de páginas, incrementa la cantidad de tiempo necesaria para explorar la tabla cuando se produce la referencia a una página. Puesto que la tabla de páginas invertida está ordenada según la dirección física, pero las búsquedas se producen según las direcciones virtuales, puede que sea necesario explorar toda la tabla hasta encontrar una correspondencia y esta exploración consumiría demasiado tiempo. Para aliviar este problema, se utiliza una tabla hash, con el fin de limitar la búsqueda a una sola entrada de la tabla de páginas o a unas pocas entradas.

Páginas compartidas

Una ventaja de la paginación es la posibilidad de compartir código común. Esta consideración es particularmente importante en un entorno de tiempo compartido.

Una copia de código de lectura solamente (reentrante) es compartido entre procesos (p.e. editores de texto, compiladores, sistemas de ventanas). El código compartido debe aparecer en la misma locación en el espacio de direcciones de todos los procesos.

No obstante, hay código privado y datos. Cada proceso guarda una copia separada de esta información y las páginas de código privado y datos pueden aparecer en cualquier lugar del espacio de direcciones lógico.

Segmentación

Los usuarios prefieren ver la memoria como una colección de segmentos de tamaño variable, sin que exista necesariamente ninguna ordenación entre dichos segmentos.

La segmentación es un esquema de gestión de memoria que soporta la visión del usuario de la memoria. Un espacio lógico de direcciones es una colección de segmentos y cada segmento tiene un número de segmento y una longitud. Un programa es una colección de segmentos. Un segmento es una unidad lógica como: programa principal, procedimiento, función, etc.

Arquitectura de Segmentación

Las direcciones especifican tanto el número del segmento como el desplazamiento dentro de este segmento. La tabla de segmentos mapea direcciones físicas en dos dimensiones; cada entrada de la tabla de segmentos tiene una dirección base del segmento y un límite del segmento. La dirección base del segmento contiene la dirección física inicial del lugar donde el segmento reside dentro de la memoria, mientas que el límite del segmento especifica la longitud de éste.

Además se cuenta con el registro base de la tabla de segmento (STBR) que apunta a la locación de la tabla de segmentos en memoria y el registro de longitud de la tabla de segmentos (STLR) indica el número de segmentos usados por el programa, donde el número de segmento s es lega si s<STLR.

Con cada entrada en la tabla de segmentos se asocia un bit de validación. Si el bit de validación es cero entonces el segmento es ilegal. También se puede extender este esquema para brindar una granularidad más fina con privilegios de lectura, escritura o ejecución.

Dado que los segmentos varían en longitud, la alocación de memoria es un problema de alocación dinámica de almacenaje. Para este problema se acude a la utilización de la política de primer lugar y mejor lugar. Claramente esto produce problemas de fragmentación externa. Esto es espacio de memoria de tamaño suficiente para el segmento libre pero no contigua.

Por último, cabe subrayar, que para compartir memoria sólo se debe poseer el mismo número de segmento.

Memoria Virtual – Módulo 9

Base

La memoria virtual incluye la separación de la memoria lógica del usuario de la memoria física. Esta separación permite proporcionar a los programadores una memoria virtual extremadamente grande, cuando sólo está disponible una memoria física de menor tamaño.

El espacio de direcciones virtuales de un proceso hace referencia a la forma lógica de almacenar un proceso en la memoria. Típicamente, esta forma consiste en que el proceso comienza en una cierta dirección lógica y está almacenado de forma contigua en la memoria.

Cabe destacar que, dejamos que el heap crezca hacia arriba en la memoria, ya que se utiliza para la asignación dinámica de memoria. De forma similar, permitimos que la pila crezca hacia abajo con las sucesivas llamadas a función. El gran espacio vacío entre el heap y la pila forma parte del espacio de direcciones virtual.

Además de separar la memoria lógica de la memoria física, la memoria virtual también permite que dos o más procesos compartan archivos y la memoria mediante mecanismos de compartición de páginas. Esto proporciona las siguientes ventajas:

- La memoria virtual reduce la memoria que ocupan los procesos

- La memoria virtual puede permitir que se compartan página durante la creación de procesos mediante la llamada al sistema fork(), acelerando así la tarea de creación de procesos.

Paginado por Demanda

Considerando que generalmente no necesitamos inicialmente todo el programa en la memoria, una alternativa viable consiste en cargar inicialmente las páginas únicamente cuando sean necesarias. Esta técnica se denomina paginación bajo demanda y se utiliza comúnmente en los sistemas de memoria virtual. Con la memoria virtual basada en paginación bajo demanda, sólo se cargan las páginas cuando así se solicita durante la ejecución del programa; de este modo, las páginas a las que nunca se acceda no llegarán a cargarse en la memoria física. También se puede emplear segmentación por demanda.

Esta mecánica permite que sean necesarias menos operaciones de E/S, menos memoria física y ayuda a alcanzar una respuesta más rápida.

Se utiliza un intercambiador perezoso el cual jamás intercambia una página con la memoria a menos que esa página vaya a ser necesaria. El intercambiador (swapper) que trata con páginas es un paginador (pager), ya que este último solo se ocupa de páginas individuales de un proceso no de manipular procesos completos como lo hace el intercambiador.

Bit Válido-Inválido

Se necesita algún tipo de soporte hardware para distinguir entre las páginas que se encuentran en memoria y las páginas que residen en disco. Podemos usar para este propósito un bit de válido-inválido el cual se asocia a cada entrada a la tabla de páginas. Cuando se configura este bit como válido, la página asociada será legal y además se encontrará en la memoria. Si el bit se configura como inválido, querrá decir que o bien la página no es válida o es válida pero está actualmente en el disco, por lo que se habla de una falta de página.

Falta de Página

Si el proceso trata de acceder a una página que no haya sido cargada en memoria, se produce una falta de página. El hardware de paginación, al traducir la dirección mediante la tabla de páginas, detectará que el bit de página inválida esta activado, provocando una interrupción dirigida al sistema operativo.

El procedimiento para tratar este fallo de página es muy sencillo

1. Comprobamos una tabla interna (que usualmente se mantiene el PCB) correspondiente a este proceso, para determinar si la referencia es un acceso de memoria válido o inválido.

2. Si la referencia era inválida, terminamos el proceso. Si era válida pero esa página todavía no había sido cargada, la cargamos en memoria.

3. Buscamos un cuadro libre (por ejemplo, tomando uno de la lista de cuadros libres).

4. Ordenamos una operación de disco para leer la página deseada en el marco recién llegado.

5. Una vez completada la lectura de disco, modificamos la tabla interna que se mantiene con los datos del proceso y la tabla de páginas para indicar que dicha página se encuentra ahora en memoria.

6. Reiniciamos la instrucción que fue interrumpida. El proceso podrá ahora acceder a la página como si siempre hubiera estado en memoria.

Performance del Paginado por Demanda

En el caso extremo, podríamos empezar a ejecutar un proceso sin ninguna página en memoria. Cuando el sistema operativo hiciera que el puntero de instrucciones apuntara a la primera instrucción del proceso, que se encontraría en

una página no residente en memoria, se produciría inmediatamente un fallo de página. Después de cargar dicha página en memoria, el proceso continuaría ejecutándose, generando los fallos de página necesarios hasta que todas las páginas requeridas se encontraran en la memoria. A partir de ahí, podría ejecutarse sin ningún fallo de página. Este esquema sería una paginación bajo demanda pura: nunca cargar una página en memoria hasta que sea requerida.

Si bien en teoría, algunos programas podrían acceder a varias nuevas páginas con cada ejecución de una instrucción, posiblemente generando múltiples fallos de página por cada instrucción, y esta situación provocaría una degradación inaceptable en el rendimiento del sistema, afortunadamente, el análisis de la ejecución de los procesos muestra que este comportamiento es bastante improbable. Los programas tienden a tener lo que se denomina localidad de referencia, por lo que la paginación bajo demanda puede ser una alternativa viable.

El hardware necesario para soportar la paginación bajo demanda es el mismo que para los mecanismos de paginación e intercambio: una tabla de páginas con bits de protección y la memoria secundaria. Un requisito fundamental para la paginación bajo demanda es la necesidad de poder reiniciar cualquier instrucción después de un fallo de página.

La paginación bajo demanda puede afectar significativamente el rendimiento de un sistema informático. Para ver porque vamos a calcular el tiempo efectivo de acceso a una memoria con paginación bajo demanda.

Sea p la probabilidad de que se produzca un fallo de página ($0 \leq p \leq 1$). Cabe esperar que p esté próxima a cero, es decir, que sólo vayan a producirse unos cuantos fallos de página. El tiempo efectivo de acceso será entonces:

TEA = (1-p) x acceso a memoria + p x (sobrecarga de falta de página + salida de la página + entrada de la página + sobrecarga de reinicio).

Ejemplo de Paginado por Demanda

Sea el tiempo de acceso a memoria = 200 nanosegundos

Sea el tiempo promedio de servicio de una falta de página = 8 milisegundos.

TEA = (1 - p) x 200 + p x (8 milisegundos) = (1 – p) x 200 + p x 8000000 = 200 + p x 7999800

Podemos ver, entonces, que el tiempo de acceso efectivo es directamente proporcional a la tasa de fallos de página. Si sólo un acceso de cada 1000 provoca un fallo de página, el tiempo efectivo de acceso será de 8,2 microsegundos. La computadora se ralentizará, por tanto, según el factor de 40 debido a la paginación bajo demanda.

¿Qué ocurre cuando no hay cuadros libres?

Imaginemos que, cuando se está ejecutando un proceso de usuario, se produce un fallo de página. El sistema operativo determina dónde reside la página deseada dentro del disco y entonces se encuentra con que no hay ningún cuadro libre en la lista de cuadros libres; toda la memoria está siendo utilizada.

Si bien el sistema operativo dispone de varias posibilidades en este punto, la solución más comúnmente utilizada es la sustitución de páginas.

Reemplazo de Páginas

La sustitución de páginas usa la siguiente técnica: Si no hay ningún marco libre, localizamos uno que no esté siendo actualmente utilizado y lo liberamos.

Modifiquemos la rutina de servicio del fallo de página para incluir este mecanismo de sustitución de páginas:

1. Hallar la ubicación de la página deseada dentro del disco

2. Localizar un marco libre:

a. Si hay un marco libre, utilizarlo

b. Si no hay ningún marco libre, utilizar un algoritmo de sustitución de páginas para seleccionar un marco víctima.

c. Escribir el marco victima en el disco; cambiar las tablas de páginas y de cuadros correspondientes.

3. Leer la página deseada y cargarla en el marco recién liberado; cambiar las tablas de páginas y de cuadros.

4. Reiniciar el proceso de usuario.

Debe observarse que, si no hay ningún marco libre, se necesitan dos transferencias de páginas (una descarga y una carga). Esta situación duplica, en la práctica, el tiempo de servicio del fallo de página e incrementa correspondientemente el tiempo efectivo de acceso. Podemos reducir esta carga de trabajo adicional utilizando un bit de modificación (o bit de sucio). Cuando se utiliza este esquema, cada página o marco tiene asociado un bit de modificación. El bit de modificación para una página es activado por el hardware cada vez que se escribe en una palabra, para indicar que la página ha sido modificada. Cuando seleccionamos una página para sustitución, examinamos su bit de modificación. Si el bit está activado, sabremos que la página ha sido modificada desde que se leyó del disco. En este caso, deberemos escribir dicha página en el disco. Sin embargo, si el bit de modificación no es activado, la página no habría sido modificada desde que fue leída y cargada en memoria. Por tanto, si la copia de la página en el disco no ha sido sobrescrita no necesitaremos escribir la página de memoria en el disco, puesto que ya se encuentra allí. Este esquema puede reducir significativamente el tiempo requerido para dar servicio a un fallo de página, ya que reduce a la mitad el tiempo de E/S si la página no ha sido modificada.

El mecanismo de sustitución de páginas resulta fundamental para la paginación bajo demanda. Completa la separación entre la memoria lógica y la memoria física y, con este mecanismo, se puede proporcionar a los programadores una memoria virtual de gran tamaño a partir de una memoria física más pequeña.

Algoritmos de Reemplazo de Páginas

Debemos resolver dos problemas principales a la hora de implementar paginación bajo demanda: hay que desarrollar un algoritmo de asignación de cuadros y un algoritmo de sustitución de páginas. Si tenemos múltiples procesos en memoria, debemos decidir cuántos cuadros asignar a cada proceso. Además, cuando se necesita una sustitución de páginas, debemos seleccionar los cuadros que hay que sustituir. Debido a que existen muchos algoritmos de sustitución de páginas, en general, intentaremos seleccionar aquel que tenga la tasa de fallos de página más baja.

Podemos evaluar un algoritmo ejecutándolo con una cadena concreta de referencias de memoria y calculando el número de fallos de páginas. La cadena de referencias a memoria se denomina cadena de referencia.

Reemplazo de Páginas PEPS

El algoritmo más simple de sustitución de páginas es el algoritmo FIFO (first-in, first out). El algoritmo de sustitución FIFO asocia con cada página el instante en que dicha página fue cargada en memoria. Cuando hace falta sustituir una página, se elige la página más antigua. Con objeto de registrar el instante, se utiliza una cola FIFO para almacenar todas las páginas y sustituir la página situada al principio de la cola. Cuando se carga en memoria una página nueva, se la inserta al final de la cola.

Este algoritmo de sustitución es fácil de entender y de programar. Sin embargo, su rendimiento no siempre es bueno. Por un lado, la página sustituida puede ser un modulo de inicialización que hubiera sido empleado hace mucho tiempo y que ya no es necesario, también podría contener una variable muy utilizada que fuera inicializada muy pronto y se utilice de forma constante.

Este algoritmo sufre de la anomalía de Belady: para algunos algoritmos de sustitución de páginas la tasa de fallos de página puede incrementarse a medida que se incrementa el número de cuadros asignados.

Reemplazo de Páginas Óptimo

Uno de los resultados de la anomalía de Belady fue la búsqueda de un algoritmo óptimo de sustitución de páginas. Un algoritmo óptimo de sustitución de páginas es aquel que tenga la tasa más baja de fallos de página de entre todos los algoritmos y que nunca este sujeto a la anomalía de Belady. A este algoritmo también se lo conoce como mínimo.

Consiste simplemente en sustituir la página que no vaya a ser utilizada durante el período de tiempo más largo.

La utilización de este algoritmo de sustitución de página garantiza la tasa de fallos de página más baja posible para un número fijo de cuadros.

Desafortunadamente, el algoritmo óptimo de sustitución de páginas resulta difícil de implementar, porque requiere un conocimiento futuro de la cadena de referencia. Como resultado, el algoritmo óptimo se utiliza principalmente con propósitos comparativos.

Reemplazo de Páginas LRU

La distinción clave entre los algoritmos FIFO y OPT es que el algoritmo FIFO utiliza el instante en el que se cargó una página en memoria, mientras que el algoritmo OPT utiliza el instante en el que hay que utilizar una página. Si utilizamos el pasado reciente como aproximación del futuro próximo, podemos sustituir la página que no haya sido utilizada durante el periodo más largo de tiempo. Esta técnica se conoce como algoritmo LRU (least-recently-used, menos recientemente usada).

Este algoritmo asocia con cada página el instante correspondiente al último uso de dicha página. Cuando hay que sustituir una página, el algoritmo LRU selecciona la página que no haya sido utilizada durante el periodo de tiempo más largo.

Esta política de LRU se utiliza a menudo como algoritmo de sustitución de páginas y se considera que es bastante buena. El principal problema es cómo implementar ese mecanismo de sustitución LRU ya que puede requerir una considerable asistencia del hardware. El problema consiste en determinar un orden para los cuadros definido por el instante correspondiente al último uso. Existen dos posibles implementaciones:

- Contadores. En el caso más simple, asociamos con cada entrada en la tabla de páginas un campo de tiempo de uso y añadimos a la CPU un contador. Cuando se realiza una referencia a una página, se copia el contenido del registro de reloj en el campo de tiempo de uso de la entrada de la tabla de páginas correspondiente a dicha página. A pesar de que este esquema nos permite contar siempre con el tiempo de la ultima referencia a cada página, cuando una página necesita ser reemplazada, se requiere realizar una búsqueda en la tabla de páginas para localizar la página menos recientemente utilizada y realizar una escritura en memoria para cada acceso a memoria.
- Pila. Aquí se mantiene una pila de números de página. Cada vez que se hace referencia a una página, se extrae esa página de la pila y se la coloca en la parte superior. De esta forma, la página más recientemente utilizada se encontrará siempre en la parte superior de la pila y la menos recientemente utilizada en la inferior. Puesto que es necesario eliminar las entradas de la parte intermedia de la pila, lo mejor para implementar este mecanismo es utilizar una lista doblemente enlazada. Si bien cada actualización es más costosa que con otros métodos, no hay necesidad de buscar la página que hay que sustituir.

Tanto el algoritmo óptimo como LRU pertenecen a una clase de algoritmos de sustitución de páginas, denominada algoritmos de pila, que jamás pueden exhibir la anomalía de Belady. Un algoritmo de pila es un algoritmo para el que puede demostrarse que el conjunto de páginas en memoria estaría formado por las n páginas más recientemente

referenciadas. Si se incrementa el número de cuadros, estas n páginas seguirán siendo las más recientemente referencias y, por tanto, continuaran estando en la memoria.

Algoritmos de Aproximación a LRU

Pocos sistemas informáticos proporcionan el suficiente soporte hardware como para implementar un verdadero algoritmo LRU de sustitución de páginas. De mínima, algunos sistemas proporcionan un bit de referencia.

Inicialmente, todos los bits son desactivados por el sistema operativo. A medida que se ejecuta un proceso de usuario, el bit asociado con cada una de las páginas referenciadas es activado por el hardware. Después de un cierto tiempo, podemos determinar qué paginas se han utilizado y cuáles no examinado los bit de referencia, aunque no podremos saber el orden de utilización de las páginas. Esta información constituye la base para muchos algoritmos de sustitución de páginas que traten de aproximarse al algoritmo de sustitución LRU.

Algoritmos de Reemplazo de Páginas Segunda Oportunidad (Reloj)

El algoritmo básico de segunda oportunidad para la sustitución de páginas es un algoritmo de sustitución FIFO. Sin embargo, cuando se selecciona una página, inspeccionamos su bit de referencia. Si el valor es 0, sustituimos dicha página, pero si el bit de referencia tiene el valor 1, damos a esa página una segunda oportunidad y seleccionamos la siguiente página FIFO. Cuando una página obtiene una segunda oportunidad, su bit de referencia se borra y el campo indica que su instante de llegada se reconfigura, asignándole el instante actual. Así, si una página se utiliza de forma lo suficientemente frecuente como para que si bit de referencia permanezca activado, nunca será sustituida.

Una forma de implementar el algoritmo del reloj es utilizando una cola circular. Debemos observar que, en el peor de los casos, cuando todos los bits estén activados, el puntero recorrerá la cola completa, dando a cada una de las páginas una segunda oportunidad y borrando todos los bits de referencia antes de seleccionar la siguiente página que hay que sustituir. En este caso resulta en un algoritmo de sustitución FIFO.

También se pude mejorar este algoritmo considerando el bit de referencia y el bit de modificación. La principal diferencia radica que ahora al poder conocer si una página fue modificada o no, damos preferencia a aquellas páginas que no hayan sido modificadas, con el fin de reducir el número de operaciones de E/S requeridas.

Algoritmos de Cuenta

En este se mantiene un contador del número de referencias que se hayan hecho a cada página y puede implementarse mediante los siguientes dos esquemas:

- El algoritmo de sustitución de páginas LFU (least frequently used, menos frecuentemente utilizada) requiere sustituir la página que tenga el valor más pequeño del contador. La razón para esta selección es que las páginas más activamente utilizadas deben tener un valor grande en el contador de referencias. Sin embargo, puede surgir un problema cuando una página se utiliza con gran frecuencia durante la fase inicial de un proceso y luego ya no se la vuelve a utilizar, ya que tendrá un valor de contador grande y no será reemplazada. Una solución consiste en desplazar una posición a la derecha los contenidos del contador a intervalos regulares, obteniendo así un contador de uso medio con decrecimiento exponencial.
- El algoritmo de sustitución de páginas MFU (most frequently used, más frecuentemente utilizada) se basa en el argumento de que la página que tenga el valor de contador más pequeño acaba probablemente de ser cargada en memoria y todavía tiene que ser utilizada.

Alocación de Cuadros

Si bien esta claro que al proceso de usuario se le asignan todos los cuadros libres, la asignación de cuadros está restringida de varias maneras. No podemos, por ejemplo, asignar un número de cuadros superior al número de cuadros disponibles. Asimismo, debemos asignar al menos un número mínimo de cuadros.

Una razón para asignar al menos un número mínimo de cuadros se refiere al rendimiento. Obviamente, a medida que el número de cuadros asignados a un proceso se reduzca, se incrementará la tasa de fallos de páginas, ralentizando la ejecución del proceso. Además, debemos recordar que, cuando se produce un fallo de página antes de completar la ejecución de una instrucción, dicha instrucción debe reiniciarse. El número mínimo de cuadros esta definido por la arquitectura informática y el número máximo está definido por la cantidad de memoria física disponible.

Alocación Fija

La forma más fácil de repartir m cuadros entre n procesos consiste en dar a cada uno un número igual de cuadros, m/n. Por ejemplo, si hay 100 cuadros y 5 procesos, a cada uno se les da 20 páginas. Este sistema se denomina alocación igualitaria.

Debemos considerar que los diversos procesos necesitaran cantidades diferentes de memoria, por lo que podría suceder que en la alocación igualitaria se desperdicien cuadros. Para resolver este problema, podemos utilizar la alocación proporcional, asignando memoria a cada proceso de acuerdo con el tamaño de este.

Alocación con Prioridad

Debemos observar que, con los mecanismos de asignación igualitaria y proporcional, a los procesos de alta prioridad se los trata igual que a los de baja prioridad. Sin embargo, por su propia definición puede que convenga proporcionar al proceso de alta prioridad más memoria con el fin de acelerar su ejecución, en detrimento de los procesos de baja prioridad. Una solución consiste en utilizar un esquema de asignación proporcional en el que el cociente de cuadros dependa no de tamaño relativo de los procesos, sino más bien de las prioridades de los procesos, o bien de una combinación del tamaño de la prioridad.

Reemplazo Global vs. Local

Otro factor importante en la forma de asignar los cuadros a los diversos procesos es el mecanismo de sustitución de página. Si hay múltiples procesos compitiendo por los cuadros, podemos clasificar los algoritmos de sustitución de páginas en dos categorías amplias: sustitución global y sustitución local. La sustitución global permite a un proceso seleccionar un cuadro de sustitución de entre el conjunto de todos los cuadros, incluso si dicho cuadro está asignado actualmente a algún otro proceso; es decir, un proceso puede quitar un marco a otro. El mecanismo de sustitución local requiere, por el contrario, que cada proceso sólo efectúe esa selección entre su propio conjunto de cuadros asignado.

Thrashing

Si un proceso no tiene el número de cuadros que necesita para soportar las páginas que están siendo usadas activamente, generará rápidamente fallos de página. En ese momento, deberá sustituir alguna página; sin embargo, como todas sus páginas se están usando activamente, se verá forzado a sustituir una página que va a volver a ser necesaria enseguida. Como consecuencia, vuelve a generar rápidamente una y otra vez sucesivos fallos de página, sustituyendo páginas que se forzado a recargar inmediatamente.

El thrashing provoca graves problemas de rendimiento como se verá a continuación. Cuando el planificador de la CPU ve que la tasa de utilización de la CPU ha descendido, incrementa el grado de multiprogramación. El nuevo proceso tratará de iniciarse quitando cuadros a los procesos que se estuvieran ejecutando, haciendo que se provoquen más fallos de página y que la cola del dispositivo de paginación crezca. Como resultado, la utilización de la CPU cae todavía más y el planificador de la CPU trata de incrementar el grado multiprogramación todavía en mayor medida. Se ha producido thrashing, la tasa de procesamiento desciende vigorosamente, a la vez que se incrementa enormemente la tasa de fallos de página.

Podemos limitar los efectos del thrashing utilizando un algoritmo de sustitución local. Con la sustitución local, si uno de los procesos entra en thrashing, no puede robar cuadros de otro proceso y hacer que este también entre en thrashing. Sin embargo, esto no resuelve el problema completamente. Si los procesos están en thrashing, se pasarán la mayor

parte del tiempo en la cola del dispositivo de paginación. De este modo, el tiempo medio de servicio para un fallo de página se incrementará, debido a que ahora el tiempo medio invertido en la cola del dispositivo de paginación es mayor. Por tanto, el tiempo efectivo se incrementará incluso para los procesos que no estén en thrashing.

Método de Conjunto de Trabajo (Working-Set)

Para prevenir el thrashing, debemos proporcionar a los procesos tantos cuadros como necesiten. Para determinar este valor pueden utilizarse varias técnicas distintas. La estrategia basada en el working set comienza examinando cuántos cuadros está utilizando realmente un proceso, esta técnica define el modelo de localidad de ejecución de un proceso. En este modelo, a medida que un proceso se ejecuta, se va desplazando de una localidad a otra. Una localidad es el conjunto de páginas que se utilizan activamente de forma combinada. Todo programa está generalmente compuesto de varias localidades diferentes, que puede estar solapadas.

Suponga que asignamos a un proceso los suficientes cuadros como para acomodar su localidad actual. El proceso generará fallos de página para todas las páginas de su localidad, hasta que todas ellas estén en memoria; a partir de ahí, no volverá a generar fallos de página hasta que cambie de localidad. Si asignamos menos cuadros que el tamaño de la localidad actual, el proceso entrará en thrashing, ya que no podrá mantener en memoria todas las páginas que esté utilizando activamente.

El modelo del conjunto de trabajo utiliza un parámetro, Δ, para definir la ventana del conjunto de trabajo. La idea consiste en examinar las Δ referencias de página más recientes. El conjunto de páginas en las Δ referencias de página más recientes es el conjunto de trabajo. Si una página está siendo utilizada, será eliminada del conjunto de trabajo Δ unidades de tiempo después de la última referencia que se hiciera a la misma. Por tanto, el conjunto de trabajo es una aproximación de la localidad del programa.

La precisión del conjunto de trabajo depende de la selección de Δ. Si Δ es demasiado pequeña, no abarcara la localidad completa; si Δ es demasiado grande, puede que se solapen varias localidades. En el caso extremo, si Δ es infinita, el conjunto de trabajo será el conjunto de páginas utilizadas durante la ejecución del proceso.

La propiedad más importante del conjunto de trabajo es, entonces, su tamaño. Si calculamos el tamaño del conjunto de trabajo, WSS_i, para cada proceso del sistema, podemos considerar que

$$D = \sum WSS_i,$$

Donde D es la demanda total de marcos. Cada proceso estará utilizando activamente las páginas de su conjunto de trabajo. Así, el proceso i necesita WSS_i cuadros. Si la demanda total es superior al número total de marcos disponibles (D > m), se producirá thrashing, porque algunos procesos no dispondrán de suficientes cuadros. En tal caso, el sistema operativo seleccionará un proceso para suspenderlo.

Esta estrategia del working-set impide el thrashing al mismo tiempo que mantiene el grado de multiprogramación con el mayor valor posible. De este modo, se optimiza la tasa de utilización de la CPU.

Esquema de Frecuencia de Falta de Página

Para controlar el thrashing hay una estrategia más directa que está basada en la frecuencia de fallos de página (PFF, page-fault frequency). Podemos establecer sendos límites superior e inferior de la tasa deseada de fallos de página.

Si la tasa real de fallos de página excede el límite superior, eliminamos un cuadro del proceso. Por tanto, podemos medir y controlar directamente la tasa de fallos de página para evitar thrashing.

Si la tasa de fallos de página se incrementa y no hay ningún cuadro disponible, deberemos seleccionar algún proceso y suspenderlo. Los cuadros liberados se distribuirán entonces entre los procesos que tengan una mayor tasa de fallos de página.

La idea es establecer un ritmo aceptable de falta de páginas. Si el ritmo actual es demasiado bajo, los procesos pierden cuadros. SI el ritmo actual es demasiado alto, el proceso gana cuadros.

Otras Consideraciones – Prepaginado

Una característica obvia del mecanismo de paginación bajo demanda pura es el gran número de fallos de página que se producen en el momento de iniciar un proceso o reactivarlo. La prepaginación es un intento de evitar este alto nivel de paginación inicial. La estrategia consiste en cargar en memoria de una sola vez todas las páginas que vayan a ser necesarias.

La prepaginación puede ser muy ventajosa en algunos casos. La cuestión es simplemente, si el coste de utilizar la prepaginación es menor que el coste de dar servicio a los correspondientes fallos de página. Podría darse el caso de que muchas de las páginas que fueran cargadas en memoria por el mecanismo de prepaginación no llegaran a utilizarse.

Supongamos que se prepaginan s páginas y que una fragmentación α de esas s páginas llega a utilizarse ($0 \leq \alpha \leq 1$). La cuestión es si el coste de los s x α fallos de página que nos ahorramos es mayor o menos que el coste de prepaginar s x (1 – α) páginas innecesarias. Si α está próxima a 0 la prepaginación no será aconsejable; si α está próxima a 1, la prepaginación será la solución más adecuada.

Otras Consideraciones – Tamaño de Página

Cuando se están diseñando nuevas plataformas, es necesario tomar una decisión en lo referente a cuál es el tamaño de página más adecuado. Como cabría esperar, no hay ningún tamaño de página que sea mejor en términos absolutos. En lugar de ello, lo que hay es una serie de factores que favorecen los distintos tamaños.

Uno de los aspectos que hay que tener en cuenta es el tamaño de la tabla de páginas. Para un espacio de memoria virtual determinado, al reducir el tamaño de página se incrementa el número de páginas y, por tanto, el tamaño de la tabla de páginas. Puesto que un proceso activo debe tener su propia copia de la tabla de páginas, conviene utilizar un tamaño de página grande. Por otro lado, la memoria se aprovecha mejor si las páginas son más pequeñas. Además esto ayuda a reducir la fragmentación interna.

Otro problema es el tiempo requerido para leer o escribir una página. El tiempo de E/S está compuesto de los tiempos de búsqueda, latencia y transferencia. El tiempo de transferencia es proporcional a la cantidad de datos transferida, un hecho que parece actuar a favor de los tamaños de página pequeños. Sin embargo, los tiempos de latencia y búsqueda son mucho mayores que el tiempo de transferencia. Por tanto, el deseo de minimizar el tiempo de E/S es un argumento a favor de los tamaños de página grandes.

Por otro lado, con un tamaño de página pequeño, la cantidad total de E/S debería reducirse, ya que se mejora la localidad. Pero si consideramos un tamaño de página de 1 byte, tendríamos un fallo de página por cada byte. Dado que un fallo de página requiere gran cantidad de trabajo adicional, para minimizar el número de fallos de página debemos disponer de un tamaño de página grande.

Otras Consideraciones – Estructura del Programa

La paginación bajo demanda está diseñada para ser transparente para el programa de usuario. En muchos casos, el usuario no será consciente de la naturaleza paginada de la memoria. En otros casos, sin embargo, puede mejorarse el rendimiento del sistema si el usuario o el compilador conocen el mecanismo subyacente de paginación bajo demanda.

Veamos un ejemplo sencillo pero bastante ilustrativo.

```
Arreglo de Enteros A[1024,1024]
```

```
Cada fila está almacenada en una página

Un cuadro

Programa 1

for j:=1 to 1024 do
      for i:=1 to 1024 do
            A[i,j]=0;

1024 * 1024 faltas de páginas


Programa 2

for i:=1 to 1024 do
      for j:=1 to 1024 do
            A[i,j]=0;

1024 faltas de páginas
```

Otras Consideraciones – Fijación de E/S

Cuando se utiliza paginación bajo demanda, en ocasiones es necesario que ciertas páginas queden fijas en memoria. Una de tales situaciones es cuando se realizan operaciones de E/S hacia o desde la memoria de usuario (virtual). La E/S suele implementarse mediante un procesador de E/S independiente.

Debemos asegurarnos de que no se produzca la siguiente secuencia de sucesos: un proceso realiza una solicitud de E/S y es puesto en una cola para dicho dispositivo de E/S; mientras tanto, la CPU se asigna a otros procesos; estos procesos provocan fallos de páginas y uno de ellos, utilizando un algoritmo de sustitución global, sustituye la página que contiene el búfer de memoria del proceso en espera, por lo que dichas páginas se descargan; algún tiempo después, cuando la solicitud de E/S llegue a la parte inicial de la cola del dispositivo, la E/S se realizara utilizando la dirección especificada; sin embargo, dicho cuadro está siendo ahora utilizado para una página distinta, que pertenece a otro proceso.

Existen dos soluciones comunes a este problema: una solución consiste en no ejecutar nunca operaciones de E/S sobre la memoria de usuario. Otra solución consiste en permitir bloquear páginas en memoria. En este caso, se asocia un bit de bloqueo con cada cuadro. Si el cuadro esta bloqueado, no podrá ser seleccionado para sustitución.

Sistema de Archivos Interfaz - Módulo 10

Concepto de Archivo

El sistema operativo realiza una abstracción de las propiedades físicas de los dispositivos de almacenamiento, con el fin de definir una unidad lógica de almacenamiento, el archivo.

Un archivo es una colección de información relacionada, con un nombre, que se graba en almacenamiento secundario. Desde la perspectiva del usuario, un archivo es la unidad más pequeña de almacenamiento secundario lógico.

Comúnmente, los archivos representan programas y datos. Los archivos de datos pueden ser alfabéticos, alfanuméricos o binarios.

Estructura de Archivo

En general, un archivo es una secuencia de bits, bytes, líneas o registros cuyo significado está definido por el creador y el usuario del archivo.

Un archivo tiene una determinada estructura definida que dependerá de su tipo.

Hay archivos de estructura simple, como por ejemplo, un archivo de texto, el cuál es una secuencia de caracteres organizada en líneas (y posiblemente en páginas). Un archivo fuente es una secuencia de subrutinas y funciones, cada una de las cuales está a su vez organizada como una serie de declaraciones, seguidas de instrucciones ejecutables.

Y archivos de estructura compleja, como un archivo objeto, que es una secuencia de bytes organizada en bloques que el programa montador del sistema puede comprender.

Atributos de Archivo

Los atributos de un archivo varían de un sistema operativo a otro, pero típicamente son los siguientes:

- Nombre: mantiene información en forma legible
- Tipo: necesario para sistemas que soportan diferentes tipo
- Locación: puntero a la locación del archivo en el dispositivo
- Tamaño: tamaño corriente del archivo
- Protección: contrala quien puede leer, escribir y ejecutar el archivo
- Tiempo, fecha, e identificación de usuario: datos para protección, seguridad y visualización de uso.

La información acerca de los archivos se almacena en la estructura de directorios, que también reside en el almacenamiento secundario.

Operaciones sobre Archivos

Un archivo es un tipo de dato abstracto. Para definir un archivo apropiadamente, debemos considerar las operaciones que pueden realizarse con los archivos. El sistema operativo puede proporcionar llamadas al sistema para crear, escribir, leer, reposicionar, borrar y truncar archivos.

Además hay dos operaciones importantes open (F_i) y close (F_i). Open (F_i) busca la estructura de directorio en el disco para la entrada F_i, y mueve el contenido de la entrada a la memoria. Close (F_i) mueve el contenido de la entrada F_i en la memoria a la estructura del directorio del disco.

Archivos Abiertos

El sistema operativo mantiene una pequeña tabla, denominada tabla de archivos abiertos, que contiene información acerca de todos los archivos abiertos. Cuando se solicita que se realice una operación con un archivo, el archivo se especifica mediante un índice a esta tabla, con lo que no hace falta realizar ninguna exploración de directorio. Cuando el archivo deja de ser utilizado activamente será cerrado por el proceso y el sistema operativo eliminará la entrada correspondiente de la tabla de archivos abiertos.

Son necesarios varios datos para administrar los archivos abiertos:

- Puntero corriente del archivo: El sistema deberá registrar la ubicación correspondiente a la última lectura-escritura, utilizando para ello un puntero de posición actual dentro del archivo. Este puntero es distinto para cada proceso que esté operando con el archivo, y deberá por tanto almacenarse de forma separada de los atributos del archivo almacenados en disco.

- Cuenta de archivo abierto: El contador de aperturas del archivo controla el número de aperturas y cierres y alcanzará el valor cero cuando el último proceso cierre el archivo. Entonces, el sistema podrá eliminar la correspondiente entrada para evitarse quedar sin espacio en la tabla de archivos abiertos.
- Locación en el disco del archivo: La mayoría de las operaciones de archivo requieren que el sistema modifique datos dentro del archivo. La información necesaria para ubicar el archivo en el disco se almacena en la memoria, para que el sistema no tenga que leer de nuevo esa información desde el disco en cada operación.
- Derechos de acceso: Cada proceso abre un determinado archivo con un cierto modo de acceso. Esta información se almacena en la tabla correspondiente a cada proceso, para que el sistema operativo pueda autorizar o denegar las siguientes solicitudes de E/S.

Locking de Archivos Abiertos

Algunos sistemas operativos y sistemas de archivos proporcionan funciones para bloquear un archivo abierto (o determinadas secciones de un archivo). Los bloqueos de archivo resultan útiles para aquellos archivos que son compartidos por varios procesos, como por ejemplo un archivo de registro del sistema que puede ser consultado y modificado por varios procesos distintos del sistema.

Un bloque compartido es similar a un bloqueo lector, en el sentido de que varios procesos pueden adquirir dichos bloqueos concurrentemente. Un bloqueo exclusivo se comparta como un bloqueo escritor; sólo puede adquirir dicho tipo de bloqueo un proceso cada vez.

Además, los sistemas operativos pueden proporcionar mecanismos de bloqueo de archivos mandatorios o flexibles. Si un bloqueo es mandatorio el acceso es rechazado dependiendo de los locks que se tienen sobre el archivo y los requeridos. Alternativamente, si el bloqueo es de tipo flexible los procesos verifican el estado de los locks y deciden que hacer.

En otras palabras, si el esquema de bloqueos es mandatorio, el sistema operativo garantiza la integridad de los bloqueos, mientras que para el esquema flexible es responsabilidad de los desarrolladores de software garantizar que se adquieran y liberen apropiadamente los distintos bloqueos.

Tipos de Archivo – nombre, extensión

Si un sistema operativo reconoce el tipo de un archivo, podrá operar con ese archivo de formas razonables.

Una técnica común para implementar los tipos de archivo consiste en incluir el tipo como parte del nombre del archivo. El nombre se divide en dos partes: un nombre y una extensión, usualmente separadas por un carácter de punto. De esta forma, el usuario y el sistema operativo pueden determinar, con sólo analizar el nombre, cuál es el tipo de cada archivo.

Métodos de Accesos

Los archivos almacenan información. Cuando hace falta utilizarla, es necesario acceder a esta información y leerla en la memoria de la computadora.

El método de acceso más simple es el acceso secuencial. La información del archivo se procesa por orden, un registro después de otro. Las lecturas y escrituras constituyen el grueso de las operaciones realizadas con un archivo. Una operación de lectura (leer siguiente) lee la siguiente porción del archivo e incrementa automáticamente un puntero de archivo, que controla la ubicación de E/S. De forma similar trabaja la operación de escritura. Dichos archivos podrán reinicializarse para situar el puntero al principio de los mismos.

Otro método es el acceso directo o acceso relativo. Un archivo esta compuesto de registros lógicos de longitud fija que permite a los programas leer y escribir registros rápidamente, sin ningún orden concreto. El método de acceso directo se

basa en un modelo de archivos que se corresponde con los dispositivos de disco, ya que los discos permiten el acceso aleatorio a cualquier bloque de un archivo. Para el acceso directo, el archivo se considera como una secuencia numerada de bloques o registros. Así tendremos operaciones tales como leer n, donde n es el número de bloque, en lugar de leer siguiente, y escribir n.

El número de bloque proporcionado por el usuario al sistema operativo es normalmente un número de bloque relativo. Un número de bloque relativo es un índice referido al comienzo del archivo. Así, el primer bloque relativo del archivo será el 0, el siguiente será el 1, aunque cuando las direcciones reales absolutas pueden no ser contiguas. La utilización de números de bloque relativos permite al sistema operativo decidir donde hay que colocar el archivo y ayuda a impedir que el usuario acceda a porciones del sistema de archivos que no formen parte de su archivo.

Ejemplo de Archivo Indexado y Relativo

Pueden construirse otros métodos por encima del método de acceso directo. Estos métodos implican, generalmente, la construcción de un índice para el archivo. El índice, contiene punteros a los distintos bloques. Para encontrar un registro dentro del archivo, primero exploramos el índice y luego usamos el puntero para acceder al archivo directamente y para hallar el registro deseado.

Con los archivos de gran tamaño, el propio archivo índice puede ser demasiado grande como para almacenarlo en memoria. Una solución consiste en crear un índice del archivo índice. El archivo de índice principal contendría punteros a los archivos de índice secundarios que a vez apuntarían a los propios elementos de datos.

Una Organización Típica de un sistema de Archivos

Un disco puede utilizarse completamente para un sistema de archivos. Sin embargo, en ciertas ocasiones es deseable colocar múltiples sistemas de archivos en un mismo disco o utilizar partes de un disco para un sistema de archivos y otras partes para otras cosas, como por ejemplo para espacio de intercambio o como espacio de disco sin formato (raw). Estas partes se conocen como particiones. Las partes también pueden combinarse para formar estructuras de mayor tamaño, conocidas con el nombre de volúmenes.

Cada volumen que contenga un sistema de archivos debe también contener información acerca de los archivos almacenados en el sistema. Esta información se almacena como entrada en una tabla de contenidos del volumen. Esta tabla también conocida como directorio almacena información de todos los archivos de dicho volumen, como por ejemplo el nombre, el tipo, la dirección, la longitud corriente, la máxima longitud, la fecha del último acceso, fecha de la última actualización, el ID del dueño y la información de protección.

La estructura de directorio al igual que los archivos reside en el disco.

Operaciones sobre un Directorio

El directorio puede considerarse como una tabla de símbolos que traduce los nombres de archivo a sus correspondientes entradas de directorio. Si adoptamos esta visión, es fácil comprender que el propio directorio puede organizarse de muchas formas.

Cuando consideramos una estructura de directorio concreta, tendremos que tener presentes las operaciones que habrá que realizar con el directorio:

- Búsqueda de un archivo
- Creación de un archivo
- Borrado de un archivo

- Listado de un directorio
- Renombrado de un archivo
- Atravesar un sistema de archivos

Organice el Directorio (Lógicamente) para Obtener

Con la organización lógica de los directorios se busca obtener eficiencia que permita localizar un archivo rápidamente. Además también se busca permitir que dos usuarios puedan tener el mismo nombre para diferentes archivos o un mismo archivo puede tener varios nombres diferentes. Por último, estas organizaciones también buscan favorecer el agrupamiento lógico de los archivos por propiedades, por ejemplo, todos los programas de Pascal juntos, todos los juegos juntos, etc.

Estructura Arbórea de Directorios

Esta estructura permite a los usuarios crear sus propios subdirectorios y organizar sus archivos correspondientes. Cada árbol tiene un directorio raíz y todos los archivos del sistema tienen un nombre de ruta distinto.

Cada directorio (o subdirectorio) contiene un conjunto de archivos y subdirectorios. Un directorio es simplemente otro archivo, pero que se trata de forma especial. Todos los directorios tienen el mismo formato interno. Un bit de cada entrada de directorio define si esa entrada es un archivo(0) o un subdirectorio(1). Se utilizan llamadas especiales para crear y borrar los directorios.

En el uso normal, cada proceso tiene un directorio actual. Ese directorio actual debe contener la mayor parte de los archivos que actualmente interesen al proceso. Cuando se hace referencia a un archivo que no se encuentre en el directorio actual, entonces el usuario deberá normalmente especificar un nombre de ruta o cambiar el directorio actual, para situarse en el directorio donde fue almacenado ese archivo. Para cambiar de directorio, se proporciona una llamada al sistema que toma como parámetro el nombre del directorio y lo utiliza para redefinir el directorio actual. Los nombres de ruta pueden ser de dos tipos: absolutos y relativos. Un nombre de ruta absoluto comienza en la raíz y sigue una ruta descendiente hasta el archivo especificado, indicando los nombres de los directorios que componen la ruta. Un nombre de ruta relativo define una ruta a partir del directorio actual.

Una decisión interesante de política dentro de un directorio con estructura de árbol es la que se refiere a qué hacer si se borra un directorio. Si el directorio esta vacío, podemos simplemente borrar la entrada correspondiente dentro del mismo directorio que lo contuviera. Sin embargo, supongamos que el directorio que hay que borrar no esta vacío, sino que contiene varios archivos o subdirectorios. Podemos adoptar una de dos soluciones. Algunos sistemas, como MS-DOS, no permiten borrar un directorio a menos que este vacío, esto provoca que el usuario deba realizar una gran cantidad de trabajo. Otra técnica alternativa, como la adoptada por el comando rm de UNIX, consiste en proporcionar una opción: cuando se hace una solicitud para borrar un directorio, también se borran todos los archivos y subdirectorios de dicho directorio. Esta ultima opción si bien es cómoda, es riesgosa ya que con un solo comando se puede destruir una estructura de directorios completa.

Grafo Acíclico de Directorios

Una estructura de árbol prohíbe la compartición de archivos o directorios. Por el contrario, un grafo acíclico permite que los directorios compartan subdirectorios de archivos. El mismo archivo o subdirectorio puede estar en dos directorios diferentes. El grafo acíclico es una generalización natural del esquema de directorio con estructura de árbol.

Una estructura de directorio en forma de grafo acíclico es más flexible que una estructura simple de árbol, pero también más compleja. Es necesario prestar cuidadosa atención a determinados problemas. Los archivos podrán tener ahora

múltiples nombres de ruta absoluta. En consecuencia, puede haber nombres de archivo distintos que hagan referencia a un mismo archivo. La situación es similar al problema de los alias en los lenguajes de programación. Si estamos intentando recorrer el sistema de archivos completo, por ejemplo, para acumular estadísticas, para encontrar un archivo o para copiar todos los archivos en una copia de seguridad; este problema cobra gran importancia, ya que no conviene recorrer las estructuras compartidas más de una vez.

Otro problema es el que se refiere al borrado, ¿cuándo puede desasignarse y reutilizarse el espacio asignado a un archivo compartido? Una posibilidad consiste en eliminar el archivo cuando un usuario cualquiera lo borre, pero esta acción puede dejar punteros colgantes al archivo que ha dejado de existir. El problema puede complicarse si los punteros de archivo restantes contienen direcciones reales de disco y ese espacio se reutiliza subsiguientemente para otros archivos; en ese caso, esos punteros colgantes pueden apuntar a un lugar cualquiera de estos nuevos archivos.

En un sistema en el que la compartición se implemente mediante enlaces simbólicos, la situación es algo más fácil de manejar. El borrado de un enlace no tiene porque afectar al archivo original, ya que sólo se elimina el enlace. Si lo que se elimina es la propia entrada del archivo, se designará el espacio del archivo, dejando que los enlaces cuelguen. Podemos buscar estos enlaces y eliminarlos también, pero a menos que se mantenga con cada archivo una lista de los enlaces asociados esta búsqueda puede ser bastante costosa. Alternativamente, podemos dejar esos enlaces hasta que se produzca un intento de utilizarlos, en cuyo momento podemos determinar que el archivo del nombre indicado por el enlace no existe y que no podemos resolver el nombre del enlace. Otra técnica de borrado consiste en preservar el archivo hasta que se borren todas las referencias al mismo. Para implementar esta técnica, debemos disponer de algún mecanismo para determinar que se ha borrado la última referencia al archivo. Podríamos mantener un contador de entradas al archivo.

Grafo General de Directorio

Uno de los problemas más graves que afectan a la utilización de una estructura en forma de grafo acíclico consiste en garantizar que no exista ningún ciclo. Si comenzamos con un directorio en dos niveles y permitimos a los usuarios crear subdirectorios, obtendremos un directorio con estructura de árbol. Es fácil darse cuenta de que la simple adición de nuevos archivos y subdirectorios a una estructura de árbol preserva la naturaleza y la estructura de árbol de ese directorio.

Sin embargo, si añadimos enlaces a un directorio existente con estructura de árbol, la estructura de árbol se destruye, dando como resultado una estructura de grafo simple.

Necesitamos poder evitar recorrer las secciones compartidas de un grafo cíclico dos veces, principalmente por razones de rendimiento. Inclusive, un algoritmo mal diseñado podría provocar un bucle infinito que estuviera explorando el ciclo continuamente, sin terminar nunca.

Un problema similar existe cuando tratamos de determinar si un archivo puede ser borrado. Con las estructuras de directorio en grafo acíclico, un valor de 0 en el contador de referencia significaría que ya no hay más referencias al archivo o directorio, y que ese archivo puede ser borrado. Sin embargo, si existen ciclos, el contador de referencias puede no ser 0 incluso aunque ya no sea posible hacer referencia a un directorio de archivo. Esta anomalía resulta de la posibilidad de auto-referencia en la estructura de directorio. Una solución consiste en permitir enlaces a archivos y no a subdirectorios. Otra solución se basaría en utilizar un esquema de recolección de memoria para determinar cuándo se ha borrado la última referencia y que, en consecuencia, puede reasignarse el espacio de disco.

La principal dificultad es la de evitar que aparezcan ciclos a medida que se añaden nuevos enlaces a la estructura. ¿Cómo podemos saber si un nuevo enlace va a completar el ciclo? Existen algoritmos para terminar la existencia de ciclos en los grafos; sin embargo, estos algoritmos son muy costosos desde el punto de vista computacional, especialmente cuando

el grafo se encuentra almacenado en disco. Un algoritmo más simple en el caso especial de directorios y enlaces consiste en ignorar los enlaces durante el recorrido de directorios. De este modo, se evitan los ciclos sin necesidad de efectuar ningún procesamiento adicional.

Montaje de Sistema de Archivos (Mounting)

Un sistema de archivos debe montarse para poder estar disponible para los procesos del sistema.

El proceso de montaje es bastante simple. Al sistema operativo se le proporciona el nombre del dispositivo y el punto de montaje que es la ubicación dentro de la estructura de archivos a la que hay que conectar el sistema de archivos que está montando. Normalmente, el punto de montaje será un directorio vacío.

A continuación el sistema operativo verifica que el dispositivo contiene un sistema de archivos válido. Para ello, pide al controlador de dispositivo que lea el directorio de dispositivo y verifique que ese directorio tiene el formato esperado. Finalmente, el sistema operativo registra en su estructura de directorios que hay un sistema de archivo montado en el punto de montaje especificado.

Archivos Compartidos

Cuando un sistema operativo tiene múltiples usuarios, las cuestiones relativas a la compartición de archivos, a la denominación de archivos y a la protección de archivos cobran una gran importancia. Dada una estructura de directorio que permite que los usuarios compartan archivos, el sistema debe adoptar un papel de mediador en lo que a la compartición de archivos respecta. El sistema puede permitir que un usuario acceda a los archivos de otros usuarios de manera predeterminada, o puede por el contrario requerir que los usuarios concedan explícitamente el acceso a sus archivos.

Para implementar la compartición y los mecanismos de protección, el sistema debe mantener más atributos de los archivos y de los directorios. La mayoría de los sistemas han optado por usar el concepto de propietario de un archivo o directorio y el concepto de grupo. El propietario es el usuario que puede cambiar los atributos y conceder el acceso y que dispone del máximo grado de control sobre el archivo. El atributo de grupo define un subconjunto de usuarios que pueden compartir el acceso al archivo.

Cuando un usuario solicita realizar una operación sobre un archivo, se puede comparar el ID del usuario con el atributo de propietario para determinar si el usuario solicitante es el propietario del archivo. De la misma manera, pueden compararse los identificadores de grupo. El resultado indicara que permisos son aplicables. A continuación, el sistema aplicará dichos permisos a la operación solicitada y la autorizara o negara.

En sistemas distribuidos los archivos pueden ser compartidos a través de la red.

Archivos Compartidos – Semántica de Consistencia

La semántica de consistencia especifica cómo pueden acceder simultáneamente a un archivo múltiples usuarios del sistema. En particular, especifica cuando las modificaciones que un usuario realice en los datos serán observados por parte de los otros usuarios. La semántica consistencia está directamente relacionada con los algoritmos de sincronización de procesos.

Por otro lado el sistema UNIX utiliza la siguiente semántica de coherencia:

- Las escrituras en un archivo abierto por parte de un usuario son visibles inmediatamente para los otros usuarios que hayan abierto ese archivo.

- Un modo de compartición permite a los usuarios compartir el puntero de ubicación actual dentro del archivo. Así, el incremento de ese puntero por parte del usuario afectara a todos los usuarios que estén compartiendo el archivo.

En la semántica UNIX, cada archivo está asociado con una única imagen física a la que se accede en forma de recurso exclusivo.

En el sistema de archivos Andrew se implemento una semántica muy compleja para compartir archivos que consta de lo siguiente:

- Las escrituras en un archivo abierto por parte de un usuario no son visibles inmediatamente para los restantes usuarios que hayan abierto el mismo archivo.
- Una vez que se cierra un archivo, los cambios realizados en él son visibles únicamente en las sesiones que den comienzo posteriormente. Las instancias ya abiertas del archivo no reflejarán dichos cambios.

De acuerdo con esta semántica, un archivo puede estar temporalmente asociado con varias imágenes al mismo tiempo. En consecuencia, se permite que múltiples usuarios realicen accesos concurrentes tanto de lectura como de escritura en sus propias imágenes del archivo, sin ningún retardo.

Protección

El creador/dueño del archivo debería poder controlar que acciones pueden realizarse con el archivo y quienes las pueden realizar. Se habla entonces de acceso controlado.

Los mecanismos de protección proporcionan este acceso controlado limitando los tipos de accesos a archivo que puedan realizarse. Podemos controlar varios tipos de operaciones diferentes entre ellas lectura, escritura, ejecución, adición, borrado y listado.

Listas de Acceso y Grupos

La técnica más común para resolver el problema de la protección consiste en hacer que el acceso dependa de la identidad del usuario. Los diferentes usuarios pueden necesitar diferentes tipos de acceso a un archivo o directorio. El esquema más general para implementar un acceso dependiente de la identidad consiste en asociar con cada archivo y directorio una lista de control de acceso (ACL, access-control list) que especifique los nombres del usuario y los tipos de acceso que se permiten para cada uno. Cuando un usuario solicita acceder a un archivo concreto, el sistema operativo comprueba la lista de acceso asociada con dicho archivo; si dicho usuario está incluido en la lista para el tipo de acceso solicitado, se permite el acceso. En caso contrario, se producirá una violación de la protección y se denegará al trabajo de usuario el acceso al archivo.

Esta técnica tiene la ventaja de permitir la implementación de complejas metodologías de acceso. El problema principal con las listas de acceso es su longitud. Si queremos permitir que todo el mundo lea un archivo, deberemos enumerar a todos los usuarios que disponen de ese acceso de lectura. La cual es una tarea tediosa y requiere que la entrada de directorio tenga tamaño variable ya que no sabemos de antemano la lista de usuarios del sistema.

Estos problemas pueden resolverse utilizando una versión condensada de la lista de acceso. Para condensar la longitud de la lista de control de acceso, muchos sistemas clasifican a los usuarios en tres grupos, en lo que respecta con cada archivo:

- Propietario: el usuario que creo el archivo será su propietario

- Grupo: Un conjunto de usuarios que están compartiendo el archivo y necesitan un acceso similar al mismo es un grupo, o grupo de trabajo.

- Universo. Todos los demás usuarios del sistema constituyen el universo.

La técnica reciente más común consiste en combinar las listas de control de acceso con el esquema más general de control de acceso para lograr una granularidad más fina.

Para que este esquema funcione adecuadamente, deben controlarse estrechamente tanto los permisos como las listas de control de acceso. Este control puede llevarse acabo de varias formas distintas. Por ejemplo, en el sistema UNIX, los grupos sólo pueden ser creados y modificados por el administrador de la instalación o cualquier superusuario.

Sistemas de Archivos Implementación - Módulo 11

Estructura del Sistema de Archivos

Los discos constituyen el principal tipo de almacenamiento secundario para mantener sistemas de archivos.

En lugar de transferir un byte cada vez, las transferencias de E/S entre la memoria y el disco se realizan en unidades de bloques, para mejorar la eficiencia de E/S. Cada bloque tiene uno o más sectores.

Para proporcionar un acceso eficiente y cómodo al disco, el sistema operativo impone uno o más sistemas de archivos, con los que los datos pueden almacenarse, localizarse y extenderse fácilmente. Un sistema de archivos acarrea dos problemas de diseño bastante diferentes. El primer problema es definir que aspecto debe tener el sistema de archivos para el usuario. Esta tarea implica definir un archivo y sus atributos, las operaciones definidas sobre los archivos y la estructura de directorio utilizada para organizar los archivos. El segundo problema es crear algoritmos y estructuras de datos que permitan mapear el sistema lógico de archivos sobre los dispositivos físicos de almacenamiento secundario.

El sistema de archivos esta organizado en capas. Cada nivel de diseño utiliza las funciones de los niveles inferiores para crear nuevas funciones que serán, a su vez, utilizadas por los niveles superiores a este.

El nivel más bajo, el control de E/S, esta compuesto por controladores de dispositivo y rutinas de tratamiento de interrupción, que se encargan de transferir la información entre la memoria principal y el sistema de disco.

El sistema básico de archivos sólo necesita enviar comandos genéricos al controlador de dispositivo apropiado, con el fin de leer y escribir bloques físicos en el disco. Cada bloque físico se identifica mediante su dirección de disco numérica.

El módulo de organización de archivos tiene conocimiento acerca de los archivos y de sus bloques lógicos, así como de sus bloques físicos. Considerando el tipo de asignación de archivos utilizado y la ubicación del archivo, el módulo de organización de archivos puede traducir las direcciones lógicas de bloque a direcciones físicas de bloque, que serán las que envíe al sistema básico de archivos para que realice las necesarias transferencias. El módulo de organización de archivos incluye también el gestor de espacio libre, que controla los bloques no asignados.

Finalmente, el sistema lógico de archivos gestiona la información de metadatos. Los metadatos incluyen toda la estructura del sistema de archivos, excepto los propios datos (es decir, el propio contenido del archivo). El sistema lógico de archivos gestiona la estructura de directorio para proporcionar al módulo de organización de archivos la información que este necesita, a partir de un nombre de archivo simbólico. Este nivel mantiene la estructura de los archivos mediante bloques de control de archivo. Un bloque de control de archivo (FCB, file-control block) contiene información acerca del archivo, incluyendo su propietario, los permisos y la ubicación del contenido del archivo. El sistema lógico también es el responsable de las tareas de protección y seguridad.

Estructuras de Archivo

Se utilizan varias estructuras en disco y en memoria para implementar un sistema de archivos.

Un bloque de control de arranque puede contener la información que se necesita para iniciar un sistema operativo a partir de dicho volumen. Si dicho disco no contiene un sistema operativo, este bloque puede estar vacío.

Un bloque de control de volumen contiene detalles acerca del volumen, tales como el número de bloques que hay en la partición, el tamaño de los bloques, el número de bloques libres y los punteros de bloques libres, así como un contador de FCB libres y punteros a FCB.

Se utiliza una estructura de directorios por cada sistema de archivos para organizar los archivos.

Estructura del Sistema de Archivos en Memoria

La información almacenada en memoria se utiliza tanto para la gestión de un sistema de archivos como para la mejora del rendimiento mediante mecanismos de caché.

Una tabla de montaje en memoria contiene información acerca de cada volumen montado. Una caché de la estructura de directorios en memoria almacena la información relativa a los directorios que se han accedido recientemente.

La tabla global de archivos abiertos contiene una copia del FCB de cada archivo abierto. La tabla de archivos abiertos de cada proceso contiene un puntero a la entrada apropiada de la entrada global de archivos abiertos.

Para crear un nuevo archivo, un programa de aplicación llama al sistema lógico de archivos. El sistema lógico de archivos asigna un nuevo FCB. El sistema lee entonces el directorio apropiado en la memoria, lo actualiza con el nombre del nuevo archivo y el nuevo FCB y lo vuele a escribir en el disco. Una vez creado el archivo, ya podemos utilizarlo para E/S. Pero primero, sin embargo, debemos abrirlo. La llamada open() pasa un nombre de archivo al sistema de archivos. Esta llamada primero busca en la tabla global de archivos abiertos para ver si el archivo está siendo ya utilizado por otro proceso. En caso afirmativo, se crea una entrada en la tabla de archivos abiertos del proceso que apunte a la tabla global de archivos abiertos existente. Cuando el archivo está abierto, se busca en la estructura de directorios para encontrar el nombre del archivo indicado. Una vez encontrado el archivo, el FCB se copia en la tabla global de archivos abiertos existente en la memoria. Esta tabla no sólo almacena el FCB, sino que también controla el número de procesos que han abierto el archivo. A continuación, se crea una entrada en la tabla de archivos abiertos del proceso, con un puntero a la entrada de la tabla global de archivos abiertos y algunos otros campos de información. La llamada a open() devuelve un puntero a la entrada apropiada de la tabla de archivos abiertos del proceso; todas las operaciones de archivo se realizan a partir de ahí mediante este puntero. El nombre del archivo que se proporciona a la caché para futuras operaciones de apertura se conoce como descriptor de archivo.

Cuando un proceso cierra el archivo, se elimina la entrada de la tabla del proceso y se decrementa el contador de aperturas existente en al tabla global. Cuando todos los usuarios que hayan abierto el archivo lo cierren, los metadatos actualizados se copian en la estructura de directorio residente en el disco y se elimina la entrada de la tabla global de archivos abiertos.

Implementación de Directorio

El método más simple para implementar un directorio consiste en utilizar una lista lineal de nombres de archivos, con punteros a los bloques de datos. Este método es simple de programar, pero requiere mucho tiempo de ejecución ya que para localizar un archivo, se requiere realizar una búsqueda lineal.

Otro tipo de estructura de datos utilizado para los directorios son las tablas hash. Con este método, se almacenan las entradas de directorio en una lista lineal, pero también se utiliza una estructura de datos hash. La tabla toma un valor

calculado a partir del nombre del archivo y devuelve un puntero a la ubicación de dicho nombre de archivo dentro de la lista lineal. Por tanto, puede reducir enormemente el tiempo de búsqueda en el directorio. La inserción y el borrado son también bastante sencillas, aunque es necesario tener en cuenta la posible aparición de colisiones, que son aquellas situaciones en las que dos nombres de archivo proporcionan, al aplicar la función hash, la misma ubicación dentro de la lista.

Las principales dificultades asociadas con las tablas hash son que su tamaño es, por regla general, fijo y que la función hash depende de dicho tamaño.

Métodos de Alocación

Casi siempre existirán múltiples archivos almacenados en un mismo disco. El principal problema es cómo asignar el espacio a esos archivos de forma rápida. Hay tres métodos principales de asignación del espacio de disco que se utilizan de manera común: alocación contigua, enlazada e indexada.

Alocación Contigua

La alocación contigua requiere que cada archivo ocupe un conjunto de bloques contiguos en el disco. Las direcciones de disco definen una ordenación lineal del disco.

La asignación contigua de un archivo está definida por la asignación de disco del primer bloque y por la longitud del archivo, en unidades de bloques. La entrada de directorio de cada archivo indicara la dirección del bloque de inicio y la longitud de área asignada al archivo.

Acceder a un archivo que haya sido asignado de manera contigua resulta sencillo. Para el acceso secuencial, el sistema de archivos recuerda la dirección de disco del último bloque al que se haya hecho referencia y leerá el siguiente bloque cuando sea necesario. Para el acceso directo al bloque i de un archivo que comience en el bloque b, podemos acceder inmediatamente al bloque b+i. Así, la asignación contigua permite el acceso aleatorio.

Sin embargo, la asignación contigua también tiene sus problemas. Una de las dificultades estriba en encontrar espacio para un nuevo archivo. El problema de la asignación contigua puede verse como un caso concreto del problema general de asignación dinámica del almacenamiento y que se refiere a como satisfacer una solicitud de tamaño n a partir de una lista de huecos libres. Las estrategias más comunes para seleccionar un hueco libre a partir de un conjunto de huecos disponibles son la de primer ajuste y mejor ajuste aunque sufren de fragmentación externa. Dependiendo de la cantidad total del espacio de almacenamiento en disco y del tamaño medio de los archivos, la fragmentación externa puede ser un problema grave o demasiado importante.

Otro problema de los mecanismos de asignación contigua es determinar cuánto espacio se necesita para un archivo. En el momento de crear el archivo, será necesario determinar la cantidad total de espacio que vamos a necesitar y asignar esa cantidad total. Si asignamos demasiado poco espacio a un archivo, podemos encontrarnos con que no puede ampliarse. Especialmente con las estrategias de asignación de mejor ajuste, puede que el espacio situado a ambos lados del archivo ya esté siendo utilizado, por tanto, no podemos ampliar el tamaño del archivo sin desplazarlo a otro lugar. Entonces, existen dos posibilidades. La primera es terminar el programa de usuario emitiendo un error apropiado o localizar un hueco de mayor tamaño, copiar el contenido del archivo al nuevo espacio y liberar el anterior.

Alocación Enlazada

El mecanismo de asignación enlazada resuelve todos los problemas de asignación contigua.

Con la asignación enlazada, cada archivo es una lista enlazada de bloques de disco, pudiendo estar dichos bloques dispersos por todo el disco. El directorio contiene un puntero al primer y al último bloque de cada archivo. Para leer un archivo, simplemente leemos los bloques siguiendo los punteros de un bloque a otro.

No hay ninguna fragmentación externa con asignación enlazada y puede utilizarse cualquier de los bloques de la lista de bloques libres para satisfacer una solicitud. No es necesario declarar el tamaño del archivo en el momento de crearlo y el archivo puede continuar creciendo mientras existan bloques libres. En consecuencia, nunca es necesario compactar el espacio en disco.

Sin embargo, este mecanismo también tiene sus desventajas. El principal problema es que sólo se lo puede utilizar de manera efectiva para los archivos de acceso secuencial. Para encontrar el bloque i-esimo de un archivo, podemos comenzar al principio de dicho archivo y seguir los punteros hasta llegar al bloque i-esimo. Cada acceso a un puntero requiere una lectura de disco y algunos de ellos requerirán un reposicionamiento de cabezales. En consecuencia, resulta muy poco eficiente tratar de soportar una funcionalidad de acceso directo para los archivos de alocación enlazada.

Otra desventaja es el espacio requerido para los punteros. Otro problema de la asignación enlazada es la fiabilidad ya que los punteros se pueden dañar arruinando la estructura.

Una variante importante del mecanismo de asignación enlazada es la que se basa en el uso de una tabla de asignación de archivos (FAT). Este método simple, pero eficiente, de asignación del espacio del disco se utiliza en los sistemas operativos MS-DOS y OS/2.

Asignación indexada

Bajo el método de alocación enlazada, sino se utiliza una FAT, no se puede soportar el acceso directo de manera eficiente. El mecanismo de alocación indexada resuelve este problema agrupando todos los punteros en una única ubicación: el bloque de índice.

Cada archivo tiene su propio bloque de índice, que es una matriz de direcciones de bloques de disco. La entrada i-esima del bloque de índice apunta al bloque i-esimo del archivo. El directorio contiene la información del bloque de índice. Para localizar y leer el bloque i-esimo, utilizamos el puntero contenido en la entrada i-esima del bloque de índice.

El mecanismo de asignación indexada soporta el acceso directo, sin sufrir el problema de la fragmentación externa, ya que puede utilizarse cualquier bloque de disco para satisfacer una solicitud de espacio.

Sin embargo, el mecanismo de asignación indexada sí que sufre el problema de desperdicio de espacio. El espacio adicional requerido para almacenar los punteros del bloque de índice es, generalmente, mayor que el que se requiere en el caso de la asignación enlazada. Este punto nos plantea la cuestión de cuál debe ser el tamaño del bloque de índice. Cada archivo debe tener un bloque de índice, así que ese bloque deberá ser lo más pequeño posible. Sin embargo, si el bloque de índice es demasiado pequeño, no podrá almacenar suficientes punteros para un archivo de gran tamaño y será necesario implementar un mecanismo para resolver este problema. Entre los mecanismos que pueden utilizarse para ello se encuentran los siguientes:

- Esquema enlazado: Cada bloque de índice ocupa normalmente un bloque de disco. Por tanto, puede leerse y escribirse directamente. Para que puedan existir archivos de gran tamaño, podemos enlazar varios bloques de índice.

- Índice multinivel: Una variante de la representación enlazada consiste en utilizar un bloque de índice de primer nivel para apuntar a un conjunto de bloques de índice de segundo nivel, que a su vez apuntaran a los bloques del archivo. Para acceder a un bloque, el sistema operativo utiliza el índice de primer nivel para encontrar un bloque de índice de segundo nivel y luego emplea dicho bloque para hallar el bloque de datos deseado. Esta técnica podría ampliarse hasta un tercer o cuarto nivel, dependiendo del tamaño de archivo que se desee.

- Esquema combinado: Otra alternativa, consiste en mantener, por ejemplo, los primeros 15 punteros del bloque de índice en el nodo del archivo. Los primeros 12 de estos punteros hacen referencia a bloques directos, es decir, contienen la dirección de una serie de bloques que almacenan datos del archivo. De este modo, los datos para los archivos de pequeño tamaño no necesitan un bloque de índice separado. Los siguientes tres punteros hacen referencia a bloques indirectos. El primero apunta a un bloque indirecto de un nivel, que es un bloque de índice que no contiene datos sino las direcciones de otros bloques que almacenan los datos. El segundo puntero hace referencia a un bloque doblemente indirecto, que contiene la dirección de un bloque que almacena las direcciones de otras series de bloques que contienen punteros a los propios bloques de datos. El último puntero contiene la dirección de un bloque triplemente indirecto.

Los esquemas de asignación indexados sufren de los mismos problemas de rendimiento que el mecanismo de asignación enlazada. Específicamente, los bloques de índice pueden almacenarse en memoria caché, pero los bloques de datos pueden estar distribuidos por todo un volumen.

Administración de Espacio Libre

Puesto que el espacio de disco está limitado, necesitamos reutilizar el espacio de los archivos borrados para los nuevos archivos, siempre que sea posible. Para controlar el espacio libre del disco, el sistema mantiene una lista de espacio libre. La lista de espacio libre indica todos los bloques de disco libres, aquellos que no están asignados a ningún archivo o directorio. Para crear un archivo, exploramos la lista de espacio libre hasta obtener la cantidad de espacio requerida y asignamos ese espacio al nuevo archivo. A continuación, este espacio se elimina de la lista de espacio libre. Cuando se borra un archivo, su espacio de disco se añade a la lista de espacio libre.

Recientemente, la lista de espacio libre se implementa como un mapa de bits o vector de bits. Cada bloque está representado por 1 bit. Si el bloque esta libre, el bit será igual a 1; si el bloque está asignado, el bit será 0. La principal ventaja de este enfoque es su relativa simplicidad y la eficiencia que permite a la hora de localizar el primer bloque libre o n bloques libres consecutivos en el disco. Una técnica para localizar el primer bloque libre en un sistema que utilice un lector de bits para asignar el espacio de disco consiste en comprobar secuencialmente cada palabra del mapa de bits para ver si dicho valor es distinto de 0, ya que una palabra con valor 0 tendrá todos los bits iguales a 0 y representará un conjunto de bloques asignado. Cuando se encuentra una palabra distinta de 0 se explora en búsqueda del primero bit 1, que corresponderá con la ubicación del primer bloque libre. El cálculo del número de bloque será:

(número de bits por palabra) x (número de palabras de valor 0) + offset del primer bit 1

Desafortunadamente, los vectores de bits son ineficientes a menos que se mantenga el vector completo en la memoria principal. Mantener ese vector en la memoria principal es posible para discos de pequeño tamaño, pero no necesariamente para los discos de tamaño grande.

Otra técnica para la gestión del espacio libre consiste en enlazar todos los bloques de disco libres, manteniendo un puntero al primer bloque libre en ubicación especial del disco y almacenándolo en memoria cache. A pesar de que no malgasta espacio, este esquema es poco eficiente; para recorrer la lista, debemos leer cada bloque, lo que requiere un tiempo sustancial de E/S.

Una modificación de la técnica basada en la lista de espacio libre consiste en almacenar las direcciones de n bloques libres en el primer bloque libre. Los primeros n-1 de estos bloques estarán realmente libres. De este modo, podrán encontrarse rápidamente las direcciones de un gran número de bloques libres, a diferencia del caso en que se utilice la técnica estándar de lista enlazada.

Otra técnica consiste en aprovechar el hecho de que, generalmente, puede asignarse o liberarse simultáneamente varios bloques contiguos, particularmente cuando se asigna el espacio mediante el algoritmo de asignación contigua o mediante un mecanismo de agrupación cluster. Así, en lugar de mantener la lista de n direcciones de bloques de disco libres, podemos mantener la dirección del primer bloque libre y el número n de bloques libres contiguos que siguen a ese primer bloque. Cada entrada en la lista de espacio libre estará entonces compuesta por una dirección de disco y un contador. Esto provoca que la lista global sea más corta, siempre y cuando el valor del contador sea mayor que 1.

Eficiencia y Desempeño

El uso eficiente del espacio de disco depende en gran medida de los algoritmos de asignación de disco y de directorio que se utilicen. También debemos considerar los tipos de datos que normalmente se almacenan en la entrada de directorio de un archivo. El resultado de mantener esta información es que, cada vez que se realiza una operación con el archivo, puede ser necesario escribir en uno de los campos de la estructura de directorio.

Incluso después de haber seleccionado los algoritmos básicos del sistema de archivos, podemos mejorar las prestaciones de diversas maneras. Algunos sistemas mantienen una sección separada de memoria principal, reservándola para caché de búfer en la que se almacenan los bloques bajo la suposición de que en breve volverán a ser utilizados de nuevo. Otros sistemas almacenan en caché los datos de archivos utilizando una caché de páginas. La cache de páginas utiliza técnicas de memoria virtual para almacenar en cache los datos de los archivos en forma de páginas, en lugar de cómo bloques de sistema de archivos, lo cual resulta mucho más eficiente.

Algunos sistemas optimizan su cache de páginas utilizando diferentes algoritmos de sustitución, dependiendo del tipo de acceso del archivo. Para los archivos que se lean o escriban secuencialmente, sus páginas no se sustituirán en orden LRU, porque las páginas más recientemente utilizadas serán las que se utilicen más tarde, o quizá no lleguen a utilizarse nunca. En lugar de ello, el acceso secuencial puede optimizarse mediante ciertas técnicas conocidas como free-behind y read-ahead. Free-behind elimina una página del búfer en cuanto se solicita la página siguiente; las páginas anteriores no volverán, probablemente, a utilizarse de nuevo, así que representan un desperdicio en el búfer. Con el mecanismo de read-ahead, cuando se solicita una página, se leen y almacenan en caché tanto esa página como varias páginas sucesivas. Dichas páginas es bastante probable que se soliciten después de procesar la página actual.

Recuperación

Los archivos y directorios se mantienen tanto en memoria principal como en disco, y debe tenerse cuidado para que los fallos del sistema no provoquen una pérdida de datos o una incoherencia en los mismos.

El comprobador de coherencia, compara los datos de la estructura de directorios con los bloques de datos del disco y trata de corregir todas las incoherencias que detecte. Los algoritmos de asignación y gestión del espacio libre dictan los tipos de problemas que el comprobador puede tratar de detectar y dictan también el grado de éxito que el comprobador puede tener al realizar esa tarea.

Los discos magnéticos fallan en ocasiones y es necesario tener cuidado para garantizar que los datos perdidos debido a esos fallos no se pierdan para siempre. Con este fin, pueden utilizarse programas del sistema para realizar una copia de seguridad de los datos del disco en otro dispositivo de almacenamiento. La recuperación de la pérdida de un archivo individual o de un disco completo puede ser entonces, simplemente, una cuestión de restaurar los datos a partir de la copia de seguridad.

Sistema de Archivos Almacenaje Secundario - Módulo 12

Revisión

Los discos magnéticos proporcionan la parte principal del almacenamiento secundario en los modernos sistemas informáticos. Desde el punto de vista conceptual, los discos son relativamente simples. Cada plato tiene una forma circular plana, como un CD. Las dos superficies de cada plato están recubiertas de un material magnético. La información se almacena grabándola magnéticamente sobre los platos.

Un cabezal de lectura-escritura vuela justo por encima de cada una de las superficies de cada plato. Los cabezales están conectados a un brazo del disco que mueve todos los cabezales como una sola unidad. La superficie de de cada plato está dividida desde el punto de vista lógico en pistas circulares, que a su vez se subdividen en sectores. El conjunto de pistas que están situadas en una determinada posición del brazo forman un cilindro. Puede haber miles de cilindros concéntricos en una unidad de disco y cada pista puede contener cientos de sectores.

Cuando se está usando el disco, un motor hace que gire a gran velocidad. La mayoría de los motores rotan entre 60 y 200 veces por segundo. La velocidad de un disco puede considerarse compuesta por dos partes diferenciadas: la velocidad de transferencia es la velocidad con la que los datos fluyen entre la unidad de disco y la computadora. El tiempo de posicionamiento, a veces denominado tiempo de acceso aleatorio, está compuesto del tiempo necesario para mover el brazo del disco hasta el cilindro deseado, denominado tiempo de búsqueda, y el tiempo requerido para el sector deseado rote hasta pasar por debajo del cabezal del disco, denominado latencia rotacional.

Puesto que el cabezal del disco vuela sobre un colchón de aire extremadamente fino, existe el peligro de que el cabezal entre en contacto con la superficie del disco. Aunque los platos del disco están recubiertos de una capa fina protectora, a veces el cabezal puede llegar a dañar la superficie magnética; este accidente se denomina aterrizaje de cabezales. Normalmente, los aterrizajes de cabezales no pueden repararse, siendo necesario sustituir el disco completo.

Un disco puede ser extraíble, lo que permite montar diferentes discos según sea necesario.

Para conectar una unidad de disco a una computadora, se utiliza un conjunto de cables denominado bus de E/S. Hay disponibles varios tipos de buses, incluyendo EIDE, ATA, SATA, USB, Fiber Channel (FC) y SCSI. Las transferencias de datos en un bus son realizadas por procesadores electrónicos especiales denominados controladoras. La controladora host es la controladora situada en el extremo del bus correspondiente a la computadora; además, en cada unidad de disco se integra una controladora de disco.

Estructura de Disco

Los dispositivos de disco son vistos como un arreglo unidimensional de bloques lógicos, donde el bloque lógico es la más pequeña unidad de transferencia. Ese arreglo de bloques lógicos es mapeado secuencialmente en sectores del disco. El sector 0 es el primer sector de la primera pista sobre el cilindro más externo. El mapeo procede en orden a través de esa pista, luego el resto de las pistas en el cilindro, y luego el resto de los cilindros desde el más externo hasta el más interno.

Almacenaje Adjunto de Red

Un dispositivo de almacenamiento conectado a la red (NAS, network-attached storage) es un sistema de almacenamiento de propósito general al que se accede de forma remota a través de una red de datos. Los clientes acceden al almacenamiento conectado a la red a través de una interfaz de llamadas a procedimientos remotos, como por ejemplo NFS para los sistemas UNIX o CIFS para las máquinas Windows. Las llamadas a procedimientos remotos RPC, se realizan mediante los protocolos TCP o UDP sobre una red IP.

El almacenamiento conectado a la red proporciona una forma conveniente para que todas las computadoras de una LAN compartan un conjunto de dispositivos de almacenamiento con la misma facilidad de denominación y de acceso que si se tratara de dispositivos de almacenamiento locales conectados al host. Sin embargo, este sistema tiende a ser menos eficiente y proporciona menos velocidad que algunas opciones de almacenamiento de conexión directa.

Almacenamiento en Área de Red (SAN)

Una red de almacenamiento (SAN, storage-area network) es una red privada (que utiliza protocolos de almacenamiento en lugar de protocolos de red) que conecta los servidores con las unidades de almacenamiento. La principal ventaja de una SAN radica en su flexibilidad, ya que pueden conectarse múltiples host y múltiples matrices de almacenamiento a la misma SAN y los recursos de almacenamiento pueden asignarse de forma dinámica a los hosts. Esta red es común en ambientes grandes de almacenamiento

Planificación de Disco

Una de las responsabilidades del sistema operativo es la de utilizar de manera eficiente el hardware disponible. Para las unidades de disco, cumplir con esta responsabilidad implica tener un tiempo de acceso rápido y un gran ancho de banda de disco. El tiempo de acceso tiene dos componentes principales. El tiempo de búsqueda es el tiempo requerido para que el brazo del disco mueva los cabezales hasta el cilindro que contiene el sector deseado. La latencia rotacional es el tiempo adicional necesario para que el disco rote y sitúe el sector deseado bajo el cabezal del disco. El ancho de banda del disco es el número total de bytes transferidos, dividido entre el tiempo total transcurrido entre la primera solicitud de servicio y la terminación de la última transferencia. Podemos mejorar tanto el tiempo de acceso como el ancho de banda planificando en el orden adecuado el servicio de las distintas solicitudes de E/S de disco. Existen varios algoritmos para planificar el servicio de los requerimientos de E/S.

Primero en Entrar - Primero en Salir FCFS

La forma más simple de planificación de disco es, por supuesto, el algoritmo FCFS (first-cone,first-served). Este algoritmo es intrínsecamente equitativo, pero generalmente no proporciona el servicio más rápido, dado que realiza demasiados movimientos de cabezales.

El Tiempo de Búsqueda más Corto Primero SSTF

Parece razonable satisfacer todas las solicitudes que estén próximas a la posición actual del cabezal antes de desplazar el cabezal hasta una posición muy alejada para dar servicio a otras solicitudes. Esta suposición es la base para el algoritmo de búsqueda más corto. Este algoritmo selecciona la solicitud que tenga el tiempo de búsqueda mínimo con respecto a la posición actual del cabezal.

La planificación SSTF es, esencialmente, un tipo de planificación SJF (shortest-job-first) y, al igual que la planificación SJF, puede provocar la muerte por inanición de algunas solicitudes, por lo cual es inequitativo. Además el tiempo medio depende de la carga.

Aunque el algoritmo SSTF representa una mejora sustancial con respecto al algoritmo FCFS, no es un algoritmo óptimo.

SCAN

En el algoritmo SCAN, el brazo del disco comienza en uno de los extremos del disco y se mueve hacia el otro extremo, dando servicio a las solicitudes a medida que pasa por cada cilindro. En ese otro extremo, se invierte la dirección del cabezal y se continúa dando servicio a las solicitudes. El algoritmo SCAN se denomina también en ocasiones algoritmo del ascensor. Cabe aclarar que por su naturaleza no tiene un tiempo de espera uniforme, ya que si llega una solicitud a la cola y esa solicitud corresponde a un cilindro situado justo delante del cabezal, dicha solicitud será servida casi inmediatamente; por el contrario, una solicitud relativa a un cilindro situado justo detrás del cabezal tendrá que esperar hasta que el brazo alcance el extremo del disco, invierta su dirección y vuelva atrás.

LOOK

Los algoritmos SCAN mueven el brazo a través de la anchura completa del disco; sin embargo, en la práctica, ninguna de las versiones del algoritmo SCAN se suele implementar de esta manera. Lo más común es que el brazo sólo vaya hasta el cilindro correspondiente a la solicitud final en cada dirección. Entonces, invierte su dirección inmediatamente, sin llegar hasta el extremo del disco.

C-SCAN

La planificación circular (C-SCAN) es una variante de SCAN diseñada para proporcionar un tiempo de espera más uniforme. Al igual que SCAN, C-SCAN mueve el cabezal de un extremo del disco al otro, prestando servicio a las solicitudes a lo largo de ese trayecto. Sin embargo, cuando el cabezal alcanza el otro extremo, vuelve inmediatamente al principio del disco, sin dar servicio a ninguna solicitud en el viaje de vuelta. El algoritmo de planificación C-SCAN trata esencialmente a los cilindros como si fueran una lista circular que se plegara sobre sí misma conectando el cilindro final con el primer cilindro.

Selección de un Algoritmo de Planificación de Disco

SSTF resulta bastante común y tiene un atractivo natural, porque mejora la velocidad que puede obtenerse con FCFS. SCAN y C-SCAN tienen un mejor comportamiento en aquellos sistemas donde el disco está sometido a una intensa carga, porque es menos probable que provoquen problemas de muerte por inanición. Con cualquier algoritmo de planificación, las prestaciones dependen en gran medida del número de solicitudes y del tipo de estas.

Las solicitudes de servicio de disco pueden verse enormemente influenciadas por el método de asignación de archivos. Un programa que esté leyendo un archivo con asignación contigua generará varias solicitudes que están próximas entre sí en el disco, lo que da como resultado un movimiento de cabezales limitado. Por el contrario, un archivo enlazado o indexado puede incluir bloques que estén muy dispersos por el disco, lo que provoca un movimiento mucho mayor de los cabezales. La ubicación de los directorios y de los bloques de índice también es importante. Debido a estas complejidades, el algoritmo de planificación debe escribirse como un módulo independiente del sistema operativo, para poderlo sustituir por otro algoritmo distinto en caso necesario. Tanto SSTF como LOOK son una elección razonable como algoritmo predeterminado.

Los algoritmos de planificación sólo tienen en cuenta las instancias de búsqueda. Para los discos modernos, la latencia rotacional puede ser casi tan grande como el tiempo medio de búsqueda. Sin embargo, resulta difícil para el sistema operativo realizar una planificación para optimizar la latencia rotacional, porque los discos modernos no revelan la ubicación física de los bloques lógicos. Los fabricantes de disco tratan de aliviar este problema implementando algoritmos de planificación del disco en el hardware controlador integrado dentro de la unidad de disco.

Administración de Disco

Un disco magnético nuevo es como una pizarra en blanco, se trata simplemente de una placa de material magnético para grabación. Antes de poder almacenar los datos en el disco, es necesario dividir este en sectores que la controladora de disco pueda leer y escribir. Este proceso se denomina formateo de bajo nivel o formateo físico.

Para utilizar un disco para almacenar archivos, el sistema operativo sigue necesitando poder grabar sus propias estructuras de datos en el disco y para ello sigue un proceso en dos pasos. El primer paso consiste en particionar el disco en uno o más grupos de cilindros. Después del particionamiento, el segundo paso es el formato lógico o creación de un sistema de archivos.

Para que una computadora comience a operar debe tener un programa inicial que ejecutar. Este programa inicial de arranque tiende a ser muy simple. Se encarga de inicializar todos los aspectos del sistema, desde los registros de la CPU hasta las controladoras de dispositivo y el contenido de la memoria principal, y luego arranca el sistema operativo. Para

llevar a cabo su tarea, el programa de arranque localiza el kernel del sistema operativo en disco, carga dicho kernel en memoria y salta hasta una dirección inicial con el fin de comenzar la ejecución del sistema operativo.

Para la mayoría de las computadoras, el programa de arranque está almacenado en memoria de solo lectura, ROM. Este programa se lo conoce como bootstrap. Si bien esta ubicación resulta muy adecuada, porque la ROM no necesita ser inicializada y se encuentra en una dirección fija y no puede verse afectada por virus informáticos, si es necesario cambiar el código deberemos cambiar los chips hardware de la ROM. Por esta razón, la mayoría de los sistemas almacenan un programa cargador de arranque de muy pequeño tamaño en la ROM de arranque, cuya única tarea consiste en cargar un programa de arranque completo en el disco.

Puesto que los discos tienen partes móviles y tolerancias muy pequeñas son bastantes propensos a los fallos. Algunas veces, el fallo que se produce es completo, en cuyo caso será necesario cambiar el disco y restaurar su contenido en el nuevo disco a partir de un soporte de copia de seguridad. Lo más frecuente es que uno o más sectores pasen a ser defectuosos. La mayoría de los discos vienen incluso de fábrica con una serie de bloques defectuosos. Dependiendo del disco y de la controladora que se utilice, estos bloques se gestionan de diversas formas.

Manejo de Espacio de Swapping

La gestión del espacio de intercambio es otra tarea de bajo nivel del sistema operativo. La memoria virtual utiliza el espacio de disco como una extensión de la memoria principal.

Debemos observar que puede resultar más seguro sobrestimar la cantidad de espacio de intercambio requerido, en lugar de subestimarla, porque si un sistema se queda sin espacio de intercambio puede verse forzado a abortar procesos o puede incluso sufrir un fallo catastrófico. La sobreestimación hace que se desperdicie un espacio de disco que podría, de otro modo, utilizarse para los archivos, pero no provoca ningún daño.

El espacio de intercambio puede residir en uno de dos lugares: puede construirse a partir del sistema de archivos normal o puede residir en una partición de disco separada. Si el espacio de intercambio es simplemente un archivo de gran tamaño dentro del sistema de archivos, pueden usarse las rutinas normales del sistema de archivos para crearlo, nombrarlo y asignarle el espacio. Está técnica, aunque resulta fácil de implementar, también es poco eficiente. Alternativamente, puede crearse un espacio de intercambio en una partición sin formato separada, ya que en este espacio no se almacena ningún sistema de archivos ni estructura de directorio. En lugar de ello, se utiliza un gestor de almacenamiento independiente para el espacio de intercambio, con el fin de asignar y desasignar los bloques de la partición sin formato.

Estructura RAID

Disponer de un gran número de discos en un sistema permite mejorar la velocidad con la que lo que datos pueden leerse o escribirse, siempre y cuando los discos operen en paralelo. Además, esta configuración también puede permitir mejorar la fiabilidad del almacenamiento de los datos y que puede almacenarse información redundante en ese conjunto de múltiples discos. De este modo, el fallo de uno de los discos no conduce a la pérdida de los datos. Existen diversas técnicas de organización de discos, colectivamente denominadas matrices redundantes de discos de bajo coste, que se utilizan comúnmente para resolver estas cuestiones de velocidad y de fiabilidad.

La técnica más simple, pero la más cara, para introducir redundancia consiste en duplicar cada uno de los discos. Esta técnica se denomina duplicación en espejo (mirroring). Así, cada disco lógico estará compuesto por dos discos físicos y toda escritura se llevará a cabo en ambos discos a la vez. Si uno de los discos falla, pueden leerse los datos del otro disco. Los datos sólo se perderán si el segundo disco fallara antes de que el primero disco estropeado se reemplace. Con este mecanismo, la velocidad a la que pueden gestionarse las solicitudes de lectura se dobla, ya que esas solicitudes

pueden enviarse a cualquier de los discos. No obstante, la escritura se reduce ya que debe ser realizada en los dos discos para mantener el espejo.

También podemos mejorar la velocidad de transferencia dividiendo los datos entre distintos discos. En su forma más simple, el mecanismo de distribución en bandas de los datos (data striping) consiste en dividir los bits de cada byte entre múltiples discos; dicha distribución se denomina distribución en bandas de nivel bit. Con este tipo de organización, todos los discos participan en cada acceso (de lectura o escritura) porque el número de accesos que pueden procesarse por segundo es más o menos el mismo que con un único disco, pero cada acceso puede leer más datos en el mismo tiempo que se si utiliza un único disco.

Niveles RAID

La duplicación en espejo proporciona una alta fiabilidad, pero resulta muy cara. La distribución en bandas proporciona una alta velocidad de transferencia de datos, pero no mejora la fiabilidad. Se han propuesto números esquemas para proporcionar redundancia a un menor coste utilizando la idea de distribución en bandas combinada con bits de paridad. Estos esquemas tienen diferentes tipos de compromisos entre el coste y las prestaciones y se clasifican de acuerdo con una serie de niveles denominados niveles RAID.

Implementación de Almacenaje Estable

La información que reside en un almacenamiento estable no se pierde nunca. Para implementar este tipo de almacenamiento, necesitamos replicar la información necesaria en múltiples dispositivos de almacenamiento con modos independientes de fallo. Necesitamos coordinar la escritura de las actualizaciones en forma que se garantice que un fallo durante una actualización no dejará todas las copias en un estado incorrecto y que, cuando nos estemos recuperando de un fallo, podamos forzar a todas las copias para que adopten un estado coherente y correcto, incluso si se produce otro fallo durante la recuperación.

Dispositivos de Almacenaje Terciarios

En una computadora puede reducirse el coste global utilizando muchos discos de bajo coste con una misma unidad. El bajo coste es una característica definitoria del almacenamiento terciario. Puesto que el coste es tan importante, en la práctica el almacenamiento terciario se construye con soportes extraíbles. Los ejemplos más comunes son los disquetes, las cintas y los CD, DVD y los pendrives.

Aspectos del Sistema Operativo

Dos de los principales cometidos de un sistema operativo son gestionar los dispositivos físicos y presentar una abstracción de máquina virtual a las aplicaciones. Para los discos duros el SO provee dos abstracciones. Una es el dispositivo crudo o sin formato, compuesto simple por una matriz de bloques de datos. La otra es un sistema de archivos.

La mayoría de los sistemas operativos pueden gestionar los discos extraíbles de forma casi exactamente igual que los discos fijos. Cuando se inserta un cartucho en blanco en una unidad, el cartucho debe formatearse, con lo que se escribe en el disco un sistema de archivos vacio. Este sistema de archivos se utiliza igual que los sistemas de archivo residentes en el disco duro.

Las cintas suelen manejarse de modo distinto. El sistema operativo, usualmente, presenta las cintas como soportes de almacenamiento sin formato. Las aplicaciones no abren un archivo en la cinta, sino que abren la unidad de cinta completa como un dispositivo sin formato. Usualmente, la unidad de cinta queda entonces reservada para el uso exclusivo por parte de dicha aplicaciones, hasta que la aplicación termina o hasta que cierra el dispositivo de cinta.

Resulta fácil ver los problemas que pueden surgir debido a esta forma de utilizar las cintas. Puesto que cada aplicación tiene sus propias reglas en cuanto al modo de organizar una cinta, cada cinta llena de datos sólo puede, por regla general, ser utilizada por el programa que la creó.

Otra cuestión que el sistema operativo necesita resolver es cómo nombrar los archivos en los soportes extraíbles.

Protección y Seguridad - Módulo 13

Principios de Protección

Podemos utilizar un principio director a lo largo del diseño de un sistema operativo. Ajustarnos a este principio simplifica las decisiones de diseño y hace que el sistema continúe siendo coherente y fácil de comprender. Uno de los principios directores clave y que ha resistido al paso del tiempo a la hora de proporcionar protección es el principio del mínimo privilegio. Este principio dicta que a los programas, a los usuarios, incluso a los sistemas se les concedan únicamente los suficientes privilegios para llevar a cabo sus tareas.

Un sistema operativo que se ajuste al principio del mínimo privilegio implementará sus características, programas, llamadas al sistema y estructuras de datos de modo que el fallo o el compromiso de un componente provoquen un daño mínimo y no permitan realizar más que un daño mínimo.

Estructura de Dominios

Un sistema informático es una colección de procesos y objetos. Cada objeto tiene un único nombre y puede ser accedido por un conjunto de operaciones bien definidas. El problema de la protección asegura que cada objeto es accedido correctamente y solo por aquellos procesos que les está permitido hacerlo.

Para facilitar este esquema, un proceso opera dentro de un dominio de protección, que especifica los recursos a los que el proceso puede acceder. Cada dominio define un conjunto de objetos y los tipos de operaciones que pueden invocarse sobre cada objeto. La capacidad de ejecutar una operación sobre un objeto es un derecho de acceso. Un dominio es una colección de derechos de acceso, cada uno de los cuales es una pareja ordenada <nombre-objeto, conjunto-derechos>.

Los dominios no tienen por qué ser disjuntos, sino que pueden compartir derechos de acceso.

Implementación de Dominios (UNIX)

En el sistema operativo UNIX, un dominio está asociado con el usuario. La conmutación de dominio se corresponde con un cambio temporal en la identificación del usuario. Este cambio se lleva a cabo a través del sistema de archivos de la forma siguiente: con cada archivo hay asociada una identificación de propietario y un bit de dominio (conocido como el bit setuid); cuando el bit setuid esta activado y un usuario ejecuta dicho archivo, el ID de usuario se configura con el valor correspondiente al propietario del archivo; sin embargo, cuando el bit está desactivado, el ID del usuario no se modifica. Cuando se completa la ejecución la identificación de usuario es retorna a su original.

Debe tenerse un gran cuidado a la hora de escribir los programas privilegiados. Cualquier descuido puede provocar una total falta de protección en el sistema. Generalmente, estos programas son los primeros en ser atacados por aquellas personas que tratan de irrumpir dentro de un sistema.

Matriz de Acceso

El modelo de protección basado en dominios puede contemplarse de forma abstracta como una matriz, denominada matriz de acceso. Las filas de la matriz de acceso representan dominios y las columnas representan objetos. Cada entrada de la matriz está compuesta de un conjunto de derechos de acceso. La entrada acceso (i,j) define el conjunto de operaciones que un proceso que se ejecute en el dominio D_i puede invocar sobre el objeto O_j.

Si un proceso en Dominio D_i trata de hacer "op" sobre el objeto O_j, entonces "Op" debe estar en la matriz de acceso.

La matriz de acceso proporciona el mecanismo apropiado para definir e implementar un control estricto de la asociación tanto estática como dinámica entre procesos y dominios. Cuando conmutamos un proceso de un dominio a otro, ejecutamos una operación (switch) sobre un objeto (el dominio). Podemos controlar la conmutación de dominios incluyendo los dominios entre los objetos de la matriz de acceso. De forma similar, cuando cambiamos el contenido de la matriz de acceso, realizamos una operación sobre un objeto: la propia matriz de acceso. Entonces debemos considerar a cada entrada de la matriz de acceso como un objeto que hay que proteger. Ahora, sólo necesitamos considerar las operaciones que sean posibles sobre estos objetos y decidir cómo queremos que los procesos puedan ejecutar dichas operaciones.

Estas operaciones sobre los dominios y la matriz de acceso no son por sí mismas importantes, pero ilustran la capacidad del modelo de la matriz de acceso para permitir la implementación y control de requisitos de protección dinámicos. Pueden crearse dinámicamente nuevos objetos y nuevos dominios e incluirlos en el modelo de la matriz de acceso. Sin embargo, sólo hemos mostrado que el mecanismo básico existe; los diseñadores del sistema y los usuarios deben tomar las decisiones de políticas relativas a qué dominios deben poder acceder a qué objetos y en qué manera.

Implementación de la Matriz de Acceso

La implementación más simple de la matriz de acceso es una tabla global compuesta de un conjunto de tripletas ordenadas <dominio, objeto, conjunto-derechos>. La tabla es usualmente muy grande y no puede, por tanto, ser conservada en memoria principal, por lo que hacen falta operaciones adicionales de E/S. A menudo se utilizan técnicas de memoria virtual para gestionar esta tabla. Además, resulta difícil aprovechar las agrupaciones especiales de objetos o dominios.

Cada columna de la matriz de acceso puede implementarse como una lista de acceso de un objeto. Obviamente, las entradas vacías pueden descartarse. La lista resultante para cada objeto estará compuesta por una serie de parejas ordenadas <dominio, conjunto-derechos>, que definen todos los dominios que tenga un conjunto de derechos de acceso no vacio para dicho objeto.

En lugar de asociar las columnas de la matriz de acceso con los objetos en forma de listas de acceso, podemos asociar cada fila con su dominio. Una lista de capacidades para un dominio es una lista de objetos junto con las operaciones permitidas sobre esos objetos. Cada objeto se suele representar mediante su dirección o nombre físico, denominada capacidad. La simple posesión de la capacidad significa que el acceso está permitido.

Control de Accesos

La protección puede ser aplicada también a recursos físicos.

Solaris 10 amplía el sistema de protección disponible en el sistema operativo añadiendo explícitamente el principio del mínimo privilegio mediante el control de acceso basado en roles. Esta funcionalidad gira en torno a los privilegios. Un privilegio es el derecho a ejecutar una llamada al sistema o usar una opción dentro de dicha llamada al sistema. Podemos asignar privilegios a los procesos, limitando éstos a exactamente el tipo de acceso que necesitan para llevar a cabo su tarea. Esta implementación de los privilegios reduce el riesgo de seguridad asociado al setuid.

Revocación de Derechos de Acceso

En un sistema de protección dinámico, puede que necesitamos en ocasiones revocar derechos de acceso a objetos compartidos por diferentes usuarios.

Con el esquema basado en lista de acceso, la revocación resulta sencilla. Se analiza la lista de acceso en busca de los derechos de acceso que hay que revocar y se los borra de la lista. La revocación es inmediata y puede ser general o selectiva, total o parcial y permanente o temporal.

Las capacidades, sin embargo, presentan un problema mucho más difícil de cara a la revocación. Puesto que las capacidades están distribuidas por todo el sistema, debemos encontrarlas antes de poder revocarlas. Entre los esquemas que pueden utilizarse para implementar la revocación en un sistema basado en capacidades podemos citar:

- Readquisición: Periódicamente, se borran las capacidades de cada dominio. Si un proceso quiere usar una capacidad, puede que se encuentre con que esa capacidad ya ha sido borrada. El proceso puede entonces tratar de volver a adquirir la capacidad. Si se ha revocado el acceso, el proceso no será capaz de efectuar esa adquisición.

- Retropunteros: Con cada objeto se mantiene una lista de punteros, que hace referencia a todas las capacidades asociadas con ese objeto. Cuando se necesita realizar una revocación, podemos seguir esos punteros, cambiando las capacidades según sea necesario.

- Indirección: Las capacidades apuntan indirectamente, en lugar de directamente, a los objetos. Cada capacidad apunta a una entrada unívoca dentro de una tabla global, que a su vez apunta al objeto. Implementamos la revocación analizando la tabla global en busca de la entrada deseada y borrándola. Entonces, cuando se intenta realizar un acceso, se verá que la capacidad está apuntado a una entrada ilegal de la tabla.

- Claves: Una clave es un patrón distintivo de bits que pueden asociarse con una capacidad. Esta clave define en el momento de crear la capacidad y no puede ser nunca modificada ni inspeccionada por el proceso que posee la capacidad. Con cada objeto hay asociada una clave maestra. El proceso de revocación sustituye la clave maestra con un nuevo valor, invalidando las capacidades anteriores del objeto.

Protección Basada en Lenguajes

Las políticas de uso de los recursos también pueden variar, dependiendo de la aplicación, y pueden estar sujetas a cambios a lo largo del tiempo. Por estas razones, la protección ya no puede ser considerada exclusivamente como una preocupación del diseñador del sistema operativo, sino que también debe estar disponible como herramienta para que la use el diseñador de aplicaciones, de modo que los recursos de un subsistema de aplicación puedan estar protegidos frente a las modificaciones o frente a la influencia de errores.

Es aquí donde entran dentro del panorama los lenguajes de programación. La especificación de protección en lenguajes de programación permite una descripción en alto nivel de políticas para la alocación y uso de recursos. La implementación del lenguaje puede forzar software para protección cuando la verificación automática soportada por hardware no esta disponible.

Código Móvil

Una técnica para protegerse de las amenazas, es ejecutar un programa en lo que se denomina cajón de arena, que es una sección emulada o controlada del sistema. El software antivirus analiza el comportamiento del código en el cajón de arena antes de dejar que se ejecute sin monitorización.

Seguridad

El problema de Seguridad

En muchas aplicaciones, merece la pena dedicar un esfuerzo considerable a garantizar la seguridad de un sistema informatico, tales como los sistemas comerciales de gran envergadura que contenga datos financieros o relativos a corporaciones.

Decimos que un sistema es seguro si sus recursos se utilizan de la forma prevista y si se accede a ellos tal como se pretendía, en todas las circunstancias. Desafortunadamente, no es posible conseguir una seguridad total; a pesar de ello, debemos prever mecanismos para hacer que los fallos de seguridad constituyan la excepción, en lugar de la norma.

Un intruso o cracker es aquella persona que intenta romper la seguridad. Una amenaza es potencialmente una violación a la seguridad. Un ataque es un intento de romper la seguridad. Este puede ser accidental o malicioso. Es más fácil protegerse contra un uso accidental que contra uno malicioso ya que principalmente los mecanismos de protección forman la base de defensa frente a posibles accidentes.

Violaciones de Seguridad

Existen diferentes categorías:

- Fallo de confidencialidad. Este tipo de violación implica la lectura no autorizada de determinados datos privados.
- Fallo de integridad. Este tipo de ataque implica la modificación no autorizada de los datos.
- Robo de servicio. Este tipo de violación de seguridad implica el uso no autorizado de recursos.
- Negación de Servicio (Denial of service) Esta violación implica impedir el uso legitimo del sistema. Los ataques de denegación de servicio son en ocasiones accidentes.

Los atacantes utilizan diversos métodos estándar en sus intentos de romper la seguridad de los sistemas. El más común es la mascarada, en la que un participante en una comunicación pretende ser otra persona. Mediante la mascarada, los atacantes rompen la autenticación, es decir, la corrección de la identificación; como consecuencia, pueden obtener tipos de acceso que normalmente no les estarían permitidos o escalar sus privilegios, es decir, obtener privilegios que normalmente no podrían disfrutar. Otro ataque común consiste en reproducir un intercambio de datos previamente capturado: los ataques de reproducción consisten en la repetición maliciosa o fraudulenta de una transmisión de datos válida. En ocasiones, esa reproducción constituye el propio ataque, como por ejemplo cuando se repite una solicitud para transferir dinero, pero frecuentemente la reproducción se realiza en conjunción con una modificación del mensaje. Otro tipo de ataque es el ataque por interposición (hombre en el medio), en el cual un atacante se introduce dentro del flujo de datos de una comunicación, haciéndose pasar por el emisor a ojos del receptor y viceversa. En una comunicación por red, un ataque de imposición puede estar precedido de un secuestro de sesión, en el que se intercepta una sesión de comunicación activa.

Niveles de Medidas de Seguridad

Para proteger un sistema, debemos adoptar las necesarias medidas de seguridad en cuatro niveles distintos:

- Físico. El nodo o nodos que contengan los sistemas informaticos deben dotarse de medidas de seguridad físicas frente a posibles intrusiones armadas por parte de los potenciales intrusos.
- Humano. La autorización de los usuarios debe llevarse a cabo con cuidado, para garantizar que sólo los usuarios apropiados tenga acceso al sistema. Sin embargo, incluso los usuarios autorizados pueden verse “motivados”

para permitir que otros usen su acceso. También pueden ser engañados para permitir el acceso a otros, mediante técnicas de ingeniería social. Uno de los tipos de ataque basado en las técnicas de ingeniería social es denominado phishing; con este tipo de ataque, un correo electrónico o página web de aspecto autentico llevan a engaño a un usuario que introduzca información confidencial. Otra técnica comúnmente utilizada es el análisis de desperdicios, un término general para describir el intento de recopilar información para poder obtener un acceso no autorizado a la computadora. Estos problemas de seguridad son cuestiones relacionadas con la gestión y con el personal, más que problemas relativos a los sistemas operativos.

- Sistemas Operativo. El sistema debe autoprotegerse frente a los posibles fallos de seguridad accidentales o premeditados.

- Red. Son muchos los datos en los modernos sistemas informaticos que viajan a través de líneas arrendadas privadas, de líneas compartidas como Internet, de conexiones inalámbricas o de líneas de acceso telefónico. La intercepción de estos datos podría ser tan dañina como el acceso a una computadora.

Si queremos garantizar la seguridad del sistema operativo, es necesario garantizar la seguridad en los primeros dos niveles. Cualquier debilidad en uno de los niveles altos de seguridad (físico o humano) permitirá puentear las medidas de seguridad que son estrictamente de bajo nivel. Así, la frase que afirma que una cadena es tan fuerte como el más débil de los eslabones es especialmente cierta cuando hablamos de la seguridad de los sistemas. Para poder mantener la seguridad, debemos contemplar todos los aspectos.

Programas Peligrosos

Los procesos son, junto con el kernel, el único medio de realizar trabajo útil en una computadora. Por tanto, un objetivo común de los piratas informáticos consiste en escribir un programa que cree una brecha de seguridad o que haga que un proceso normal cambie su comportamiento y cree esa brecha de seguridad.

- Caballo de Troya. Muchos sistemas tienen mecanismos para permitir que programas escritos por unos usuarios sean ejecutados por otros. Si estos programas se ejecutan en un dominio que proporcione los derechos de acceso del usuario ejecutante, los otros usuarios podrían utilizar inapropiadamente estos derechos. Un segmento de código que utilice inapropiadamente su entorno se denomina caballo de Troya.

  Otra variante del caballo de Troya es el spyware. Los programas spyware acompañan en ocasiones a ciertos programas que el usuario haya decido instalar. En la mayoría de los casos, se incluyen con programas tipo freeware o shareware, pero en otras ocasiones también se incluyen en software de carácter comercial. El objetivo del spyware es descargar anuncios para mostrarlos en el sistema del usuario, crear ventajas de explorador emergentes cuando se visiten sitios o capturar información del sistema del usuario y enviarla a un sitio central. Este ultimo modo es un ejemplo de una categoría general de ataques conocida como canales encubiertos, a través de los cuales tienen lugar comunicaciones subrepticias.

- Puerta trasera o puerta trampa. El diseñador de un programa o sistema puede dejar detrás suyo un agujero en el software que sólo él sea capaz de utilizar. Por ejemplo, el código puede tratar de detectar un ID de usuario o una contraseña especificos y, al detectarlo, evitar los procedimientos de seguridad normales. Podría incluirse una puerta trasera inteligente dentro del propio compilador. Las puertas traseras representan un problema porque para detectarlas, tenemos que analizar todo el código fuente de todos los componentes de un sistema, una tarea muy extensa considerando la cantidad de líneas de código de los sistemas de software.

- Bomba Lógica. Considere un programa capaz de iniciar un incidente de seguridad sólo cuando se dan determinadas circunstancias. Sería difícil de detectar porque, en condiciones normales de operación, no existiría ningún agujero de seguridad.

- Rebalse de Stack y Buffer. El ataque por desbordamiento de pila o de búfer es la forma más común para que un atacante externo al sistema a través de una conexión de red obtenga acceso no autorizado al sistema operativo. Esencialmente, lo que el ataque hace es aprovechar un error de un programa. El resultado de la ejecución del código de este programa de ataque será una shell raíz o la ejecución de otro comando privilegiado.

Amenazas al Sistema y Red

Las amenazas del sistema y de la red implica el abuso de los servicios y las conexiones de red.

- Gusanos. Un gusano es un proceso que utiliza un mecanismo de reproducción para afectar al rendimiento del sistema. El gusano crea copias de sí mismo, utilizando recursos del sistema y en ocasiones impidiendo operar a todos los demás procesos. El Worm Internet explotaba características de red UNIX y bugs en los programas finger y sendmail.

- Barrido de Pórticos. El escaneo de puertos no es un ataque, sino más bien un método para que los piratas informáticos detecten las vulnerabilidades del sistema que puedan ser atacadas. El escaneo de puertos se realiza normalmente de forma automatizada, lo que implica utilizar una herramienta que trate de crear una conexión TCP/IP a un puerto o rango de puertos específicos.

- Denegación de servicio. Los ataques DOS están dirigidos no a obtener información o robar recursos, sino a impedir el uso legítimo de un sistema o funcionalidad. Este tipo de ataques se realizan generalmente a través de la red. Se los puede clasificar en dos categorías. El primer caso es el de los ataques que consumen tantos recursos de la máquina atacada que prácticamente no pueden realizarse con ella ningún trabajo útil. El segundo caso de ataque implica hacer caer la red o la instalación. Generalmente, es imposible prevenir los ataques de denegación de servicio ya que utilizan mismos mecanismos que la operación normal. Todavía es más difícil de prevenir y de solucionar los ataques distribuidos de denegación de servicio. Estos ataques se inician desde múltiples sitios a la vez, dirigidos hacia un objeto común.

Criptografía como Herramienta de Seguridad

En una computadora aislada, el sistema operativo puede determinar de manera fiable quiénes son el emisor y el receptor de todas las comunicaciones interprocesos, ya que el sistema operativo controla todos los canales de comunicaciones de la computadora. En una red de computadoras, la situación es bastante distinta. Una computadora conectada a la red envía y recibe bits desde el exterior y no tiene ninguna forma inmediata y fiable de determinar qué maquina o aplicación han recibido/enviado esos bits. Si bien hay direcciones de red que permiten saber quien envía o recibe, puede haber suplantación de identidad por lo que no se puede fiar de las mismas.

Por tanto, la única alternativa es eliminar, de alguna manera, la necesidad de confiar en la red; este es el trabajo de la criptografía. Desde un punto de vista abstracto la criptografía se utiliza para restringir los emisores y/o receptores potenciales de un mensaje. La criptografía moderna se basa en una serie de secretos, denominados claves, que se distribuyen selectivamente a las computadoras de una red y se utilizan para procesar mensajes. La criptografía permite al receptor de un mensaje verificar que el mensaje ha sido creado por alguna computadora que posee una cierta clave: esa clave es el origen del mensaje. De forma similar, un emisor puede codificar su mensaje de modo que sólo una computadora que disponga de cierta clave pueda decodificar el mensaje, de manera que esa clave se convierte en el destino. Las claves están diseñadas de modo que no sea computacionalmente factible calcularlas a partir de los mensajes que se han generado con ellas, ni a partir de ninguna información publica

Bases de la Criptografía

El cifrado es un medio de restringir los posibles receptores de un mensaje. Un algoritmo de cifrado permite al emisor de un mensaje garantizar que sólo pueda leer el mensaje una computadora que posea cierta clave.

Un algoritmo de cifrado consta de los siguientes componentes:

- Un conjunto K de claves
- Un conjunto M de mensajes
- Un conjunto C de mensajes de texto cifrado
- Una función E: K-> (M->C). Es decir, para cada k ε K, E(k) es una función para generar mensajes de texto cifrado a partir de los mensajes de texto plano. Tanto E como E(k) para cualquier k deben ser funciones computables de manera eficiente.
- Una función D: K -> (C->M). Es decir, para cada k ε K, D(k) es una función para generar mensajes de texto plano a partir de los mensajes de texto cifrado. Tanto D como D(k) para cualquier k deben ser funciones eficientemente computables.

Un algoritmo de cifrado debe proporcionar esta propiedad esencial: dado un mensaje de texto cifrado c ε C, una computadora puede calcular m tal que E(k)(m) = c sólo si posee D(k). Así, una computadora que posea D(k) puede descifrar los mensajes de texto cifrado para obtener los mensajes de texto plano que se usaron para producirlos, pero una computadora que no posea D(k) no puede descifrar esos mensajes cifrados. Puesto que los mensajes cifrados están generalmente expuestos es importante que no se pueda deducir D(k) a partir de los mensajes cifrados.

Existen dos tipos principales de algoritmos de cifrado: simétricos y asimétricos.

En un algoritmo de cifrado simétrico, se utiliza la misma clave para cifrar y descifrar, es decir, E(k) puede deducirse a partir de D(k) y viceversa. Por tanto, es necesario proteger el secreto de E(k) con el mismo grado que el de D(k).

En un algoritmo de cifrado asimétrico, las claves de cifrado y descifrado son distintas.

Resulta impracticable, desde el punto de vista computacional, deducir $D(k_d, N)$ a partir de $E(k_e, N)$ por lo que no es necesario mantener $E(k_e, N)$ en secreto y dicha clave puede ser ampliamente diseminada; por tanto, $E(k_d,N)$ es la clave publica y $D(k_d,N)$ es la clave privada. N es el producto de dos números primos de gran magnitud p y q aleatoriamente seleccionados. El algoritmo de cifrado es $E(k_e,N)(m)=m^{ke} \bmod N$, donde k_e satisface la ecuación $k_e k_d \bmod (p-1)(q-1) = 1$. El algoritmo de descifrado será entonces $D(k_d,N)(c)=c^{kd} \bmod N$.

El uso del mecanismo de cifrado asimétrico comienza con la publicación de la clave pública del destino. Para la comunicación bidireccional, el origen debe también publicar su clave pública. Esa “publicación” puede ser tan simple como entregar una copia electrónica de la clave, o puede tratarse de un mecanismo más complejo. La clave privada (o “clave secreta”) debe ser guardada celosamente, ya que cualquiera que disponga de esa clave podrá descifrar cualquier mensaje creado a partir de la correspondiente clave publica.

Firma Digital

El proceso de restringir el conjunto de potenciales emisores de un mensaje se denomina autenticación. La autenticación es, por tanto, complementaria al cifrado. La autenticación sirve para demostrar que un mensaje no ha sido modificado.

Un algoritmo de autenticación consta de los siguientes componentes

- Un conjunto K de claves.
- Un conjunto M de mensajes.
- Un conjunto A de autenticadores.
- Una función S: K -> (M->A). Es decir, para cada k ε K, S(k) es una función para generar autenticadores a partir de los mensajes. Tanto S como S(k) para cualquier k deben ser funciones computacionalmente eficientes.
- Una función V:K -> (MxA -> {true,false}). Es decir, para cada k ε K, V(k) es una función para verificar autenticadores de mensajes. Tanto V como V(k) para cualquier k deben ser funciones completamente computables.

La propiedad critica que un algoritmo de autenticación debe poseer es esta: para un mensaje m, una computadora puede generar un autenticador a ε A tal que V(k)(m,a)=true sólo si posee S(k). Así, una computadora que posea S(k) podrá generar autenticadores para los mensajes de modo que cualquier otra computadora que posea V(k) pueda verificarlo. Sin embargo, una computadora que no tenga S(k) no podrá generar autenticadores para los mensajes que puedan verificarse utilizando V(k). Puesto que los autenticadores están generalmente expuestos no debe ser factible deducir S(k) a partir de los autenticadores.

Al igual que hay dos algoritmos de cifrado, hay también dos variedades principales de algoritmos de autenticación. El primer tipo de algoritmo de autenticación utiliza cifrado simétrico. En un código de autenticación de mensajes (MAC, message-authentication code) se genera una suma de comprobación criptográfica a partir del mensaje utilizando una clave secreta. El conocimiento de V(k) y el conocimiento de S(k) son equivalentes: podemos derivar una función a partir de la otra, por lo que es necesario mantener en secreto el valor de k. Debido a la propiedad de resistencia a las colisiones de la función hash, podemos estar razonablemente seguros de que ningún otro mensaje permita crear el mismo valor MAC. Un algoritmo de verificación apropiado será entonces V(k)(m,a) = (f(k,m)=a)

El segundo tipo principal de algoritmo de autenticación es el algoritmo de firma digital, y los autenticadores generados por uno de estos algoritmos se denominan firmas digitales. En un algoritmo de firma digital, no resulta computacionalmente factible deducir $S(k_s)$ a partir de $V(k_v)$; en particular, V es una función de una sola dirección. Por tanto, k_v es la clave pública y K_s es la clave privada.

Autenticación de Usuario

Generalmente, la autenticación de usuario se basa en uno o más de tres cuestiones: la posesión de algo por parte del usuario (una clave o tarjeta), el conocimiento de algo (un identificador de usuario y una contraseña) por parte del usuario y/o un atributo del usuario (huella digital, patrón retinal o firma).

La autenticación de usuario es crucial para identificar correctamente al usuario, dado que el sistema de protección depende del user ID.

El método más habitual para autenticar la identidad de un usuario consiste en usar contraseñas. Cuando el usuario se identifica a sí mismo mediante un ID de usuario o un nombre cuenta, se le pide una contraseña. Si la contraseña suministrada por el usuario coincide con la contraseña almacenada en el sistema, el sistema supone que el propietario de la cuenta está accediendo a la misma.

Las contraseñas pueden considerarse un caso especial de las claves o de las capacidades.

Las contraseñas son extremadamente comunes porque son fáciles de de comprender y utilizar. Lamentablemente, a menudo pueden adivinarse, ser mostradas por accidente, ser interceptadas o ser ilegalmente transferidas por un usuario autorizado a otro que no lo está. Para permitir que las contraseñas permanezcan en secreto, se las debe cambiar frecuentemente y no se deben usar contraseñas adivinables.

Un problema que se plantea es la dificultad de mantener en secreto la contraseña dentro de la computadora. Los sistemas UNIX utilizan el cifrado para evitar la necesidad de mantener en secreto su lista de contraseñas. El sistema contiene una función que es extremadamente difícil de invertir, pero fácil de calcular. El problema de este método es que el sistema ya no tiene el control sobre las contraseñas. Aunque las contraseñas se cifren, cualquiera que disponga de una copia del archivo de contraseñas puede ejecutar rutinas de cifrado rápido, cifrando cada palabra en un diccionario, por ejemplo, y comparando los resultados con las contraseñas almacenadas.

Por otro lado, para evitar los problemas de intercepción de contraseñas un sistema podría usar un conjunto de contraseñas emparejadas. Cuando se inicia una sesión, el sistema selecciona aleatoriamente una pareja de contraseñas y presenta una parte de la misma; el usuario debe suministrar la otra parte. En este sistema de contraseña de un solo uso, la contraseña es diferente en cada caso. Cualquiera que capture la contraseña de una sesión e intente reutilizarla en otra sesión no tendrá éxito. Las contraseñas de un solo uso constituyen un método para impedir las autenticaciones erróneas debidas a la exposición de la contraseña.

Contrafuegos para proteger los sistemas y redes

Para conectar una computadora de confianza a una red que no sea de confianza, podemos utilizar un contrafuegos. Un contrafuegos es una computadora, dispositivo o enrutador que se sitúa entre el sistema seguro y el que no lo es. Un contrafuegos de red limita el acceso a la red entre los dos dominios de seguridad y monitoriza y registra todas las conexiones. También puede limitar las conexiones basándose en la dirección de origen o de destino, el puerto de origen o de destino o la dirección de la conexión.

Un contrafuegos permite dividir la red en múltiples dominios. Una implementación habitual define varios dominios distintos: el dominio no seguro constituido por Internet, otro dominio semi-seguro, denominado zona desmilitarizada (DMZ, demilitarized zone); y un tercer dominio formado por las computadoras de la empresa. Las conexiones entre Internet y las computadoras DMZ, y entre las computadoras de la empresa e Internet se permite, pero no se permiten las comunicaciones entre Internet o las computadoras DMZ y las computadoras de la empresa.

Cabe aclarar que los contrafuegos no impiden los ataques de tipo túnel, es decir, los ataques contenidos dentro de los protocolos o conexiones que el contrafuegos permita. Por ejemplo, un contrafuegos no detendrá un ataque por desbordamiento de un búfer a un servidor web, ya que la conexión http está permitida y es el propio contenido de la conexión http lo que se utiliza como mecanismo de ataque. Del mismo modo, los ataques por DOS pueden afectar a los contrafuegos al igual que a otras máquinas. Otra vulnerabilidad de los contrafuegos es la suplantación.

Un contrafuegos persona es una capa de software incluida en el sistema operativo o que se añade como una aplicación. En lugar de limitar la comunicación entre dominios de seguridad, limita la comunicación a un determinado host. Por ejemplo, un usuario puede añadir un contrafuegos personal a su PC con el fin de denegar a un caballo de Troya el acceso a la red a la que esté conectado el PC. Un contrafuegos del tipo proxy de aplicación entiende los protocolos que utilizan las aplicaciones a lo largo de la red, como por ejemplo SMTP, que usa para la transferencia de correo electrónico. Los contrafuegos de llamadas al sistema se ubican entre las aplicaciones y el kernel, monitorizando la ejecución de las llamadas al sistema.

Clasificaciones de Seguridad

El departamento de defensa de los EE.UU especifica cuatro clasificaciones de seguridad para los sistemas: A, B, C y D.

La clasificación de nivel más bajo es la división D, o protección mínima. La división D incluye solo una clase y se usa para los sistemas que no cumplan los requisitos mínimos de ninguna de las otras clases de seguridad. Por ejemplo, MS-DOS y Windows 3.1 pertenecen a la división D.

La división C, el siguiente nivel de seguridad, proporciona un protección discrecional y mecanismos de atribución de responsabilidad a los usuarios y control de sus acciones, mediante el uso de capacidades de auditoria. La división C tiene dos niveles C1 y C2. Un entorno C1 es aquél en el que una serie de usuarios cooperantes acceden a un conjunto de datos con el mismo nivel de confidencialidad. La mayoría de las versiones de UNIX son de clase C1. Un sistema C2 añade un control de acceso de nivel individual a los requisitos de un sistema C1. Por ejemplo, los derechos de acceso de un archivo pueden especificarse en el nivel de un solo individuo. Además, el administrador del sistema puede auditar selectivamente las acciones de un solo individuo. Algunas versiones seguras especiales de UNIX han sido certificadas con el nivel C2.

Los sistemas de protección obligatoria de la división B tienen todas las propiedades de un sistema de clase C2 y además asignan un nivel de seguridad de cada objeto. Esta divida en B1, B2 y B3.

La clasificación de nivel superior es la división A. Estos sistemas usan técnicas formales de diseño y verificación para asegurar la seguridad.

Seguridad Multinivel

La seguridad multinivel, también conocida por las siglas MLS (del inglés Multiple Level of Security), es un tipo de política de seguridad que clasifica a los usuarios en distintos niveles de seguridad, permitiendo el acceso simultáneo a distintos usuarios con diferentes permisos y asegurándose de que cada usuario acceda a aquellos recursos que tiene autorizados y de la forma que tiene autorizada. Por tanto es una tecnología que permite proteger secretos y así evitar que algún usuario pueda acceder a cierta información cuyo acceso no tiene permitido.

Podemos extender el concepto de seguridad multinivel desde un computador individual a redes de computadores. A la aplicación de la seguridad multinivel en ese entorno se le llama sistemas inter-dominio o CDS (del inglés Cross Domain Systems). En ella hablamos de dominios en lugar de niveles y los datos se comparten en las redes de comunicaciones.

El modelo de seguridad Bell-Lapadula , consiste en dividir el permiso de acceso de los usuarios a la información en función de etiquetas de seguridad. Se basa en seguridad multinivel.

Este modelo se centra en la confidencialidad y no en la integridad por lo que también se lo conoce como modelo de confidencialidad. Se distinguen 2 tipos de entidades, sujetos y objetos. Se define estados seguros y se prueba que cualquier transición se hace de un estado seguro a otro. Un estado se define como estado seguro si el único modo de acceso permitido de un sujeto a un objeto está en concordancia con la política de seguridad. Para determinar si un modo de acceso específico está permitido, se compara la acreditación de un sujeto con la clasificación del objeto para determinar si el sujeto está autorizado para el modo de acceso especificado. El esquema de clasificación/acreditación se expresa en términos de un retículo. El modelo define 2 reglas de control de acceso mandatorio (MAC) y una regla de control de acceso discrecional (DAC) con 3 propiedades:

- Propiedad de seguridad simple: Un sujeto de un determinado nivel de seguridad no puede leer un objeto perteneciente a un nivel de seguridad más alto.
- Propiedad (Propiedad estrella): Un sujeto de un determinado nivel de seguridad no puede escribir un objeto perteneciente a un nivel de seguridad más bajo. (También llamada propiedad de confinamiento).

- Propiedad de seguridad discrecional: Se utiliza una matriz de acceso para especificar el control de acceso discrecional.

El método Biba se basa en seguridad multinivel y permite garantizar la integridad de los datos. Se basa en dos principios que son opuestos al método de Bell-La Padula.

- Write-Down. El proceso puede escribir solamente objetos en su nivel de seguridad o inferior.
- Read-Up. El proceso puede leer solamente objetos de su nivel de seguridad o más alto.

Canales Encubiertos

Un canal encubierto, es un canal que puede ser usado para transferir información desde un usuario de un sistema a otro, usando medios no destinados para este propósito por los desarrolladores del sistema. Para que la comunicación sea posible suele ser necesario un preacuerdo entre el emisor y el receptor codifique el mensaje de una forma que el receptor sea capaz de interpretar.

Canales Cubiertos

Hay un tipo especial de canales encubiertos a los que se les llama canales cubiertos (en inglés side channel) en los que el emisor de forma accidental comunica al receptor. En estos sistemas el desafío del receptor es descubrir y decodificar el canal oculto para obtener la información.

Esteganografía

Uno de los métodos más fáciles de implementar es el de inyección o agregado. Consiste esencialmente en agregar o adosar al final de un archivo, de cualquier tipo, otro archivo que será el contenedor del "mensaje a ocultar", también de cualquier tipo. Esta metodología es la más versátil, pues permite usar cualquier tipo de archivo como portador (documentos, imágenes, audio, vídeos, ejecutables, etc) y adosar el "paquete enviado", que es otro archivo, también de cualquier tipo.

Esta es una técnica que no se vale de las limitaciones humanas (vista y oído) para implementar la estrategia esteganográfica, sino que se vale de la forma de funcionamiento de la aplicaciones software que utilizan el portador. No degradan el contenido del portador de ninguna forma, por ejemplo, si es una imagen, permanecerá intacta; ya que el "mensaje" se le inyecta o adosa al final de la misma y la aplicación usada para visualizarla la mostrará normalmente hasta donde ella finalice. Esto es debido que todo tipo de archivo, en su cabecera, entre otros, contiene ciertos bytes fijos (en cantidad y ubicación) usados exclusivamente para indicar el tamaño del archivo. La aplicación que utilice un archivo, de cualquier tipo, siempre lee su cabecera primero, adquiere ese valor como su tamaño (en cantidad de bytes) y seguidamente lee el resto del archivo hasta el final indicado por dicho valor. De modo que si se coloca algo (mensaje) más allá del valor de ese parámetro, no será leído por la aplicación normal, por tanto no detectado, y el archivo portador funcionará normalmente.

Si bien es la técnica más sencilla de implementar, y de uso muy difundido, tiene la gran desventaja que provoca crecimiento del portador, tanto como el tamaño de su mensaje, siendo por tanto una estrategia fácilmente detectable. Otra desventaja, aunque muy relativa y eventual, es que el crecimiento del portador podría ser limitante a la hora de trasferirlo por las redes, particularmente por Internet.

Si no se requiere reunir requisitos de indetectabilidad es uno de los métodos preferidos, por su sencillez, flexibilidad y escasas limitaciones. Prácticamente cualquier tipo de portador es admitido, con o sin compresión, incluso módulos ejecutables. En algunos casos provoca corrupción del portador, lo cual no es gran problema: practicada la técnica e

inyectado el mensaje se prueba el portador con su aplicación correspondiente, si se ha degradado y/o no funciona bien, sencillamente toma otro, del mismo u otro tipo y se repite la operación.