Universidad de Guadalajara

Centro Universitario De Ciencias Económico Administrativas

División de Gestión Empresarial

Departamento de Sistemas de Información

Temas Selectos de Computación

Apuntes para el Curso de Sistemas Operativos 1

1.1 Sistemas Operativos a través de las generaciones de computadoras

Definición de Sistema Operativo (S.O.)

- Programa que controla la ejecución de los programas de aplicación y sirve de interfaz entre el usuario y el hardware de una computadora**.**

Los objetivos principales de un S.O. son:

− Aprovechar los recursos del sistema de manera eficiente

− Lograr una interfaz cómoda

− Crear un entorno donde el usuario pueda ejecutar programas

Historia y evolución de los sistemas operativos.

Primeras computadoras (1944):

− El programa se cargaba manualmente en la memoria a través de los

interruptores de un tablero

− Se utilizaban cintas de papel o tarjetas perforadas

− El programador supervisaba la ejecución con las luces del tablero

− La salida se imprimía o perforaba en cintas de papel o tarjetas

− Después había asignaciones de tiempo

Acceso por operador (monitor sencillo):

- Se nombra a un operador para que haga todo el trabajo
- El operador agrupaba los trabajos similares **(agrupación en lotes)**

y tenía habilidad para montar dispositivos y detectar errores

- Se crearon ensambladores, ligadores y cargadores y bibliotecas de funciones comunes
- Surgieron los manejadores de dispositivos (Device Drivers)
- Aparecieron los compiladores de Fortran, Cobol
- Había mucho desperdicio en tiempo de preparación

Monitor Residente:

− Programa que se encuentra residente en la memoria

− El monitor residente transfiere el control al programa y este al terminar, lo

devuelve al monitor

− Se utilizaron tarjetas de control con instrucciones especiales para indicar al

monitor cuál programa ejecutar

− Se utilizó el lenguaje **JCL (Job Control Language)** de IBM, utilizando el

símbolo $: **$JOB, $FORTRAN, $LOAD, $RUN, $END**

Operación fuera de línea:

− Es una solución a la lentitud de los dispositivos

− Hubo mejoras en los dispositivos pero también en las CPU

− Para eficientar las tareas de lectura de tarjetas e impresión estos trabajos se

hacían aparte dejando que la computadora únicamente se dedicara a leer y

escribir en cintas magnéticas

- ¿Qué ventajas trajo? Que la cinta magnética simulaba un solo tipo de registro para tarjetas perforadas, perforadoras, cintas de papel e impresoras

- Se logró la independencia del dispositivo **Buffer**
- Almacenamiento temporal para hacer simultánea la E/S de un trabajo con el propio sistema

− Mantienen ocupado al procesador

− Se manejan a través de interrupciones para saber si está lleno o no

− ¿Qué pasa si el dispositivo de E/S es muy rápido?

- ¿Qué soluciones hay? **DMA, Spoolers (Simultaneous Peripheral**

Operation On−Line)

- Con la aparición del disco magnético se dio solución al problema de las cintas; estas únicamente podían utilizarse después de haberse leído o escrito

Multiprogramación:

− Se ejecutan varios programas a la vez

− Planificación de trabajos (trabajos limitados por proceso y trabajos limitados

por E/S)

−Tiempo compartido

− Permiten la comunicación interactiva, a través de intérpretes de comandos

− Atiende a varios usuarios

− Tiempo real

− Respuestas inmediatas

−Se utilizan para procesos de control

−Sistemas distribuidos

− Compartir trabajos en varias computadoras a través de la red

− Se comparten recursos, se aceleran cálculos y son sistemas confiables

Sistemas multiproceso:

− Equipos con varios procesadores con la capacidad de trabajar con varios

procesos al mismo tiempo

Sistemas monousuarios:

- Computadoras personales

1.2 Modelos: jerárquico, capas, orientado a objetos

Estructura Monolítica

Es la estructura de los primeros sistemas operativos constituidos fundamentalmente por un solo programa compuesto por un conjunto de rutinas entrelazadas de tal forma que cada una puede llamar a otra.

Las características fundamentales de este tipo de estructuras son:

- Construcción del programa final basándose en módulos compilados separadamente que se unen a través del editor de enlace.
- Buena definición de parámetros de enlace entre las distintas rutinas existentes.
- Carecen de protecciones y privilegios.
- Generalmente están hechos a la medida, por lo que son eficientes y rápidos en su ejecución y gestión.

Modelo por capas

A medida que fueron creciendo las necesidades de los usuarios y se perfeccionaron los sistemas, se hizo necesaria una mayor organización del software.

Se dividió el sistema operativo en pequeñas partes, de tal forma que cada una de ellas estuviera perfectamente definida y con un claro interfase con el resto de elementos.

Se constituyó una estructura jerárquica o de niveles en los sistemas operativos, el primero en los cuales fue el denominado **THE (Technishe Hogeschool Eindhove)**

de Dijkstra, que se utilizó con fines didácticos.

En este sistema operativo pueden verse las distintas capas en su orden jerárquico:

− Hardware **(Nivel −1).**

− Planificación del procesador **(Nivel 0).**

− Gestión de la memoria **(Nivel 1).**

− Controlador de la consola del operador **(Nivel 2).**

− Control de las operaciones de entrada / salida **(Nivel 3).**

− Gestión de archivos **(Nivel 4).**

− Control de programas de usuario **(Nivel 5).**

En la estructura anterior se basan prácticamente la mayoría de los sistemas operativos actuales. Otra forma de ver este tipo de sistema es la denominada de anillos concéntricos o <rings>

En el sistema de rings, cada anillo tiene una apertura, conocida como **puerta o trampa (trap),** por donde pueden entrar las llamadas de las capas inferiores.

De esta forma, las zonas más internas del sistema operativo o **núcleo** del sistema estarán más protegidas de accesos indeseados desde las capas más externas. Las capas más internas serán, por tanto, más privilegiadas

que las externas.

Modelos jerárquicos:

Nivel

Objetos

Nombre

Ejemplos de Operaciones

13

Shell

Entorno de

programación del

usuario.

Sentencias de un lenguaje

de shell.

12

Directorios

Procesos de

usuario.

Salir, eliminar, suspender,

reanudar.

11

Procesos de

usuario

Directorios.

Crear, destruir, conectar,

desconectar, buscar, listar.

Nivel

Objetos

Nombre

Ejemplos de Operaciones

10

Dispositivos

Dispositivos

externos tales

como

impresoras,

pantallas y

teclados.

Crear, destruir, abrir,

cerrar, leer, escribir.

9

Sistema de

archivos

Archivos.

Crear, destruir, abrir,

cerrar, leer, escribir.

8

Comunicaciones

Tubos (pipes).

Crear, destruir, abrir,

cerrar, leer, escribir.

7

Memoria Virtual

Segmentos,

páginas .

Leer, escribir, traer (fetch).

6

Almacenamiento

secundario local

Bloques de

datos, canales de

dispositivos.

Leer, escribir, asignar,

liberar.

Nivel

Objetos

Nombre

Ejemplos de Operaciones

5

Procesos

Primitivos

Procesos

primitivos,

semáforos, colas

de procesos

listos.

Suspender, reanudar,

esperar, señalizar.

4

Interrupciones

Programas de

tratamiento de

interrupciones.

Invocar, enmascarar,

desenmascarar, reintentar.

3

Procedimientos

Procedimientos,

pila de llamadas,

visualización.

Marcar la pila, llamar,

retornar.

2

Conjunto de

instrucciones

Evaluación de la

pila, intérprete de

microprogramas,

vectores de datos

y escalares.

Cargar, almacenar, sumar,

restar, bifurcar.

1

Circuitos

electrónicos

Registros,

puertas, buses,

etc.

Borrar, transferir activar,

complementar.

Modelo orientado a objetos:

Ej. Windows NT

1.3 Interrupciones

Casi todas las computadoras tienen un mecanismo mediante el cual otros módulos (E/S, memoria) pueden interrumpir la ejecución normal del procesador. La tabla siguiente enumera las clases más comunes de interrupciones. Las interrupciones aparecen, principalmente, como vía para mejorar la eficiencia del procesamiento. Por ejemplo, la mayoría de los dispositivos externos son mucho más lentos que el procesador.

TABLA : Clases de Interrupciones

De programa

Generadas por alguna condición que se produce como resultado de la

ejecución de una instrucción, como el desbordamiento aritmético, la

división por cero, el intento de ejecutar una instrucción ilegal de la

máquina o una referencia a una zona de memoria fuera del espacio

permitido al usuario.

De reloj

Generadas por un reloj interno del procesador. Esto permite al sistema

operativo llevar a cabo ciertas funciones con determinada regularidad.

De E/S

Generadas por un controlador de E/S, para indicar que una operación

ha terminado normalmente o para indicar diversas condiciones de error

por fallo de hardware.

De error

Generadas por fallos tales como un corte de energía o un error de

paridad de la memoria.

Las interrupciones y el ciclo de instrucción.

Con las interrupciones, el procesador se puede dedicar a la ejecución de otras instrucciones mientras una operación de E/S está en proceso. Cuando el dispositivo E/S esté disponible, es decir, cuando esté preparado para aceptar más datos desde el procesador, el módulo de E/S de dicho dispositivo enviara una señal de solicitud de interrupción al procesador. El procesador responde suspendiendo la operación del programa en curso y saltando a un programa que da servicio al dispositivo de E/S en particular, conocido como rutina de tratamiento de la interrupción (interrupt handler), reanudando la ejecución original después de haber atendido al dispositivo.

Para dar cabida a las interrupciones, se añade un ciclo de interrupción al ciclo de instrucción. En el ciclo de interrupción, el procesador comprueba si ha ocurrido alguna interrupción, lo que se indicará con la presencia de una señal de interrupción. Si no hay interrupciones pendientes, el procesador sigue con el ciclo de lectura y trae la próxima instrucción del programa en curso. Si hay una interrupción pendiente, el procesador suspende la ejecución del programa en curso y ejecuta la rutina de tratamiento de la interrupción.

La rutina de tratamiento de la interrupción forma parte generalmente del sistema operativo. Normalmente este programa determina la naturaleza de la interrupción y realiza cuantas acciones sean necesarias.

1.4 Modo usuario y modo supervisor

Los primeros sistemas de computación eran sistemas monousuario controlados por el programador. Cuando este manejaba el computadora desde la consola, tenia un control completo del sistema. Sin embargo, al desarrollarse los sistemas operativos, este control se le otorgo al sistema operativo.

Empezando por el monitor residente, el sistema operativo comenzó a desempeñar varias de las funciones, sobre todo las de E/S, que antes eran responsabilidad del programador.

Además, para mejorar la utilización del sistema, el sistema operativo comenzó a compartir los recursos del sistema entre varios programas al mismo tiempo. Con la utilización de spoolers, un programa se podía ejecutar mientras se realizaban operaciones de E/S para otros trabajos; el disco almacenaba simultáneamente datos de varios trabajos.

Con la multiprogramación se colocaron en la memoria varios programas a la vez.

Este modo de compartir mejora la utilización pero también aumentaron los problemas. Cuando el sistema se ejecutaba sin compartir, un error en el programa solo podía causar problemas al programa que se ejecutaba. Con el compartir, un error en el programa podía afectar (de manera adversa) a varios trabajos.

Por ejemplo, considere el primer monitor residente, el cual solo proporcionaba una secuenciación automática de trabajos. Suponga que un programa se atasca en el ciclo al leer tarjetas de entrada. El programa leerá todos sus datos y, si nada lo detiene, continuara leyendo las tarjetas del siguiente trabajo, el que sigue a este, etc. Esto podía impedir el funcionamiento correcto de varios trabajos.

En un sistema multiprogramado pueden ocurrir errores mas sutiles, en los que un programa imperfecto podría modificar el programa o los datos de otro, e incluso al mismo monitor residente.

Sin protección frente a este tipo de errores, el computadora tendría que ejecutar en cada ocasión un solo trabajo o habría que sospechar de todas las salidas. Un sistema operativo correctamente diseñando debe asegurar que un programa incorrecto (o malintencionado) no provoque la ejecución incorrecta de otros

programas.

El hardware detecta muchos errores de programación, normalmente tratados por el sistema operativo. Si el programa de usuario falla de alguna manera (por ejemplo, al intentar ejecutar una instrucción no permitida, o tener acceso a un área de memoria que no corresponde al espacio de direcciones del usuario), el hardware dirigirá una trampa al sistema operativo. La trampa transfiere el control al sistema operativo a través del vector de interrupciones, igual que una interrupción.

Al ocurrir un error de programa, el sistema operativo debe terminarlo anormalmente. Esta situación la maneja el mismo código utilizado para una terminación anormal solicitada por el usuario. Se envía un mensaje de error

apropiado y se vuelca la memoria del programa. En un sistema por lotes, este volcado puede imprimirse de la memoria, lo que permite al usuario hallar la causa del error examinando la impresión. En un sistema interactivo, el volcado puede escribirse en un archivo; el usuario puede entonces imprimirlo o examinarlo directamente, y quizá corregir y reiniciar el programa.

Este método funciona bien siempre que el hardware detecte el error. Sin embargo, debemos asegurarnos que se detecten todos los errores y proteger al sistema operativo, así como a todos los demás programas y datos, de cualquier programa que no funcione correctamente. Se requiere protección para cualquier recurso compartido. La estrategia que se utiliza es proporcionar apoyo del hardware que permita distinguir entre diversos modos de ejecución.

Por lo menos necesitamos dos modos de operación distintos: modo usuario y modo monitor (también llamado modo de supervisión o modo de sistema). Se añade un bit, llamado bit de modo, al hardware del computadora para indicar el modo actual: monitor (0) o usuario (1). Con el bit de modo podemos distinguir entre una ejecución efectuada por el sistema operativo y una efectuada por el

usuario.

Al ocurrir una trampa o interrupción, el hardware cambia de modo usuario a modo monitor (es decir, cambia a 0 el estado del bit de modo). Así, cada vez que el sistema operativo obtiene el control del computador, se encuentra en modo monitor. El sistema siempre cambia a modo usuario (fijando a 1 el bit de modo) antes de pasar el control a un programa de usuario.

El modo dual de operación nos proporciona un medio para proteger al sistema operativo de los usuarios erráticos, y a estos de ellos mismos. Logramos esta protección clasificando algunas de las instrucciones de la maquina que pueden causar daño como instrucciones privilegiadas. A las cuales el hardware permite ejecutarse solo en modo monitor. Si se intenta ejecutar una instrucción privilegiada

en modo usuario, el hardware no lo hace, sino que la considera como una instrucción incorrecta y dirige una trampa al sistema operativo.

Si se carece de un modo dual apoyado por el hardware, se puede provocar serios defectos en un sistema operativo. Por ejemplo, MS−DOS se escribió para la arquitectura Intel 8088, que no tiene bit de modo y, por tanto, tampoco posee modo dual. Un programa que pierda el control puede destruir el sistema operativo escribiendo datos encima de el, y varios programas pueden escribir en un dispositivo al mismo tiempo, con resultados posiblemente desastrosos.

1.5 Máquina virtual

Es una ilusión de máquina real. La crea un S.O. de máquina virtual.

Ejemplos:

− Java

− Inferno

− Máquina virtual VM de IBM

Inferno

Inferno es un sistema operativo para creación de servicios distribuidos. Es un producto de la Unidad de Negocios Inferno de Lucent Technologies y fue desarrollado por ellos y por el Centro de Investigación de Ciencias de la Computación de los Laboratorios Bell, el brazo derecho de Lucent Technologies.

Fue lanzado comercialmente en Marzo de 1997.Aunque si bien Inferno es un nuevo sistema desarrollado específicamente como un producto comercial este condujo por muchos años la investigación de los Laboratorios Bell en sistemas

operativos, lenguajes, compiladores, gráficos, seguridad y trabajo en red.

Seguridad en Inferno

Toda la seguridad es opcional: una aplicación puede usarse o evitarse. Inferno provee un fuerte mecanismo de autentificación, mensajes encriptados y firmas digitales.

La autenticación y las firmas digitales son desarrolladas mediante el uso de claves criptográficas públicas (el algoritmo de cifrado o clave se hace público no el de descifrado), el algoritmo de descifrado se hace extraordinariamente difícil deducirlo a partir del algoritmo de cifrado − al ponerse en un archivo que cualquiera que lo quisiera pudiese leerlo. Así cuando dos interlocutores A y B se quieran comunicar. A recibe el mensaje y lo encripta con el mecanismo de cifrado de B lo manda y B con su mecanismo de descifrado secreto lo descifra, nadie más podría leer el mensaje que A encripta con el mecanismo de B.

La autentificación con claves públicas consiste en que el usuario debe de desarrollar un mecanismo con el que cifre y otro de descifrado. Al tratar de ingresar al sistema este le proporciona al usuario un número o frase aleatoria con el mecanismo de cifrado que el usuario proporciona, y el usuario debe devolver la frase o número ya descifrado. Ya que solo el usuario tiene el algoritmo de descifrado sólo el pasa la prueba.

Firmas digitales.

Suponiendo que A y B desean hablarse el mecanismo de cifrado es E y el de descifrado es D. Primero A aplica el algoritmo de descifrado propio al mensaje Da(M) después lo cifra con el mecanismo de cifrado de B Eb(Da(M)) lo envía a B que aplica el algoritmo de descifrado suyo haciendo que sólo quede Da(M) que al aplicar el mecanismo de cifrado público de A queda solucionado.

Esto implica que la criptografía de clave publica para transmitir mensajes firmados tengan la propiedad :

E(D(M))= M, además de la propiedad normal D(E(M))= M.

Modalidades

Inferno provee dos soluciones una fija el Inferno nativo (para sistemas operativos fijos) y una solución de red (Hosted Inferno). En su implementación nativa funciona como un sistema operativo primario. Inferno en su

ambiente de servidor funciona como una aplicación bajo un sistema operativo nativo como Windows NT, Win95,

y Solaris, Silicon Graphics IRIX, DEC Alpha, HP−UX, y provee un ambiente sencillo para el desarrollo de soluciones de red.

Provee alto grado de conectividad ya sea por medio de la red pública de teléfono, Internet, redes empresariales, cable de televisión y por satélite. Inferno provee todos los servicios de un sistema operativo funcional incluidos el manejo de memoria, capacidades gráficas, manejo de procesos.

Inferno también provee un alto grado de programabilidad. El lenguaje de programación Limbo en el cual fue creado va incluido con el sistema operativo para crear aplicaciones propias.

Inferno es una solución compacta. El diseño del sistema permite cargar el Kernel y sus aplicaciones por separado. Al correr sólo se carga el Kernel y va cargando las aplicaciones como las va necesitando, así se ahorra espacio considerable en memoria que permite correr otras aplicaciones, y reducir el código de la interfaz.

Inferno corre en pequeñas computadoras que usan procesadores como Hitachi SH3, DEC Strong Arm, Intel Architecture (la familia de procesadores x86), MIPS, Motorola 68030, PPC, SPARC y ARM.

Con el sistema operativo se incluye:

− Aplicaciones Básicas de Inferno

− Ejemplo de código en Limbo

− Herramientas para desarrollo de Software

− Compilador de Limbo

− Debbuger

− Editor de Texto Brutus

− Administrador de tareas

− Documentación.

− Guía del usuario. Presenta información necesaria para presentar el S.O.

- Manual de referencia. Proporciona información necesaria para desarrollar y

crear aplicaciones escritas en Limbo.

− Guía para el programador: Guía general para programar. Proporciona

información como crear y correr programas de Limbo en Inferno.

Lenguajes que soporta: limbo, Java, C y C++.

La versión 2.0 SDK trae integrado un ED para Windows, un editor con constructor para desarrollar programas

Limbo. Las características del editor son:

− Poder de edición.

− Fácil de usar.

− Búsqueda y remplazamiento de frases, palabras o bloques de texto.

− Compilación dentro del editor.

− Resalta palabras de sintaxis.

− Impresión con fuentes.

− Ayuda a preguntas frecuentes.

− Desplegado de múltiples ventanas.

− Funciones de Cut/copy/paste a través de las ventanas.

− Lista de búsqueda

Preguntas frecuentes de Inferno y sus soluciones

Máquina virtual VM de IBM:

La máquina virtual soporta generalmente otros S.O. como DOS, MVS, PC DOS, VM/370, Opción de tiempo compartido de MVS, AIX, CMS.

Componentes principales:

1. El programa de control (CP) Crea el ambiente para que se ejecuten las

maquina virtuales. Proporciona el apoyo para diversos S.O.

2. El subsistema supervisor de conversación (CMS) Es un sistema de

aplicaciones con potentes características para el desarrollo de programas.

Contiene editores, traductores de lenguajes, paquetes de aplicaciones y

depuradores.

3. El subsistema spool remoto y de comunicaciones (RSCS) Se encarga de

transmitir y recibir archivos en un sistema distribuido.

4. El sistema interactivo para control de problemas (IPCS) Se utiliza para el

análisis en línea y corregir problemas de software de VM

5. El recurso CMS por lotes El usuario puede introducir trabajos largos para

procesarse por lotes mientras continua su trabajo interactivo con una

maquina virtual CMS

Ventajas:

− Facilita la migración de diferentes versiones de S.O.

− Sirve para capacitar

− El desarrollo del S.O. se lleva a cabo junto con la producción

− Se ejecutan diversos S.O.

− Como respaldo a fallas

2.1 Procesos (definición y estados)

Un proceso es una entidad dinámica que nace cuando se carga un programa en memoria y muere al finalizar la ejecución del programa cargado en memoria.

El término proceso fue utilizado, en principio, por los diseñadores del sistema Multics en la década de 1960. Desde entonces, el proceso, en alguna medida intercambiado con tarea, se le han dado muchas definiciones.

Las que siguen son algunas de ellas:

un programa que se está ejecutando

una actividad asincrónica

el "espíritu animado" del procedimiento

el "emplazamiento del control" de un procedimiento que está siendo ejecutado

aquello que se manifiesta por la existencia en el sistema operativo de un

"bloque de control de proceso"

aquella entidad a la cual son asignados los procesadores

la unidad "despachable".

Se le han dado muchas otras definiciones. Aunque no hay un acuerdo universal sobre su definición, parece hacerse referencia más frecuentemente al concepto de programa que se está ejecutando.

Estados de un proceso

Durante su existencia, un proceso pasa por una serie de estados discretos. Varias circunstancias pueden hacer que un proceso cambie de estado. Se dice que un programa se está **ejecutando** (es decir, está en estado de ejecución), si tiene en ese momento el CPU. Un proceso se dice que está **listo** (es decir, en estado de listo), cuando podría usar un CPU, si hubiera uno disponible. Se dice que un

proceso está **bloqueado** (es decir, en estado bloqueado), si espera que ocurra algo (como la terminación de una entrada / salida) para poder ponerse en marcha.

Por simplicidad consideraremos un solo CPU, aun cuando hacer la extensión hacia el multiprocesamiento no es difícil. Sólo puede estar corriendo un proceso al mismo tiempo, pero puede haber varios procesos listos y varios más bloqueados. Por tanto, se establece una lista de listos para los procesos listos, y una lista de

bloqueados, para los bloqueados. La lista de listos se mantiene en orden prioritario, para que el siguiente proceso que reciba el CPU sea el primer proceso de la lista. La lista de bloqueados no está ordenada, los procesos no se desbloquean (o sea, no quedan listos) en orden prioritario; en lugar de esto, los procesos se desbloquean en el orden en que tienen lugar los eventos que están esperando.

Transiciones entre los estados del proceso

Cuando un trabajo es admitido en el sistema, se crea un proceso equivalente, y es insertado en la última parte de la lista de listos. El proceso se va moviendo gradualmente hacia la cabeza de esta relación a medida que se completan los procesos anteriores. Cuando el proceso llega a la cabeza de la lista, y cuando el CPU se encuentre disponible, el proceso recibe el CPU y se dice que hace una

transición de estado, del estado listo al estado de ejecución . La asignación del CPU al primer proceso de la lista de listos es llamado despacho, y es ejecutado por la entidad del sistema llamado despachador. Mientras el proceso tenga el CPU, se dice que está en ejecución.

Para prevenir que cualquier proceso monopolice el sistema, ya sea de manera accidental o maliciosamente, el sistema operativo ajusta un reloj de interrupción del hardware para permitir al usuario ejecutar su proceso durante un intervalo de tiempo específico o quantum. Si el proceso no abandona voluntariamente el

CPU antes de que expire el intervalo, el reloj genera una interrupción, haciendo que el sistema operativo recupere el control. El sistema operativo, entonces, hace que el proceso que anteriormente se hallaba en estado de ejecución pase al de listo, y hace que el primer proceso de la lista de listos pase al estado de ejecución.

Si un proceso que se encuentra en estado de ejecución inicia una

operación de entrada/salida antes de que termine su quantum, el proceso voluntariamente abandona el CPU (es decir, el proceso se bloquea a sí mismo hasta la terminación de la operación de entrada / salida). La única otra transición posible en este modelo de tres estados ocurre cuando termina una operación de entrada / salida (o alguna otra causa por la que esté esperando el proceso). El proceso cambia del estado bloqueado al estado listo.

El estado de un proceso forma parte del estado global del sistema. El SO mantiene listas de **bloques de control de procesos (BCP)** clasificados por el estado actual de cada proceso. Así tendremos:

− Lista de procesos listos (o preparados)

− Lista de procesos bloqueados (o suspendidos)

− Lista de procesos en ejecución.

Vale la pena mencionar que cuando se trata de sistemas multiprocesadores puede haber una Lista de procesos en ejecución global y cuando son monoprocesadores solo habrá una sola entrada. Es decir un solo proceso en ejecución a la vez.

Transiciones de estado de un proceso.

Los SO multitarea están guiados en gran medida por sucesos, en el sentido de que hay operaciones de gestión en respuesta a sucesos del sistema que pueden provocar cambios de estado, y llevar a la reasignación de recursos.

SUCESOS EXTERNOS

SUCESOS INTERNOS

Ocurrencia asíncrona

Ocurrencia síncrona

Por medio de interrupciones

Llamadas al SO por el proceso

Sin mayores problemas

Las interrupciones deben ser

dirigidas al Sistema Operativo

Cada que un proceso cambia de estado, el SO coloca el BCP del proceso en la lista que corresponde a su nuevo estado.

Conmutación de procesos

Se llama conmutación de proceso o conmutación de tarea a la transición entre dos procesos residentes en memoria. Las conmutaciones de proceso casi siempre son el resultado de un suceso.

Los SO de multiprogramación se ejecutan normalmente en modo protegido y en espacio de direcciones aparte del de usuario, pero la transición desde el proceso de usuario al SO necesita cruzar la frontera de protección. A esta operación se le conoce como conmutación de modo.

En seguida se guarda el estado del proceso en ejecución que esta por quedar bloqueado o listo. Debe guardarse:

− Contador del programa

− Apuntador de la pila

− Palabra de estado de procesador

− Los registros restantes (accesibles)

− El estado hardware completo

Y dependiendo de la causa que lo saco de ejecución, su BCP se lleva a la lista de preparados o de suspendidos. Es en este momento en el que se atiende al suceso que provocó el cambio de estado, y casi nunca se le considera parte de la conmutación del proceso.

Luego el SO invoca al planificador para que seleccione a otro proceso para ejecución. Una vez hecho lo anterior, su BCP se usa para preparar el entorno de ejecución, además de que se restaura el estado hardware:

− Cálculo de máscara de interrupción asociada a su prioridad

− Establecimiento de punteros para activar archivos y lista de recursos

− Actualización de registros de gestión de memoria

− Otras actividades

Finalmente, el SO inicia una conmutación de modo en el espacio del usuario y cede el control al proceso recién planificado.

La conmutación de proceso es una operación considerablemente compleja y dada esa complejidad y su relativa alta frecuencia de ocurrencia, la implementación de la conmutación de proceso puede afectar significativamente al rendimiento de un sistema operativo de multiprogramación.

2.2 Descriptor de procesos

El **Bloque de Control del Proceso (PBC)** es la parte que define a un proceso en el Sistema Operativo.

Cada proceso cuenta con su PBC y los campos más importantes de éste son:

Identificador del proceso

Identificador del padre

Apuntadores de la pila

Apuntadores a la memoria

Apuntadores a los recursos que utiliza

Palabra de estado de procesador

Los registros restantes (accesibles) utilizados por el proceso

El estado del proceso

El PCB es un almacenamiento central de información que permite al sistema operativo localizar toda la información clave sobre el proceso. Cuando el sistema operativo cambia la atención del CPU entre los procesos, utiliza las áreas de preservación del PCB para mantener la información que necesita para reiniciar el proceso cuando consiga de nuevo el CPU.

Un proceso se presenta, desde el punto de vista del sistema operativo, por un conjunto de datos donde se incluyen el estado en cada momento, recursos utilizados, registros, etc., denominado Bloque de Control del Proceso (PBC).

Los objetivos que se pretenden cubrir con el bloque de control del proceso son los siguientes:

− Localización de la información sobre el proceso por parte del sistema operativo.

− Mantener registrados los datos del proceso en caso de tener que suspender

temporalmente su ejecución o reanudarla.

En general, la información contenida en el bloque de control es la siguiente:

· Estado del proceso. Información relativa al contenido del contador del programa

(Program Counter, PC), estado del procesador en cuanto a prioridad del proceso, modo de ejecución, etc., y por último el estado de los registros internos de la computadora.

· Estadísticas de tiempo y ocupación de recursos para la gestión de la planificación

del procesador.

· Ocupación de la memoria interna y externa para el intercambio (swapping).

· Recursos en uso (normalmente unidades de entrada / salida).

· Archivos en uso.

· Privilegios.

Estas informaciones se encuentran en memoria principal o en disco y se accede a ellas en los momentos en que se hace necesaria su actualización o consulta. Los datos relativos al estado del proceso siempre se encuentran en memoria principal.

De igual forma existe un **Bloque de Control del Sistema (SCB),** con unos objetivos globales similares al anterior y entre los que se encuentra el enlazado de los bloques de control de los procesos existentes en el sistema.

Trataremos de ver a continuación cómo se realiza el cambio de un proceso a otro, para lo cual supondremos que estamos en una computadora con un solo procesador (sólo un proceso puede estar ejecutándose en cada momento), y existen varios procesos activos compitiendo por el acceso al procesador (se está ejecutando un proceso B). Suponemos que los programas de los procesos A y B están ambos en memoria principal.

Las acciones que realiza el sistema operativo para cambiar el proceso A por el B se denominan cambio de proceso, y son los siguientes:

· Deja de ejecutar el proceso en curso (A), cediéndose el control al núcleo del sistema operativo, y aparece lo que se denomina un cambio de contexto pasando del modo usuario al modo supervisor. Antes de realizarse el cambio de contexto se salva el estado del proceso A para su posterior vuelta al punto donde fue interrumpido.

· El núcleo estudia si el proceso B está preparado para su ejecución y, si es así, realiza el cambio de contexto correspondiente pasando del modo supervisor al modo usuario. A continuación repone el estado del proceso B (en caso de haber sido interrumpido con anterioridad) y, por último pone en ejecución el proceso B.

El cambio de contexto se producirá en caso de ejecución de una instrucción privilegiada, una llamada al sistema operativo o una interrupción, es decir, siempre que se requiera la atención de algún servicio del sistema operativo.

En el cambio de contexto, el núcleo salva el estado de proceso que se estaba ejecutando en su bloque de

control y restaura el proceso que va a ejecutarse

2.3 Cambio de proceso y cambio de contexto

Cambio de proceso

En una interrupción ordinaria, el control se transfiere primero a un gestor de interrupciones, quien lleva a cabo algunas tares básicas y, después, se salta a una rutina del sistema operativo que se ocupa del tipo de interrupción que se ha producido. Se puede hacer un cambio de proceso o, simplemente, reanudar el mismo proceso que se estaba ejecutando.

Cambio de contexto

Si hay alguna interrupción pendiente, el procesador hace lo siguiente:

1. Salva el contexto del programa que está ejecutándose.

2. Asigna al contador de programa el valor de la dirección de comienzo de un programa de tratamiento de la interrupción

El procesador continúa entonces con el ciclo de lectura de instrucción y trae la primera instrucción del programa de tratamiento de interrupciones, que atenderá a la interrupción. Así pues, debe salvarse la parte del bloque de control del proceso que se denomina información de estado del procesador. Esto incluye el contador de programa, otros registros del procesador y la información de la pila.

2.4 Algoritmos de Planificación

Niveles de planificación

a) Planificación de alto nivel. Determina cuáles trabajos podrán competir activamente por los recursos del sistema o cuáles deben admitirse en el sistema

b) Planificación de nivel intermedio. Determina qué procesos pueden competir por la UCP. Responde a las fluctuaciones temporales en la carga del sistema mediante la suspensión temporal y reanudación.

c) Planificación de bajo nivel. Determina cuál proceso listo se le asignará la UCP cuando ésta se encuentre disponible. El despachador asigna la UCP a un proceso.

Objetivos de la planificación

− Ser justa

− Elevar al máximo el rendimiento

− Aumentar al máximo el número de usuarios interactivos con respuesta en

tiempos aceptables

− Ser predecible: tareas al mismo tiempo y casi al mismo costo

− Reducir al mínimo el gasto extra

− Equilibrar el aprovechamiento del sistema

− Equilibrar la respuesta y el aprovechamiento

− Evitar el aplazamiento indefinido

− Imponer prioridades

− Dar preferencia a procesos que ocupan recursos decisivos

− Dar mejor trato a los procesos que muestren comportamientos deseables

− Degradarse paulatinamente con las cargas pesadas

Criterios de la planificación

− Limitación de un proceso por E/S

− Limitación de un proceso por CPU

− Atención a procesos por lotes o interactivos

− Cuán urgente es la respuesta

− Prioridades de un proceso

− Frecuencia con la que un proceso genera fallas de páginas

− Frecuencia con la que los recursos de un proceso son apropiados por otro de

mayor prioridad

− Cuánto tiempo real de ejecución ha recibido un proceso

− Cuánto tiempo más necesitará para terminar

Planificación apropiativa y no apropiativa

− No apropiativa: Después que la CPU ha sido asignada al proceso ya no se le puede arrebatar

− Apropiativa: Cuando se puede arrebatar la CPU al proceso por medio del Cronómetro de intervalos o reloj de interrupciones. Se utiliza para evitar que los usuarios monopolicen el sistema. El SO genera interrupciones con base a un reloj. Si el usuario se encuentra en ejecución y es interrumpido por el reloj, el sistema operativo entra en ejecución y decide a qué proceso se asignará la CPU.

Prioridades

Deben ser asignadas en forma automática por el sistema o asignarse externamente. Pueden ganarse o comprarse. Pueden ser estáticas o dinámicas. Pueden asignarse en forma racional o en forma arbitraria.

− Estáticas. No cambian.

− Dinámicas. Responden a los cambios. La prioridad inicial tiene una duración

corta, después de la cual se le asigna un valor más apropiado.

− Compradas. Se utilizan para cuando los usuarios requieren de mayores

prioridades en sus proceso.

Planificación a plazo fijo

Se programan ciertos trabajos para terminarse en un tiempo específico o plazo fijo.

− El usuario debe informar por adelantado sus necesidades de recursos

− El sistema debe ejecutar la tarea en plazo fijo sin degradar el sistema

− El sistema planifica los trabajos de manera que es muy difícil aceptar nuevos

− Si hay muchos trabajos de plazo fijo al mismo tiempo se necesitan niveles de

optimización muy grandes del sistema

Planificación de primeras entradas−primeras salidas (PEPS)

Es la más sencilla. Los procesos se despachan de acuerdo con su tiempo de llegada a la cola de procesos listos. Cuando un proceso tiene la CPU se ejecuta hasta terminar. Es no apropiativa. No es útil para la planificación de usuarios interactivos. Ya no se usa, aunque a veces se usa combinado con otros métodos de planificación: agrupación de procesos con la misma prioridad y con una cola PEPS.

Planificación por turno (RR)

Los procesos se despachan en forma de PEPS, pero se les asigna una cantidad limitada de tiempo de CPU. El proceso que se le acabó su quantum pasa la cola para volver a ser ejecutado.

Tamaño del cuanto (Analizar cuando el cuanto es muy grande y muy pequeño)

Planificación por prioridad del trabajo más corto (SJF)

No apropiativo. Se ejecuta primero el proceso en espera que tiene el menor tiempo estimado de ejecución hasta terminar.

Planificación por el tiempo restante más corto (SRT)

Es la contraparte apropiativa de SJF. En SJF un proceso se atiende y se ejecuta hasta que termine. En SRT a un proceso se le puede arrebatar la CPU si entra otro trabajo con menor tiempo de ejecución.

Planificación por prioridad de la tasa de respuesta más alta (HRN)

Corrige las deficiencias de SJF, del retraso excesivo de trabajos largos y favoritismo para los cortos. No apropiativa, en donde no solo están en función del tiempo de servicio sino también del tiempo de espera para ser atendidos.

Prioridad = (tiempo de espera + tiempo de servicio)/tiempo de servicio

Colas de retroalimentación en múltiples niveles

Los planificadores deben:

− Favorecer los trabajos cortos

− Favorecer los trabajos limitados por E/S para lograr un mejor aprovechamiento de los recursos

− Determinar la naturaleza de un trabajo lo más pronto posible y planificarlo de acuerdo a su naturaleza Las colas de retroalimentación cubren estos objetivos.

2.5 Procesos e hilos

La eficiencia de la conmutación de proceso puede ser mejorada con la ayuda de asistencias hardware y una técnica software de estructuración de procesos especial denominada hebras o hilos.

El esquema hardware comúnmente usado es contar con varios conjuntos estructuralmente idénticos de registros del procesador. El mínimo son dos, uno para el SO y otro para los procesos de usuario.

Hebras de ejecución.

Una hebra es un proceso ligero con un estado reducido. Esto se logra compartiendo recursos como memoria y archivos. Aquí el proceso sirve como entorno para la ejecución de las hebras.

Cada hebra pertenece solo a un proceso y ninguna hebra puede existir sin su proceso. Cada hebra representa un flujo separado de control y está caracterizado por su propia pila y estado hardware. La conmutación entre hebras relacionadas (de un mismo proceso) es rápida y eficiente. Pero la conmutación entre hebras de distintos procesos implica la conmutación de proceso completa, con el consiguiente retardo.

Hebras

relacionadas

(formas de

comunicación y

sincronización)

Memoria compartida accesible dentro del proceso

que las engloba.

Señales, con memoria compartida y su interacción

no requiere cruzar la frontera de protección de

memoria.

El compromiso de las hebras entre las dos filosofías de

implementación de los Sistemas Operativos:

Los tradicionales y pesados

procesos con protección que

afectan al rendimiento en tiempo real

(UNIX).

Los SO de tiempo real ligeros y

rápidos que sacrifican protección

por rendimiento.

Las hebras se han utilizado con éxito en:

En servidores de red.

Ejecución paralela con multiprocesadores de

memoria compartida.

Las hebras son un desarrollo relativamente reciente en los SO. Pueden encontrarse en los S.O. Mach, Unix, NT y OS2.

3.1 Ejecución concurrente

Se dice que los procesos son concurrentes si existen al mismo tiempo. Los procesos concurrentes pueden funcionar con total independencia unos de otros, o pueden ser asincrónicos, lo cual significa que requieren una sincronización y cooperación ocasionales.

El asincronismo es un tópico complejo; en este trabajo se exponen la organización y administración de sistemas que soportan procesos concurrentes asincrónicos. Se presentan muchos problemas importantes de asincronismo. Sus soluciones se presentan como programas concurrentes codificados utilizando el lenguaje Pascal concurrente desarrollado inicialmente por Wirth (Pascal−S), extendido por Ben Ari y modificado en la Universidad de Bradford (UK) por G.L. Davies (Pascal−FC).

Procesamiento en Paralelo

A medida que disminuyen tanto el tamaño como el precio del hardware de las computadoras se irá produciendo una tendencia hacia el multiprocesamiento y la masificación del paralelismo. Si ciertas operaciones pueden ser ejecutadas en paralelo de forma lógica, entonces las computadoras con múltiples procesadores las ejecutarán físicamente en paralelo, aunque en el nivel de paralelismo se den miles o, tal vez, millones de actividades concurrentes.

El procesamiento en paralelo es interesante por varias razones. La gente parece más capaz de centrar su atención en una sola actividad a la vez que de pensar en paralelo. (Intente leer dos libros al mismo tiempo, leyendo una línea de un libro, una línea del otro, la segunda línea del primero, y así sucesivamente.) Es difícil determinar cuáles actividades pueden ejecutarse o no en paralelo. Los programas en paralelo son mucho más difíciles de depurar que los programas secuenciales; después de haber arreglado supuestamente un error,

puede resultar imposible reconstruir la secuencia de eventos que han ocasionado el error. Por ello, sería en cierto modo inapropiado asegurar que se ha corregido el error. Los procesos asincrónicos deben interactuar ocasionalmente entre sí, y esas interacciones pueden ser complejas. Trataremos varios ejemplos de interacción de procesos en este

trabajo.

Estructura para indicar el paralelismo: ParBegin/ParEnd (CoBegin/CoEnd)

Son muchas las construcciones de lenguajes de programación para indicar paralelismo que han aparecido en la literatura. Estas implican pares de proposiciones como las siguientes:

Una proposición indicando que la ejecución secuencial debe ser dividida entre varias secuencias de ejecución en paralelo (trayectoria de control).

Una proposición indicando que ciertas secuencias de ejecución en paralelo están a punto de producirse y se reanudará la ejecución secuencial.

Estas proposiciones ocurren en pares y suelen denominarse parbegin/parend (para comenzar y finalizar una ejecución en paralelo), o cobegin/coend (para comenzar o finalizar una ejecución concurrente). La forma general es para utilizar estas proposiciones son:

cobegin

proposición 1;

proposición 2;

:

.

proposición n;

coend

Supóngase que un programa está ejecutando una sola secuencia de instrucciones cuando se encuentra con la construcción cobegin anterior. Esto determina que la trayectoria de control simple se divida en n trayectorias separadas de control, una por cada proposición de la construcción cobegin/coend. Estas pueden ser proposiciones simples, llamadas a procedimientos, bloques de proposiciones

secuenciales delineados por begin/end, o combinaciones de éstos. Cada una de las trayectorias de control individuales acaba por alcanzar y terminar al coend. Cuando todas las trayectorias de control paralelas llegan al final, se reanuda una trayectoria de control simple y el sistema prosigue más allá del coend.

Como ejemplo, considérese el siguiente cálculo de una raíz de la ecuación cuadrática:

x := (−b + (b**2 − 4*a*c) ** 0.5) / (2*a)

Esta asignación puede ser evaluada en un procesador secuencial (poseedor de una instrucción de exponenciación) de la siguiente manera:

1 b**2

2 4*a

3 (4*a)*c

4 (b**2) − (4*a*c)

5 (b**2 − 4*a*c) ** 0.5

6 − b

7 (−b) + ((b**2 − 4*a*c)**0.5)

8 2*a

9 (−b+(b**2−4*a*c)**0.5)/(2*a)

Aquí se ejecutan de una en una cada una de las nueve operaciones en una secuencia determinada por las reglas de un sistema de precedencia de operadores. En un sistema que soporte procesamientos en paralelo, la expresión puede evaluarse de la manera siguiente:

1 cobegin

t1:= −b;

t2:= b ** 2;

t3:= 4 * a;

t4:= 2 * a;

coend

2 t5:= t3 * c;

3 t5:= t2 − t5;

4 t5:= t5 ** 0.5;

5 t5:= t1 + t5;

6 x:= t5 / t4;

Aquí se evalúan en paralelo las cuatro operaciones contenidas dentro de la construcción cobegin/coend; las cinco operaciones restantes deben ser ejecutadas de forma secuencial. Al ejecutar los cálculos en paralelo se reduce en gran medida el tiempo real de ejecución.

3.2 Exclusión Mutua

Considérese un sistema con varias terminales de tiempo compartido. Supóngase que los usuarios acaban cada línea, que teclean al sistema con un retorno de carro. Supóngase que se desea supervisar continuamente el

número total de líneas que los usuarios han introducido al sistema desde el comienzo del día, y que cada terminal está supervisada por un proceso diferente. Cada vez que uno de estos

procesos recibe una línea de la terminal de un usuario incrementa en 1 la variable global compartida NOLINEAS. Considérese lo que pasaría si dos procesos intentaran incrementar NOLINEAS simultáneamente.

Supóngase que cada proceso tiene su propia copia del código:

LOAD NOLINEAS

ADD 1

STORE NOLINEAS

Suponga que NOLINEAS tiene un valor de 2345. Suponga ahora que el primer proceso ejecuta las instrucciones LOAD y ADD, dejando así el valor de 2346 en un acumulador. Entonces el proceso pierde el procesador (debido a la terminación de un quantum) en beneficio del segundo proceso. Este último ejecuta ahora las tres instrucciones, ajustando así NOLINEAS con el valor 2346. Pierde el procesador y lo obtiene de nuevo el primer proceso, el cual continua con la ejecución de la instrucción STORE, colocando también el valor 2346 en NOLINEAS. Debido al acceso incontrolado a la variable compartida NOLINEAS el sistema ha perdido la pista de una de las líneas, el total correcto debería ser 2347.

A continuación, se muestra el programa pro00 que simula este concepto así como diferentes corridas del mismo. El proceso uno mete 60 líneas mientras que el proceso dos mete 40. Por supuesto, la suma total de líneas debería ser 100, pero observe lo que pasa.

− Pascal−FC for IBM PC compatibles −

− Compiler Versión P5.2

G L Davies & A Burns, University of Bradford

Compiler listing

1 0

2 0 program pro00;

3 0

4 0 var

5 0 nolineas: integer;

6 0

7 0 (* El proceso uno contará 60 lineas *)

8 0 process uno;

9 0 var

```
10 0 lin: integer;
11 0
12 0 begin
13 0 for lin := 1 to 60 do
14 4 begin
15 4 nolineas := nolineas + 1
16 7 end
17 9 end; (* uno *)
18 11
19 11 (* El proceso dos contará 40 líneas *)
20 11 process dos;
21 11 var
22 11 lin: integer;
23 11
24 11 begin
25 11 for lin := 1 to 40 do
26 15 begin
27 15 nolineas := nolineas + 1
28 18 end
29 20 end; (* dos *)
30 22
31 22 (* Programa principal *)
32 22 begin
33 22 nolineas := 0;
34 25 cobegin
35 26 uno;
```

```
36 30 dos

37 30 coend;

38 35 writeln('Total de líneas =',nolineas)

39 39 end.

− Interpreter Version P5.3 −

Program pro00 ... execution begins ...

Total de líneas = 57

Program terminated normally

Type r and RETURN to rerun

Program pro00 ... execution begins ...

Total de líneas = 74

Program terminated normally

Type r and RETURN to rerun

Program pro00 ... execution begins ...

Total de líneas = 78

Program terminated normally

Type r and RETURN to rerun

Program pro00 ... execution begins ...

Total de líneas = 76

Program terminated normally

Type r and RETURN to rerun

Program pro00 ... execution begins ...

Total de líneas = 69

Program terminated normally
```

Este problema puede solucionarse dándole a cada proceso acceso exclusivo a NOLINEAS. Mientras un proceso incrementa la variable compartida, los demás procesos que deseen hacer lo mismo al mismo tiempo deben permanecer a la espera; cuando ese proceso termine de accesar la variable deseada, le será permitido proceder a uno de los procesos. De esta manera, cada proceso que esté

accesando el dato compartido impide a todos los demás hacer lo mismo al mismo tiempo. Esto se denomina exclusión mutua.

Secciones Críticas

La exclusión mutua necesita ser aplicada sólo cuando un proceso accesa a datos compartidos; cuando los procesos ejecutan operaciones que no estén en conflicto entre sí, debe permitírseles proceder de forma concurrente. Cuando un proceso está accesando datos compartidos se dice que el proceso se encuentra en su sección crítica (o región crítica). Esta claro que para prevenir el tipo de problema experimentado en la sección anterior debe asegurarse que, cuando un proceso esté en su sección crítica, todos los demás procesos (o al menos aquellos que tengan acceso a los mismos datos compartidos) sean excluidos de sus propias secciones críticas.

Mientras un proceso se encuentre en su sección crítica, los demás procesos pueden continuar su ejecución fuera de su secciones críticas. Cuando un proceso abandona su sección crítica, entonces debe permitírsele proceder a otros procesos que esperan entrar en su propia sección crítica (si hubiera un proceso en espera). La aplicación de la exclusión mutua es uno de los problemas clave de la programación concurrente. Se han diseñado muchas soluciones para esto: algunas de software y

algunas de hardware, más de bajo nivel y otras de alto nivel; algunas que requieren de cooperación voluntaria entre los procesos, y algunas que demandan una adherencia rígida a protocolos estrictos.

Estar dentro de una sección crítica es un estado muy especial asignado a un proceso. El proceso tiene acceso exclusivo a los datos compartidos, y todos los demás procesos que necesitan accesar a esos datos permanecen en espera. Por tanto, las secciones críticas deben ser ejecutadas lo más rápido posible, un programa no debe bloquearse dentro de su sección crítica, y las secciones críticas deben ser codificadas con todo cuidado (para evitar, por ejemplo, la posibilidad de incurrir en ciclos infinitos).

Si un proceso dentro de una sección crítica termina, tanto de forma voluntaria como involuntaria, entonces, al realizar su limpieza de terminación, el sistema operativo debe liberar la exclusión mutua para que otros procesos puedan entrar en sus secciones críticas.

Primitivas de Exclusión Mutua

El programa concurrente anterior implementa correctamente el mecanismo contador de líneas de la sección anterior. Por simplicidad, trataremos tan sólo dos procesos concurrentes en los programas presentados en esta y en las próximas secciones. El manejo de n procesos concurrentes es mucho más complejo.

Las construcciones entraexcluisionmutua y saledeexclusionmutua introducidas en el programa concurrente anterior encapsulan el código en cada proceso que accesa la variable compartida NOLINEAS, es decir, estas construcciones demarcan las secciones críticas. Estas operaciones se llaman, a veces, primitivas de exclusión mutua; o sea, que invocan las operaciones más fundamentales inherentes a la exclusión mutua.

```
program exclusionmutua;

var

nolineas: integer;

process uno;

var
```

```
lin: integer;
begin
for lin := 1 to 60 do
begin
entraexclusionmutua;
nolineas := nolineas + 1;
saledeexclusionmutua
end
end; (* uno *)
process dos;
var
lin: integer;
begin
for lin := 1 to 40 do
begin
entraexclusionmutua;
nolineas := nolineas + 1;
saledeexclusionmutua
end
end; (* dos *)
begin
nolineas := 0;
cobegin
uno;
dos
coend;
```

end.

En el caso de los dos procesos, estas primitivas operan como sigue: cuando proceso

uno ejecuta entraexclusionmutua, si proceso dos no está en su sección crítica,

entonces proceso uno entra a su sección crítica, accesa la variable deseada y

después ejecuta saledeexclusionmutua para indicar que ha abandonado su sección

crítica.

Si proceso dos está en su sección crítica cuando proceso uno ejecuta entraexclusionmutua, entonces proceso uno entra en espera hasta que proceso dos ejecute saledeexclusionmutua. Proceso uno puede, entonces, proceder a entrar en su sección crítica. Si proceso uno y proceso dos ejecutan entraexclusionmutua simultáneamente, entonces le será permitido proceder a alguno, permaneciendo el otro en espera.

Implementación de las primitivas de Exclusión Mutua

Se busca una implementación de entraexclusionmutua código de entrada a exclusión mutua y saledeexclusionmutua código de salida de exclusión mutua que satisfaga las cuatro restricciones siguientes:

1. La solución se implementa sólo en software en una máquina que no tenga instrucciones de exclusión mutua especialmente diseñadas. Cada instrucción de lenguaje de máquina se ejecuta de forma indivisible; es decir, una vez iniciada una instrucción, ésta se termina sin interrupción.

2. Si procesadores múltiples tratan de accesar el mismo dato, debemos suponer que una característica del hardware llamada interbloqueo de almacenamiento resuelve todos los conflictos. El interbloqueo de almacenamiento secuencializa las referencias en conflicto por medio de procesadores separados, esto es, se hace que las referencias sucedan una a la vez. Se da por supuesto que las referencias separadas son servidas en orden aleatorio.

3. No deberá hacerse ninguna suposición en relación con las velocidades relativas de procesos concurrentes asincrónicos.

4. Los proceso que se encuentren operando fuera de sus secciones críticas no pueden evitar que entren otros procesos en sus propias secciones críticas. No deben postergarse indefinidamente la entrada de los procesos en sus secciones críticas.

Una implementación elegante de software de la exclusión mutua fue la presentada por primera vez por el matemático holandés Dekker. En la siguiente sección seguimos el desarrollo de Dijkstra del algoritmo de Dekker.

3.3 Abrazo Mortal: causas, condiciones, prevención

En la teoría de los sistemas operativos, se puede definir el problema del Abrazo Mortal como la situación de un conjunto de procesos en un estado de espera tal que ninguno de ellos tiene suficientes criterios para continuar su ejecución.

Circunstancias parecidas suceden a menudo en la vida real, por ejemplo, en una carretera de dos direcciones, donde un determinado punto de cruce con la vía férrea, se ha construido un puente que por problemas urbanísticos o de presupuesto sólo deja pasar vehículos de un sentido. Dado este punto crítico en la

mencionada carretera, se presentan las siguientes situaciones:

· Un vehículo llega al puente y no se encuentra ningún otro en sentido contrario. En este caso, cruza haciendo uso del puente y no ocurre nada anormal.

· Si el paso por el puente es controlado por un semáforo en cada lado de manera que 100 metros antes de cada semáforo se sitúen sendos detectores de presencia de vehículos cuya finalidad sea poner en rojo el semáforo del sentido contrario ante la presencia de un vehículo, podría suceder que si llegan al mismo tiempo vehículos en los dos sentidos se pongan los dos semáforos en rojo impidiendo el paso de vehículos en ambos sentidos. En este caso el camino queda bloqueado, lo que representa un silogismo al Abrazo Mortal entre procesos.

· Si no habiendo semáforos, el conductor situado en uno de los extremos es lo

suficientemente educado que deja pasar en primer lugar al del otro extremo y antes de terminar de cruzar este último aparece por el mismo extremo otro vehículo, y así

sucesivamente mientras aparezcan vehículos por el lado extremo. Esta situación podemos emparejarla con la de postergación indefinida que estudiaremos más delante.

Visto el ejemplo anterior que nos introduce en los conceptos básicos que vamos a tratar a continuación referentes al Abrazo Mortal y a la postergación indefinida, damos paso al estudio de recursos y del modelo de sistema en los que basaremos dichos conceptos.

Recursos

Se entiende como recurso un elemento que un programa o proceso puede utilizar en la computadora donde se está ejecutando. Se engloban bajo el concepto de recurso, tanto los dispositivos hardware (por ejemplo, una impresora), como una cierta cantidad de información (por ejemplo, un registro de un archivo).

No obstante, en una computadora pueden existir muchos tipos de recursos, e incluso varios del mismo tipo. Por ello definiremos un recurso como algo que puede ser utilizado por un solo proceso en un instante dado. Para que el proceso pueda utilizar un recurso, deberá realizar la siguiente secuencia de operaciones:

· Solicitar el recurso. Si no estuviera disponible el proceso, quedará bloqueado hasta que le pueda ser asignado.

· Utilizar el recurso.

· Liberar el recurso

Modelo

En primer lugar vamos a definir lo que entendemos por Abrazo Mortal. Se dice que un conjunto de procesos han alcanzado un estado de Abrazo Mortal si cada uno de ellos espera que ocurra algo que sólo puede ser producido por uno de los procesos del propio conjunto (no necesariamente tiene que ser el mismo suceso). Como todos los procesos están en espera, ninguno de ello será el primero en producir el suceso deseado y por tanto permanecerá esperando indefinidamente.

Para formalizar todo lo expuesto hasta el momento, vamos a fijar los principios en que se basa todo sistema informático:

· Posee un número finito de recursos.

· Existe un número finito de procesos que compiten por los recursos.

· Los recursos se pueden dividir en tipos de tal forma que cada uno de ellos esté

compuesto por recursos idénticos entre sí.

· Los procesos deben realizar las tres acciones expuestas anteriormente sobre los

recursos: solicitar, utilizar, liberar.

· Un proceso puede pedir tantos recursos como necesite para realizar su trabajo, ya sean del mismo tipo o no, siempre que no excedan del total existente en el sistema.

· Las operaciones sobre los recursos se realizarán a través de llamadas al sistema operativo, de manera que si se solicita un recurso que está siendo utilizado, quedará bloqueado en espera de que se libere dicho recurso.

3.4.1 Espera activa

Son aquellos algoritmos que basan todo su funcionamiento en establecer la espera de entrada a la sección crítica con un bucle que será roto en el momento en que se cumpla una determinada condición. Se llaman de espera activa porque el proceso no queda bloqueado durante su ejecución, sino que estará compitiendo por el procesador constantemente. Por este motivo, estos algoritmos sobrecargan el sistema innecesariamente. Fueron los primeros en utilizarse y han sido sustituidos por otros más eficientes. Entre los distintos algoritmos de este tipo existentes podemos citar:

· Espera con mutex. Algoritmo que utiliza un switch (MUTEX) a través del cual se produce la sincronización.

· Alternancia. Ligeramente mejor que el anterior, utiliza también una variable TURNO para realizar el sincronismo entre los procesos.

· Algoritmo de DEKKER. Resuelve el problema mediante la solución propuesta por

Dekker, basando su funcionamiento en una tabla unidimensional de dos

elementos lógicos (Switches).

Espera no activa

Son los algoritmos que establecen la espera para entrar en la sección crítica bloqueando el proceso, haciendo que deje de competir por el procesador hasta que se cumpla la condición de desbloqueo.

Entre los algoritmos existentes, citaremos los siguientes:

· Semáforos. Para eliminar los problemas que se producen con los algoritmos de espera activa, fundamentalmente los referidos a la sobrecarga que producen en el sistema, E.W. Dijkstra (1965) diseñó un mecanismo basado en una variable entera utilizada como contador de peticiones de entrada a una sección crítica. Esta variable es compartida por todos los procesos del sistema. Este nuevo tipo de variable se denomina semáforo por su capacidades gestionar el tráfico de procesos que desean acceder a datos compartidos. Con este sistema, cuando un proceso intente entrar en una sección crítica mientras otro está accediendo a los datos compartidos, se bloqueará de igual manera que cuando un proceso accede a un recurso que está ocupado.

· Regiones críticas. Son sistemas que permiten establecer protecciones contra una mala utilización de los usuarios. Para ello sólo permiten que los datos compartidos se puedan acceder desde determinadas regiones quedando transparentes desde el resto. Tienen pequeños inconvenientes relacionados con la sincronización y no permite que varias actividades puedan realizar operaciones de lectura simultánea.

· Regiones críticas condicionales. Consiste en una mejora del método anterior tratando de resolver algunos problemas de sincronización que se presentaban.

· Monitores. Uno de los problemas existentes en los mecanismos anteriores es que el programador tiene que proporcionar de forma explícita el modo de sincronización. Para evitarlo B Hansen y C.A.R. Hoare desarrollaron un nuevo mecanismo denominado Monitor, que debe ser soportado por el lenguaje correspondiente. Un monitor permite compartir, segura y eficientemente, datos entre varias actividades, garantizando la exclusión mutua, sin necesidad de que el programador tenga que suministrarla explícitamente. Se basa en dos premisas:

la primera es la abstracción de datos consistente en una técnica capaz de

separar las operaciones a ejecutar sobre los datos, de los detalles de diseño, propios de los mismos (los lenguajes Modula y Ada soportan este tipo de estructuras). La segunda es que realizan la exclusión mutua de forma implícita. La finalidad más útil de los monitores es reunir todas las funciones que operar sobre un conjunto de datos compartidos en un solo módulo, de manera que todos los accesos a esos datos estarán forzados a utilizar dichas funciones.

· Contadores de eventos. Es un mecanismo para sincronizar actividades sin que sea necesario forzar la exclusión mutua, ya sea por que no deseamos limitar la concurrencia de las actividades, o simplemente porque no lo necesitemos. Se basa en una variable entera cuya misión en contar determinadas operaciones.

· Mensajes. Es un mecanismo que permite a los procesos intercambiar aquella información que sea necesaria durante el desarrollo normal de su ejecución. Por tanto, es más un mecanismo de cooperación que de sincronización. Esta cooperación se realiza por medio de mensajes que se envían entre sí los procesos. Se basan en una zona de memoria compartida que gestiona el sistema operativo directamente y que permanece oculta a los procesos. De esta manera el proceso que envía un mensaje a otro deposita la información en la zona de memoria compartida, y el receptor lo lee de ella posteriormente. En este sistema. Las directrices de envío y recepción establecen una sincronización entre los

procesos al tener que esperar por dichos mensajes, antes de continuar con su ejecución.

Existen dos tipos de comunicación entre procesos:

− Directa. Los procesos envían y reciben los mensajes directamente entre si, de manera que se asocia un enlace bidireccional único entre cada dos procesos.

− Indirecta. Los mensajes son enviados y recibidos a través de mailboxes o buzones de correos. Con este método cada proceso puede estar relacionado con tantos buzones como desee consiguiendo comunicarse con tantos procesos como sea necesario.

− Llamadas remotas. El paso de mensajes de un proceso a otro es un mecanismo muy utilizado para resolver los problemas de concurrencia entre procesos. Es análogo al paso de parámetros en una llamada a una rutina o procedimiento. Si unimos los conceptos de mensajes y paso de parámetros, tendremos lo que se conoce como rutinas o procedimientos remotos, creándose una copia del mismo

cada vez que son ejecutadas (llamadas remotas), siendo dicha ejecución concurrente. Este tipo de interacción

asegura que hasta que un proceso no termine determinadas operaciones, el siguiente permanecerá en espera.

− Rendez−vous. Es la culminación de todos los mecanismos anteriores, tratándose de una ligera modificación del método de las llamadas remotas, desde la llamada, en lugar de ser de todo un procedimiento, lo es solamente a un grupo de sentencias dentro de él. De igual forma, se trata de un procedimiento similar al monitor, pero con más versatilidad y potencia. Este método se ha llevado a la práctica con el lenguaje Ada.

3.4.2 Bloqueo Mutuo

La implantación de un semáforo con una cola de espera puede dar como

resultado una situación donde dos o más procesos esperen indefinidamente un suceso que sólo puede ocasionar uno de los procesos en espera. El suceso en cuestión consiste en la ejecución de una operación signal. Cuando se llega a este estado, se dice que los procesos están en bloqueo mutuo.

Para ilustrarlo, consideremos un sistema que consta de dos procesos, Uno y Dos, cada uno con acceso a dos semáforos, S y Q, con valor inicial 1:

Uno Dos wait(S) wait(Q)

wait(Q) wait(S)

. .

. .

. .

signal(S) signal(Q)

signal(Q) signal(S)

Suponga que Uno ejecuta wait(S) y luego Dos ejecuta wait(Q). Cuando Uno ejecuta wait(Q), debe esperar a que Dos ejecute signal(Q). De manera parecida, cuando Dos ejecuta wait(S), debe esperar a que Uno ejecute signal(S). Como estas

operaciones de signal no pueden ejecutarse, Uno y Dos están en bloqueo mutuo.

Decimos que un conjunto de procesos está en un estado de bloqueo mutuo cuando cada uno de los procesos del conjunto está esperando un suceso que únicamente puede ser provocado por otro proceso del conjunto.

Otro problema relacionado con el bloqueo mutuo es el bloqueo indefinido o inanición, en que los procesos pueden esperar indefinidamente dentro del semáforo. El bloqueo indefinido se puede presentar si añadimos y eliminamos procesos de la lista asociada con un semáforo en orden LIFO.

A continuación se muestra el listado de un programa sencillo que ilustra el concepto de bloqueo mutuo. La segunda columna de números en el listado del compilador, corresponde a la dirección de memoria de esa instrucción en lenguaje de máquina. Este tipo de listado se presenta para determinar, el punto exacto, de cada proceso en el programa fuente, donde ocurre el bloqueo.

− Pascal−FC for IBM PC compatibles −

− Compiler Version P5.2

G L Davies & A Burns, University of Bradford

Compiler listing

Line PC

1 0 program sem03;

2 0 (*

3 0 Proceso de sincronización de bloqueo/despertar

4 0 con semáforos

5 0 *)

6 0

7 0 var

8 0 S, Q: semaphore;(* declaración de los semáforos *)

9 0

10 0 process Uno;

11 0 begin

12 0 writeln('Entramos al Uno');

13 3 wait(S); (* Espera por el semáforo *)

14 5 writeln('El Uno toma S y espera por Q');

15 8 wait(Q);

16 10

17 10 writeln('Uno entra a la sección crítica ');

18 13

19 13 signal(S);

20 15 signal(Q);

21 17 end; (* uno *)

22 18

```
23 18

24 18 process Dos;

25 18 begin

26 18 writeln('Entramos al Dos');

27 21 wait(Q); (* Espera por el semáforo *)

28 23 writeln('El Dos toma Q y espera por S');

29 26 wait(S);

30 28

31 28 writeln('Dos entra a la sección crítica ');

32 31

33 31 signal(Q);

34 33 signal(S);

35 35 end; (* dos *)

36 36

37 36

38 36 begin

39 36 initial(S,1);

40 40 initial(Q,1);

41 44 cobegin

42 45 Uno;

43 49 Dos

44 49 coend;

45 54 end.
```

3.4.3 Algoritmos de Dekker

Secciones Críticas

La exclusión mutua debe ponerse en practica sólo cuando los procesos obtienen acceso a datos compartidos modificables; cuando los procesos realizan operaciones que no entran en conflicto con otras, debe permitirse

que procedan concurrentemente.

Mientras un proceso se encuentra en su sección crítica, otros procesos pueden, claro está, seguir ejecutándose fuera de sus secciones críticas. El matemático holandés Dekker fue el primero en presentar una elegante realización en software para la exclusión mutua:

Primera versión de las primitivas de exclusión mutua:

Este es un primer intento por especificar el código que permite poner en práctica la Exclusión Mutua en el contexto de un programa concurrente con dos procesos. La construcción parbegin/parend hace que proceso_uno y proceso_dos operen como procesos concurrentes. Cada uno de estos procesos se encuentra dentro de un ciclo infinito, entrando una y otra vez en sus secciones críticas. Si usted observa en la siguiente página, verá que se pone en práctica entrar_ewxclusión_mutua mediante un simple ciclo while que se repite hasta que numero_proceso es igual al numero del proceso; salir_exclusión mutua se pone en practica con una sola instrucción que asigna a número_proceso el numero del otro proceso Proceso_uno ejecuta el ciclo while do. Como en un principio numero_proceso vale 1, proceso_uno entra en su sección crítica. Proceso_dos encuentra que numero_proceso equivale a 1 y se mantiene dentro de su ciclo while do. Cada vez que proceso_dos obtiene el procesador, se limita a repetir el ciclo en espera de que numero_proceso valga 2, de manera que proceso_dos no entra en su sección crítica y la exclusión mutua está garantizada. Como el procesador esta en uso mientras proceso_dos no hace nada (excepto verificar el valor el valor de numero_proceso), esto se conoce como espera activa. La espera activa se considera una técnica aceptable cuando las esperas anticipadas son cortas; de lo contrario, la espera activa puede resultar costosa.

Algoritmo Versión 1

Segunda versión:

En la primera versión había solamente una variable global y ello ocasionó el problema de la sincronización en Tandem. Por eso, en la segunda versión se usan dos variables: p1adentro, que es cierta si proceso_uno se encuentra en su sección crítica y p2adentro que es cierta si proceso_dos se encuentra a su vez en su sección crítica. Ahora proceso_uno permanece bloqueado en una espera activa mientras sea cierta p2adentro. Con el tiempo proceso_dos abandona su sección crítica y ejecuta su propio código de salida de exclusión mutua, asignando a p2adentro el valor falso. En ese momento, proceso_uno asigna a p1 adentro el valor cierto y entra en su sección crítica. De nuevo surgen las sutilezas de la programación concurrente. Como proceso_uno y proceso_dos son procesos concurrentes, podrían intentar ejecutar simultáneamente sus códigos de entrada a exclusión mutua. En un principio p1adentro y p2adentro son falsas. Proceso_uno podría verificar el valor de

p2adentro y encontrar que es falso; entonces, antes que proceso_uno pueda asignar a p1adentro y encontrar que es falso. En ese instante proceso_uno asigna el valor cierto a p1adentro y entra en su sección crítica; por su parte, proceso_dos asigna el valor cierto a p2adentro y entra en su sección crítica. Ambos procesos están simultáneamente en sus secciones críticas, de manera que la versión dos ni siquiera garantiza la exclusión mutua.

Algoritmo Versión 2

Tercera versión:

La versión trata de resolver el problema de la segunda haciendo que cada proceso asigne el valor cierto a su bandera antes de realizar la verificación. Así, proceso_uno indica su deseo de entrar en su sección crítica asignando el valor cierto ap1deseantrar. Si p2deseaentrar es falso, entonces proceso_uno entra en su sección crítica y hace que proceso_dos espere, si acaso desea entrar en su sección crítica. De esta manera se garantiza la exclusión mutua y se tiene al parecer una solución correcta. Se ha resuelto un problema pero se ha creado

otro. Si cada proceso asigna el valor cierto a su bandera antes de realizar la verificación, entonces cada proceso podrá encontrar la bandera del otro con valor cierto y los dos entrarán en un ciclo infinito. Este es un ejemplo de bloqueo mutuo de dos procesos.

Algoritmo Versión 3

Cuarta versión

El problema de la versión tres, es que cada proceso puede bloquearse en su verificación respectiva. Se necesita una forma de "escapar" de estas verificaciones. La cuarta versión resuelve el problema ligando a un proceso que se encuentre dentro del ciclo a asignar repetidamente el valor falso a su bandera por periodos cortos; esto permitirá que el otro proceso salga de su ciclo while, con el valor

cierto todavía en su propia bandera. La exclusión mutua esta garantizada y no se puede producir un bloqueo mutuo, pero puede originarse otro problema potencialmente grave: un aplazamiento indefinido. Los procesos podrían ejecutarse en tandem, es decir, cada proceso puede asignar el valor cierto a su bandera, realizar la verificación, entrar en el cuerpo del ciclo while, asignar el valor falso a

su bandera, asignar el valor cierto a su bandera y después repetir la secuencia comenzando con la verificación. Mientras ocurre esto, las condiciones verificadas s siguen cumpliendo. Es claro que esta situación tiene muy pocas posibilidades de ocurrir, pero podría darse el caso. Por tanto la versión cuatro resulta inaceptable. Si un sistema que usara este tipo de exclusión mutua se empleara para controlar un vuelo espacial, un marcapasos o un sistema de control de tráfico aéreo, estarían latentes la posibilidad de un aplazamiento indefinido y la consiguiente falla del sistema.

Algoritmo Versión 4

Algoritmo de Dekker

El algoritmo de Dekker resuelve la posibilidad de aplazamiento indefinido de la versión cuatro. Proceso_uno indica su deseo de entrar en su sección crítica. Si la bandera de proceso_dos tiene el valor cierto a su bandera. En seguida realiza la verificación, en la cual comprueba si proceso_dos también desea entrar en su sección crítica. Si la bandera de proceso_dos tiene el valor falso, proceso_uno pasa por alto el cuerpo del ciclo y entra en su sección crítica. Supóngase empero que cuando proceso_uno realiza su verificación descubre que la bandera de proceso_dos tiene el valor cierto. Esto hace que proceso_uno ejecute el cuerpo de su ciclo while. Dentro del ciclo examina la variable proceso_favorecido que simultáneamente en sus secciones críticas. Si proceso_uno es el favorecido, pasa por alto el cuerpo de la condicional y ejecuta repetidamente el ciclo while en espera de que proceso_dos asigne el valor falso a su bandera.

3.4.4 Semáforos

Dijkstra extractó las nociones clave de la exclusión mutua en su concepto de semáforo. Un semáforo es una variable protegida cuyo valor puede ser accesado y alterado tan sólo por las operaciones wait, signal y una operación de inicialización del semáforo initial. Los semáforos binarios sólo pueden tomar los valores 0 y 1. Los semáforos contadores pueden tomar valores enteros no negativos.

La operación wait en el semáforo S, escrito wait(S), opera de la siguiente manera:

if $S > 0$

then $S:=S-1$

else espera S

La operación signal en el semáforo S, escrito signal(S), opera de la siguiente manera:

if (uno o mas procesos esperan S

then deja que esperen

else S:= S+1

Supondremos que hay una disciplina de colas del primero en entrar − primero en salir para los procesos que esperan a completar un wait(S). La ejecución de las instrucciones wait y signal son indivisibles. La exclusión mutua en el semáforo, S, es aplicada en wait(S) y signal(S). Si varios procesos intentan ejecutar wait(S) al mismo tiempo, sólo uno podrá proseguir; los otros permanecerán en espera. Los semáforos y las operaciones de semáforos pueden implementarse en software o hardware. En general, se implementan en el núcleo del sistema operativo, donde se controlan los cambios de estado de un proceso. El programa sem01 que se lista a continuación, muestra como pueden utilizarse los semáforos para aplicar la exclusión mutua.

```
program sem01;

(*

solución por semáforos al problema de la

exclusión mutua

*)

var

nolineas: integer;

mutex: semaphore; (* declaración del semáforo *)

process uno;

var

lin: integer;

begin

for lin := 1 to 20 do

begin

wait(mutex); (* Espera por el semáforo *)

nolineas := nolineas + 1; (* sección crítica *)

signal(mutex) (* libera el semáforo *)
```

```
end

end; (* uno *)

process dos;

var

lin: integer;

begin

for lin := 1 to 20 do

begin

wait(mutex);

nolineas := nolineas + 1;

signal(mutex)

end

end; (* dos *)

begin

nolineas := 0;

initial(mutex,1); (* se inicializa el semáforo *)

cobegin

uno;

dos

coend;

writeln('Total de Líneas = ',nolineas)

end.
```

Sincronización de procesos con semáforos

Cuando un proceso emite una petición de entrada/salida, se bloquea a sí mismo para esperar a que termine esta operación. Algunos procesos deben despertar al proceso bloqueado. Tal interacción es un ejemplo de un protocolo de bloqueo/despertar.

En forma más general, supóngase que un proceso desee ser informado de la ocurrencia de un evento. Supóngase que otro proceso es capaz de detectar que ese acontecimiento ya ha ocurrido. El siguiente

programa (sem02) muestra cómo pueden utilizarse las operaciones de semáforos para implementar un mecanismo simple de sincronización de bloqueo/despertar de dos procesos.

```
program sem02;

(*

Proceso de sincronización de bloqueo/despertar

con semáforos

*)

var

mutex: semaphore; (* declaración del semáforo *)

process uno;

begin

writeln('Cosas preliminares de Uno');

wait(mutex); (* Espera por el semáforo *)

writeln('Otras cosas del Uno');

end; (* uno *)

process dos;

begin

writeln('Cosas preliminares del Dos');

signal(mutex);

writeln('Otras cosas del Dos.');

end; (* dos *)

begin

initial(mutex,0);

cobegin

uno;

dos

coend;
```

end.

El proceso uno ejecuta algunas cosas preliminares y después ejecuta wait(mutex). El semáforo ha sido inicializado a cero, de modo que el proceso debe esperar. El proceso dos ejecuta signal(mutex) para señalar que el evento ha ocurrido. Esto permite proceder al proceso uno. Obsérvese que este mecanismo trabaja incluso cuando proceso dos detecta y señala el evento antes de que el proceso uno ejecute wait(mutex); el semáforo habrá sido incrementado de 0 a 1, así es que wait(mutex), decremento el semáforo de 1 a 0, y proceso uno proseguirá sin tener que esperar a que ocurra el evento.

La implantación de un semáforo con una cola de espera puede dar como resultado una situación donde dos o más procesos esperen indefinidamente un suceso que sólo puede ocasionar uno de los procesos en espera. El suceso en cuestión consiste en la ejecución de una operación signal. Cuando se llega a este estado, se dice que los procesos están en bloqueo mutuo.

Semáforos Contadores

Los semáforos contadores son particularmente útiles cuando un recurso va a ser asignado desde una bolsa de recursos idénticos. El semáforo se inicializa para el número de recursos de la bolsa. Cada operación wait decrementa el semáforo en 1, indicando que ha extraído otro recurso de la bolsa y está siendo usado por un proceso. Cada operación signal incrementa el semáforo en 1, indicando que un

proceso ha devuelto un recurso a la bolsa, y puede ser asignado a otro proceso. Si se intenta una operación wait cuando el semáforo ha sido decrementado hasta cero, el proceso debe esperar hasta que se devuelva algún recurso a la bolsa por medio de una operación signal.

3.4.5 Productor−Consumidor

En un programa secuencial, cuando un procedimiento llama a otro y le trasmite datos, los procedimientos son parte de un proceso simple y no operan de forma concurrente. Pero cuando un proceso transmite datos a otro, los problemas son mucho más complejos. Tal transmisión es un ejemplo de comunicación interprocesos.

Considérese la siguiente relación productor consumidor. Suponga que un proceso, un productor, está generando información que utiliza un segundo proceso, un consumidor. Suponga que se comunican por medio de la variable compartida, buffer. El productor agrega datos en el arreglo buffer, el consumidor lee los datos de buffer y los imprime. Es posible que los procesos productor y consumidor trabajen bien en tándem, o que sus velocidades de ejecución sean muy parecidas. Si cada vez que el productor deposita un dato en buffer, el consumidor lo lee y lo imprime de inmediato,

entonces la salida impresa representará fielmente la corriente de números generada por el productor.

Pero supóngase que las velocidades de los procesos son dispares. Si el consumidor está operando con más rapidez que el productor, el consumidor puede leer e imprimir datos antes de que el productor los deposite. Si el productor está operando más rápido que el consumidor, el productor podría sobrescribir los datos previos antes de que el consumidor tenga la oportunidad de leerlos e imprimirlos; un productor muy veloz podría hacer esto muchas veces, con lo cual muchos resultados

se perderían.

Como es evidente, el comportamiento que deseamos aquí es que el productor y el consumidor cooperen de forma tal que los datos escritos en buffer no se dupliquen ni se pierdan. La aplicación de tal comportamiento se conoce como sincronización de procesos.

El siguiente programa (pcsem) muestra un programa concurrente que utiliza las operaciones de semáforos para implementar una relación productor − consumidor. Los procesos utilizan la variable compartida buffer. El productor va poniendo en el arreglo los números del 65 al 90 (código ASCII de las letras A−Z) mientras que el consumidor los va tomando e imprimiendo. Aquí hemos utilizado tres semáforos: mutex se utiliza para aplicar el acceso de exclusión mutua a la variable compartida buffer; listos controla la sincronización de los procesos para determinar si existen datos en el buffer; espacios controla en la sincronización la existencia de espacio en el buffer.

```
program pcsem; (* Solución al problema del Productor − Consumidor mediante el uso de semáforos*)

const buffmax = 5;

var buffer: array[0..buffmax] of integer; sigentra, sigsale: integer; (* apuntadores a buffer para el

manejo de la cola*)

espacios, listos: semaphore; mutex: semaphore;

procedure pon(ch: integer);

begin (* el buffer es una cola circular *)

buffer[sigentra] := ch;

sigentra := (sigentra + 1) mod (buffmax + 1) end;

(* pon *)

procedure toma(var ch: integer);

begin ch := buffer[sigsale];

sigsale := (sigsale + 1) mod (buffmax + 1) end;

(* toma *)

process productor;

var local: integer;

begin for local := 65 to 90 do

begin wait(espacios); (* Espera que haya espacio *)

wait(mutex);

(* Entrada a la sección crítica *) pon(local);

(* Sección crítica *) write(chr(local));

signal(mutex);
```

```
(* Sale de sección crítica *) signal(listos)

(* Avisa que ya hay un dato listo *)

end

end; (* productor *)

process consumidor;

var local: integer;

begin

repeat

begin

wait(listos);

(* Espera que haya datos listos *) wait(mutex);

(* Entra en la sección crítica *) toma(local);

(* Sección crítica *) signal(mutex);

(* sale de la sección crítica *) signal(espacios);

(* Avisa que hay un espacio *) write(chr(local+32));

end until local = 90;

writeln end;

(* consumidor *)

begin

initial(espacios,buffmax + 1);

initial(listos,0);

initial(mutex,1);

sigentra := 0;

sigsale := 0;

cobegin

productor;
```

consumidor

coend

end.

3.4.6 Monitores

Un monitor es una construcción de concurrencia que contiene los datos y procedimientos necesarios para realizar la asignación de un recurso compartido determinado, o de un grupo de recursos compartidos. Para cumplir con la función de asignación de recurso, un proceso debe llamar a determinada entrada al monitor. Muchos procesos pueden tratar de entrar al monitor en diversos momentos. Pero la exclusión mutua es aplicada rígidamente en los límites del monitor. Sólo se permite

la entrada a un proceso a la vez. Los procesos que deseen entrar cuando el monitor está en uso deben esperar. La espera es administrada de forma automática por el monitor. Como la exclusión mutua está garantizada, se evitan los desagradables problemas de concurrencia (como los resultados indeterminados) señalados con anterioridad.

Los datos contenidos en el monitor pueden ser globales, para todos los procedimientos dentro del monitor, o locales, para un procedimiento específico. Todos estos datos sólo son accesibles dentro del monitor, no hay forma de que un proceso de fuera del monitor pueda accesar sus datos. Esto se llama encapsulamiento de información, una técnica de estructuración del sistema que facilita en gran medida el desarrollo de sistemas de software más confiables.

Si el proceso que llama a la entrada al monitor encuentra que el recurso ha sido asignado, entonces el procedimiento del monitor llama a wait. El proceso podría permanecer dentro del monitor, pero esto violaría la exclusión mutua si en este momento entrara otro proceso al monitor. Por tanto, al proceso que llama a wait se le hace esperar fuera del monitor hasta que el recurso sea liberado.

El proceso que tiene al recurso llamará finalmente una entrada al monitor para devolver el recurso al sistema. Esta entrada tan sólo aceptaría el recurso de regreso y esperaría la llegada de otro proceso de petición. Pero puede haber varios procesos esperando por el recurso, así es que el monitor llama a signal para permitir que uno de los procesos en espera adquiera el recurso y salga del monitor. Si un proceso señala el regreso (o liberación) del recurso, y no hay otro esperando, entonces la señal no tiene efecto alguno (pero el monitor ha recapturado el recurso). Está claro que un proceso que espera un recurso debe hacerlo fuera del monitor, para permitir que otro proceso dentro del monitor regrese el recurso.

Para asegurar que un proceso en espera de un recurso acabe consiguiéndolo, el monitor le otorga prioridad sobre un nuevo proceso que pida entrar al monitor. De otra manera, el nuevo proceso toma el recurso antes de que el que está en espera vuelva al monitor. Si esta desafortunada secuencia tuviera lugar de forma repetida, entonces un proceso en espera sería postergado indefinidamente.

De hecho, los procesos podrían desear (necesitar) esperar fuera del monitor por varias razones, como veremos en los ejemplos. Así que se introduce la noción de una variable de condición. Se asocia una variable de condición individual a cada una de las razones por las que un proceso pueda necesitar esperar. Las operaciones delay y resume son entonces modificadas para incluir el nombre de la variable de condición que se está esperando o señalando:

delay(nombredelavariabledecondicion) resume(nombredelavariabledecondicion)

Las variables de condición son muy distintas de las "convencionales" con las que estamos familiarizados. Al

definir una variable de condición se establece una cola. Un proceso que llama a delay se asocia a la cola y un proceso que llama a resume provoca la extracción de un proceso en espera de la cola y su entrada al monitor. Puede darse por sentada la existencia de una disciplina de colas FIFO, aunque los esquemas de prioridades pueden ser útiles en algunas situaciones.

A continuación se muestra el listado de un programa (mons01) que resuelve el mismo problema de exclusión mutua, descrito anteriormente, utilizando monitores.

```
program mons01;

(*

Problema de conteo de líneas usando monitores

*)

const

nprocs = 10;

var

lin: integer;

monitor contador; (* se define el monitor *)

export (* define los procedimientos a exportar *)

inc, escribe;

var (* variables locales al monitor *)

nolineas: integer;

procedure inc;

begin

nolineas := nolineas + 1

end; (* inc *)

procedure escribe;

begin

writeln('Total de líneas = ',nolineas:2)

end; (* escribe *)

begin (* cuerpo del monitor *)
```

```
nolineas := 0

end; (* monitor contador *)

process type turnos;

(* se define un tipo de proceso llamado turnos*)

var

loop: integer;

begin

for loop := 1 to 20 do

contador.inc

(* se invoca el procedimiento inc del monitor *)

end; (* turnos *)

var

proceso: array[1..nprocs] of turnos;

begin

cobegin

for lin := 1 to nprocs do

proceso[lin]

coend;

contador.escribe

end.
```

Llamada a recursos

Una llamada a un recurso es una llamada a un procedimiento (custodiado o no) exportado de ese recurso y tiene la siguiente forma:

Id_resource.

exported_procedure_Id[(parameters=]

Como con los monitores, si el procedimiento mencionado está declarado dentro del recurso actual, se permite una notación corta. Si se llama a un procedimiento custodiado cuyo guardia evalúa a falso, el proceso que lo llama quedará suspendido y dejará el recurso. Cualquier proceso que esté dejando un recurso debe intentar

encontrar un candidato, el cual herede la exclusión mutua en el recurso. El candidato será seleccionado, de aquellos procesos suspendidos por los guardias y el cual evalúa a true.

4.1 Memoria Física

Necesidad del manejador de memoria

Para lograr el rendimiento de los procesos en necesario conservarlos en memoria y para ello se debe compartir la memoria.

La selección de un esquema de administración de la memoria depende del diseño del hardware del s.o.. Los algoritmos dependen del diseño del hardware.

¿Qué es la memoria?

Arreglo de palabras o bytes con su propia dirección. Se tiene acceso a ella por medio de una secuencia de lecturas o escrituras a direcciones específicas de memoria. La UCP busca información de la memoria o la almacena en ella.

Un proceso debe estar cargado en la memoria para que se pueda ejecutar. Por lo general el proceso reside en disco como archivo binario ejecutable. El conjunto de procesos que esperan entrar en la memoria para ejecutarse forma una cola de entrada.

El procedimiento normal consiste en seleccionar uno de los procesos de la cola de entrada y cargarlo en la memoria. El proceso se carga, accede a las instrucciones y datos en memoria; cuando termina su ejecución se libera el espacio en memoria.

Disposición de la memoria:

1. En un programa fuente las direcciones son generalmente simbólicas (I)

2. Un compilador enlazará estas direcciones simbólicas con direcciones

relocalizables (como 14 bytes a partir del inició de este módulo)

3. El cargador (o editor de enlaces) enlazará estas direcciones relocalizables con

direcciones absolutas (como 74014). Cada enlace es una correspondencia

entre un espacio de direcciones y otro.

El enlace de las instrucciones y datos casi puede efectuarse en cualquier etapa del camino:

a) Compilación: Si al momento de la compilación se sabe dónde residirá el programa en memoria puede generarse un código absoluto. Si se sabe que el proceso de usuario residirá en una posición R, entonces el código generado por el compilador comenzará en esa posición. Si después se cambia la posición de inicio, será necesario volver a compilar el código. Los programas de MS−DOS .COM se enlazan de manera absoluta durante la compilación.

b) Carga. Si durante la compilación no se conoce dónde residirá el programa en la memoria, entonces el compilador deberá generar código relocalizable. En este caso, el enlace final se hace al momento de la carga. Si cambia la dirección de inicio, se tendrá que cargar de nuevo el código del usuario

c) Ejecución. Si durante la ejecución el proceso puede moverse de un segmento de memoria a otro, entonces el enlace final se hace al momento de la ejecución. Es necesario tener un hardware especial para que este esquema funcione.

Carga dinámica

Se utiliza la carga dinámica para tener una mejor utilización del espacio de memoria, en la cual las rutinas se cargan conforme se llaman.

1. Todas las rutinas se almacenan en un disco en formato de carga relocalizable.

2. El programa principal se carga en memoria y se ejecuta

3. Cuando una rutina tiene que llamar a otra, la primera comprueba si la segunda se ha cargado en la memoria

4. En caso de que no se haya cargado se llama al cargador de enlace relocalizable para que cargue en memoria la rutina deseada y actualice las tablas para reflejar dicho cambio

5. El control se transfiere a la rutina recién llegada.

Ventaja: Las rutinas no se cargan hasta que no se usen o sean llamadas. Se utiliza para grandes cantidades de código que manejan situaciones con de poca frecuencia, como las rutinas de error.

Enlace dinámico

Las bibliotecas también pueden ser enlazadas dinámicamente. La mayoría de los s.o. sólo permiten el enlace estático, donde las bibliotecas del lenguaje se tratan como otro módulo objeto. El enlace dinámico posterga en enlace de las bibliotecas hasta el momento de su ejecución. Sin este recurso, los programas tendrían que incluir en su ejecutable una copia de la biblioteca del lenguaje. Con el enlace dinámico se incluye un fragmento (stub) en el código binario para cada referencia a la rutina de la biblioteca:

1. El fragmento es un código que indica cómo localizar la rutina de biblioteca residente en memoria

2. Al ejecutarse el fragmento, se reemplaza a sí mismo con la dirección de la rutina y la ejecuta

Ventaja: Para las nuevas actualizaciones de bibliotecas, en donde se cargan las versiones anteriores y nuevas con un número de versión, de tal forma que cada programa pueda saber que versión utilizar.

Superposiciones

Como todo el espacio lógico de direcciones de un proceso debe encontrarse en la memoria física antes de ejecutarse, la dimensión del proceso está limitada por el tamaño de la memoria física. Para que un proceso pueda utilizar más memoria que la disponible se utiliza la técnica de superposiciones.

Esto significa conservar en memoria sólo las intrusiones y datos que se requieren en un momento determinado. Cuando se necesitan otras instrucciones, se cargan en el espacio que antes ocupan las que ya no se requieren.

Ventaja: El usuario programa su programa para usar las superposiciones aunque se ejecutara más despacio por las E/S adicionales.

Intercambios

Un proceso necesita estar en memoria para ejecutarse. Sin embargo, un proceso puede intercambiarse temporalmente saliendo de la memoria a un almacenamiento secundario y regresando luego a la memoria para continuar con su ejecución.

Cada vez que un proceso termina su quantum, se intercambia con otro. El quantum debe tener el tamaño suficiente para que se efectúe una cantidad de cálculos razonables.

Para los algoritmos basados en prioridades se selecciona el proceso de mayor prioridad y después el de menor prioridad. A esto se le conoce como salida y entrada por intercambio.

Casi siempre un proceso que sale regresará al mismo espacio de memoria que antes ocupaba. Esta restricción la determina el método de enlace de direcciones. Si el enlace se lleva a cabo durante el ensamblador o carga, el proceso no puede moverse a otras localidades. Si el enlace se hace durante la ejecución entonces si se puede colocar en distinta localidad de memoria.

El almacenamiento auxiliar se lleva a cabo en un disco rápido, y debe contar con espacio suficiente para tener una copia de los procesos de memoria y tener un acceso directo a esta imagen. El sistema tiene una cola de procesos listos que son los procesos cuya imagen de memoria se encuentra en el disco o en la memoria y están listos para ejecutarse.

Cambio de contexto: (Ej.)

1 proceso de 100K

Velocidad de transferencia del disco: 1000K por seg.

100/1000 = 1/10 segundo

100 milisegundos

Tiempo de latencia: 8 milisegundos

Intercambio de entrada y salida: 216 milisegundos

Por lo tanto, el tiempo de ejecución del proceso (quantum) debe ser mayor que el tiempo de intercambio. También se utiliza saber el espacio que cada proceso ocupará de memoria y para ello el usuario informa al s.o. de los requisitos de memoria.

Restricciones para los intercambios:

Estar seguros que el proceso está inactivo

Procesos de E/S

4.2 Estrategias de almacenamiento

Obtención.

Cuando se debe transferir una página o segmento del almacenamiento secundario al almacenamiento primario. Se utilizan los esquemas de obtención anticipada, o depende de la capacidad de almacenamiento primario

Colocación.

Determinan el lugar donde se colocará el segmento entrante ¿ Porqué esto no es válido para la paginación

Reemplazo.

Decidir cuál página o segmento se debe desplazar para dejar espacio a una página o segmento cuanto está completamente ocupado el almacenamiento primario

4.3 Particiones fijas y variables

Una sola partición

a) El esquema de administración más sencillo es no tener ninguno, es decir, al usuario se le proporciona la máquina básica.

Ventajas:

Tiene el control absoluto de todo el espacio de memoria.

El usuario controla a su antojo el uso de la memoria

Es sencillo y de costo mínimo

No requiere de hardware especial para administrar la memoria ni software para S.O.

Desventajas:

No ofrece servicios

El SO no tiene control sobre las interrupciones

No hay llamadas al sistema o errores, ni espacio para la multiprogramación

Se utiliza para sistemas dedicados en donde se requieren rutinas de apoyo propias. Los primeros sistemas en donde se cobraba por el tiempo de uso de la máquina se utilizó este esquema, aunque se perdía de mucho tiempo y dinero ya que cada usuario tenía que escribir su propio código de E/S. En estos casos el tamaño del programa del usuario estaba limitado por el tamaño de la memoria.

b) El siguiente esquema consiste en dividir la memoria en dos particiones: una para el usuario y otra para el so residente. Podemos colocar el so en memoria baja o alta, aunque el factor decisivo es la ubicación del vector de interrupciones. En los sig. casos se supondrá que se trabaja con la memoria baja.

Para proteger que los programas del usuario no afecten al s.o. se utilizan mecanismos de protección como registros base y límite. Como en este caso sólo hay un usuario no se necesita el registro límite.

Todas las direcciones generadas por el programa del usuario se revisan con el registro base para que no sean menores a éste. Cuando el usuario desea entrar al área del sistema operativo como son los servicios de E/S, se le da la usuario una instrucción específica, que es una llamada al sistema, de modo que el so ejecute esta instrucción en modo sistema y después devolver el control al programa

del usuario.

Otro problema son la carga de los procesos de los usuarios. Aunque la primera dirección de la computadora es

00000, la primera dirección del programa del usuario es la siguiente del valor del registro base. Si durante la compilación se conoce la dirección base, entonces puede generarse código absoluto, que comenzará en la dirección base. Si cambia la dirección base entonces se tendrá que volver a compilar el programa. El compilador puede generar código relocalizable, y en este caso, el

enlace de direcciones se hace al momento de cargar al programa. Si la dirección base cambia solo hay que volver a cargar el código.

Un problema con este esquema es que el valor base debe quedar estático, ya que si se mueve o cambia entonces se deberán compilar y cargar de nuevo los programas. La dirección base solo puede cambiar cuando no se ejecuta ningún programa de usuario. Existen casos donde se desea cambiar el tamaño del so y por consiguiente la dirección base (por ej. Espacio de almacenamiento temporal para manejadores de dispositivos).

Hay dos formas para cambiar dinámicamente el tamaño del so:

1. Cargar el proceso de usuario en memoria alta, descendiendo el valor del registro base, en lugar de hacerlo desde la base hacia la memoria alta. La ventaja es que todo el espacio no utilizado se encuentra en la parte media , por lo que los procesos de usuario o de s.o. se extienden hacia la memoria no utilizada.

2. Postergar el enlace de direcciones hasta la ejecución. Este esquema requiere de apoyo de hardware. Al registro base, ahora de relocalización se le suma cada dirección generada por un proceso de usuario al momento de enviarlo a la memoria. Por ej. Si la base es 14000 y el usuario se dirige a la localidad 0, se relocalizará dinámicamente hacia la 14000.

El programa del usuario nunca ve las direcciones físicas reales. El programa ve direcciones lógicas. El hardware hace la correspondencia de direcciones lógicas en direcciones físicas.

El concepto de espacio de direcciones lógicas enlazado con un espacio de direcciones físicas separado es parte esencial de una administración de memoria adecuada.

Particiones múltiples en la memoria

En un sistema multiprogramado, varios procesos residen en la memoria y la CPU conmuta con ellos para ser ejecutados. La administración de la memoria debe asignar memoria a estos procesos que esperan en una cola de entrada para ser transferidos a la memoria.

Particiones fijas

Consiste en dividir la memoria en particiones de tamaño fijo. Cada partición puede contener un proceso. El nivel de multiprogramación está limitado al número de particiones. Cuando una partición está libre se selecciona un proceso de la cola de entrada y se carga en la partición libre; cuando el proceso termina, la partición queda disponible para otro. (IBM OS/360, so MTF). Este esquema se utilizó en los so de procesamiento por lotes.

Compiladores absolutos y reubicables

Los trabajos se traducían con compiladores y ensambladores absolutos en un principio, para ejecutarse en una partición específica. Si el trabajo estaba listo para ejecutarse y su partición estaba ocupada, aunque hubiera una disponible este trabajo tenía que esperar. Después surgieron los compiladores, ensambladores y cargadores reubicables, que producían programas que se podían ejecutar en cualquier partición disponible, aunque son mucho más complejos que los primeros.

Registro base y registro límite

En este sistema multiprogramado pueden existir varios procesos en la memoria al mismo tiempo, por lo que se deben proteger de no invadir sus zonas de memorias. Para ellos se utilizan los registros base y límite. El registro base contiene el valor de la menor dirección de memoria física; el registro límite contiene el intervalo de direcciones lógicas.

Cuando la CPU selecciona un proceso carga los valores de los registros base y límite. Cada dirección se compara con estos registros. De igual forma, cuando el usuario necesita llamar al so lo hace a través de llamadas al sistema.

Fragmentación

La fragmentación se presenta en todos los sistemas de cómputo y ocurren cuando las se dejan de utilizar particiones porque son muy pequeñas.

Particiones variables

Para dar solución a los problemas generados por las particiones fijas, los diseñadores de so decidieron que era mejor permitir que los trabajos ocuparan tanto espacio como necesitaran. En este esquema no hay desperdicio pues la partición de un trabajo es de su mismo tamaño y se atiende a éste conforme a sus propias necesidades. Sin embargo, comienza a haber un desperdicio ya que los huecos que van dejando los proceso después de ejecutarse son de tamaño más pequeños, y

así continúa hasta que quedan demasiado pequeños para contener otro proceso

Condensación de huecos

Cuando un trabajo termina de ejecutarse y forma un hueco, éste se revisa con sus vecinos para ver si hay huecos contiguos de tal forma que puedan formar un solo hueco más grande.

Compactación

La condensación no es suficiente porque aún pueden quedar huecos individuales distribuidos en el almacenamiento ocupando espacio. A veces, un trabajo puede requerir cierta cantidad de almacenamiento que no puede ser proporcionada por ningún hueco, pero sí por la suma de éstos. La compactación consiste en trasladar todas las áreas ocupadas a algún extremo de la memoria principal. Esto deja un hueco único de almacenamiento libre (eructo del almacenamiento).

Las desventajas de la compactación son que consume recursos del sistema; el sistema debe detener todas sus actividades mientras se realiza la compactación; puede implicar reubicar trabajos que estén en almacenamiento; de manera continua se debe esta compactando.

Estrategias de colocación en almacenamiento

Sirven para determinar en qué lugar del almacenamiento se deben colocar los programas.

− Primer ajuste. Consiste en asignar el trabajo en el primer hueco disponible con tamaño suficiente.

− Mejor ajuste. Consiste en colocar el proceso en el hueco más pequeño que tenga tamaño suficiente. Para ello se recorre toda la lista de huecos, a menos que ésta esté ordenada por tamaño.

− Peor ajuste. Consiste en asignar el trabajo en el hueco más grande posible.

4.4 Almacenamiento Virtual

Los sistemas de gestión de al memoria vistos hasta ahora están sujetos a la exigencia de que un programa, para ejecutarse, debe estar cargado en memoria principal en su totalidad. Sin embargo, no todas las partes del programa se ejecutan normalmente.

El programador diseña rutinas que sólo se ejecutan en determinadas circunstancias. Por ejemplo, aquellas que tratan situaciones de error. Con este enfoque no parece necesario que todo el programa esté cargado en memoria para poder ser procesado.

La memoria virtual es una técnica de gestión que, combinando hardware y software, permite la ejecución de programas parcialmente cargados en memoria real.

Esta forma de trabajar aporta ventajas importantes:

· Si los programas se pueden ejecutar por partes, la memoria lógica puede ser mayor que la real disponible.

· Puesto que cada programa ocupa menos memoria real, se puede elevar el índice de multiprogramación, y por tanto, la eficiencia del sistema.

· Al cargar menos cantidad de cada programa, se necesitan menos operaciones de entrada / salida para las operaciones de carga e intercambio de los mismos.

Las diferentes partes de un programa se van cargando en memoria a medida que se necesitan, y por ello, esta técnica debe considerar tres aspectos importantes:

· Carga. Las porciones del programa se cargan cuando se necesiten (petición de página) o bien se pueden cargar por adelantado (anticipación o prepaginación).

· Colocación. Los sistemas de memoria virtual que utilicen segmentación deben decidir al cargar un nuevo segmento, si lo hacen en el hueco más adecuado bien en el primeros posible.

· Sustitución. Lo normal será que toda la memoria real esté ocupada, y cuando se necesite cargar una nueva parte de un programa habrá que reemplazar alguna de las existentes. Es importante definir la selección de la parte a reemplazar.

Esta técnica es difícil llevar a la práctica. Los sistemas que la utilizan son complejos, pero a la vez, deben ser muy eficientes. De lo contrario, la ejecución de los programas puede empeorar notablemente.

4.5 Paginación

Cuando la memoria presenta el problema de la fragmentación, se utiliza la compactación para disponer de espacio libre contiguo, o bien, se utiliza la paginación, que permite que la memoria de un proceso no sea contigua y que a un proceso se le asigne memoria física donde quiera que ésta esté disponible.

La paginación evita el gran problema de acomodar trozos de memoria de tamaño variable en el almacenamiento auxiliar. Cuando es necesario intercambiar fragmentos de código o datos que residen en la memoria principal, hay que encontrarles espacio en el almacenamiento auxiliar. Este último también sufre de fragmentación.

Hardware

La memoria física se divide en bloques de tamaño fijo llamados marcos. La memoria lógica también se divide en bloques del mismo tamaño llamados páginas. Cuando un proceso se va a ejecutar, sus páginas se cargan desde el almacenamiento auxiliar en cualesquiera de los marcos disponibles. El almacenamiento auxiliar se divide en bloques de tamaño fijo del mismo tamaño que los marcos de la memoria.

A través de hardware, la dirección generada por la CPU se divide en dos partes: un número de página (p) y un desplazamiento de página (d). El número de página se utiliza como índice en una tabla de páginas, que contiene la dirección base para cada página de la memoria física. La dirección base se combina con el desplazamiento en la página para definir la dirección de memoria física que se envía a la unidad de memoria.

El tamaño de la página, al igual que el tamaño del marco, está definido por el hardware. Generalmente es una potencia de 2, entre 512 y 2048 palabras por página. Si el tamaño de la página es de 2&n, entonces los n bits de orden inferior de la dirección lógica representan el desplazamiento en la página y los bits restantes, de orden superior, indican el número de página.

Planificación a largo plazo

El planificador a largo plazo examina un proceso cuando llega para ejecutarse: es decir, examina su tamaño expresado en páginas y comprueba la memoria física disponible. Si hay suficientes marcos el planificador los asigna al proceso. La primera página del proceso se carga en uno de los marcos asignados y el número de marcos se registra en la tabla de páginas para este proceso; la sig. página se

carga en otro marco, etc.... Con los marcos no hay fragmentación externa pero sí interna, ya que el último marco asignado puede estar vacío. Ej. Un proceso de 72776 bytes, con páginas de 2048, necesita 35 páginas pero le faltan 1086 bytes, lo

que hace requerir otro marco más. Se podría tener un tamaño de página pequeño pero se genera más gasto en la administración.

¿Dónde definen los procesos a la tabla de páginas?

Implantación de la tabla de páginas

La manera más sencilla es implantar las páginas utilizando un conjunto de registros dedicados. Estos registros deben ser de alta velocidad (por hardware) para que la traducción de direcciones de páginas sea eficiente. Las instrucciones para cargar o modificar los registros de la tabla de páginas son privilegiadas. Actualmente las computadoras manejan tablas de registros de un millón de entradas, pero como los

registros de alta velocidad son muy caros se utiliza el siguiente esquema:

1. La tabla de páginas se conserva en memoria principal

2. Un registro base de tabla de páginas (PTBR, page−table base register) apunta a la tabla de páginas.

3. Para cambiar entre tablas de páginas entonces se modifica el registro

El problema con esta forma es el tiempo de acceso a una localidad de memoria:

1. Para llegar a i, primero se debe accesar el índice de tabla de página a través del PTBR para el número de página i. Esto requiere de acceso a la memoria.

2. Después hay que accesar al número de marco, mediante el desplazamiento en la página para llegar a la dirección real.

3. Por último podemos acceder al lugar deseado de la memoria.

En este esquema se necesitan dos accesos a la memoria: uno para la entrada de tabla de páginas y otro para la palabra. Este acceso puede ser muy lento.

La solución es utilizar una memoria especial, de tamaño pequeño, llamada registros asociativos de traducción con búsqueda anticipada (TLB, translation look−aside buffers). Estos registros son de memoria de muy alta velocidad y cada registro consta de dos partes: una clave y un valor, y cuando se presenta un elemento a los registros asociativos, se compara simultáneamente a todas las claves. Si el elemento se encuentra se devuelve el valor correspondiente. La búsqueda es muy rápida pero el hardware costoso.

Los registros asociativos contienen sólo algunas entradas de la tabla de páginas. Cuando la CPU genera una dirección lógica, su número de página se presenta a un conjunto de registros asociativos que contienen números de páginas y sus números de marco correspondientes. Si el número de página se encuentra en los registros asociativos, su número de marco está inmediatamente disponible.

Si el número de página no se encuentra en los registros asociativos, hay que efectuar una referencia de memoria ala tabla de páginas. Cuando se obtiene el número de marco se puede usar para tener acceso a la memoria. Además, se añaden los números de página y marco a los registros asociativos para que

se puedan localizar con rapidez en la siguiente referencia. El procesador INTEL 80486 tiene una tasa de aciertos del 98% con 32 registros asociativos.

Páginas compartidas

Otra ventaja de la paginación es compartir código común. El código reentrante (código puro) es un código que no se modifica a sí mismo (como partes de código de un software de aplicación). Si el código es reentrante, nunca cambia durante su ejecución, de manera que dos o más procesos pueden ejecutar el mismo código al mismo tiempo. Cada proceso tiene su propia copia de los registros y su almacenamiento para contener datos durante la ejecución. Se utiliza el código puro de programas de uso frecuente como compiladores, sistemas de ventanas, sistemas de BD, etc. para paginación compartida.

Protección

La protección de memoria en un entorno paginado se logra por medio de bits de protección asociados a cada marco. Estos bits se conservan en la tabla de páginas. Un bit define si una página es de sólo lectura o de lectura y escritura. Cada referencia a memoria pasa por la tabla de páginas para encontrar el marco correcto. Al mismo tiempo que se calcula la dirección física, pueden consultarse los bits de

protección que se calcula para la dirección física.

Dos perspectivas de la memoria.

Un aspecto importante en la paginación es la clara separación entre la perspectiva del usuario de la memoria y la memoria física real. El programa del usuario cree que la memoria es un espacio contiguo que contiene un solo programa. En realidad, este programa está dispersos por la memoria física en donde también corren más programas.

El S.O. controla la asignación de memoria para el usuario y para el so. Puesto que éste administra la memoria

física, debe estar la tanto de los detalles de la asignación de memoria física, además de los procesos de usuarios, como el espacio y transformaciones de direcciones lógicas y produciendo direcciones físicas.

4.6 Estrategias de reemplazo

El principio de optimalidad establece que para obtener un rendimiento óptimo, la página que se debe reemplazar es aquélla que tardará más tiempo en volver a ser utilizada. Esto es irrealizable porque no se puede predecir el futuro. La estrategia conocida como OPT o MIN intenta aproximarse al principio de optimalidad utilizando

técnicas de reemplazo de páginas que se aproximen al reemplazo óptimo de páginas.

Reemplazo de páginas aleatorio

Es una técnica sencilla para no gastar tiempo de la máquina y sin discriminar al usuario. Este esquema ya casi no se utiliza por que su enfoque aleatorio puede acertar o errar.

Reemplazo de páginas de primeras entradas−primeras salidas (PEPS).

Por cada página se registra el instante en que entró al almacenamiento primario. Cuando se necesita reemplazar una página, se escoge la que ha permanecido en el almacenamiento durante mayor tiempo. El problema es que se pueden reemplazar páginas muy utilizadas ya que si una página esta mucho tiempo en almacenamiento primario puede ser debido a que es muy utilizada (por ej. En los sistemas de tiempo compartido). Esta estrategia se puede implementar también con una cola PEPS: si llega una página se coloca al final de la cola, y las páginas que se reemplazan son las del principio de la cola.

Anomalías PEPS.

Pensar que entre más marcos de página se asignen a un proceso, menos fallas de página tendrá es erróneo. Belady, Nelson y Shedler descubrieron que ciertos patrones de referencias a páginas originan más fallas de página cuando aumenta el número de marcos de página asignados a un proceso.

Reemplazo de páginas de la menos recientemente utilizada (LRU)

Se reemplaza a la página que no ha sido utilizada durante el mayor tiempo. Este esquema exige que cada vez que se accese una página se registre esta referencia y esto podría requerir de mucho trabajo adicional. Para ello se utiliza una estructura de listas que contenga una entrada por cada marco de página ocupado. Cada vez que se hace referencia al marco de página, la entrada correspondiente a esa página se coloca al principio de la lista, y las entradas mas antiguas se llevan al final de la lista. Para reemplazar una página se selecciona la entrada final de la lista. Esto implica mucho trabajo.

Reemplazo de páginas de la menos frecuentemente utilizada (LFU)

Es una aproximación a LRU. Lo importante es la intensidad con la que se ha utilizado coda página: se reemplaza aquella que se ha usado menos frecuentemente o a la que se ha hecho referencia con menos frecuencia. Pero puede ser que la página recién llegada sea la menos utilizada o menos referenciada.

Reemplazo de páginas de la no utilizada recientemente (NUR)

Bits de hardware de página:

Bit de referencia: 0 si no se ha hecho referencia al página 1 se ha hecho referencia a la página.

Bit de modificación: 0 si la página no ha sido modificada 1 si la página ha sido modificada

Modificaciones de PEPS: reemplazo de páginas por reloj y reemplazo de páginas con segunda oportunidad

El problema de PEPS es que una página muy utilizada que ha permanecido en memoria mucho tiempo puede ser reemplazada. Esto se elimina reemplazando las páginas cuyos bits de referencia tengan 0. La variante de PEPS con segunda oportunidad examina el bit de referencia de la página más antigua : si vale 0, se selecciona para ser reemplazada. SI vale 1, se le asigna el valor 0 y la página se pasa al final de la lista y se considera como página nueva. La variación de reloj, dispone de una lista circular en donde el apuntador se desplaza por la lista circular.

Localidad

Es fundamental en las estrategias de administración de la memoria y significa que los procesos tienden a hacer referencia a la memoria en patrones no uniformes y altamente localizados. En la memoria es una propiedad empírica más que teórica. Nunca está garantizada, pero a menudo es muy probable. Por ej. en la paginación se observa que los procesos tienen a favorecer ciertos subconjuntos de sus páginas y que estas páginas tienden a ser adyacentes en el espacio de direcciones virtuales de un proceso. De hecho, la localidad es bastante razonable en los sistemas de cómputo, si se toma en cuenta como se escriben los programas y se organizan los datos :

1. Localidad temporal. Es probable que las localidades de memoria a las que se haya hecho referencia recientemente sean objeto de otra referencia en un futuro cercano:

a) ciclos

b) subrutinas

c) pilas

d) variables más utilizadas para cuenta y totalización

2. Localidad espacial. Las referencias a memoria tienden a estar concentradas, y una vez que se hace referencia a una localidad, es muy probable que se haga referencia a las localidades cercanas.

a) los recorridos de arreglos

b) ejecución secuencial de código

c) tendencia de los programadores a colocar las definiciones de variables afines

próximas unas de otras

Denning formuló la teoría del conjunto de trabajo (working set) en el comportamiento de un programa, con base a observaciones sobre la localidad.

Conjuntos de Trabajo

Denning desarrolló la teoría del conjunto de trabajo en el comportamiento de un programa. En general., un conjunto de trabajo es una colección de páginas a las que hace referencia activamente un proceso. Un programa se ejecuta eficientemente si se mantiene en el almacenamiento principal en conjunto de trabajo, si no hay excesiva paginación.

Se puede dar espacio suficiente a la mitad de las páginas de un proceso pero esto limitaría el número de procesos activos. Una política de administración del almacenamiento mediante conjuntos de trabajo intenta mantener en el almacenamiento primario a los conjuntos de trabajo de los programas activos.

El conjunto de trabajos de un proceso, W(t,w) en el tiempo t, es el conjunto de páginas a las que hizo referencia el proceso durante el intervalo de tiempo de proceso t−w a

Paginación por demanda.

Las páginas de un proceso deben cargarse hasta que son llamadas por un proceso en ejecución por:

a) Los resultados de la teoría de la computabilidad, indican que no se puede predecir con precisión la trayectoria de ejecución que seguirá un programa.

b) La paginación por demanda garantiza que las únicas páginas que se transfieren al almacenamiento principal son aquellas que requieren los procesos

c) El trabajo extra por decidir cuáles páginas se deben transferir a el almacenamiento principal es mínimo. Sin embargo, existe mucha espera mientras un proceso espera acumular sus páginas una por una.

Paginación anticipada.

Dedica una área de la memoria a los procesos y otra a anticipar y cargar páginas que se requerirán en el futuro.

a) Se reducen los tiempos de los procesos si se tomó o cargó la página correcta

b) Como el hardware se vuelve más económico las consecuencias de una mala decisión no son tan serias.

Liberación de páginas

Tamaño de páginas.

a) Cuanto menor sea el tamaño dela página, más páginas y marcos de páginas habrá y mayores tendrán que ser las tablas de páginas

b) Con tamaños de páginas grandes se paginan en el almacenamiento primario grandes cantidades de información a la que quizá nunca se haga referencia.

c) Si las páginas son pequeñas y muchas, la transferencia al disco es relativamente lento por lo que podría afectar al sistema.

4.7 Segmentación

La segmentación es el hecho de tener dos o más espacios independientes de direcciones lo cual es mucho mejor que tener uno solo. Por ejemplo, un compilador tiene muchas tablas, las cuales se integran al proceder la compilación; entre éstas está las siguientes:

1. El texto fuente que se resguarda para el listado impreso (en los sistemas de

procesamiento por lotes).

2. La tabla de símbolos, con los nombres y atributos de las variables

3. La tabla que contiene todas las constantes enteras y de punto flotante.

4. El árbol del léxico, con el análisis sintáctico del programa.

5. La pila que se utiliza para las llamadas a los procedimientos dentro del compilador.

Cada una de las cuatro primeras tablas crece en forma continua mientras que se lleva a cabo la compilación. La ultima crece y decrece en forma impredecible durante la compilación. En una memoria unidimensional estas cinco tablas tendrían asignados bloques adyacentes en el espacio de direcciones virtuales.

Considerando lo que ocurre si un programa tiene un número excepcionalmente grande de variables. El bloque de espacio de direcciones asignado para la tabla de símbolos podría ocuparse en su totalidad, aunque hubiera espacio en las demás tablas. Por supuesto, el compilador podría emitir un mensaje para indicar que la compilación no puede continuar debido a un número excesivo de variables, pero

esto no parece divertido, pues existe espacio en las demás tablas.

Otra posibilidad es tomar de las tablas con mucho espacio y dando este a tablas con poco. Este cambio se puede llevar a cabo, pero es análogo al control del propio proyecto: puede surgir al menos un conflicto, pero también representar un trabajo tedioso y poco reconfortante.

Lo que se necesita realmente es una forma de liberar al programador de la tarea de control de las tablas de expansión y contracción, de la misma forma que la memoria virtual elimina la preocupación por organizar el programa en una serie de proyectos.

Una solución directa y muy general consiste en dotar a la maquina de varios espacios independientes de direcciones, llamados segmentos. Cada segmento tiene una serie lineal de direcciones, desde 0 hasta cierto máximo. La longitud de cada segmento puede variar de 0 hasta un máximo permitido. Los distintos segmentos pueden tener, y de hecho tienen generalmente, longitudes distintas. Además, la

longitud de un segmento de la pila puede crecer si algo entra a la pila y decrecer si algo sale de ella.

Puesto que cada segmento constituye un espacio independiente de direcciones, los distintos segmentos pueden crecer o reducirse en forma independiente, sin afectar a las demás. Si la pila de cierto segmento necesita más espacio para crecer, lo puede tener, puesto que ya no posee lugar en su espacio de direcciones para poderlo ocupar. Por supuesto, un segmento puede ser utilizado en su totalidad, pero lo común es que los segmentos sean grandes, de modo que lo anterior ocurre muy

poco. Para especificar una dirección en esta memoria segmentada o bidimensional, el programa debe proporcionar una dirección con dos partes, un numero de segmento y una dirección de este.

Conviene enfatizar que un segmento es un objeto lógico, de lo cual tiene conciencia el programador, por lo que lo utiliza como tal. Un segmento puede contener un procedimiento, un arreglo, una pila, o bien un conjunto de variables escalares, pero, por lo general no contiene una mezcla de varios.

Una memoria segmentada tiene otras ventajas, además de hacer más sencilla la administración de las estructuras de datos que crecen o se reducen. Sí cada procedimiento ocupa un segmento independiente, con la dirección 0 como dirección inicial, el ligado independiente de los procedimientos compilados es mucho más sencillo. Después de compilar y ligar todos los procedimientos que constituyen un programa, una llamada al procedimiento en el segmento n utilizara la dirección (n,0)

para dirigirse a la palabra 0 (en donde está el dato).

Una memoria segmentada permite que cada tabla crezca o se reduzca en forma independiente de las demás. Si el procedimiento del segmento n se modifica y vuelve a compilar más tarde, no hay que cambiar los demás procedimientos (puesto que no se han modificado las direcciones iniciales), incluso en caso de que la nueva versión sea mas grande que la anterior. En una memoria unidimensional los

procedimientos se empaquetan muy cercanos entre sí, sin que exista espacio de direcciones entre ellos. En consecuencia el cambio de tamaño de un procedimiento puede afectar las direcciones iniciales de otros no relacionados con aquel. Esto a su vez, requiere modificar todos los procedimientos que llaman a cualquiera de los procedimientos desplazados, para poder incorporar sus nuevas direcciones iniciales. Si un programa contiene cientos de procedimientos; todo este procedimiento puede resultar costoso.

La segmentación también facilita el uso de procedimientos o datos compartidos entre varios procesos. Un ejemplo común es la biblioteca compartida. Es frecuente que las estaciones de trabajo modernas que ejecutan sistemas avanzados, con ventanas, tengan bibliotecas graficas de tamaño muy grande que se compilan en casi todos los programas. En un sistema segmentado, la biblioteca grafica se puede colocar en un segmento y compartirse entre varios procesos, sin necesidad de tenerla en el espacio de direcciones de cada proceso. Aunque también es posible tener bibliotecas compartidas sin los sistemas de con paginación pura, es mucho mas complejo. De hecho, estos sistemas simulan la segmentación.

Puesto que cada segmento es una unidad lógica de la que es consiente el programador, como por ejemplo un procedimiento, arreglo o pila, los distintos segmentos pueden tener distintos tipos de producción. Un segmento de procedimiento podría utilizarse solo para su ejecución, si se prohíben los intentos por leerlo o almacenar algo en el. Un arreglo de punto flotante podría utilizarse en modo lectura−escritura, pero no para su ejecución, en forma que los intentos por entrar a él se vieran frustrados. Tal protección es útil para determinar errores de programación.

Se debe comprender porque la protección tiene sentido en una memoria segmentada pero no en una memoria paginada unidimensional. En una memoria segmentado el usuario es consiente del contenido de cada segmento. Por lo general, un segmento no contendría a la vez un procedimiento y una pila sino

una cosa u la otra. Puesto que cada segmento sólo contiene un tipo de objeto, el segmento puede tener la segmentación adecuada para ese tipo particular.

El contenido de una página es en cierta medida incidental. El programador no se da cuenta de que la paginación se lleva a cabo. Aunque es posible colocar unos cuantos bits en cada dato de la tabla de páginas para determinar el tipo de acceso permitido en cada página, si se desea utilizar esto, el programador debería llevar un registro del lugar donde se encuentran las fronteras de la página dentro de su espacio de direcciones y eso es precisamente el tipo de administración para cuya eliminación se inventó la paginación. Puesto que el usuario de una memoria segmentada tiene la ilusión de que todos los segmentos está dentro de la memoria principal todo el tiempo (es decir, él puede dirigirse a ellas como si lo estuvieran), el puede proteger cada segmento en forma independiente, sin tener que preocuparse por su administración.

Implantación de la segmentación pura

La implementación de la segmentación difiere del caso de la paginación en un sentido esencial: las páginas tienen un tamaño fijo y los segmentos no

Considerando paginación segmentación

¿Necesita saber el programador si esta utilizando esta técnica?

No

Si

¿Cúantos espacios lineales de direcciones existen?

1

Muchos

¿Puede el espacio total de direcciones exceder el tamaño de la memoria física?

Si

Si

¿Pueden distinguirse los procedimientos y los datos, además de protegerse en forma independiente?

No

Si

¿Pueden adecuarse con facilidad las tablas con tamaños fluctuantes?

No

Si

¿Se facilita el uso de procedimientos compartidos

entre los usuarios?

No

Si

¿Para qué se inventó la técnica?

Para obtener un

espacio lineal de

direcciones sin

tener que adquirir

mas memoria

física

Para permitir que los

programas y datos

fueran separados en

espacios independientes

de direcciones y poder

proporcionar la

protección y uso de

objetos compartidos.

Control de acceso en los sistemas con segmentación

No se recomienda otorgar a cada proceso un acceso ilimitado a todos los segmentos del sistema. De hecho una de las cualidades de los sistemas con segmentación es el cuidadoso control del acceso que puede realizarse, lo cual se logra asignando a cada proceso ciertos derechos de acceso a cada segmento y, de hecho, denegando por completo el acceso a muchos segmentos.

Protección de almacenamiento con claves en una asignación de almacenamiento no contigua en un sistema de multiprogramación. Mientras la clave de protección de la UCP sea 2 (correspondiente al usuario B), el programa del usuario B sólo podrá hacer referencia a otros bloques de almacenamiento con la misma clave de protección. Estas claves está bajo el estricto control del sistema operativo y

solo pueden manipularse mediante instrucciones privilegiadas.

Numero de segmentos s

Desplazamiento d

Dirección virtual v=(s,d)

Formato de dirección virtual en un sistema de segmentación pura

Organización del almacenamiento virtual

Traducción de direcciones virtuales en un sistema con segmentación pura.

Tipos de acceso Abreviatura Explicación

Lectura

R

Este segmento se puede leer

Escritura

W

Este segmento se puede modificar

Ejecución

E

Este segmento es puede ejecutar

Adición

A

Se puede agregar información al final de

este segmento

Tipos de control de acceso.

Aquí se muestran los tipos de control de acceso más comunes empleados por los sistemas actuales.

Si un proceso tiene acceso para lectura a un segmento, podrá obtener cualquier información contenida en el segmento. Si lo desea el proceso puede realizar una copia del segmento.

Si el proceso tiene acceso para escritura a un segmento, entonces puede modificar cualquier parte del contenido del segmento y colocar información adicional. Si lo desea, el proceso puede destruir toda la información del segmento.

Un proceso con acceso para ejecución a un segmento puede ejecutar ese segmento como programa. El acceso de ejecución se suele denegar para segmentos de datos.

Traducción de direcciones con segmentación por correspondencia directa

Al igual que los sistemas paginados, existen muchas estrategias para realizar la traducción de direcciones con segmentación. Puede hacerse por correspondencia directa, asociativa o asociativa / directa combinada. Puede realizarse con memorias cache de tamaño suficiente para contener toda la tabla de correspondencia de segmentos o con almacenamientos asociativos parciales de tamaño suficiente para contener entradas para los segmentos a los que se ha hecho referencia más

recientemente.

Comportamiento de un sistema con segmentación

Una de las ventajas que tiene la segmentación sobre la paginación es que la segmentación es un concepto lógico no físico. En su forma más general los segmentos no está restringidos arbitrariamente a cierto tamaño, sino que pueden ser (dentro de limites razonables) tan grandes (o tan pequeños) como sea necesario. Un segmento correspondiente a una estructura a un arreglo tiene el tamaño del

arreglo. Un segmento correspondiente a una estructura de datos dinámica puede crecer o decrecer según lo haga la estructura de datos misma. Un segmento correspondiente a código de procedimientos generado por un compilador será tan grande como sea necesario para contener el código.

El comportamiento de segmentos es bastante sencillo comparado con el de paginación pura. Si un arreglo en

un sistema de paginación pura tiene tres paginas y media de longitud, entonces, en lugar de tener entradas simples para indicar "arreglo compartido", habrá que tener entradas individuales para cada una de las paginas donde reside el arreglo. El manejo de la pagina parcial puede ser difícil. La situación empeora con una estructura de datos dinámica. Cuando una estructura así crece y ocupa una nueva pagina, las indicaciones de comportamiento deberá ajustarse en el momento de la ejecución. En un sistema con segmentación cuando un segmento se declara como compartido las estructuras de datos pueden crecer o decrecer sin afectar el hecho lógico de que residen en segmentos compartidos.

5.1 Dispositivos de E/S y sus controladores

Una de las principales funciones de un sistema operativo es controlar todos los dispositivos de entrada y salida. Debe enviar comandos a los dispositivos, detectar las interrupciones y controlar todos los errores. También debe proporcionar una interfaz entre los dispositivos y el resto del sistema.

Los dispositivos externos que deben hacer E/S con las computadoras pueden clasificarse básicamente en 3 categorías.

− Dispositivos legibles por los humanos: apropiados para la comunicación con el usuario. Ejemplo: Una terminal sin cpu.

− Dispositivos legibles por la máquina. Adecuados para comunicarse con equipos

electrónicos como discos, unidades de cinta.

−Dispositivos de comunicación: Apropiados para comunicarse con dispositivos lejanos como los módems y adaptadores de línea digital.

Los dispositivos externos se pueden dividir en otras 2 categorías que son:

− Dispositivos de bloque: Son los que almacenan la información en bloques de tamaño fijo cada uno con su propia dirección. Los tamaños más comunes de bloques van desde 128 hasta 1024 bytes. Ejemplo Los discos duros.

− Dispositivos de carácter: Estos envían o reciben un flujo de caracteres sin sujetarse a una estructura de bloques. No se pueden utilizar direcciones de memoria ni métodos de búsqueda. Ejemplo: impresoras.

Cada tipo de dispositivo encarga la parte que depende de él para su manejo al manejador del dispositivo o unidad de E/S. El componente electrónico se llama controlador del dispositivo o adaptador y por lo general es una tarjeta de circuitos impresos que se inserta en la computadora. El componente mecánico es el propio dispositivo. La interfaz ente el controlador y el dispositivo entre el controlador y el dispositivo comúnmente es de bajo nivel.

Existen tres técnicas para realizar la E/S:

− E/S programada: El procesador emite una orden de E/S de arte de un proceso a un módulo de E/S; el proceso espera entonces a que termine la operación, antes de seguir.

− E/S dirigida por interrupciones: El procesador emite una orden de E/S de parte de un proceso, continua la ejecución de las instrucciones siguientes y es interrumpido por el módulo de E/S cuando este ha completado su trabajo. Las instrucciones siguientes pueden ser del mismo proceso, si no es necesario para este esperar la terminación de la E/S. En otro caso, el proceso se ve suspendido a la espera de la interrupción, mientras se realiza otro trabajo.

− Acceso dirigido a memoria (DMA): Un módulo de DMA controla el intercambio de datos entre la memoria principal y un módulo de E/S. El procesador envía una petición de transferencia de un bloque de datos al módulo de DMA y se ve interrumpido sólo cuando el bloque entero se haya transferido.

Dispositivos de E/S

Los dispositivos de E/S se pueden dividir de manera general en dos categorías: dispositivos de bloque y dispositivos de carácter. Un dispositivo de bloque es aquel que almacena la información en bloques de tamaño fijo, cada uno con su propia dirección. Los tamaños comunes de los bloques van desde 128 bytes hasta 1024 bytes. La propiedad esencial de un dispositivo de bloque es la posibilidad de leer o escribir en un bloque de forma independiente de los demás. En otras palabras, en todo momento, el programa puede leer o escribir en cualquiera de los bloques. Los discos son dispositivos de bloque.

Vista de cerca, no está bien definida la frontera entre los dispositivos que se manejan mediante direcciones de bloques y los que no. Todos coinciden en que un disco es un dispositivo que trabaja mediante direcciones de bloques, puesto que no importa dónde se encuentre el brazo, siempre es posible buscar otro cilindro y después esperar hasta que el bloque necesario rote debajo de la cabeza. Consideremos ahora una cinta magnética con bloques de 1 K bytes. Si la unidad de cinta recibe un comando para la lectura del bloque N, siempre puede rebobinar la cinta e ir hacia adelante hasta llegar a dicho bloque. Esta operación es análoga al proceso de búsqueda en un disco, excepto que tarda más tiempo. Además, podría ser posible rescribir un bloque a mitad de la cinta. Incluso en caso de que las cintas se pudieran utilizar como dispositivos de bloque, eso dificulta la distinción, pues por lo general no se emplean de esa forma.

El otro tipo de dispositivo de E/S es el dispositivo de carácter. Un dispositivo de carácter envía o recibe un flujo de caracteres, sin sujetarse a una estructura de bloques, No se pueden utilizar direcciones ni tienen una operación de búsqueda. Las terminales, ratones, impresoras de línea, cintas de papel, tarjetas perforadas, interfaces de una red y muchos otros dispositivos no parecidos a los discos son dispositivos de carácter.

Este esquema de clasificación no es perfecto. Algunos dispositivos no se ajustan a él. Por ejemplo, los relojes no tienen direcciones por medio de bloques. Lo único que hacen es provocar que interrumpan al microprocesador, intervalos bien definidos. Las pantallas mapeadas a memoria tampoco se ajustan. Aun así, el modelo de dispositivos de bloque y de carácter es lo bastante grafica para ser utilizado

como base para trabajar el software de sistemas operativos trabaja independiente del dispositivo de E/S. El sistema de archivos encarga la parte que depende del dispositivo a un software de menor nivel, llamado manejador del dispositivo.

Bus del sistema

La interfaz entre el controlador y el dispositivo es con frecuencia de muy bajo nivel. Un disco, por ejemplo, puede recibir un formato de 8 sectores de 512 bytes por pista. Sin embargo, lo que realmente sale de la unidad es un flujo de bits en serie, que comienza con un preámbulo, seguido de 4096 bits en un sector y por último una suma p Para verificación o un códificador o corrector de errores. El preámbulo se escribe al dar formato al disco y contiene el número de cilindro y sector, el tamaño de sector y otros detalles similares. La labor del controlador es convertir el flujo de bits en serie en un bloque de bytes.

Con frecuencia es posible separar las dos partes para tener un diseño o modelo electrónico general. Mencionamos esa distinción entre controlador y dispositivo porque el sistema operativo casi siempre trabaja con el controlador y no con el dispositivo. Casi todas las minicomputadoras utilizan el modelo de un bus para la comunicación entre la CPU y los controladores. Los grandes mainframes utilizan

con frecuencia otro modelo, con varios buses y computadoras especializadas en E/S llamadas canales de E/S

que toman cierta carga de E/S fuera de la CPU principal.

5.2 Organización de las funciones de E/S

Evolución de las funciones de E/S.

A medida que los sistemas informáticos han evolucionado, se ha producido una tendencia creciente en la complejidad y sofisticación individual.

Las etapas de evolución pueden resumirse de la manera siguiente:

1. El procesador controla directamente los dispositivos periféricos. Estos se nota en dispositivos simples controlados por microcontroladores.

2. Se añade un controlador o módulo de E/S. El procesador utiliza E/S programada sin interrupciones. El procesador parece aislarse de los detalles específicos de las interfaces con dispositivos externos.

3. Es muy similar a la segunda pero ahora se emplean interrupciones y el procesador no tiene que desperdiciar tiempo esperando a que se realice una operación de E/S, incrementando la eficiencia.

4. El módulo de E/S recibe control directo de la memoria, a través de DMA. Ahora puede mover un bloque de datos a la memoria o desde la misma sin que intervenga el procesador, excepto al principio y a final de la transferencia.

5. El módulo de E/S es mejorado para constituir un procesador separado con un conjunto de instrucciones especializado para realizar E/S. El procesador central ordena al procesador de E/S la ejecución de los programas de E/S en la memoria principal. El procesador de E/S va en busca de estas instrucciones y las ejecuta sin la intervención de la cpu. Esto permite a la cpu precisar que una secuencia de actividades de E/S se vea interrumpido sólo cuando haya terminado la secuencia entera.

6. El módulo de E/S posee su memoria local y es un computador independiente. Con esta arquitectura se pueden controlar un gran número de dispositivos de E/S con una participación mínima de la cpu. Un uso muy común de tal arquitectura ha sido el control con terminales interactivos. El procesador de E/S se encarga de la mayoría de las tareas implicadas en le control de los terminales.

Una de las funciones principales de un sistema operativo es el control de todos los dispositivos de entrada / salida de la computadora. Debe enviar comandos a los dispositivos, detectar las interrupciones y controlar los errores. También debe proporcionar una interfaz entre 1os dispositivos y el resto del sistema, sencilla y fácil de usar. En la medida de lo posible, la interfaz debe ser la misma para todos los dispositivos (independencia del dispositivo). El código de E/S representa una fracción significativa del sistema operativo. La forma de administrar la E/S por parte del sistema operativo es el tema de esta sección.

Este es un esbozo. En primer lugar, analizaremos algunos de los principios del hardware de E/S y después del software de E/S en general. El software de E/S se puede estructurar por capas, cada una de las cuales tiene una tarea bien definida por realizar. Analizaremos esas capas, con el fin de observar lo que realizan y la forma en que se complementan entre sí. Después de esta introducción, estudiaremos con detaIle tres dispositivos comunes de E/S: discos. relojes y terminales. Revisaremos los aspectos del hardware y software de cada dispositivo.

Principios de hardware de E/S

Distintas personas analizan de varias maneras el hardware de E/S. Los ingenieros electrónicos lo hacen en

términos de los chips, cables, fuentes de poder, motores y demás componentes físicos que conforman el hardware. Los programadores se fijan en la interfaz que se presenta al software (los comandos que acepta el hardware, las funciones que realiza y los errores que puede informar). En estos apuntes, nos interesaremos Por la programación de los dispositivos de E/S, no por su diseño, construcción mantenimiento; así, nuestro interés estará restringido a la forma de programar el hardware a su funcionamiento interno. Sin embargo, es frecuente que la programación de dichos dispositivos de E/S esté íntimamente ligada con

su operación interna.

5.3 Procesamiento de interrupciones (drivers)

SOFTWARE DE CONTROL DE ENTRADA / SALIDA (DRIVER)

Se define driver como "el software formado por un conjunto de rutinas y tablas que, formando parte del núcleo del sistema operativo, ejecutan y controlan todas las operaciones de entrada y salida sobre cualquier periférico conectado a la computadora, siendo particulares para cada dispositivo".

Un driver no es un proceso o tarea independiente gestionado por el sistema operativo, sino un conjunto de tablas en las que se aloja la información que caracteriza a cada periférico conectado a la computadora, y una serie de rutinas que controlan toda la gestión de los mismos y las informaciones que fluyen en un sentido o en otro. Se encuentran permanentemente alojados en memoria principal y requieren una elevada rapidez de ejecución sin formar parte del proceso de

usuario que los utilice.

El tratamiento por el núcleo de un sistema operativo de toda la información de entrada / salida desde, o a un periférico, se puede dividir en dos niveles para su estudio:

Tratamiento independiente del periférico

Está formado por el conjunto de rutinas que procesan información sin atender a las características propias del periférico.

Tratamiento dependiente del periférico

Es el conjunto de rutinas que el núcleo del sistema operativo ofrece para controlar el propio dispositivo periférico.

Funciones de un driver

· Entre las funciones que realiza un driver podemos citar las siguientes:

· Definir las características del periférico al resto del sistema operativo.

· Inicializar los registros asociados al periférico en el momento del arranque

(bootstrap) del sistema operativo.

· Habilitar y deshabilitar el dispositivo para un proceso.

· Procesar todas las operaciones de entrada / salida solicitadas por un proceso.

· Cancelar toda operaci6n de entrada / salida en el momento que sea necesario por

cualquier motivo.

· Procesar todas las interrupciones hardware generadas por el propio periférico.

· Tratar los errores y estado del dispositivo haciendo la correspondiente

comunicación al usuario.

Rutinas de un driver

Son los puntos de entrada al driver y pueden ser llamadas directamente por el núcleo del sistema operativo o por una interrupción hardware del dispositivo periférico. En general, en un driver podemos encontrar las siguientes rutinas:

· Inicialización. Es llamada por el núcleo del sistema operativo en la inicialización

del sistema. La rutina se encarga de Inicializar el dispositivo incluyendo la

información correspondiente en los registros de estado y operación del mismo.

· Atención de peticiones de entrada / salida. Esta rutina atiende todas las peticiones de los procesos de usuario para realizar operaciones de entrada / salida.

· Gestión de interrupciones. Es la rutina que maneja todas las interrupciones (Interrupt handler) del dispositivo. Toma el control cuando el dispositivo periférico origina una interrupción en el procesador.

· Cancelación de operaciones de entrada / salida. Es una rutina que da por finalizadas las operaciones de entrada / salida sobre el dispositivo cuando se produzca alguna circunstancia que le obligue a ello.

· Otras. Existen otras rutinas menos importantes, como pueden ser: el time−out que controla el tiempo de proceso de la operación y el Power−fail que actúa en el

arranque y al reanudarse el proceso después de un corte de la alimentación del

sistema.

Estructuras de datos de un driver

Las rutinas de un driver para dar un correcto servicio a las peticiones de entrada / salida necesitan para cada dispositivo una serie de datos que se encuentran en estructuras de datos en forma de tabla de manera que su composición depende del sistema operativo, aunque tiene forma y nombres similares a los siguientes:

Bloque de control del driver (BCD)

Es la representación del driver desde el punto de vista del sistema operativo. Contiene aquellos parámetros que son susceptibles de ser variados dinámicamente y aquellos que definen el tipo de dispositivo que puede ser atendido por el driver. Los datos que suele contener son:

− Dirección del siguiente BCD.

− Nombre del driver.

− Dirección de comienzo de los bloques de control de unidades (BCU) que controle el driver.

− Numero de unidades a servir.

− Dirección de comienzo de la rutina de inicialización del driver.

− Estado del driver (On/off line...).

− Dirección de comienzo de la cola de bloques de entrada / salida (BES) en modo

serie.

− Dirección del BES que esta siendo servido.

− Variables particulares del driver.

Bloque de control de la unidad (BCU)

Cada dispositivo físico se relaciona desde el punto de vista del sistema operativo como una unidad dentro del tipo al que le corresponda y es definido e identificado por el sistema operativo por medio de su BCU. En general, contiene los siguientes datos:

− Dirección del siguiente BCU del driver.

− Número de unidad.

− Número del vector de interrupción asociado.

− Dirección de la rutina de gestión de la interrupción.

− Dirección del puerto (port) de entrada / salida.

− Dirección del BCD al que pertenece.

− Dirección del PCB del proceso que tiene reservada esta unidad.

− Dirección del comienzo de la cola de bloques de entrada / salida (BES) en modo

paralelo.

− Dirección del BES que esta siendo servido.

− Características de la unidad.

− Variables particulares del drive.

Paquete de petición de entrada/salida (PES)

Cuando un proceso de usuario intenta hacer una operación de entrada/salida, el sistema operativo crea un paquete asociado a dicho proceso y a dicha petición para ser tratado por el driver. Este paquete se coloca en

una cola prioritaria para ser atendido por el driver al que va dirigido. Los datos que normalmente contiene son:

− Dirección del siguiente PES en la cola.

− Prioridad de la petición de entrada/salida.

− Proceso que ha lanzado la petición.

− Dirección donde devolver el resultado de la petición.

− Función a realizar (entrada o salida).

− identificador del dispositivo.

− identificador de la unidad.

− Dirección de la lista de parámetros de entrada de la llamada al sistema operativo.

Interrupciones Vectorizadas

Los sistemas operativos realizan diversidad de operaciones y están preparados para aceptar interrupciones que provienen de los dispositivos periféricos. Para poder reconocer que dispositivo ha sido el causante de una interrupción y poder darle el tratamiento adecuado, el sistema operativo destina parte de su memoria (la mas baja) para almacenar las direcciones de los ya mencionados manejadores de interrupciones asociados a cada dispositivo. Cada palabra almacenada que contiene la dirección de un manejador de interrupción se conoce con el nombre de vector de interrupción. Por tanto, el vector de interrupción es un numero que nos indica la palabra que contiene la dirección de una rutina que debe tratar una interrupción.

Un sistema operativo admite un máximo de vectores de interrupciones que es fijado en el momento de la generación del mismo.

5.4 Unidades de disco

Hardware y software en disco de RAM.

Los dos comandos principales son escribir en bloque y leer en bloque. Un disco de RAM es mucho más sencillo que un disco duro porque es una parte de la memoria que lee y escribe bloques pero que no tiene pérdidas de tiempo en localización o demora rotatoria. Se utiliza para almacenar datos o programas que se utilicen con frecuencia. Ref. En Minix, el sistema de archivos root /, está montado en el disco RAM, lo que permite montar y desmontar lo que se desee después (hasta al propio root). En los sistemas con sistemas de archivos montados el dispositivo raíz siempre está presente y en una localidad fija y los sistemas de archivos removibles (disco flexible) pueden montarse bajo una misma estructura de un solo sistema de archivos.

5.5 Terminales de carácter y gráficas

En general, se denomina terminal al conjunto formado por un teclado y una pantalla conectados a la computadora para introducir datos a través del primero y recibirlos a través de la segunda.

Los terminales pueden dividirse en dos categorías: los que se conectan a través del estándar R~232, y los mapeados en memoria.

Terminales RS−232

Constan de teclado y pantalla que transmiten bit a bit en serie. La velocidad de transmisión de estos bits viene dada en baudios (bits/seg.), siendo valores utilizados: 300, 1.200, 2.400, 4.800 y 9.600. Estos terminales se conectan a la computadora a través de un cable físico. Este cable en su entrada a la computadora termina en una tarjeta hardware o interfaz que a su vez se conecta al bus de dicha computadora.

Se conocen, generalmente, con el nombre genérico ITY.

Terminales mapeados en memoria

No necesitan línea de conexión a la computadora ya que están directamente conectados al bus del mismo. En estos terminales, el teclado se conecta directamente al bus y es independiente de la pantalla.

6.1 Concepto de sistemas de archivos

El sistema operativo maneja la información de los dispositivos masivos como discos o cintas a través del concepto abstracto de "sistema de archivos".

Organización del sistema de archivos

Un sistema de cómputo puede almacenar la información de varias maneras físicas, con dispositivos diferentes entre sí y de características propias. Por ello el s.o. debe ofrecer una manera clara, sencilla y fácil para manejar el almacenamiento de la información.

El S.O. debe ocultar las características propias de cada dispositivo y definir una unidad lógica de almacenamiento: el archivo.

El sistema de archivos está formado por dos estructuras bien definidas : el conjunto de archivos, y la estructura de directorios.

Concepto de archivo.

Conjunto de información relacionada que ha sido definido por su creador. Los archivos contienen datos numéricos, alfabéticos, alfanuméricos o binarios. Pueden tener formato libre o rígido. En general, es una secuencia de bits, bytes, líneas o registros, cuyo significado está definido por su creador y el usuario.

Un archivo tiene nombre y las referencias a él se hacen a través de dicho nombre. Además tiene propiedades como tipo, hora de creación, nombre o id del creador, longitud,... La información del archivo puede ser: programas fuente, programas objeto, datos numéricos, texto, registros de nómina, imágenes, gráficas, sonido, video. Cada archivo tiene una estructura definida de acuerdo a su tipo. Así, un archivo de texto es una secuencia de caracteres, un archivo fuente es una serie de subrutinas y funciones, un archivo objeto es una secuencia de bytes organizados en bloques de registros para el cargador.

La función del S.O. es reconocer los formatos y estructura de los archivos, pero cuántos más tipos de formatos de archivo conozca el so más grande será éste.

Ej. Archivo de texto y archivo ejecutable vs. Archivo cifrado Archivos en Unix

6.2 Directorios: contenido y estructura

Estructura de los directorios

Los archivos se representan mediante entradas en un directorio de dispositivo o tabla de contenido del volumen. El directorio guarda información del nombre, ubicación, tamaño y tipo de todos los archivos que le pertenezcan.

La estructura de directorios nace por la necesidad de aumentar la información o números de usuarios en un sistema de cómputo. Además, evita que el usuario tenga que manejar información diferente de acuerdo a cada dispositivo, de manera tal que el usuario trabaje con directorios lógicos.

Muchos sistemas tienen dos estructuras para la información: el directorio del dispositivo y los directorios de los archivos. El primero define características físicas de las entradas de los archivos (dónde se ubica, longitud, cómo está asignado...). El segundo es una organización lógica de los archivos en todos los dispositivos y describen características lógicas (nombre del archivo, tipo, dueño, protección de acceso...). Una entrada del directorio de archivos puede apuntar a la entrada del directorio del dispositivo o puede duplicar dicha información.

Entradas de directorio:

Nombre del archivo

Tipo de archivo

Ubicación

Tamaño

Posición actual

Protección

Recuento de uso

Hora, fecha e id de proceso

Esta información puede almacenarse de 16 hasta 1000 bytes.

El directorio del sistema es como una tabla de símbolos que traduce los nombres de los archivos a sus entradas en el directorio, con la capacidad de insertar entradas, eliminar, buscar por su nombre y mostrar las entradas de un directorio.

Una lista lineal de entradas del directorio requiere una búsqueda secuencial para encontrar una entrada determinada. Esto es sencillo de programar pero tarda mucho la ejecución.

Lista de entradas secuencial

Lista de entradas secuencial ordenada

Árbol binario o enlazado

Tabla de dispersión

Operaciones sobre archivos

Un archivo es un tipo de datos abstracto. El conjunto de operaciones mínimas sobre archivos son:

Creación de archivos

Escritura de Archivos

Lectura

Reposicionamiento de archivos

Eliminación

Las operaciones adicionales son : editar, añadir, copiar, renombrar

Métodos de acceso

Se refiere al proceso de cargar la información en la memoria o bajarla al disco o unidad de almacenamiento. Existen varias formas de acceder a la información.

Acceso secuencial

La información en el archivo se procesa en orden, un registro tras otro. P. ej. los programas de edición, o archivos en unidades de cinta.

Acceso directo.

Se basa como modelo en el disco. Se lee o escribe en bloques. Se utiliza para grandes cantidades de información, como las bases de datos.

Acceso por medio de índices.

El índice contiene apuntadores a los distintos bloques. Para encontrar una entrada en el archivo primero se busca en el índice y luego se usa el apuntador para acceder directamente al archivo y encontrar la entrada deseada. El índice generalmente se queda en la memoria por lo que no es conveniente para grandes cantidades de datos. Para ello se puede construir un índice para el archivo índice.

Organización de estructuras de directorio

En esencia, el directorio es una tabla de símbolos, donde el sistema operativo encuentra un archivo utilizando su nombre simbólico.

Las operaciones de los directorios son:

Búsqueda

Creación de archivos

Eliminación de archivos

Listado del directorio

Copia de seguridad

Directorios de un solo nivel

Es la estructura más sencilla, por ej. el directorio del dispositivo. Todos los archivos se encuentran en el mismo directorio. Hay limitaciones cuando aumenta el número de archivos o hay más de un usuario: pueden nombrar de la misma forma a un archivo.

Directorio 2 de niveles

Es una propuesta para crear un directorio para cada usuario. Ej. de como se accesa a un directorio de un usuario cuando se introduce el login La desventaja es que no permite compartir archivos entre varios usuarios o tener acceso al directorio de otro usuario.

Directorio con estructura de árbol.

Consiste en extender el directorio de dos niveles a varios subniveles. Cada directorio contiene un bit para definir la entrada de un archivo o bien la entrada a un subdirectorio.

Nombres de rutas: absolutos y relativos

Directorios de grafo acíclico.

Ej. Un archivo compartido por dos programadores A través del grafo acíclico, o sea un grafo sin ciclos, se puede compartir un archivo, ya que es una generalización natural del esquema de directorios con estructura de árbol. Con varios usuarios, que tr4abajan en equipo, los archivos compartidos se colocan en un directorio, cada directorio de archivos de usuarios de los integrantes del equipo contiene, como

subdirectorio, al directorio compartido. Para lograr lo anterior se utilizan los enlaces (links).

Directorios de grafo general

El problema con el grafo acíclico es asegurar que no existan ciclos.

6.3 Archivos: organización, manipulación, bloqueo y almacenamiento en buffers

Asignación del espacio de almacenamiento

El sistema operativo es el responsable de plasmar en disco los archivos que crean los usuarios suministrándoles el espacio necesario. Ante una petición para crear un archivo de N bits, el subsistema deberá comprobar si existe suficiente espacio disponible y, a continuación podrá optar por dos estrategias:

− Asignar N bits contiguos de espacio en disco.

− Almacenar el archivo en trozos no contiguos.

Veamos las características de cada una de estas opciones.

Asignación contigua

Según este criterio, se coloca cada archivo en un grupo de bloques contiguos del disco. Cada entrada del directorio del disco contendrá, además del nombre del archivo y otros datos, la dirección del bloque inicial del archivo y él número de bloques que ocupa. Por ejemplo, un archivo que ocupa 5 bloques empezando en el número 3, se encontrara situado en los bloques 3, 4, 5, 6 y 7. De esta forma, el acceso a un bloque n+1

partiendo del bloque n no exigirá, en la mayoría de los

casos, movimiento del brazo del disco. Si el acceso a un archivo se hace de forma secuencial, el subsistema de archivos recordara en cada momento cual será el siguiente bloque a leer o escribir y el tiempo de búsqueda serán prácticamente nulo. La colocación contigua también permite el acceso directo de forma eficaz ya que cualquier bloque puede ser accedido rápidamente partiendo de la dirección inicial del archivo.

El inconveniente mayor en este caso es el de la fragmentación externa que se produce, va que en un momento dado puede producir un numero de huecos grande con suficiente espacio total en disco para un nuevo archivo, pero sin que exista ninguno capaz de contenerlo.

Otro inconveniente de esta técnica es el de tener que conocer por anticipado el tamaño del archivo, lo que suele llevar a los usuarios al normal sobredimensionado de los mismos con la consiguiente perdida de espacio en disco. Si a pesar de ello el archivo se quedase pequeño, seria necesario definir uno mayor y trasvasar la información con la sobrecarga que esto lleva consigo.

Algunos sistemas pequeños que utilizan discos flexibles los compactan periódicamente con el fin de agrupar los huecos libres en uno de mayor tamaño. Esta solución no es valida para aquellos sistemas que utilicen discos de gran capacidad pues agrupar sus archivos exige mover grandes cantidades de información y esto puede afectar al proceso de otros trabajos del sistema.

Otros sistemas de computadoras de gran tamaño utilizan una técnica de asignación contigua que permite ampliar los archivos después de su creación. Al definir un archivo se deben definir dos valores, el tamaño del mismo y la cantidad de espacio a añadirle en cada crecimiento (llamada extensión). Cuando el archivo necesite ser ampliado, el subsistema le ira asignando nuevas extensiones que no tienen por que ser contiguas a aquel. El ultimo bloque del espacio inicial apuntara a la primera extensión y el ultimo de esta apuntara a la siguiente. El subsistema limitara él numero máximo de extensiones utilizables.

Asignación enlazada

Este método es de asignación no contigua, en el que cada archivo es una lista enlazada de bloques del disco que pueden estar en cualquier dirección del mismo.

En el directorio del disco cada entrada de archivo contendrá, además del nombre del mismo y otros datos, un apuntador al primer bloque de la cadena (en algunos sistema también se apunta al ultimo bloque). A partir de este primer bloque, cada uno de los siguientes contiene un apuntador al que le sigue.

Ahora, para crear un archivo, el subsistema le asignara un bloque físico libre y añadirá una entrada en el directorio. A medida que van aumentando las necesidades de espacio del archivo, el subsistema le ira enlazando bloques libres.

Esta técnica de asignación no provoca fragmentación externa, ya que todos los bloques son del mismo tamaño y por tanto utilizables para las nuevas necesidades de cualquier archivo. Estos podrán crecer mientras existan bloques libres. El usuario no necesita declarar el tamaño del archivo y no puede por ello acaparar espacio innecesario en el disco.

El encadenamiento de bloques facilita el tratamiento de archivos secuenciales, pero en el caso de los accesos directos la solución no puede ser mas desfavorable, ya que siempre hay que seguir la cadena de apuntadores. La asignación enlazada es también bastante vulnerable, ya que la perdida, por cualquier razón (error de

hardware o software) de un apuntador de bloque, significa la perdida del archivo. Con el fin de paliar este peligro se utiliza a veces el doble encadenamiento, es decir, cada bloque apunta al que le sigue y al que le

precede. En este caso, aumenta la sobrecarga del método y la ocupación de espacio ocasionado por los apuntadores empieza a ser considerable.

Asignación indexada

El origen de los principales inconvenientes de la asignación enlazada esta en la distribución desordenada de los apuntadores de los bloques de un archivo. Por ello, se hace inviable el acceso directo. La asignación indexada trata de resolver este problema agrupando todos los apuntadores en un bloque de índices. Cada archivo tendrá su propio bloque de índices y su dirección quedara reflejada en el directorio del disco al crear el archivo.

No se produce fragmentación externa y se agiliza notablemente el acceso directo a los datos. El acceso a cualquier bloque de un archivo solo requiere dos acceso a disco, uno al bloque de índices y otro al bloque deseado.

Este método no es apropiado para archivos de pequeño tamaño. Para archivos grandes los apuntadores pueden llenar el bloque de índices, en cuyo caso, el subsistema le encadenará otro bloque de índices o bien podrá recurrir a definir un bloque de índices a bloques de índices.

6.4 Archivos secuenciales

Una vez creado un archivo estando su información almacenada en el soporte correspondiente, supongamos que queremos utilizar su información. ¿Cómo se puede acceder desde el punto de vista lógico a la información de uno de sus bloques?

Independientemente, hasta cierto punto, de la forma en que se haya almacenado la información en el soporte físico, se podrá acceder a ella según un esquema lógico secuencial, directo o bien casi directo por medio de parte de la información contenida en el archivo. El subsistema de archivos del sistema operativo define que formas de acceso lógico permite y que métodos de acceso soporta.

Un método de acceso es un conjunto de rutinas y tablas que posibilitan acceder a la información de los archivos según un esquema lógico determinado. Algunos sistemas ofrecen un solo método de acceso a sus usuarios. Otros, por el contrario permiten utilizar varios y será el usuario el que decida cual usar.

Acceso secuencial

Este método permite el acceso a los registros de un archivo en un orden preestablecido, del primero al ultimo y de uno en uno. El efecto es como si el archivo, para el usuario, estuviera organizado por registros consecutivos que solo permiten llegar a uno cualquiera de ellos pasando previamente por los anteriores.

Las rutinas del método de acceso secuencial mantendrán un apuntador al siguiente registro lógico a acceder. Una operación de lectura o escritura, lee o escribe el registro y avanza el apuntador al siguiente.

Este método requiere que los registros lógicos se almacenen siguiendo el orden en el que serán accedidos para su tratamiento, es decir, debe coincidir el orden lógico de los registros y su orden físico.

Los registros pueden estar ordenados según un campo a través del cual serán accedidos siendo este campo el denominado comúnmente como campo clave.

El método de acceso secuencial es sencillo de programar y permite accesos rápidos a la información siempre que se realicen según el orden establecido. Además es el método que aprovecha mejor el soporte de información.

La actualización de la información contenida en los registros es complicada, pues es necesario recolocar el contenido total del archivo una vez añadidos los cambios ocurridos. Por ello las operaciones de actualización se realizan de forma masiva (gran número de registros a la vez), copiando el archivo en otro.

Si se requiere acceder a los registros lógicos del archivo según el contenido de otro campo distinto del campo clave, este método obliga a crear un nuevo archivo cuya secuencia guarda el nuevo orden. Ambos archivos tendrán consiguientemente la misma información ordenada de distinta forma. En estos casos, manejar distintas claves provoca el almacenamiento de información redundante. Es un método muy utilizado por su simplicidad, especialmente en archivos con poca información.

6.5 Archivos no−secuenciales

Acceso directo

Este método permite el acceso directo a cualquier parte del archivo, es decir, no es necesario pasar por la información anterior para acceder a un determinado registro.

Sólo puede existir este tipo de acceso en aquellos soportes que por su naturaleza lo permitan. Es el caso de los discos, mientras las cintas no pueden tener este tipo de acceso. El usuario ve el archivo como un conjunto de registros individualizados (numerados con respecto al inicio) a los que puede acceder en cualquier orden. Para ello, ante la petición de un registro determinado el software del método de acceso calcula la dirección del bloque físico que lo contiene y accede a él directamente.

Este cálculo es necesario puesto que el usuario utiliza direcciones relativas de registro (respecto al inicio del archivo), desconociendo el posicionamiento de los registros en el disco. Por ello, si el usuario pide un registro contenido en el bloque 4 de un archivo que comienza en el bloque 75, el subsistema accederá al 78.

Esta sería la forma más sencilla de trabajar por parte del sistema, pero existen otras formas de hacerlo ya que en ocasiones se desconoce el número relativo correspondiente a cada registro y por tanto será necesario relacionar el contenido del mismo con la posición relativa que ocupará dentro del archivo. Una forma de hacerlo es mantener una tabla que contenga las claves y sus respectivos bloques. Para atender una petición, el método de acceso buscará en dicha tabla la clave solicitada y obtendrá la dirección del bloque correspondiente. Puede también obtenerse un método de transformación de clase que convierta el contenido de uno o varios campos de cada registro en la posición correspondiente. Este último suele producir sinónimos que son registros que a través del método original la misma

posición, con lo que obligan a la existencia de una zona de excedentes que será siempre de acceso secuencial puesto que los registros llegarán en cualquier orden.

El algoritmo utilizado en la figura sustituye cada letra de la clave por su número de orden en el abecedario y suma sus valores para calcular el bloque asociado.

Este método de acceso es, por sus características, el más adecuado para acceder con rapidez a grandes cantidades de información.

Acceso directo indexado

En este caso se construye un índice o tabla de las relaciones entre las claves y sus bloques físicos para cada archivo. La localización de un registro se realizará accediendo primero a ese índice y con la dirección del bloque correspondiente a la clave solicitada, se alcanzará el bloque adecuado.

Para archivos grandes se puede utilizar un índice maestro o índice de índices. El maestro apunta al índice

secundario que contiene la clave y éste directamente al bloque físico.

Para agilizar los accesos, normalmente se mantiene el índice de mayor nivel en memoria principal. Estos índices se crearán durante la carga del archivo.

.

7.1 Necesidad de la seguridad

Conforme los sistema de computación se han vuelto mas complejos y sus aplicaciones se han difundido mas, también ha crecido la necesidad de proteger su integridad. En un principio, la protección se concibió como una añadidura a los sistemas operativos de multiprogramación para que varios usuarios, en los que no se podía confiar, pudieran compartir con seguridad un mismo espacio lógico, como un directorio de archivos o un mismo espacio físico, como la memoria. Se ha

evolucionado hasta los modernos conceptos de protección para aumentar la seguridad de cualquier sistema complejo que utilice recursos compartidos.

La protección se refiere a un mecanismo para controlar el acceso de programas, procesos o usuarios a los recursos definidos por un sistema de computación. Este mecanismo debe ofrecer un medio para especificar los controles que se impondrán, así como la manera de aplicarlos. Debemos establecer la diferencia entre protecci6n y seguridad, la cual representa una medida de la confianza en que se conservara la integridad del sistema y sus datos. El control de la seguridad es un tema mucho mas

amplio que la protección y en este libro solo lo trataremos brevemente.

Hay varias razones para proporcionar la protección, la mas obvia es la necesidad de evitar una violación mal intencionada de una restricción de acceso. Sin embargo, tiene una importancia más general la necesidad de asegurar que cada componente de un programa activo utilice los recursos sólo de manera consistente con las políticas establecidas para su uso. Este es un requisito absoluto para un sistema seguro.

La protección puede mejorar la seguridad detectando errores latentes en las interfaces entre los subsistemas componentes. Sí en las primeras etapas se detecta un error en las interfaces, muchas veces se puede evitar la contaminación de un sistema integro debido al mal funcionamiento de un subsistema.

Un recurso desprotegido no puede defenderse del uso (o abuso) no autorizado o de un usuario incompetente. Un sistema orientado a la protección ofrece un medio para distinguir entre el uso autorizado y el no autorizado.

7.2 Métodos de seguridad: protección, acceso y autenticación

Confidencialidad

Consiste en proteger la información contra la lectura no autorizada explícitamente. Incluye no solo la protección de la información en su totalidad, sino también las piezas individuales que pueden ser utilizadas para inferir otros elementos de información confidencial.

La confidencialidad se refiere a la capacidad de mantener un documento electrónico inaccesible a todos, excepto a una lista determinada de personas.

Integridad.

Es necesario proteger la información contra la modificación sin el permiso del dueño. La información a ser protegida incluye no solo la que está almacenada directamente en los sistemas de cómputo, además se deben de considerar elementos menos obvios como respaldos, documentación, registros de contabilidad del sistema, tránsito en una red, etc.

Esto comprende cualquier tipo de modificaciones:

Causadas por errores de hardware y/o software.

Causadas de forma intencional.

Causadas de forma accidental

Cuando se trabaja con una red, se debe comprobar que los datos no fueron modificados durante su transferencia.

Autenticidad.

En cuanto a telecomunicaciones se refiere, la autenticidad garantiza que quien dice ser "X" es realmente "X", es decir, se deben implementar mecanismos para verificar quien está enviando la información.

La autenticidad se refiere a la capacidad de determinar si una lista determinada de personas han establecido su reconocimiento y/o compromiso sobre el contenido del documento electrónico. El problema de la autenticidad en un documento tradicional se soluciona mediante la firma autógrafa.

Mediante su firma autógrafa, un individuo, o varios, manifiestan su voluntad de reconocer el contenido de un documento, y en su caso, a cumplir con los compromisos que el documento establezca para con el individuo.

No − repudiación.

Ni el origen ni el destino en un mensaje deben poder negar la transmisión. Quien envía el mensaje puede probar que, en efecto, el mensaje fue enviado y viceversa.

Disponibilidad de los recursos y de la información.

De nada sirve la información si se encuentra intacta en el sistema pero los usuarios no pueden acceder a ella , por lo que se deben proteger los servicios de cómputo de manera que no se degraden o dejen de estar disponibles a los usuarios de forma no autorizada. La disponibilidad también se entiende como la capacidad de un sistema para recuperarse rápidamente en caso de algún problema.

Consistencia.

Es asegurar que el sistema siempre se comporte de la forma esperada, de tal manera que los usuarios no encuentren variantes inesperadas.

Control de acceso a los recursos.

Consiste en controlar quien utiliza el sistema o cualquiera de los recursos que ofrece y como lo hace.

Auditoria.

Consiste en contar con los mecanismos para poder determinar que es lo que sucede en el sistema, que es lo

que hace cada uno de los usuarios, los tiempos y fechas de dichas acciones.

En cuanto a los dos últimos puntos es de extrema importancia, cuando se trata de los derechos de los usuarios, diferenciar entre espiar y monitorear a los mismos, la ética es algo que todo buen administrador debe conocer y poseer.

Finalmente, todos estos servicios de seguridad deben ser tomados en cuenta al elaborar las políticas y procedimientos de una organización para evitar pasar por alto cuestiones importantes como las que señalan dichos servicios. De esta manera es posible sentar de forma concreta y clara los derechos y límites de usuarios y administradores. Sin embargo, antes de realizar cualquier acción para lograr

garantizar estos servicios, es necesario asegurarnos de que los usuarios conozcan sus derechos y obligaciones (es decir, las políticas), de tal forma que no se sientan agredidos por los procedimientos organizacionales.

Incidentes de Seguridad

Generalmente los incidentes de seguridad se asocian con una intrusión ilegal a un sistema. A pesar de que este es el incidente de seguridad más frecuentemente reportado, no es el único. Hay otros tipos de incidentes de seguridad, por ejemplo, los que tienen que ver con la disponibilidad de los recursos del sistema o con la integridad de la información del mismo.

En algunos casos no es posible recuperar la información, pero sí se puede evitar que se vuelva a presentar este problema.

Intrusos

A veces, la manera en que detectamos una intrusión es bastante evidente y desagradable. Por ejemplo, si notamos que nuestro servidor de web presenta una página alterada, el monitor muestra mensajes obscenos, o si el intruso borra toda la información del sistema para no dejar rastro. Otras veces no es tan sencillo. Incluso muchas veces se da de manera casual. Por ejemplo, al descubrir que un usuario

que se dedica a la historia del arte está compilando programas de dinámica de fluidos, lo cual es muy extraño. Otro ejemplo es el famoso caso de Cliff Stoll, quien detectó un error de 75 centavos en la contabilidad del sistema.

No hay una receta a seguir para descubrir a un intruso. Sin embargo, damos una lista de hechos que pueden dar un indicio de una posible intrusión:

1.Cuentas nuevas no creadas por el administrador

2.Cuentas distintas de root con uid=0

3.Una considerable baja en el rendimiento del sistema

4.Ejecución de programas de seguridad (tales como crack, cops, etc) por usuarios que no son del staff de seguridad

5.Usuarios trabajando en horarios no acostumbrado

6.Usuarios ejecutando programas que no son de su área

7.Accesos al sistema de direcciones poco comunes

8.Bitácoras cortadas o borradas

9.Nombres de archivos o directorios que empiecen con: / (diagonal), .. (dos puntos), uno o varios espacios en blanco, etc.

7.3 Estrategias de seguridad

Las primeras recomendaciones para estrategias de seguridad son:

Establecer contacto con los proveedores de equipos.

Acreditar un punto de contacto técnico ante el MxCERT

Se propone un curso básico de PGP.

Se propone instalar PGP con el objeto de analizar el impacto de uso a nivel de usuario, para de ahí partir hacia una campaña de uso global.

Se recomienda altamente la instalación de Secure Shell, con el objeto de mantener sesiones seguras, en los principales servidores universitarios; así como promover el uso del programa cliente por medio de una campaña

Es importante la instalación de TCP Wrappers, para monitorear y restringir los accesos a servicios de red, en los principales servidores universitarios, aunque en algunos de ellos se debe de analizar primero desde que equipos se reciben accesos.

En base a las recomendaciones de herramientas de seguridad que emiten organismos internacionales, como lo son el CERT, el FIRST, el CIAC; y algunos nacionales como lo son el ASC−UNAM; es conveniente actualizarlas constantemente.

7.4 Virus y gusanos

Virus

Programa de ordenador que se reproduce a sí mismo e interfiere con el hardware de una computadora o con su sistema operativo (el software básico que controla la computadora). Los virus están diseñados para reproducirse y evitar su detección. Como cualquier otro programa informático, un virus debe ser ejecutado para que funcione: es decir, el ordenador debe cargar el virus desde la memoria del ordenador y seguir sus instrucciones. Estas instrucciones se conocen como carga activa

del virus. La carga activa puede trastornar o modificar archivos de datos, presentar un determinado mensaje o provocar fallos en el sistema operativo.

Existen otros programas informáticos nocivos similares a los virus, pero que no cumplen ambos requisitos de reproducirse y eludir su detección.

Estos programas se dividen en tres categorías:

− Caballo de Troya: Aparenta ser algo interesante e inocuo, por ejemplo un juego, pero cuando se ejecuta puede tener efectos dañinos.

− Una bomba lógica libera su carga activa cuando se cumple una condición determinada, como cuando se

alcanza una fecha u hora determinada o cuando se teclea una combinación de letras.

− Un gusano se limita a reproducirse, pero puede ocupar memoria de la computadora y hacer que sus procesos vayan más lentos.

Gusanos

Un gusano es similar a un virus, excepto por el hecho de que el gusano se reproduce a sí mismo como un programa independiente, en vez de infectar y ocultarse en otros programas. Por ejemplo, el virus Jerusalén infecta cierto programa A.EXE, luego el programa B.EXE, etc.. Pero si WORM.EXE encuentra otra computadora conectada a una red, hará una copia de WORM.EXE en ella.

Los gusanos prevalecen más en los grandes sistemas multitareas que en un PC.

El incidente Robert Morris− InterNet, que detuvo 6,000 computadoras en 1988, fue un gusano. Se diseño para una difusión lenta y probar la seguridad del sistema de cómputo, pero un error en el gusano hizo que se difundiera con rapidez, de modo que cada copia del gusano se volvía a reproducir en las demás computadoras de la red, hasta que la reproducción creció de manera geométrica.

También es importante saber que WORM tiene otro significado en computación, relacionado con la tecnología de los discos ópticos: Write Once, Read Many (una sola escritura y varias lecturas).

Los gusanos Internet de 1988 no eran el primer programa de su tipo. Aquí está una descripción abreviada de otros gusanos históricos.

El término " gusano " viene realmente de una historia de la ciencia ficción llamada el jinete de la onda de choque escrito por Juan Brunner en 1975. En cortocircuito, la historia está sobre un gobierno totalitario que controle a sus ciudadanos a través de una red de ordenadores de gran alcance. Un combatiente de la libertad infesta esta red con un programa llamado una " solitaria " que fuerza el gobierno cerrar la red, de tal modo destruye su base de la potencia.

Entre esto y el gusano 1988, los programas del gusano están consiguiendo un mal nombre. Sin embargo, los primeros programas del gusano fueron diseñados realmente para facilitar un uso mejor de una red. El primer programa que podría razonablemente llamado un gusano fue escrito en 1971 por Bob Thomas. Este programa estaba en respuesta a las necesidades de los reguladores de tráfico

aéreo y ayudaría a notificar a operadores de cuando el control de cierto aeroplano se movió a partir de un ordenador a otro. En actualidad, el programa, llamado "camilla" viajó solamente de la pantalla a la pantalla en la red que visualizaba el mensaje " que soy camilla! Cójame si usted puede! " El programa de la camilla no se reprodujo. Después de esto, varios otros programadores intentaron sus manos en los programas similares, pero la idea muerta gradualmente hacia fuera en un par de meses.

En los años 80 tempranos, el choque de Juan y Jon Hepps del centro de investigación de Palo Alto de Xerox comenzaron a experimentar con programas del gusano. (éste era la primera vez que el gusano del término fue aplicado realmente a esta clase de código.) Los 5 gusanos desarrollados, cada uno de los cuales fue diseñado para preformar tareas provechosas alrededor de la red. Algunos gusanos eran absolutamente simples, por ejemplo el gusano de la ciudad, que viajó simplemente a través de los avisos de fijación de la red.

Otros gusanos eran absolutamente listos y complejos, por ejemplo el gusano del "vampiro". Este gusano era ocioso durante el día, pero en la noche, se

aprovecharía de los ordenadores en gran parte ociosos y los aplicaría a las

tareas complejas que necesitaron la potencia de proceso adicional. En el

amanecer, salvaría el trabajo que había hecho hasta ahora, y la marcha lenta

entonces convertida, esperando la tarde próxima.

Sin embargo, aunque estos programas eran intrínsecamente útiles, llegó a ser

evidente, que el gusano podría ser herramienta peligrosa si se utilizan correctamente. Esto fue demostrado suficientemente cuando uno de los gusanos de Xerox funcionó incorrectamente durante la noche. Cuando la gente llegó al trabajo el día siguiente, y encontró que se habían estrellado los ordenadores a través del centro de investigación. Por otra parte, cuando intentaron reiniciar las máquinas, el gusano que funcionaba incorrectamente se estrelló inmediatamente en ellas otra vez. Una "vacuna" tuvo que ser escrita para evitar que el gusano estrelle los sistemas.

Los gusanos difieren de los virus al ser autocontenidos y tienen que

ser eliminados. Mientras los virus son como parásitos de otros

programas o archivos y en la mayoría de los casos pueden ser reparados.

7.5 Tolerancia a fallas

Sistemas tolerantes a fallos.

Se utilizan en sistemas donde se pueda perder información debido a un mal funcionamiento de los mismos. Este aspecto es muy importante en los sistemas de control y supervisión en tiempo real. Existen mecanismos que ante situaciones de mal funcionamiento consiguen recuperar y controlar el entorno, protegiendo fundamentalmente la información. Este tipo de mecanismos se basa en redes de

dos o más computadoras conectadas entre sí de manera que, ante el mal funcionamiento de una de ellas, este se pondrá en situación de inactivo, tomando el control cualquiera de los otros que estén conectados.

Bibliografía:

− Deitel, Sistemas Operativos Addison−Wesley, 1993

− Stallings William, Sistemas Operativos Prentice Hall, 1997

−Tanenbaum Andrew, Sistemas Operativos Prentice Hall, 1997

−Silberschatz A., Galvin P., Sistemas Operativos Addison Wesley, 1999

Ligas en el Web:

− http://www.cs.arizona.edu/people/bridges/oses.html

− http://java.sun.com

− http://jos.org/

− http://osweb.8m.com

− http://www.tunes.org/Review/OSes.html